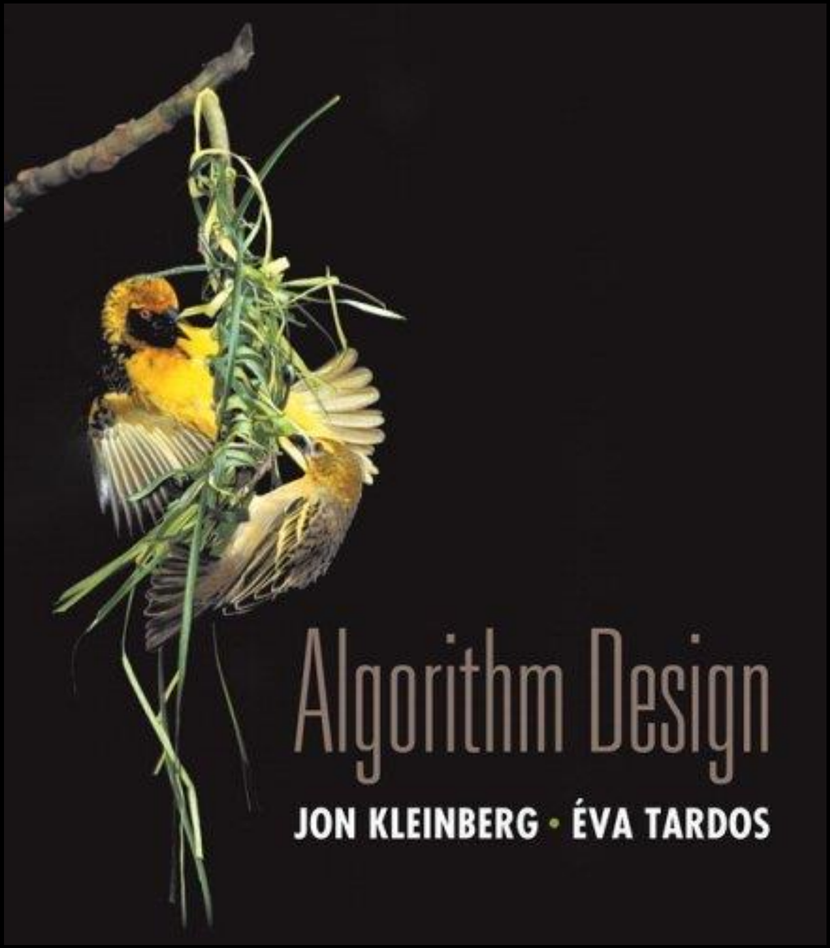


Chapter 8

NP and Computational Intractability

CS 350 Winter 2018

Algorithm Design Patterns and Anti-Patterns

Algorithm design patterns.	Ex.
- Greedy.	$O(n \log n)$ interval scheduling.
- Divide-and-conquer.	$O(n \log n)$ FFT.
- Dynamic programming.	$O(n^2)$ edit distance.
- Duality.	$O(n^3)$ bipartite matching.
- Reductions.
- Local search.
- Randomization.

Algorithm design anti-patterns.

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

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? (the Halting Problem)
- 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 polynomial 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. $X \leq_P Y$.

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

Reduction By Simple Equivalence

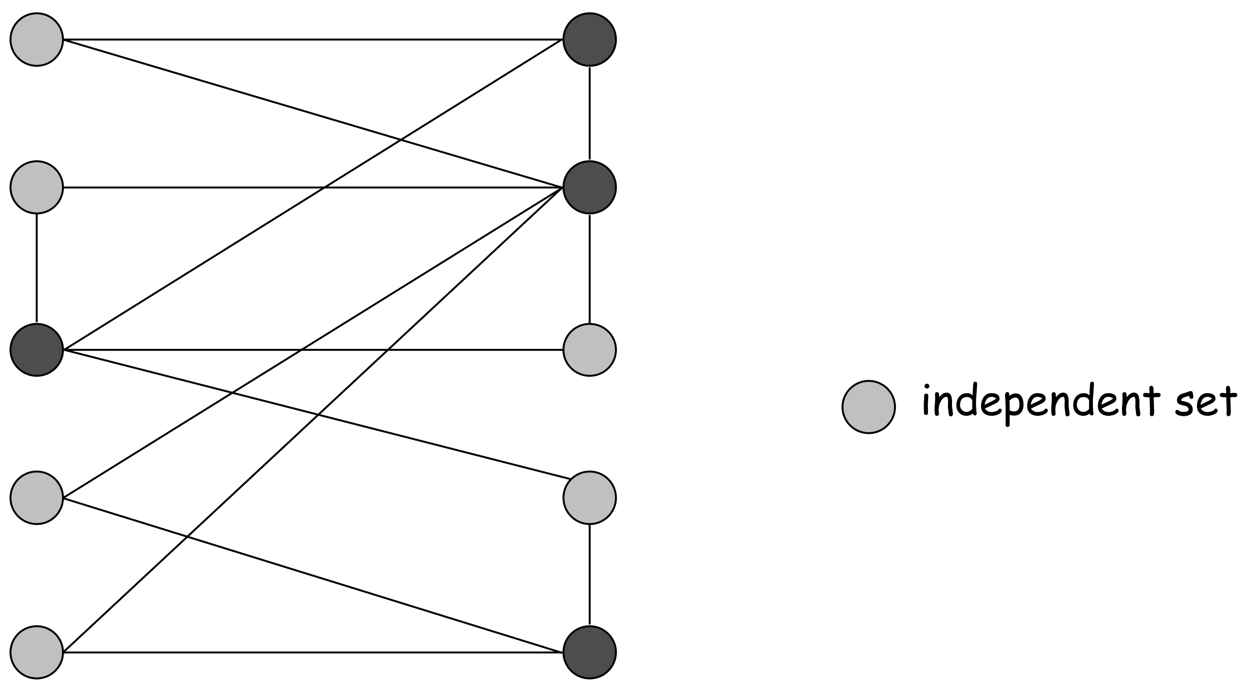
Basic reduction strategies.

