COMPSCI 311: Introduction to Algorithms

Lecture 19: Intractability: Polynomial-Time Reductions

Marius Minea

University of Massachusetts Amherst

slides credit: Dan Sheldon

Algorithm Design

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

Sometimes you can't find an efficient algorithm.

Example: Graph Searches / 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 set of nodes W (called terminals)
  - No polynomial-time algorithm is known.
  - but: for $W = V$: spanning tree $O(m \log n)$
  - for $W = \{u, v\}$: shortest path $O(m \log n)$

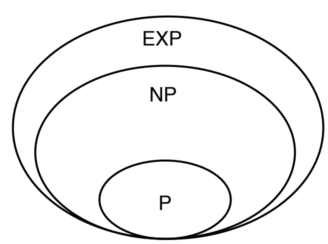
Example: Knapsack Problem

- **Input**: n items with costs and weights, capacity W
- **Goal**: select items to maximize total cost without exceeding W
- **Fractional knapsack**: select fraction in $[0, 1]$ of each item
  - $O(n \log n)$ greedy algorithm
- **0-1 Knapsack**: select all or none of each item
  - $O(nW)$ pseudo-polynomial time algorithm
  - No polynomial time algorithm known!
  - (Also none known for real weights)
- **Subset-Sum Problem** (Knapsack, no values)
  - maximum weight $\leq W$: $O(nW)$ pseudo-polynomial
  - weight sum $= W$: no polynomial algorithm known

Tractability

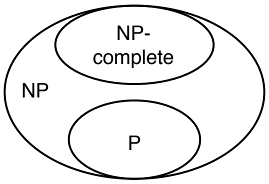
- Working definition of efficient: polynomial time
  - $O(n^d)$ for some d.
- Huge class of natural and interesting problems for which
  - We don't know any polynomial time algorithm
  - We can't prove that none exists
- **Goal**: develop mathematical tools to say when a problem is hard or "intractable"

Preview of Lansdscape: Classes of Problems



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

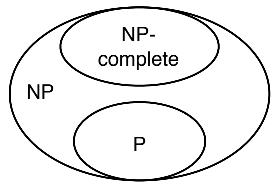
NP-Completeness



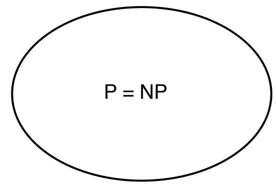
- NP-complete: problems that are "as hard as" every other problem in NP.
- A polynomial time algorithm for any NP-complete problem implies one for *every problem in NP*

P $\neq$ NP?

Two possibilities:



P $\neq$ NP

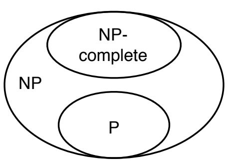


P $=$ NP

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

Outline

Goal: develop technical tools to make this precise



- **Polynomial-time reductions**: one problem is "as hard as" another (what does this mean?)
- **Define NP**: characterize mystery problems
- **NP-completeness**: some problems in NP are "as hard as" all others

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
```

- . . . if any instance of Problem Y can be solved using
  1. A polynomial number of standard computational steps
  2. A polynomial number of calls to a black box that solves problem X
- **Notation** $Y \leq_P X$

Intractability: quiz 1

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

A. If X can be solved in polynomial time, then so can Y.

B. X can be solved in poly time iff Y can be solved in poly time.

C. If X cannot be solved in polynomial time, then neither can Y.

D. If Y cannot be solved in polynomial time, then neither can X.

slide credit: Kevin Wayne / Pearson

Polynomial-Time Reduction

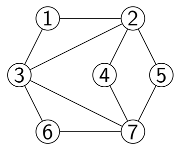
- $Y \leq_P X$

