

Algorithm Design and Analysis

**CSE
565**

LECTURE 26

Computational

Intractability

- Polynomial Time Reductions

Sofya Raskhodnikova

What algorithms are (im)possible?

Algorithm design patterns.

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Augmenting paths.
- Simplex method
- Reductions.
- ... (lots more out there)

Examples.

$O(n \log n)$ interval scheduling.
 $O(n \log n)$ sorting.
 $O(n^2)$ edit distance.
Max flow.
Linear programming
Maximum matching

New goal: understand what is hard to compute.

- NP-completeness. $O(n^k)$ algorithm unlikely.
- PSPACE-completeness. $O(n^k)$ certification algorithm unlikely.
- Undecidability. No algorithm possible.

Intractability: Central ideas we'll cover

- Poly-time as "feasible"
 - most natural problems **either** are easy (say n^3) **or** have no known poly-time algorithms
- P = problems that are easy to answer
 - e.g. minimum cut
- NP = {problems whose answers are easy to **verify** given **hint**}
 - e.g. graph 3-coloring
- Reductions: X is no harder than Y
- NP-completeness
 - many **natural** problems are easy **if and only if** P=NP

Polynomial-Time Reductions

Classify Problems According to Computational Requirements

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

A working definition. [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 lecture. 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 **poly-time 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 .



computational model supplemented by special piece of hardware that solves instances of Y in a single step

Notation. $X \leq_{p, \text{Cook}} Y$ (or $X \leq_p Y$).

Later in the lecture. $X \leq_{p, \text{Karp}} Y$.

Remarks.

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

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 $X \equiv_p Y$.

↑
up to cost of reduction

Simplifying Assumption: Decision Problems

Search problem. Find some structure.

Example. Find a minimum cut.

Decision problem.

- X is a set of strings.
- Instance: string s .
- If $x \in X$, x is a YES instance; if $x \notin X$ is a NO instance.
- Algorithm A solves problem X : $A(s) = \text{yes}$ iff $s \in X$.

Example. Does there exist a cut of size $\leq k$?

Self-reducibility. Search problem $\leq_{P, \text{Cook}}$ decision version.

- Applies to all (NP-complete) problems in Chapter 8 of KT.
- Justifies our focus on decision problems.

Polynomial Transformation

Def. Problem X **poly-time** reduces (Cook) 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.

Def. Problem X **poly-time transforms** (Karp) to problem Y if given any input x to X, we can construct an input y such that

x is a yes instance of X iff

y is a yes instance of Y.

↑
we require $|y|$ to be of size polynomial in $|x|$

Note. Poly-time transformation is poly-time reduction with just one call to oracle for Y, exactly at the end of the algorithm for X.

Open question. Are these two concepts the same?

↑
Caution: KT abuses notation \leq_p and blurs distinction

Basic reduction strategies

Basic reduction strategies

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

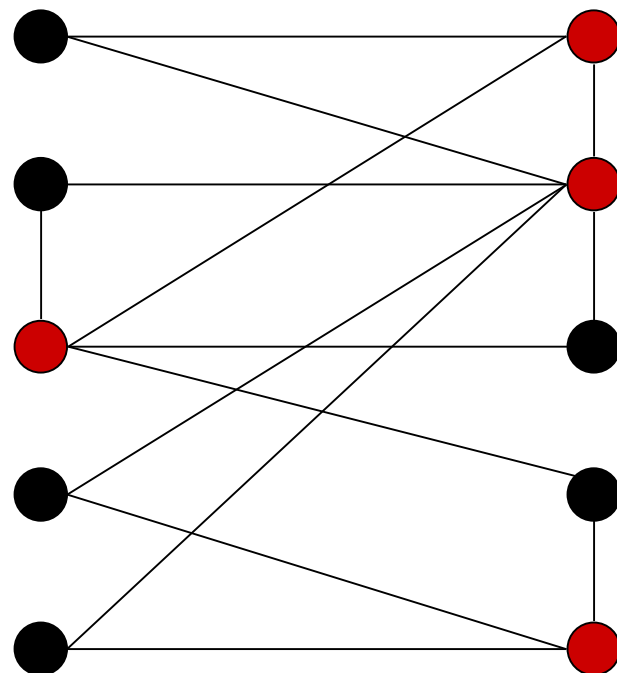
Independent Set

Given an undirected graph G , an **independent set** in G is a set of nodes, which includes at most one endpoint of every edge.

INDEPENDENT SET = $\{\langle G, k \rangle \mid G \text{ is an undirected graph which has an independent set with } k \text{ nodes}\}$

- Is there an independent set of size ≥ 6 ?
 - Yes.
- Is there an independent set of size ≥ 7 ?
 - No.

