CSE 417, Winter 2012

P, NP, and Intractability

Ben Birnbaum

Widad Machmouchi

Slides adapted from Larry Ruzzo and Kevin Wayne

The Simpson's: P = NP?

Looking for a Job?

Some writers for the Simpsons and Futurama.

- J. Steward Burns. M.S. in mathematics, Berkeley, 1993.
- David X. Cohen. M.S. in computer science, Berkeley, 1992.
- Al Jean. B.S. in mathematics, Harvard, 1981.
- Ken Keeler. Ph.D. in applied mathematics, Harvard, 1990.
- Jeff Westbrook. Ph.D. in computer science, Princeton, 1989.

What can we feasibly compute?

Focus so far in the course has been to give good algorithms for specific problems (and general techniques that help do this).

Now shifting focus to problems where we think this is impossible.

Overview

Researchers found many problems with obvious exponential solutions, but no polynomial time algorithm known.

Eventually, researchers gave up and started trying to prove that it was *impossible* to solve these problems efficiently.

Didn't quite succeed here either.

However, they did develop a beautiful theory that allows us to show that many problems "probably" can't be solved efficiently.

Theory of NP-Completeness

Our goals

1. Explain how this theory works.
2. Show how to use it to prove a problem is "probably" not solvable in polynomial time.

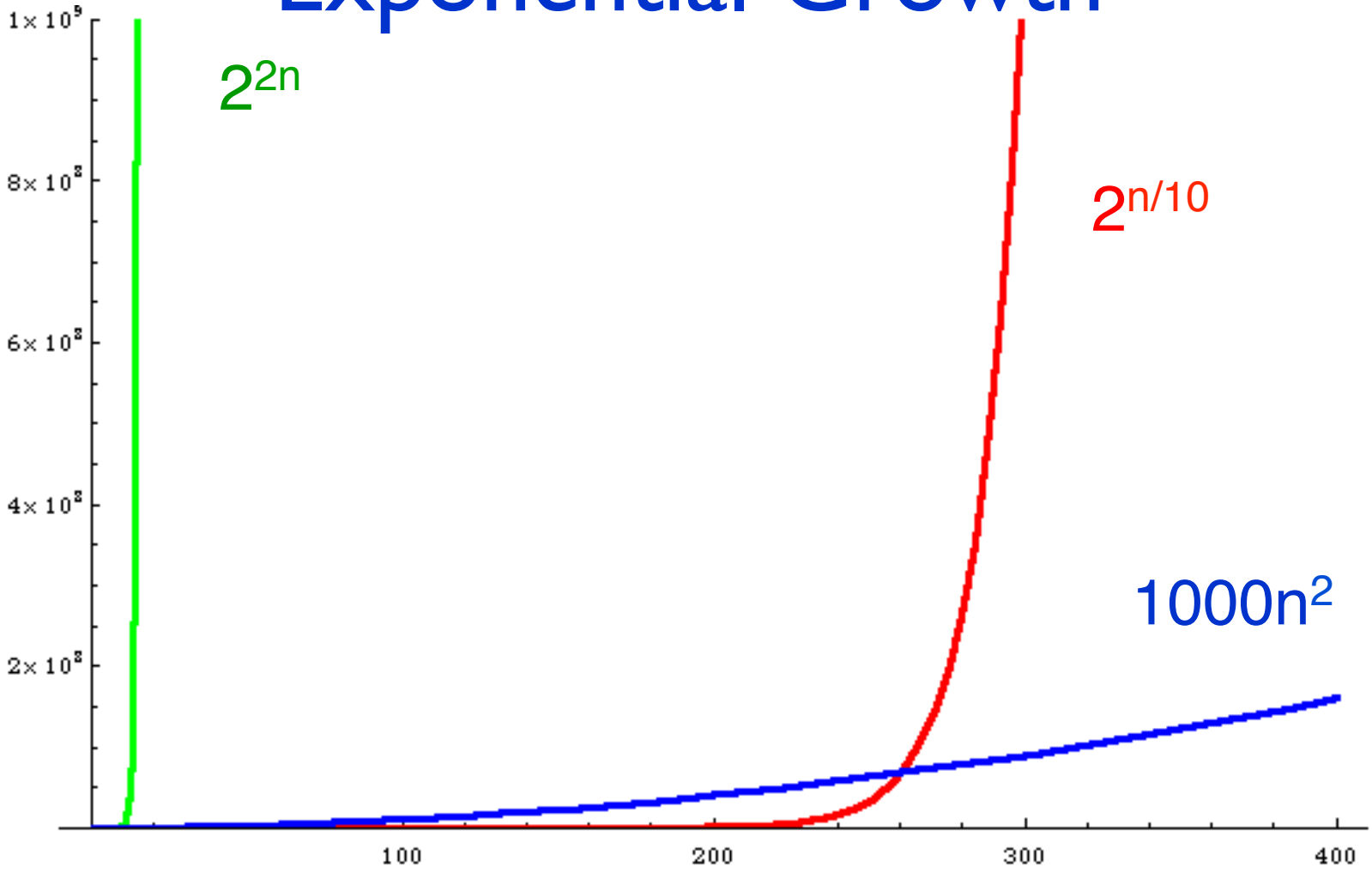
This is the most theoretical part of the course, but it is very important.

