COMPSCI 311: Introduction to Algorithms
Lecture 21: Intractability: Intro and Polynomial-Time Reductions

Dan Sheldon
University of Massachusetts Amherst

Algorithm Design

- Formulate the problem precisely
- Design an algorithm
- Prove correctness
- Analyze running time

Sometimes you can’t find an efficient algorithm.

Example: Network Design

- **Input**: undirected graph $G = (V, E)$ with edge costs
- **Minimum spanning tree problem**: find min-cost subset of edges so there is a path between any $u, v \in V$.
  - $O(m \log n)$ greedy algorithm
- **Minimum Steiner tree problem**: find min-cost subset of edges so there is a path between any $u, v \in W$ for specified terminal set $W$.
  - No polynomial-time algorithm is known.

Example: Subset Sum / Knapsack

- **Input**: $n$ items with weights, capacity $W$
- **Goal**: maximize total weight without exceeding $W$
  - $O(nW)$ pseudo-polynomial time algorithm (DP)
  - No polynomial time algorithm known!

Tractability

- Working definition of efficient: polynomial time
  - $O(n^d)$ for some $d$.
- Huge class of natural and interesting problems for which
  - We don’t know any polynomial time algorithm
  - We can’t prove that none exists
- **Goal**: develop mathematical tools to say when a problem is hard or “intractable”

Preview of Landscape: Classes of Problems

- **P**: solvable in polynomial time
- **NP**: includes most problems we don’t know about
- **EXP**: solvable in exponential time
**NP-Completeness**

- **NP-complete**: problems that are “as hard as” every other problem in NP.
- A polynomial time algorithm for any NP-complete problem implies one for every problem in NP.

**P ≠ NP?**

Two possibilities:

- We don’t know which is true, but think $P \neq NP$
- $1M prize if you can find out (Clay Institute Millenium Problems)

**Outline**

- **Goal**: develop technical tools to make this precise

**Polynomial-Time Reduction**

- Problem $Y$ is polynomial-time reducible to Problem $X$
  
  ```
  solveY(yInput)
  Construct xInput // poly-time
  foo = solveX(xInput) // poly # of calls
  return yes/no based on foo // poly-time
  ```

- Statement about relative hardness. Suppose $Y \leq_P X$, then:
  1. If $X$ is solvable in poly-time, so is $Y$
  2. If $Y$ is not solvable in poly-time, neither is $X$

**Clicker**

Suppose that $Y \leq_P X$. Which of the following can we infer?

A. If $X$ can be solved in polynomial time, then so can $Y$.
B. If $Y$ cannot be solved in polynomial time, then neither can $X$.
C. Both A and B.
D. Neither A nor B.
Consider the following graph $G$. Which are true?

A. The white vertices are a vertex cover of size 7.
B. The black vertices are an independent set of size 3.
C. Both A and B.
D. Neither A nor B.

**Independent Set $\leq_P$ Vertex Cover**

**Claim:** INDEPENDENT SET $\leq_P$ VERTEX COVER. **Reduction:**

- On INDEPENDENT SET instance $(G, k)$
- Construct VERTEX COVER instance $(G, n - k)$
- Return Yes iff $\text{solveVC}((G, n - k)) = \text{YES}$

**Correctness** for Yes output:

- Suppose $G$ has independent set $S$ with $\geq k$ nodes
- Then $T = V - S$ is a vertex cover with $\leq n - k$ nodes
- The algorithm correctly outputs Yes

**Correctness** for No output:

- Suppose $G$ has no independent set $S$ with $\geq k$ nodes
- Then there is no vertex cover with $T$ with $\leq n - k$ nodes, otherwise $S = V - T$ is an independent set with $\geq k$ nodes.
- The algorithm correctly outputs No
Aside: Decision versus Optimization

- For intractability and reductions we will focus on decision problems (Yes/No answers)
- Algorithms have typically been for optimization (find biggest/smallest)
- Can reduce optimization to decision and vice versa. Discuss.

Reduction to General Case: Set Cover

Problem. Given a set $U$ of $n$ elements, subsets $S_1, \ldots, S_m \subset U$, and a number $k$, does there exist a collection of at most $k$ subsets $S_i$ whose union is $U$?

- Example: $U = \{A, B, C, D, E\}$ is the set of all skills, there are five people with skill sets:
  $S_1 = \{A, C\}, \quad S_2 = \{B, E\}, \quad S_3 = \{A, C, E\}$
  $S_4 = \{D\}, \quad S_5 = \{B, C, E\}$

- Find a small team that has all skills. $S_1, S_4, S_5$

Theorem. \textsc{VertexCover} $\leq_P$ \textsc{SetCover}

Clicker

Vertex Cover is a special case of Set Cover with:

A. $U = V$ and $S_e$ is the two endpoints of $e$ for each $e \in E$.
B. $U = E$ and $S_e$ is the set of edges incident to $v$ for each $v \in V$.
C. $U = V \cup E$ and $S_v$ is the set of neighbors of $v$ together with edges incident to $v$ for each $v \in V$.

Intractability: quiz 4

Given the universe $U = \{1, 2, 3, 4, 5, 6, 7\}$ and the following sets, which is the minimum size of a set cover?

\begin{itemize}
  \item A. 1
  \item B. 2
  \item C. 3
  \item D. None of the above.
\end{itemize}

Reduction Strategies

- Reduction by equivalence
- Reduction to a more general case
- Reduction by “gadgets”

Intractability: quiz 4

Reduction of Vertex Cover to Set Cover

Theorem. \textsc{VertexCover} $\leq_P$ \textsc{SetCover}

Reduction.

- Given \textsc{VertexCover} instance $(G, k)$
- Construct \textsc{SetCover} instance $(U, S_1, \ldots, S_m, k)$ with $U = E$, and $S_e$ is the set of edges incident to $v$
- Return Yes iff $\text{solveSC}((U, S_1, \ldots, S_m, k)) = \text{Yes}$

Proof

- Straightforward to see that $S_1, \ldots, S_{k}$ is a set cover of size $\ell$
- This implies the algorithm correctly outputs Yes if $G$ has a vertex cover of size $\leq k$ and No otherwise
- Polynomial # of steps outside of $\text{solveSC}$
- Only one call to $\text{solveSC}$