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

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?

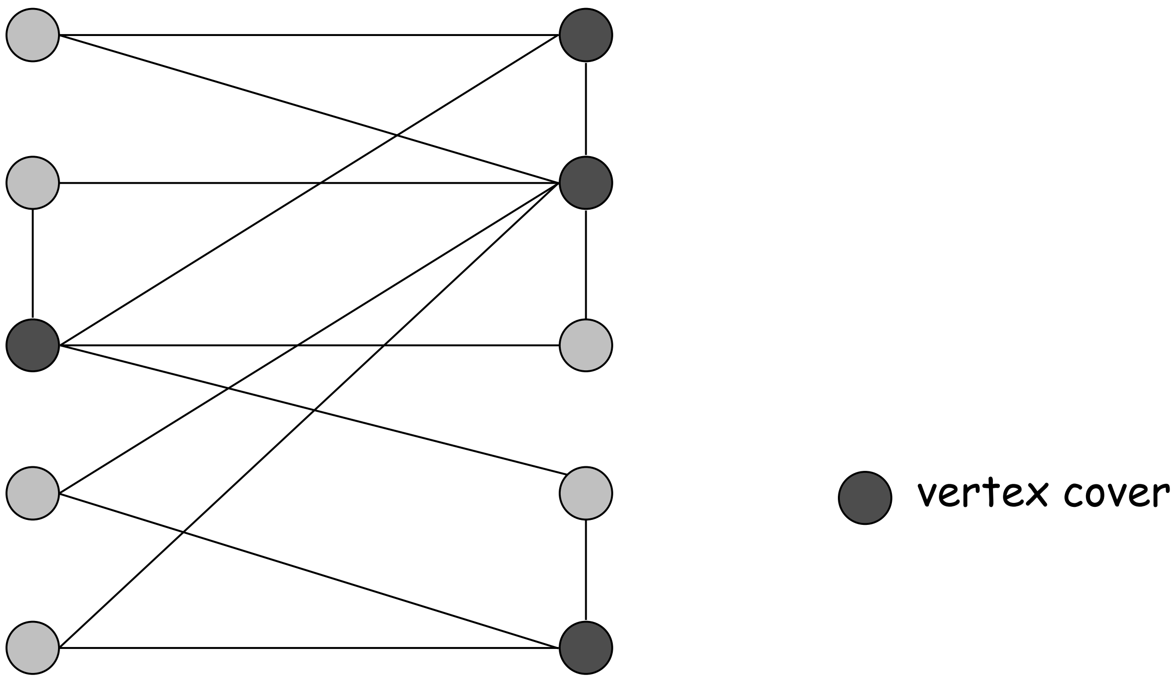
Ex. Is there an independent set of size ≥ 6? Yes.
Ex. Is there an independent set of size ≥ 7? No.



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?

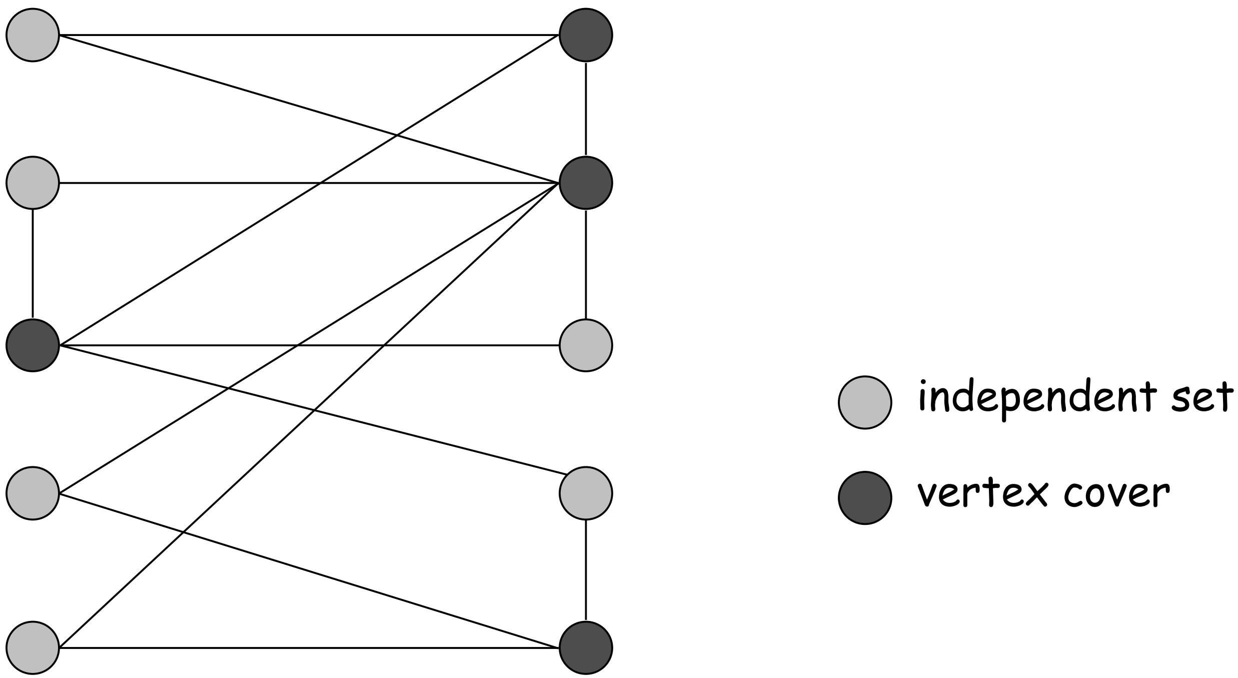
Ex. Is there a vertex cover of size ≤ 4? Yes.
Ex. Is there a vertex cover of size ≤ 3? No.



Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

Pf. We show S is an independent set iff V – S is a vertex cover.



Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.
Pf. We show 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 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. ▪

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.