```
solveY(yInput)
  Construct xInput           // poly-time
  foo = solveX(xInput)       // poly # of calls
  return yes/no based on foo // poly-time
```

- Statement abut relative hardness
  1. If $Y \leq_P X$ and $X \in P$, then $Y \in P$
  2. If $Y \leq_P X$ and $Y \notin P$ then $X \notin P$
- 1: design algorithms, 2: prove hardness

First Reduction: Independent Set and Vertex Cover

Given a graph $G = (V, E)$,



- $S \subset V$ is an **independent set** if no nodes in S share an edge. Examples: $\{3, 4, 5\}, \{1, 4, 5, 6\}$.
- $S \subset V$ is a **vertex cover** if every edge has at least one endpoint in S. Examples: $\{1, 2, 6, 7\}, \{2, 3, 7\}$

IND. SET. Does G have independent set of size at least k?
VERTEX COVER. Does G have a vertex cover of size at most k?

Intractability: quiz 3

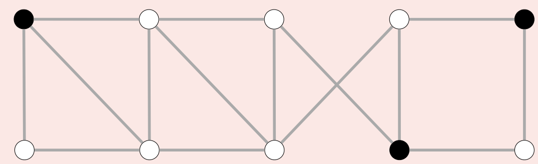
Consider the following graph G. Which are true?

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.



slide credit: Kevin Wayne / Pearson

Independent Set and Vertex Cover

- **Claim**: S is independent set if and only if $V - S$ is vertex cover.

1. S independent set $\Rightarrow V - S$ vertex cover
   - Consider any edge (u, v)
   - S independent $\Rightarrow$ either $u \notin S$ or $v \notin S$
   - I.e., either $u \in V - S$ or $v \in V - S$
   - $\Rightarrow V - S$ is a vertex cover
2. $V - S$ vertex cover $\Rightarrow S$ independent set
   - Similar.

Independent Set $\leq_P$ Vertex Cover

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

- On INDEPENDENT SET instance $\langle G, k\rangle$
- Construct VERTEX COVER instance $\langle G, n - k\rangle$
- Return YES iff `solveVC`$(\langle G, n - k\rangle) =$ 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

Vertex Cover $\leq_P$ Independent Set

- **Claim**: VERTEX COVER $\leq_P$ INDEPENDENT SET
- **Reduction**:
  - On VERTEX COVER input $\langle G, k\rangle$
  - Construct INDEPENDENT SET input $\langle G, n - k\rangle$
  - Return YES if `solveIS`$(\langle G, n - k\rangle) =$ YES
- **Proof**: similar

Decision versus Optimization

- For intractability and reductions we focus on decision problems (YES/NO answers)
- Algorithms have typically been for optimization (find biggest/smallest)
- Can reduce optimization to decision and vice versa.
- If we can solve MAXINDSET(G) and result is S then INDSET(G, k) has solution iff $k \leq |S|$
- solve MAXINDSET(G) by solving INDSET(G, k), $k = 1, \ldots, n$ or faster by doing binary search

Reduction Strategies

- Reduction by equivalence (Vertex Cover and Indpendent Set)
- Reduction to a more general case
- Reduction by “gadgets” (e.g., Satisfiability)

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

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?

A. 1

B. 2

C. 3

D. None of the above.

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$S_a = \{ 1, 4, 6 \}$	$S_b = \{ 1, 6, 7 \}$
$S_c = \{ 1, 2, 3, 6 \}$	$S_d = \{ 1, 3, 5, 7 \}$
$S_e = \{ 2, 6, 7 \}$	$S_f = \{ 3, 4, 5 \}$

Reduction of Vertex Cover to Set Cover

Reduction.

- Given $\textsc{Vertex Cover}$ instance $\langle G, k\rangle$
- Construct $\textsc{Set Cover}$ instance $\langle U, S_1, \ldots, S_m, k\rangle$ with $U = E$, and $S_v =$ the set of edges incident to v
- Return $\textsc{Yes}$ iff `solveSC`$(\langle U, S_1, \ldots, S_m, k\rangle) = \textsc{Yes}$

Proof

- Straightforward to see that $S_{v_1}, \ldots, S_{v_\ell}$ is a set cover of size ℓ if and only if $v_1, \ldots, v_\ell$ is a vertex cover of size ℓ
- This implies the algorithm correctly outputs: $\textsc{Yes}$ if G has a vertex cover of size $\leq k$ and $\textsc{No}$ otherwise
- Polynomial # of steps outside of `solveSC`
- Only one call to `solveSC`

A Bad Reduction

Reduction

- Given $\textsc{Vertex Cover}$ instance $\langle G, k\rangle$
- Construct $\textsc{Set Cover}$ instance $\langle U, S_0, S_1, \ldots, S_m, k\rangle$ as before but with additional set $S_0 = U$
- Return $\textsc{Yes}$ iff `solveSC`$(\langle U, S_0, S_1, \ldots, S_m, k\rangle) = \textsc{Yes}$

Analysis

- “$\textsc{Yes}$” instance: G has a vertex cover of size $\leq k$
  - U has a set cover of size $\leq k$
  - Output is $\textsc{Yes}$—correct
- “$\textsc{No}$” instance: G does not have a vertex cover of size $\leq k$
  - U *does* have a set cover of size $\leq k$ for $k \geq 1$
  - Output is $\textsc{Yes}$—incorrect