


# NP and Computational Intractability: Polynomial-time Reductions

CMSC 451, Summer 2009



Some slides by Kevin Wayne.  
Copyright © 2009 Pearson-Addison Wesley.  
All rights reserved.

## Second Half of this Course

Monday	Tuesday	Wednesday	Thursday	Friday
27	28	29	30	31
<p><b>Chapter 11: Interval Scheduling</b></p> <p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 3 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 3 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 3 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 3 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 3 Due</p> <p>12:00 AM Office Hours</p>
3	4	5	6	7
<p><b>Chapter 9: NP and PSPACE-completeness</b></p> <p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 4 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 4 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 4 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 4 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 4 Due</p> <p>12:00 AM Office Hours</p>
10	11	12	13	14
<p><b>Chapter 10: Hamiltonian Paths</b></p> <p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 5 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 5 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 5 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 5 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 5 Due</p> <p>12:00 AM Office Hours</p>
17	18	19	20	21
<p><b>Chapter 12: Hamiltonian Paths</b></p> <p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 6 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 6 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 6 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 6 Due</p> <p>12:00 AM Office Hours</p>	<p>12:00 AM Class Project Check-out</p> <p>12:00 AM Class Project Check-in</p> <p>12:00 AM Homework 6 Due</p> <p>12:00 AM Office Hours</p>

## Homework

- Homework 3 due now.
- Homework 4 (part 1): 8.1, 8.14

## Survey

- How's the class going so far?
- What do you like about the class?
- What don't you like about the class?
- Do you have any suggestions?

### Algorithm Design Patterns and Anti-Patterns

**Algorithm design patterns.**

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Duality.
- **Reductions.**
- Local search.
- Randomization.

**Algorithm design anti-patterns.**


- **NP-completeness.**
- PSPACE-completeness.
- Undecidability.

**Ex.**

- $O(n \log n)$  interval scheduling.
- $O(n \log n)$  closest pair of points.
- $O(n^2)$  edit distance.
- $O(n^2)$  bipartite matching.


**Algorithm design anti-patterns.**

- $O(n^*)$  algorithm unlikely.
- $O(n^*)$  certification algorithm unlikely.
- No algorithm possible.



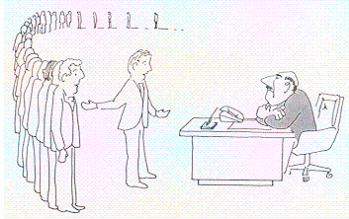
I can't find an efficient algorithm, I guess I'm just too dumb.

from *Computers and Intractability*  
by Garey and Johnson, 1979



I can't find an efficient algorithm because no such algorithm is possible!

from *Computers and Intractability* by Garey and Johnson, 1979



I can't find an efficient algorithm, but neither can all these famous people.

from *Computers and Intractability* by Garey and Johnson, 1979

## 8.1 Polynomial-Time Reductions

Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?

A working definition. [von Neumann 1953, Godel 1956, Cobham 1964, Edmonds 1965, Rabin 1966]  
Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality testing	Factoring

Classify Problems

**Desiderata.** Classify problems according to those that can be solved in polynomial-time and those that cannot.

**Provably requires exponential-time.**

- Given a Turing machine, does it halt in at most  $k$  steps?
- Given a board position in an  $n$ -by- $n$  generalization of chess, can black guarantee a win?

**Frustrating news.** Huge number of fundamental problems have defied classification for decades.

**This chapter.** Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one **really hard** problem.

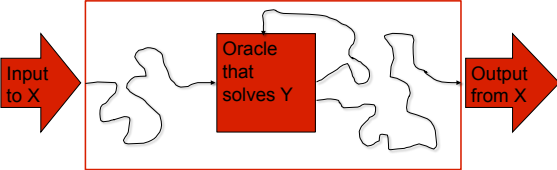
Polynomial-Time Reduction

**Desiderata'.** Suppose we could solve  $X$  in polynomial-time. What else could we solve in polynomial time?

don't confuse with reduces from

**Reduction.** Problem  $X$  **polynomially reduces to** problem  $Y$  if arbitrary instances of problem  $X$  can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem  $Y$ .



### Polynomial-Time Reduction

**Desiderata.** Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

**Reduction.** Problem X **polynomially reduces to** problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

**Notation.**  $X \leq_p Y$ .

**Remarks.**

- We pay for time to write down instances sent to black box  $\Rightarrow$  instances of Y must be of polynomial size.
- Note: Cook reducibility.

don't confuse with reduces from  
computational model supplemented by special piece of hardware that solves instances of Y in a single step  
in contrast to Karp reductions

### Polynomial-Time Reduction

**Purpose.** Classify problems according to **relative** difficulty.

**Design algorithms.** If  $X \leq_p Y$  and Y can be solved in polynomial-time, then X can also be solved in polynomial time.