Set Cover

SET COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ 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 ith piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

Ex:

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

$k = 2$

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

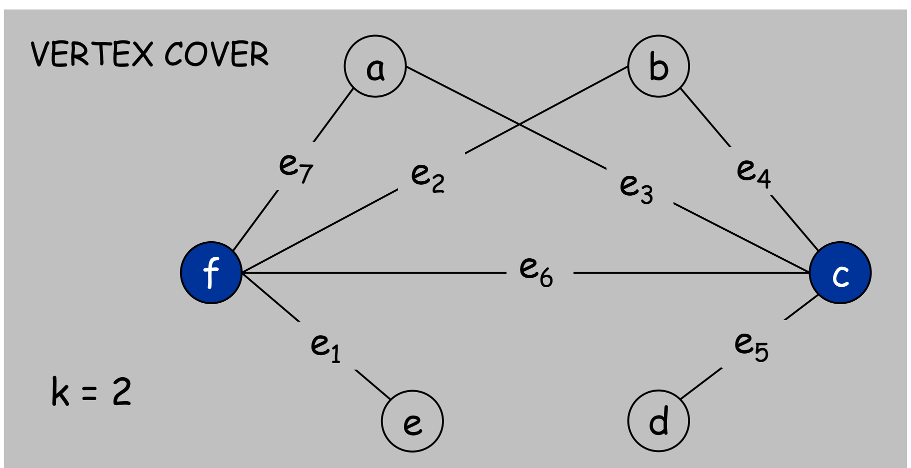
Vertex Cover Reduces to Set Cover

Claim. VERTEX-COVER $\le_P$ 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 $\le k$ iff vertex cover of size $\le k$. ▪



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."

Satisfiability

Literal: A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause: A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

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

SAT: Given CNF formula Φ, does it have a satisfying truth assignment?

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

each corresponds to a different variable

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

Yes: x_1 = true, x_2 = true x_3 = false.

3 Satisfiability Reduces to Independent Set

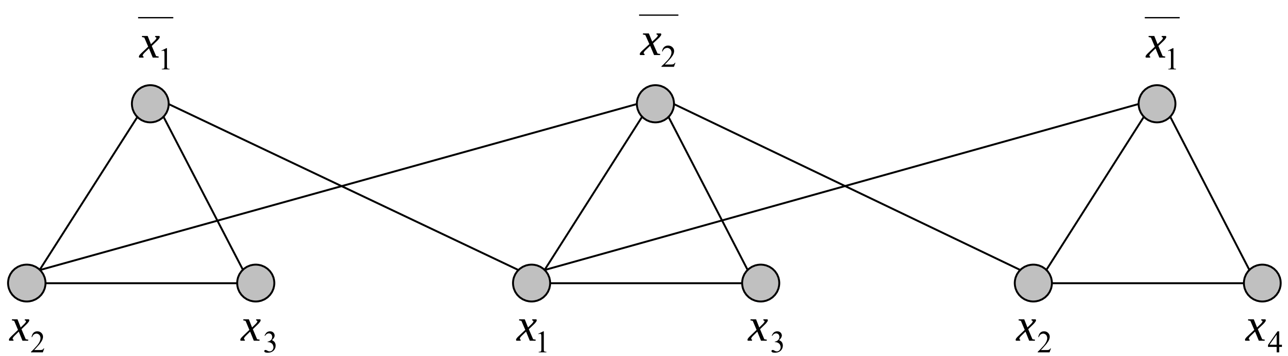
Claim. 3-SAT $\leq_P$ INDEPENDENT-SET.

Pf. Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ 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.

G



k = 3

$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

3 Satisfiability Reduces to Independent Set

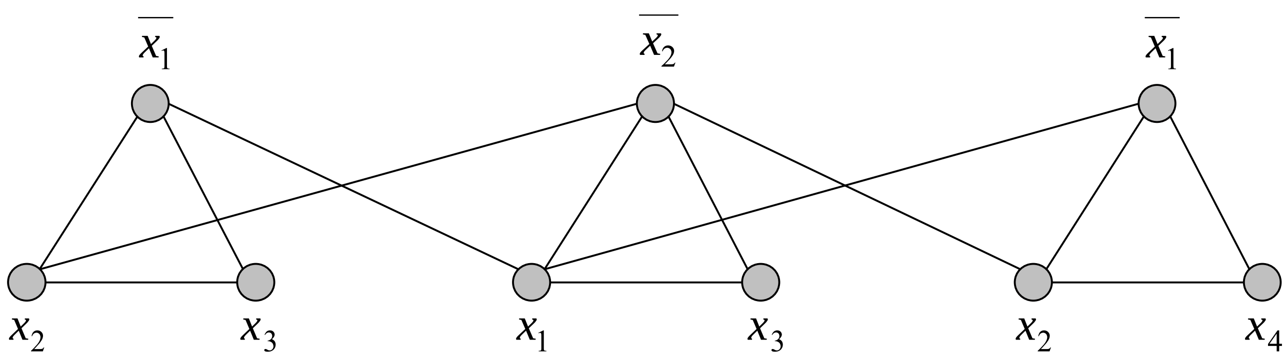
Claim. G contains independent set of size k = $|\Phi|$ iff Φ is satisfiable.

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

- S must contain exactly one vertex in each triangle.
- Set these literals to true. $\leftarrow$ 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. ▪

G



k = 3

$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Review

Basic reduction strategies.

- Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ 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-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER.

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

8.3 Definition of NP

Decision Problems

Decision problem.
- X is a set of strings.
- Instance: string s.
- Algorithm A solves problem X: A(s) = `yes` iff $s \in X$.

Polynomial time. Algorithm A runs in poly-time if for every string s, A(s) terminates in at most p(|s|) "steps", where p(·) is some polynomial.

length of s

PRIMES: X = { 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, …. }
Algorithm. [Agrawal-Kayal-Saxena, 2002] $p(|s|) = |s|^8$.