● independent set



Vertex Cover

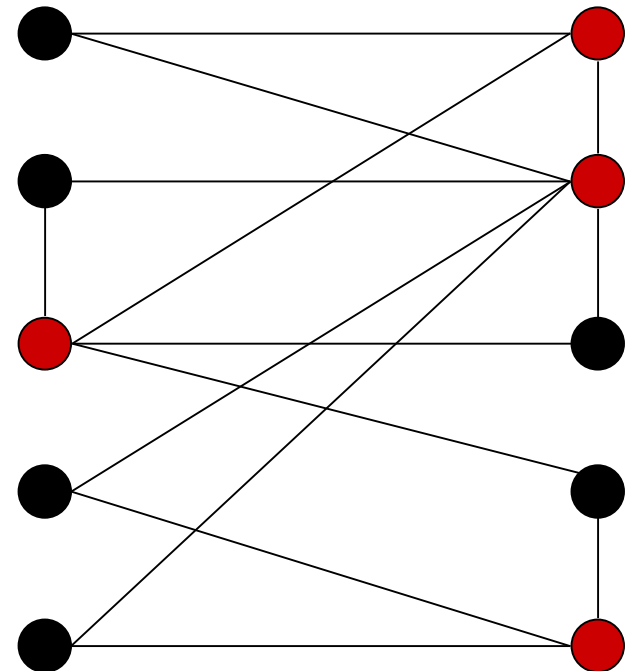
Given an undirected graph G , a **vertex cover** in G is a set of nodes, which includes at *least* one endpoint of every edge.

VERTEX COVER = $\{\langle G, k \rangle \mid G \text{ is an undirected graph which has a vertex cover with } k \text{ nodes}\}$

- Is there vertex cover of size ≤ 4 ?
 - Yes.

 vertex cover

- Is there a vertex cover of size ≤ 3 ?
 - No.



Independent Set and Vertex Cover

Claim. S is an independent set iff $V - S$ is a vertex cover.

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

INDEPENDENT SET reduces to VERTEX COVER

Theorem. INDEPENDENT-SET \leq_p VERTEX-COVER.

Proof. “On input $\langle G, k \rangle$, where G is an undirected graph and k is an integer,

1. Output $\langle G, n - k \rangle$, where n is the number of nodes in G .”

Correctness:

- G has an independent set of size k iff it has a vertex cover of size $n - k$.
- Reduction runs in linear time.

Reduction: special case to general case

Basic reduction strategies

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

Set Cover

Given a set U , called a *universe*, and a collection of its subsets S_1, S_2, \dots, S_m , a **set cover** of U is a subcollection of subsets whose union is U .

- SET COVER = $\{ \langle U, S_1, S_2, \dots, S_m; k \rangle \mid$

U has a set cover of size k }

- 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.

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

$$k = 2$$

$$S_1 = \{ 3, 7 \}$$

$$S_4 = \{ 2, 4 \}$$

$$S_2 = \{ 3, 4, 5, 6 \}$$

$$S_5 = \{ 5 \}$$

$$S_3 = \{ 1 \}$$

$$S_6 = \{ 1, 2, 6, 7 \}$$

VERTEX COVER reduces to SET COVER

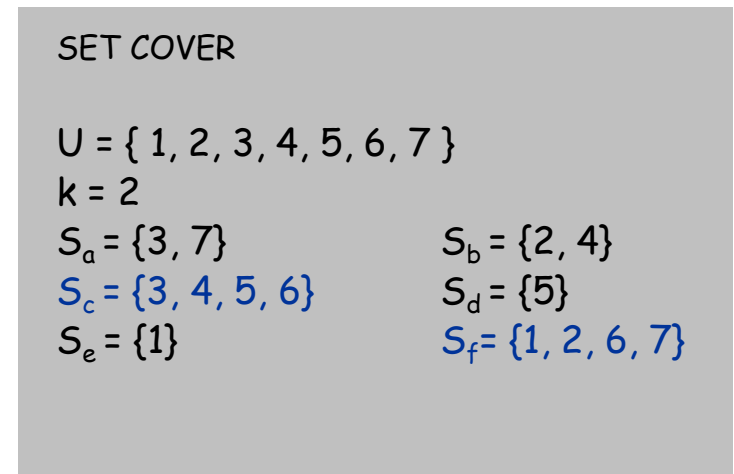
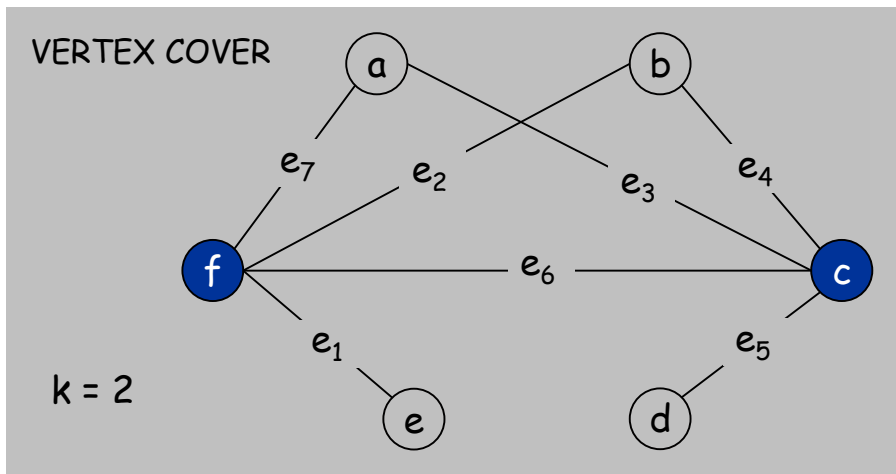
Theorem. VERTEX-COVER \leq_P SET-COVER.