Polynomial versus exponential

Polynomial

Bigger than polynomial

Polynomial vs Exponential Growth



Another view of Poly vs Exp

Next year's computer will be 2x faster. If I can solve problem of size n_0 today, how large a problem can I solve in the same time next year?

Complexity	Increase	E.g. $T=10^{12}$	
$O(n)$	$n_0 \rightarrow 2n_0$	10^{12}	2×10^{12}
$O(n^2)$	$n_0 \rightarrow \sqrt{2}\, n_0$	10^6	1.4×10^6
$O(n^3)$	$n_0 \rightarrow \sqrt[3]{2}\, n_0$	10^4	1.25×10^4
$2^{n/10}$	$n_0 \rightarrow n_0+10$	400	410
2^n	$n_0 \rightarrow n_0+1$	40	41

Polynomial versus exponential

Of course there are exceptions:

- n^{100} is bigger than $(1.001)^n$ for most practical values of n but usually such run-times don't show up
- There are algorithms that have run-times like $O(2^{sqrt(n)/22})$ and these may be useful for small input sizes, but they're not too common either

Decision problems

Computational complexity usually analyzed using decision problems: answer just YES or NO (1 or 0)

Example: "Find the minimum spanning tree" ➔
"Is there a spanning tree of size ≤ k?"

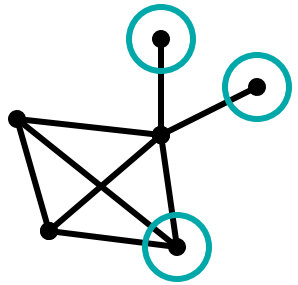
Why?

- Much simpler to deal with
- Deciding whether G has a k-clique is certainly no harder than finding a k-clique in G or finding the size of the maximum k-clique. So proving decision problem is hard is a strong result.
- Less important, but if you have a good decider, you can often use it to get a good finder.

Some Decision Problems

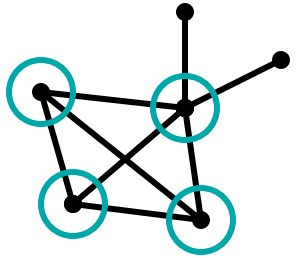
Independent-Set:

Given a graph G=(V,E) and an integer k, is there a subset U of V with |U| ≥ k such that no two vertices in U are joined by an edge.



Clique:

Given a graph G=(V,E) and an integer k, is there a subset U of V with |U| ≥ k such that every pair of vertices in U is joined by an edge.



Some Terminology

"Problem" – the general case

Ex: The Clique Problem: Given a graph G and an integer k, does G contain a k-clique?

"Problem Instance" – the specific cases

Ex: Does  contain a 4-clique? This is a "NO instance"

Ex: Does  contain a 3-clique? This is a "YES instance"

The class P

Definition: **P** = set of (decision) problems solvable by computers in *polynomial time*. i.e.,

$T(n) = O(n^k)$ for some fixed k (indp of input).

These problems are sometimes called *tractable* problems.

Examples: shortest path, MST, connectivity, interval scheduling, dynamic programming – most of this quarter

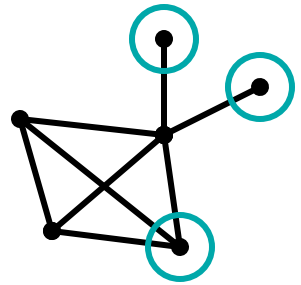
Beyond P?