Definition of P

P. Decision problems for which there is a poly-time algorithm.

Problem	Description	Algorithm	Yes	No
MULTIPLE	Is x a multiple of y?	Grade school division	51, 17	51, 16
RELPRIME	Are x and y relatively prime?	Euclid (300 BCE)	34, 39	34, 51
PRIMES	Is x prime?	AKS (2002)	53	51
EDIT-DISTANCE	Is the edit distance between x and y less than 5?	Dynamic programming	niether neither	acgggt ttttta
LSOLVE	Is there a vector x that satisfies Ax = b?	Gauss-Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

NP

Certification algorithm intuition.

- Certifier views things from "managerial" viewpoint.
- Certifier doesn't determine whether $s \in X$ on its own; rather, it checks a proposed proof t that $s \in X$.

Def. Algorithm $C(s, t)$ is a certifier for problem X if for every string s, $s \in X$ iff there exists a string t such that $C(s, t)$ = `yes`.

"certificate" or "witness"

NP. Decision problems for which there exists a poly-time certifier.

$C(s, t)$ is a poly-time algorithm and $|t| \le p(|s|)$ for some polynomial $p(\cdot)$.

Remark. NP stands for nondeterministic polynomial-time.

Certifiers and Certificates: Composite

COMPOSITES. Given an integer s, is s composite?

Certificate. A nontrivial factor t of s. Note that such a certificate exists iff s is composite. Moreover $|t| \leq |s|$.

Certifier.

```
boolean C(s, t) {
   if (t ≤ 1 or t ≥ s)
      return false
   else if (s is a multiple of t)
      return true
   else
      return false
}
```

Instance. s = 437,669.
Certificate. t = 541 or 809. ← 437,669 = 541 × 809

Conclusion. COMPOSITES is in NP.

Certifiers and Certificates: 3-Satisfiability

SAT. Given a CNF formula Φ, is there a satisfying assignment?

Certificate. An assignment of truth values to the n boolean variables.

Certifier. Check that each clause in Φ has at least one true literal.

Ex.

$$\left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(x_1 \vee x_2 \vee x_4\right) \wedge \left(\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}\right)$$

instance s

$$x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 1$$

certificate t

Conclusion. SAT is in NP.

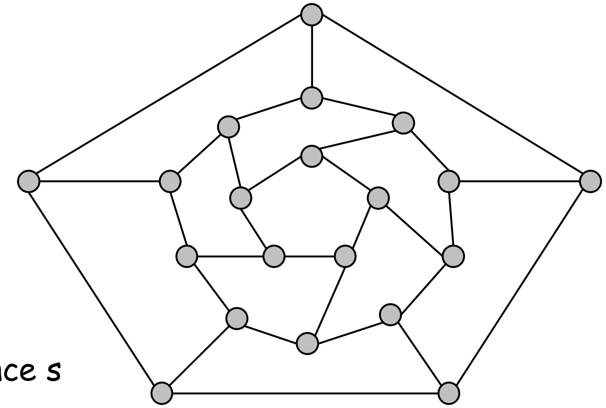
Certifiers and Certificates: Hamiltonian Cycle

HAM-CYCLE. Given an undirected graph G = (V, E), does there exist a simple cycle C that visits every node?

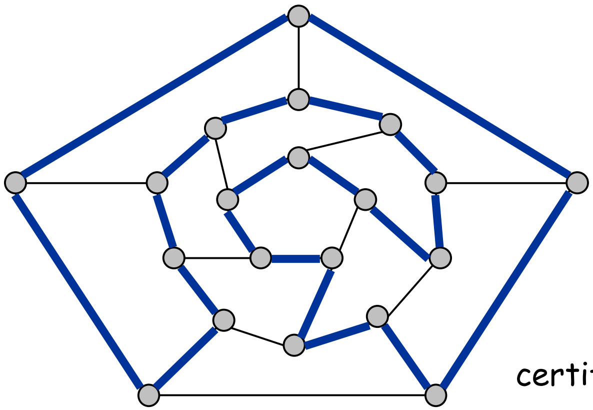
Certificate. A permutation of the n nodes.

Certifier. Check that the permutation contains each node in V exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

Conclusion. HAM-CYCLE is in NP.



instance s



certificate t

P, NP, EXP

P. Decision problems for which there is a poly-time algorithm.
EXP. Decision problems for which there is an exponential-time algorithm.
NP. Decision problems for which there is a poly-time certifier.

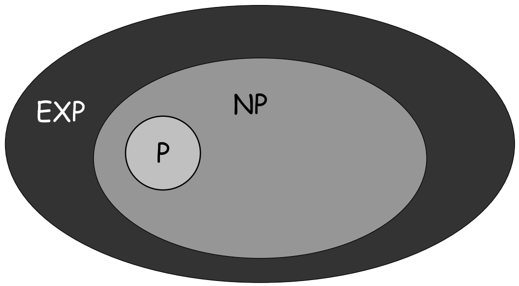
Claim. P $\subseteq$ NP.
Pf. Consider any problem X in P.
- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate: $t = \varepsilon$, certifier $C(s, t) = A(s)$. ▪

Claim. NP $\subseteq$ EXP.
Pf. Consider any problem X in NP.
- By definition, there exists a poly-time certifier $C(s, t)$ for X.
- To solve input s, run $C(s, t)$ on all strings t with $|t| \leq p(|s|)$.
- Return `yes`, if $C(s, t)$ returns `yes` for any of these. ▪

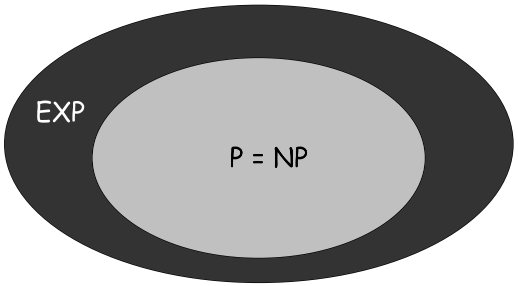
The Main Question: P Versus NP

Does P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

- Is the decision problem as easy as the certification problem?
- Clay $1 million prize.