**Establish intractability.** If  $X \leq_p Y$  and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.

**Establish equivalence.** If  $X \leq_p Y$  and  $Y \leq_p X$ , we use notation  $X =_p Y$ .

Example: X = bipartite matching, Y = max flow/ min cut  
Contrapositive of previous statement  
up to cost of reduction

### Reduction By Simple Equivalence

**Basic reduction strategies.**

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Interpret result in the context of the original problem

### Problem Types

**Optimization Version:** A problem with an optimal solution as the answer.

**INDEPENDENT SET:** Given a graph  $G = (V, E)$ , find the maximum cardinality subset of vertices  $S \subseteq V$  such that for each edge at most one of its endpoints is in S.

**Decision Version:** A problem with a yes or no answer.

**INDEPENDENT SET:** Given a graph  $G = (V, E)$  and an integer k, is there a subset of vertices  $S \subseteq V$  such that  $|S| \geq k$ , and for each edge at most one of its endpoints is in S?

Optimization problems are equivalent to decision problems

### Self-Reducibility

**Decision problem.** Does there exist a vertex cover of size  $\leq k$ ?

**Search problem.** Find vertex cover of minimum cardinality.

**Self-reducibility.** Search problem  $\leq_p$  decision version.

- Applies to all (NP-complete) problems in this chapter.
- Justifies our focus on decision problems.

**Ex:** to find min cardinality vertex cover.

- (Binary) search for cardinality  $k^*$  of min vertex cover.
- Find a vertex v such that  $G - \{v\}$  has a vertex cover of size  $\leq k^* - 1$ .  
 - any vertex in any min vertex cover will have this property
- Include v in the vertex cover.
- Recursively find a min vertex cover in  $G - \{v\}$ .

delete v and all incident edges

### Independent Set

**INDEPENDENT SET:** Given a graph  $G = (V, E)$  and an integer k, is there a subset of vertices  $S \subseteq V$  such that  $|S| \geq k$ , and for each edge at most one of its endpoints is in S?

**Ex.** Is there an independent set of size  $\geq 6$ ? Yes.

**Ex.** Is there an independent set of size  $\geq 7$ ? No.

### Vertex Cover

**VERTEX COVER:** Given a graph  $G = (V, E)$  and an integer  $k$ , is there a subset of vertices  $S \subseteq V$  such that  $|S| \leq k$ , and for each edge, at least one of its endpoints is in  $S$ ?

**Ex.** Is there a vertex cover of size  $\leq 4$ ? Yes.  
**Ex.** Is there a vertex cover of size  $\leq 3$ ? No.

● vertex cover

### Vertex Cover and Independent Set

**Claim.** VERTEX-COVER  $\equiv_p$  INDEPENDENT-SET.  
**Lemma.**  $S$  is an independent set iff  $V - S$  is a vertex cover.

○ independent set  
● vertex cover

### Vertex Cover and Independent Set

**Claim.** VERTEX-COVER  $\equiv_p$  INDEPENDENT-SET.  
**Lemma.** We show  $S$  is an independent set iff  $V - S$  is a vertex cover.  
**Proof.**

$\Rightarrow$

- Let  $S$  be any independent set.
- Consider an arbitrary edge  $(u, v)$ .
- $S$  independent  $\Rightarrow u \notin S$  or  $v \notin S \Rightarrow u \in V - S$  or  $v \in V - S$ .
- Thus,  $V - S$  covers  $(u, v)$ .

$\Leftarrow$

- Let  $V - S$  be any vertex cover.
- Consider two nodes  $u \in S$  and  $v \in S$ .
- Observe that  $(u, v) \notin E$  since  $V - S$  is a vertex cover.
- Thus, no two nodes in  $S$  are joined by an edge  $\Rightarrow S$  independent set. ■

### Vertex Cover and Independent Set

**Claim.** VERTEX-COVER  $\equiv_p$  INDEPENDENT-SET.  
**Proof.** ( $|V|=n$ )

INDEPENDENT-SET  $\leq_p$  VERTEX-COVER .  
 Suppose we have an oracle to solve vertex cover.  
 Ask the oracle if there is a vertex cover of size at most  $n-k$ .  
 Return this answer as the answer for an independent set of size  $k$ .