There are many natural, practical problems for which we don't know any polynomial-time algorithms

Some Examples

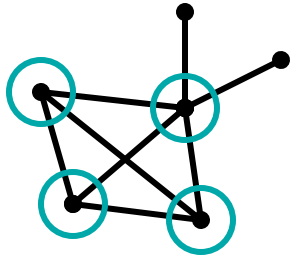
Independent-Set:

Given a graph G=(V,E) and an integer k, is there a subset U of V with |U| ≥ k such that no two vertices in U are joined by an edge.



Clique:

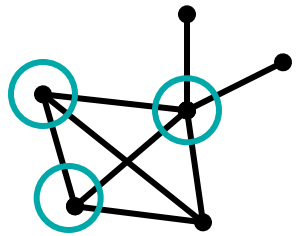
Given a graph G=(V,E) and an integer k, is there a subset U of V with |U| ≥ k such that every pair of vertices in U is joined by an edge.



Some Examples

Vertex-Cover:

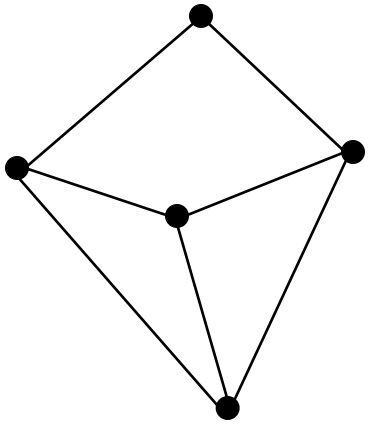
Given a graph G=(V,E) and an integer k, is there a subset U of V with $|U| \leq k$ such that every edge touches a vertex in U.

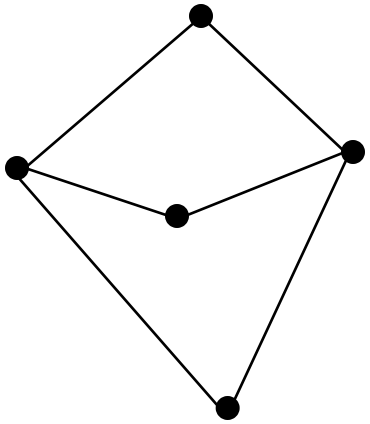


Some Examples

Hamiltonian Cycle:

Given a graph G = (V, E), is there a cycle that visits each node exactly once?

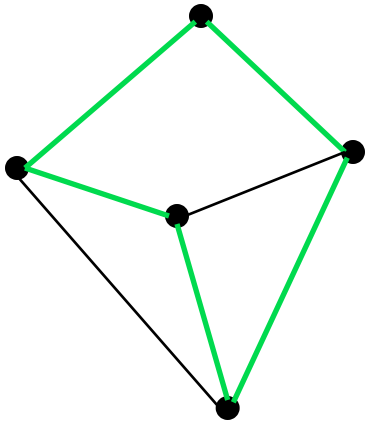




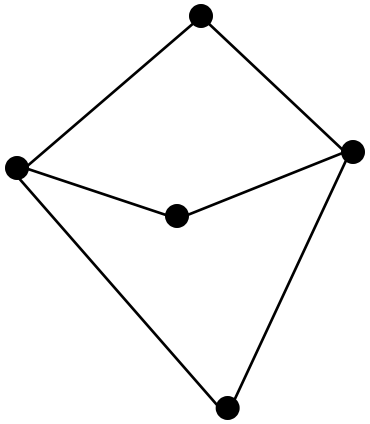
Some Examples

Hamiltonian Cycle:

Given a graph G = (V, E), is there a cycle that visits each node exactly once?