If P ≠ NP



If P = NP

would break RSA cryptography
(and potentially collapse economy)

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, …

If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, …

Consensus opinion on P = NP? Probably no.

8.4 NP-Completeness

Polynomial Transformation

Def. Problem X **polynomial 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 **polynomial 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. Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all previous reductions were of this form.

Open question. Are these two concepts the same with respect to NP?

we abuse notation $\leq_p$ and blur distinction

NP-Complete

NP-complete. A problem Y in NP with the property that for every problem X in NP, $X \leq_p Y$.

Theorem. Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff P = NP.

Pf. $\Leftarrow$ If P = NP then Y can be solved in poly-time since Y is in NP.

Pf. $\Rightarrow$ Suppose Y can be solved in poly-time.

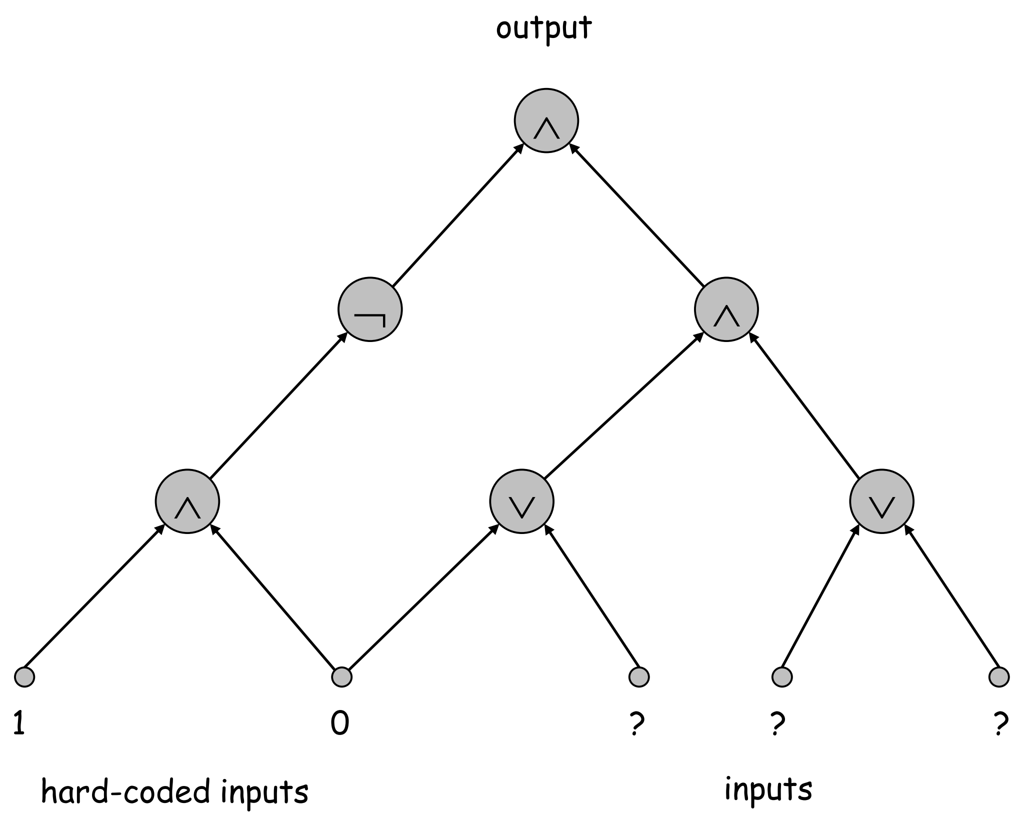
- Let X be any problem in NP. Since $X \leq_p Y$, we can solve X in poly-time. This implies NP $\subseteq$ P.
- We already know P $\subseteq$ NP. Thus P = NP. ▪

Fundamental question. Do there exist "natural" NP-complete problems?

Circuit Satisfiability

CIRCUIT-SAT. Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?

yes: 1 0 1



The "First" NP-Complete Problem

Theorem. CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

Pf. (sketch)