VERTEX-COVER  $\leq_p$  INDEPENDENT-SET.  
 Suppose we have an oracle to solve independent set.  
 Ask the oracle if there is an independent set of size at least  $n-k$ .  
 Return this answer as the answer for a vertex cover of size  $k$ . ■

### Reduction from Special Case to General Case

**Basic reduction strategies.**

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Create new, translated problem instance.

### Set Cover

**SET COVER:** Given a set  $U$  of elements, a collection  $S_1, S_2, \dots, S_m$  of subsets of  $U$ , and an integer  $k$ , does there exist a collection of  $\leq k$  of these sets whose union is equal to  $U$ ?

**Sample application.**

- $m$  available pieces of software.
- Set  $U$  of  $n$  capabilities that we would like our system to have.
- The  $i$ th piece of software provides the set  $S_i \subseteq U$  of capabilities.
- Goal: achieve all  $n$  capabilities using fewest pieces of software.

**Ex:**

$U = \{1, 2, 3, 4, 5, 6, 7\}$   
 $k = 2$   
 $S_1 = \{3, 7\}$      $S_4 = \{2, 4\}$   
 $S_2 = \{3, 4, 5, 6\}$      $S_5 = \{5\}$   
 $S_3 = \{1\}$      $S_6 = \{1, 2, 6, 7\}$

### Vertex Cover Reduces to Set Cover

**Claim.** VERTEX-COVER  $\leq_p$  SET-COVER.

**Pf.** Given a VERTEX-COVER instance  $G = (V, E)$ ,  $k$ , we construct a set cover instance whose size equals the size of the vertex cover instance.

**Construction.**

- Create SET-COVER instance:
  - $k = k$ ,  $U = E$ ,  $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size  $\leq k$  iff vertex cover of size  $\leq k$ . ■

**VERTEX COVER**

$k = 2$

**SET COVER**

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

$k = 2$

$S_a = \{3, 7\}$      $S_b = \{2, 4\}$

$S_c = \{3, 4, 5, 6\}$      $S_d = \{5\}$

$S_e = \{1\}$      $S_f = \{1, 2, 6, 7\}$

### Polynomial-Time Reduction

**Basic strategies.**

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

## 8.2 Reductions via "Gadgets"

**Basic reduction strategies.**

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction via "gadgets."

Use components in problem Y to represent X

### Satisfiability

**Literal:** A Boolean variable or its negation.  $x_i$  or  $\bar{x}_i$

**Clause:** A disjunction of literals.  $C_j = x_1 \vee \bar{x}_2 \vee x_3$

**Conjunctive normal form:** A propositional formula  $\Phi$  that is the conjunction of clauses.  $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

**SAT:** Given CNF formula  $\Phi$ , does it have a satisfying truth assignment?

**3-SAT:** SAT where each clause contains exactly 3 literals.

each corresponds to a different variable

**Ex:**  $(\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$

**Yes:**  $x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}$ .

### 3 Satisfiability Reduces to Independent Set

**Claim.** 3-SAT  $\leq_p$  INDEPENDENT-SET.

**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance  $(G, k)$  of INDEPENDENT-SET that has an independent set of size  $k$  iff  $\Phi$  is satisfiable.

**Construction.**

- $G$  contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

$k = 3$

$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$

### 3 Satisfiability Reduces to Independent Set

**Claim.**  $G$  contains independent set of size  $k = |\Phi|$  iff  $\Phi$  is satisfiable.

**Pf.  $\Rightarrow$**  Let  $S$  be independent set of size  $k$ .

- $S$  must contain exactly one vertex in each triangle.
- Set these literals to true. — and any other variables in a consistent way
- Truth assignment is consistent and all clauses are satisfied.

**Pf.  $\Leftarrow$**  Given satisfying assignment, select one true literal from each triangle. This is an independent set of size  $k$ . ■

$k = 3$

$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$

Review

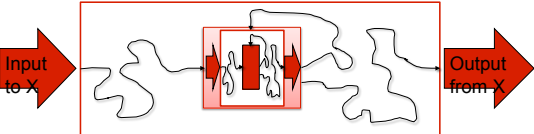
**Basic reduction strategies.**

- Simple equivalence:  $\text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER}$ .
- Special case to general case:  $\text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .
- Encoding with gadgets:  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$ .

**Transitivity.** If  $X \leq_p Y$  and  $Y \leq_p Z$ , then  $X \leq_p Z$ .

**Pf idea.** Compose the two algorithms.

**Ex:**  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .



## Midterm Exams

- Mean: 75
- 100-90:A, 89-80:B, 79-70:C, 69-60:D, 59-0:F
  
- You can still pass the course, but you must work hard.
  
- The last day to withdraw is this Friday.