YES



NO

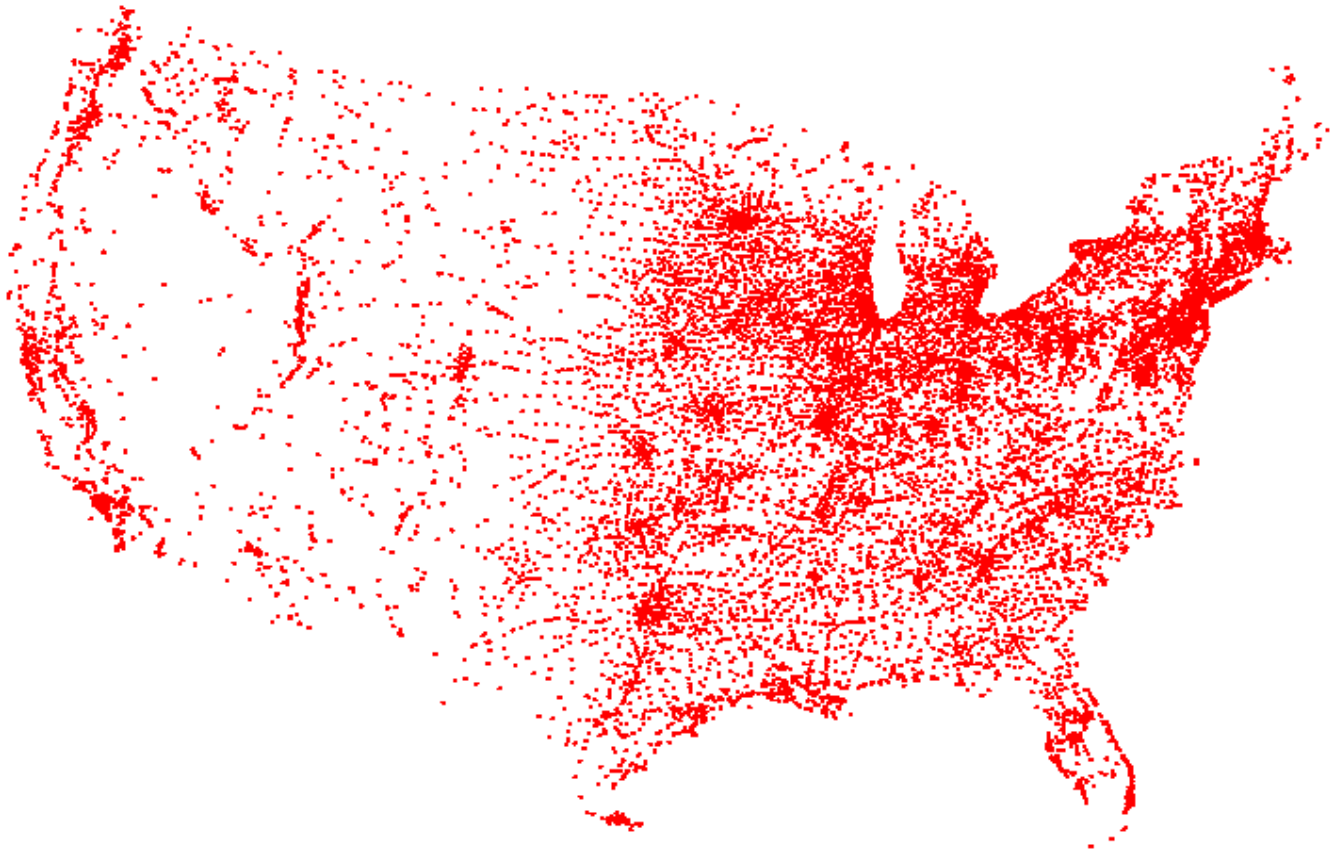
Some Examples

Traveling Salesperson Problem:

Given a weighted graph G=(V,E,w) and an integer k, is there a Hamiltonian cycle with total weight ≤ k?