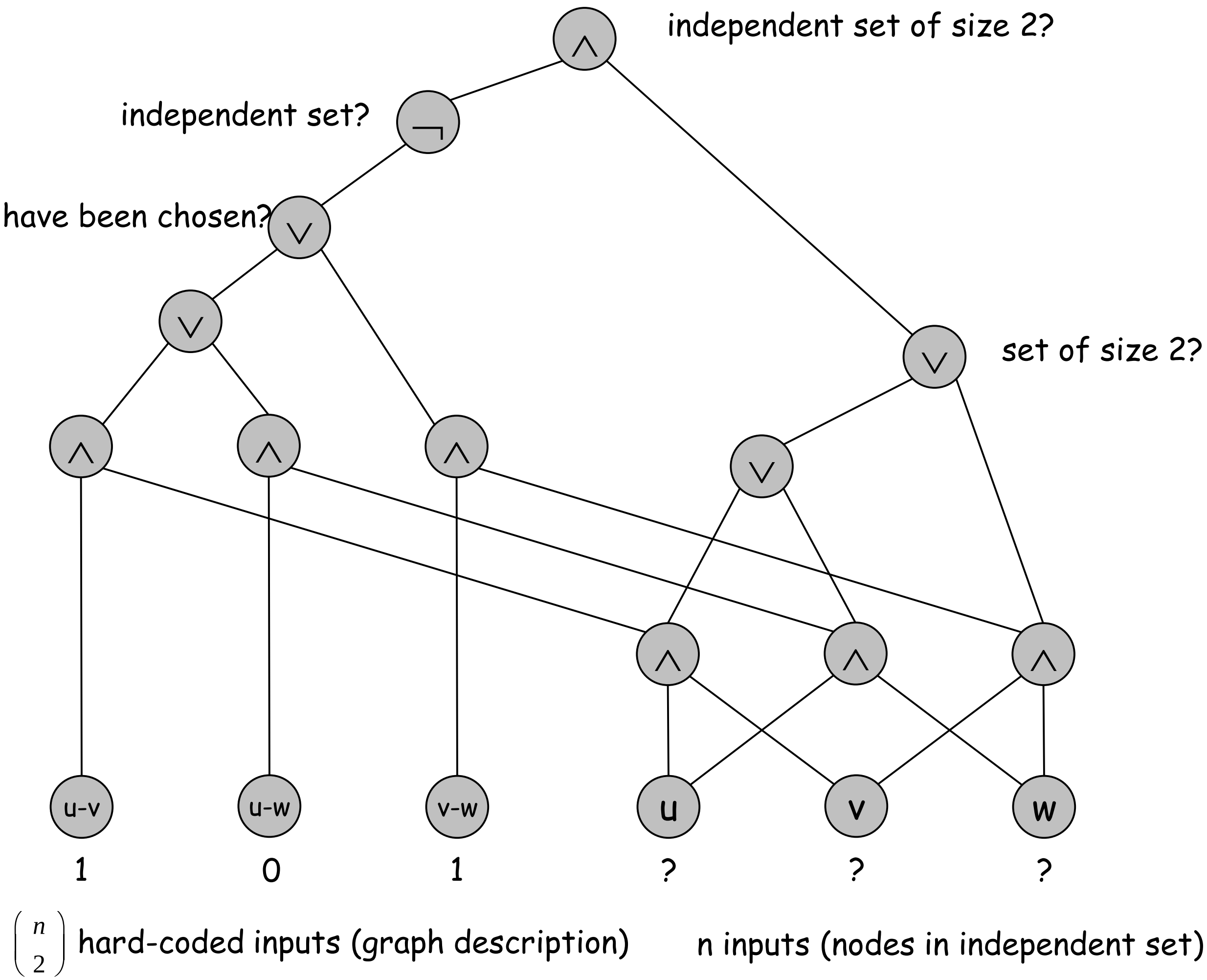
- Any algorithm that takes a fixed number of bits n as input and produces a yes/no answer can be represented by such a circuit. Moreover, if algorithm takes poly-time, then circuit is of poly-size.

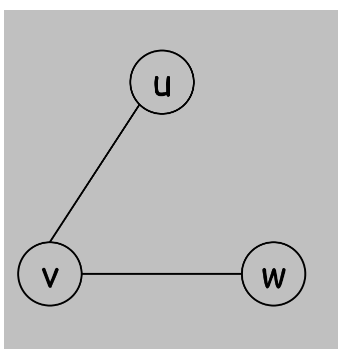
sketchy part of proof; fixing the number of bits is important, and reflects basic distinction between algorithms and circuits

- Consider some problem X in NP. It has a poly-time certifier $C(s, t)$. To determine whether s is in X, need to know if there exists a certificate t of length $p(|s|)$ such that $C(s, t)$ = `yes`.
- View $C(s, t)$ as an algorithm on $|s| + p(|s|)$ bits (input s, certificate t) and convert it into a poly-size circuit K.
  - first $|s|$ bits are hard-coded with s
  - remaining $p(|s|)$ bits represent bits of t
- Circuit K is satisfiable iff $C(s, t)$ = `yes`.

Example

Ex. Construction below creates a circuit K whose inputs can be set so that K outputs true iff graph G has an independent set of size 2.





G = (V, E), n = 3

Establishing NP-Completeness

Remark. Once we establish first "natural" NP-complete problem, others fall like dominoes.

Recipe to establish NP-completeness of problem Y.

- Step 1. Show that Y is in NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that $X \le_p Y$.

Justification. If X is an NP-complete problem, and Y is a problem in NP with the property that $X \le_P Y$ then Y is NP-complete.

Pf. Let W be any problem in NP. Then $W \underbrace{\le_P}_{\text{by definition of NP-complete}} X \underbrace{\le_P}_{\text{by assumption}} Y$.

- By transitivity, $W \le_P Y$.
- Hence Y is NP-complete. ▪

3-SAT is NP-Complete

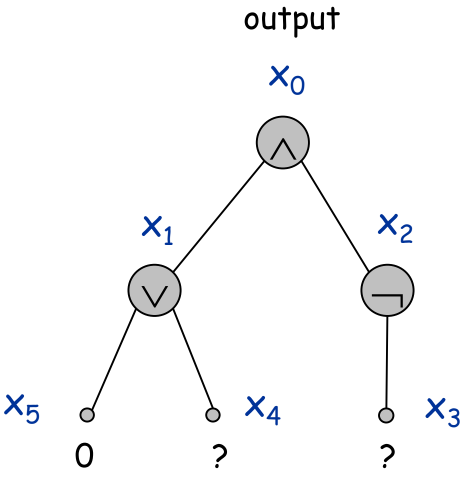
Theorem. 3-SAT is NP-complete.

Pf. Suffices to show that CIRCUIT-SAT $\le_P$ 3-SAT since 3-SAT is in NP.

- Let K be any circuit.
- Create a 3-SAT variable x_i for each circuit element i.
- Make circuit compute correct values at each node:
  - $x_2 = \neg x_3 \Rightarrow$ add 2 clauses: $x_2 \vee x_3$, $\overline{x_2} \vee \overline{x_3}$
  - $x_1 = x_4 \vee x_5 \Rightarrow$ add 3 clauses: $x_1 \vee \overline{x_4}$, $x_1 \vee \overline{x_5}$, $\overline{x_1} \vee x_4 \vee x_5$
  - $x_0 = x_1 \wedge x_2 \Rightarrow$ add 3 clauses: $\overline{x_0} \vee x_1$, $\overline{x_0} \vee x_2$, $x_0 \vee \overline{x_1} \vee \overline{x_2}$