Proof. “On input $\langle G, k \rangle$, where $G = (V, E)$ is an undirected graph and k is an integer,

1. Output $\langle U, S_1, S_2, \dots, S_m; k \rangle$, where $U = E$ and
$$S_v = \{e \in E : e \text{ incident to } v\}$$
”

Correctness:

- G has a vertex cover of size k iff U has a set cover of size k .
- Reduction runs in linear time.



Reduction by encoding with gadgets

Basic reduction strategies

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

Satisfiability

- **Boolean variables:** variables that can take on values T/F (or 1/0)
- **Boolean operations:** \vee , \wedge , and \neg
- **Boolean formula:** expression with Boolean variables and ops

SAT = { $\langle \Phi \rangle$ | Φ is a satisfiable Boolean formula}

- **Literal:** A Boolean variable or its negation. x_i or $\overline{x_i}$
- **Clause:** OR of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$
- **Conjunctive normal form (CNF):** AND of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

3SAT = { $\langle \Phi \rangle$ | Φ is a satisfiable Boolean CNF formula, where each clause contains exactly 3 literals}

↑
each corresponds to a different variable

Ex: $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

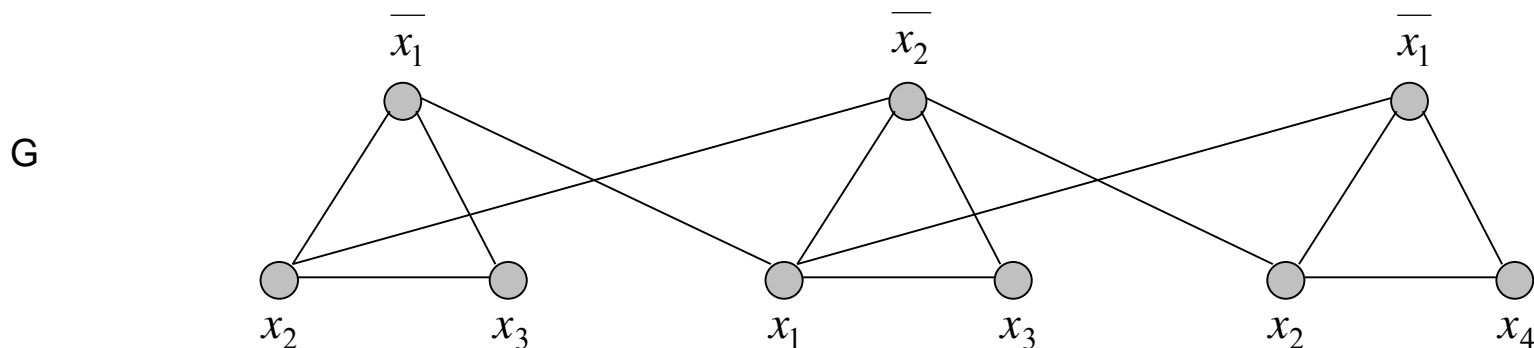
Yes: $x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}$.

3SAT reduces to INDEPENDENT SET

Theorem. $3\text{-SAT} \leq_{\text{P}} \text{INDEPENDENT-SET}$.

Proof. “On input $\langle \Phi \rangle$, where Φ is a 3CNF formula,

1. Construct graph G from Φ
 - 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.
2. Output $\langle G, k \rangle$, where k is the number of clauses in Φ .”



$k = 3$

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

3SAT reduces to INDEPENDENT SET

Correctness. Let $k = \#$ of clauses and $\ell = \#$ of literals in Φ .

Φ is satisfiable iff G contains an independent set of size k .

- \Rightarrow Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k .
- \Leftarrow Let S be an independent set of size k .
 - S must contain exactly one vertex in each triangle.
 - Set these literals to true, and other literals in a consistent way.
 - Truth assignment is consistent and all clauses are satisfied.

Run time. $O(k + \ell^2)$, i.e. polynomial in the input size.

Summary

- Basic reduction strategies.
 - Simple equivalence: $\text{INDEPENDENT-SET} \equiv_{\text{P}} \text{VERTEX-COVER}$.
 - Special case to general case: $\text{VERTEX-COVER} \leq_{\text{P}} \text{SET-COVER}$.
 - Encoding with gadgets: $3\text{-SAT} \leq_{\text{P}} \text{INDEPENDENT-SET}$.
- **Transitivity.** If $X \leq_{\text{P}} Y$ and $Y \leq_{\text{P}} Z$, then $X \leq_{\text{P}} Z$.
- Proof idea. Compose the two algorithms.
- **EX:** $3\text{-SAT} \leq_{\text{P}} \text{INDEPENDENT-SET} \leq_{\text{P}} \text{VERTEX-COVER} \leq_{\text{P}} \text{SET-COVER}$.