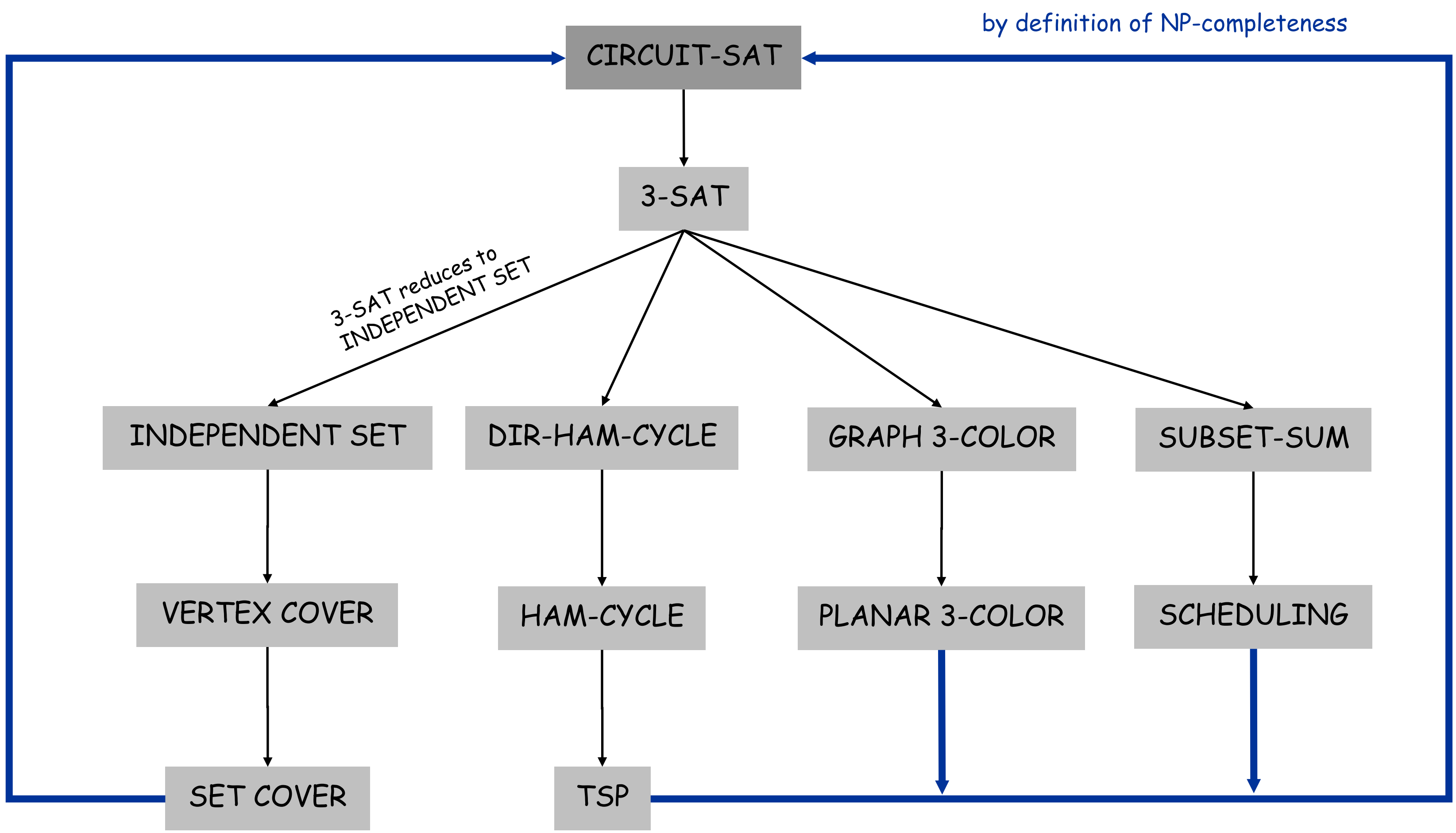
- Hard-coded input values and output value.
  - $x_5 = 0 \Rightarrow$ add 1 clause: $\overline{x_5}$
  - $x_0 = 1 \Rightarrow$ add 1 clause: x_0

- Final step: turn clauses of length < 3 into clauses of length exactly 3. ▪



NP-Completeness

Observation. All problems below are NP-complete and polynomial reduce to one another!



Some NP-Complete Problems

Six basic genres of NP-complete problems and paradigmatic examples.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Practice. Most NP problems are either known to be in P or NP-complete.

Notable exceptions. Factoring, graph isomorphism, Nash equilibrium.

Extent and Impact of NP-Completeness

Extent of NP-completeness. [Papadimitriou 1995]

- Prime intellectual export of CS to other disciplines.
- 6,000 citations per year (title, abstract, keywords).
  - more than "compiler", "operating system", "database"
- Broad applicability and classification power.
- "Captures vast domains of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasibly."

NP-completeness can guide scientific inquiry.

- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager solves 2D case in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: Istrail proves 3D problem NP-complete.

More Hard Computational Problems

Aerospace engineering: optimal mesh partitioning for finite elements.
Biology: protein folding.
Chemical engineering: heat exchanger network synthesis.
Civil engineering: equilibrium of urban traffic flow.
Economics: computation of arbitrage in financial markets with friction.
Electrical engineering: VLSI layout.
Environmental engineering: optimal placement of contaminant sensors.
Financial engineering: find minimum risk portfolio of given return.
Game theory: find Nash equilibrium that maximizes social welfare.
Genomics: phylogeny reconstruction.
Mechanical engineering: structure of turbulence in sheared flows.
Medicine: reconstructing 3-D shape from biplane angiocardiogram.
Operations research: optimal resource allocation.
Physics: partition function of 3-D Ising model in statistical mechanics.
Politics: Shapley-Shubik voting power.
Pop culture: Minesweeper consistency.
Statistics: optimal experimental design.

Kleinberg HW 8.1

#1: Decide whether the answer for each is yes, no or "unknown" because it would resolve whether P=NP.

(a) Let's define the decision version of the Interval Scheduling Problem from Chapter 4 as follows: Given a collection of intervals on a time-line, and bound k, does the collection contain a subset of nonoverlapping intervals of size at least k?

Question: Is it the case that Interval Scheduling $\leq_P$ Vertex Cover?

Kleinberg HW 8.1

#1: Decide whether the answer for each is yes, no or “unknown” because it would resolve whether P=NP.

(a) Let’s define the decision version of the Interval Scheduling Problem from Chapter 4 as follows: Given a collection of intervals on a time-line, and bound k, does the collection contain a subset of nonoverlapping intervals of size at least k?

Question: Is it the case that Interval Scheduling $\leq_P$ Vertex Cover?

Yes. One solution: Interval Scheduling can be solved in polynomial time, and so it can also be solved in polynomial time with access to a black box for Vertex Cover.

Another solution: Interval Scheduling is in NP, and anything in NP can be reduced to Vertex Cover.

Kleinberg HW 8.1

#1: Decide whether the answer for each is yes, no or "unknown" because it would resolve whether P=NP.

(b) Question: Is it the case that Independent Set $\leq_P$ Interval Scheduling?

Kleinberg HW 8.1

#1: Decide whether the answer for each is yes, no or "unknown" because it would resolve whether P=NP.

(b) Question: Is it the case that Independent Set $\leq_P$ Interval Scheduling?

This is equivalent to P = NP.

If P=NP, then Independent Set can be solved in polynomial time, so Independent Set $\leq_P$ Interval Scheduling. Conversely, if Independent Set $\leq_P$ Interval Scheduling, then since Interval Scheduling can be solved in polynomial time, so could Independent Set. But Independent Set is NP-complete, so solving it in polynomial time would imply P=NP.

Kleinberg HW 8.2

#2: A store trying to analyze the behavior of its customers will often maintain a 2-d array A, where the rows correspond to its customers, and the columns correspond to the products it sells. The entry A[i,j] specifies the quantity of product j that has been purchased by customer i.

Example:

	Liquid detergent	beer	diapers	Cat litter
Raj	0	6	0	3
Alanis	2	3	0	0
Chelsea	0	0	0	7

One thing that a store might want to do with these data is the following: Let us say that a subset S of the customers is diverse if no two of the customers in S have ever bought the same product (i.e. for each product, at most one of the customers in S has ever bought it).A diverse set of customers can be useful, for example, as a target pool for market research.

We now define the Diverse Subset Problem as follows: Given an m x n array A as defined above, and a number k ≤m, is there a subset of at least k of the customers that is diverse? Show that Diverse Subset is NP-complete.

Kleinberg HW 8.2

#2: Let us say that a subset S of the customers is diverse if no two of the customers in S have ever bought the same product (i.e. for each product, at most one of the customers in S has ever bought it).A diverse set of customers can be useful, for example, as a target pool for market research.

We now define the Diverse Subset Problem as follows: Given an m x n array A as defined above, and a number k $\leq$m, is there a subset of at least k of the customers that is diverse? Show that Diverse Subset is NP-complete.

First: The problem itself is NP because we can exhibit a set of k customers, and in polynomial time we can check that no two bought any product in common.

Next we show NP-complete…

Kleinberg HW 8.2

#2: We now define the Diverse Subset Problem as follows: Given an m x n array A as defined above, and a number k ≤m, is there a subset of at least k of the customers that is diverse? Show that Diverse Subset is NP-complete.

First: The problem itself is NP because we can exhibit a set of k customers, and in polynomial time we can check that no two bought any product in common.

Next we show NP-complete…in particular, we show that: Independent Set $\leq_P$ Diverse Subset.

Given a graph G and a number k, <u>we construct a customer for each node of G, and a product for each edge of G</u>. We then build an array that says customer v bought product e if edge e is incident to node v. Finally, we ask whether this array has a diverse subset of size k.

Claim: this holds if and only if G has an independent set of size k (you should be able to argue this directly, then we're done).

Kleinberg HW 8.3

#3: Suppose you're helping to organize a summer sports camp, and the following problem comes up. The camp is supposed to have at least one counselor who's skilled at each of the n sports covered by the camp (baseball, volleyball, etc.). They have received job applications from m potential counselors. For each of the n sports, there is some subset of the m applicants qualified in that sport.

The question is: For a given number $k < m$, is it possible to hire at most k of the counselors and have at least one counselor qualified in each of the n sports? Call this the Efficient Recruiting Problem.

Show that Efficient Recruiting is NP-complete.

Kleinberg HW 8.3

#3: Suppose you're helping to organize a summer sports camp, and the following problem comes up. The camp is supposed to have at least one counselor who's skilled at each of the n sports covered by the camp (baseball, volleyball, etc.). They have received job applications from m potential counselors. For each of the n sports, there is some subset of the m applicants qualified in that sport.

The question is: For a given number $k < m$, is it possible to hire at most k of the counselors and have at least one counselor qualified in each of the n sports? Call this the Efficient Recruiting Problem.

Show that Efficient Recruiting is NP-complete.

First: The problem is in NP since, given a set of k counselors, we can check that they cover all the sports.

Next: Suppose we had such an algorithm A: here is how we would solve an instance of Vertex Cover.

Kleinberg HW 8.3

#3:The question is: For a given number $k < m$, is it possible to hire at most k of the counselors and have at least one counselor qualified in each of the n sports? Call this the Efficient Recruiting Problem.

Show that Efficient Recruiting is NP-complete.

First: The problem is in NP since, given a set of k counselors, we can check that they cover all the sports.

Next: Suppose we had such an algorithm A: here is how we would solve an instance of Vertex Cover.

Given a graph $G=(V,E)$ and an integer k, we would define a sport Se for each edge e, and a counselor C_v for each vertex v. C_v is qualified in sport S_e if and only if e has an endpoint equal to v.

Finally show that a vertex cover of size k in this graph corresponds with having k counselors that are qualified in all sports (and vice versa). Now you should have the answer…