Traveling Salesperson Problem

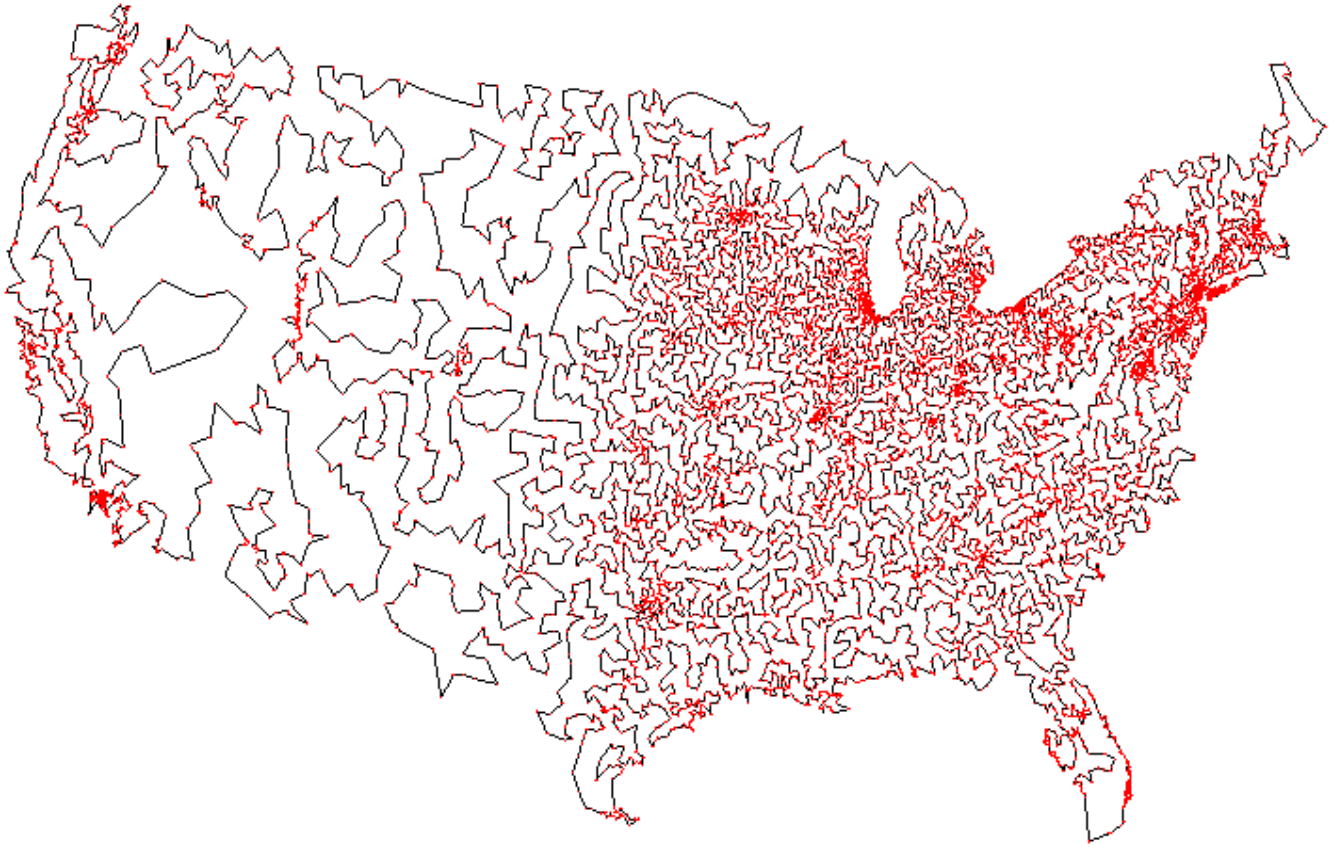
- TSP. Given a weighted graph G=(V,E,w) and an integer k, is there a Hamiltonian cycle with total weight ≤ k?



All 13,509 cities in US with a population of at least 500
Reference: http://www.tsp.gatech.edu

Traveling Salesperson Problem

- TSP. Given a weighted graph G=(V,E,w) and an integer k, is there a Hamiltonian cycle with total weight ≤ k?



Optimal TSP tour
Reference: http://www.tsp.gatech.edu

Satisfiability – Boolean Formulas

Boolean variables $x_1, ..., x_n$

taking values in {0,1}. 0=false, 1=true

Literals

x_i or $\neg x_i$ for i = 1, ..., n

Clause

a logical OR of one or more literals

e.g. $(x_1 \vee \neg x_3 \vee x_7 \vee x_{12})$

CNF formula ("conjunctive normal form")

a logical AND of a bunch of clauses

Satisfiability

CNF formula example

$(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4)$

If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable

the one above is, the following isn't

$x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$

Satisfiability: Given a CNF formula F, is it satisfiable?

Satisfiable?

$$(\ x \vee y \vee z\ ) \wedge (\ \neg x \vee y \vee \neg z\ ) \wedge$$
$$(\ x \vee \neg y \vee z\ ) \wedge (\ \neg x \vee \neg y \vee z\ ) \wedge$$
$$(\ \neg x \vee \neg y \vee \neg z\ ) \wedge (\ x \vee y \vee z\ ) \wedge$$
$$(\ x \vee \neg y \vee z\ ) \wedge (\ x \vee y \vee \neg z\ )$$

$$(\ x \vee y \vee z\ ) \wedge (\ \neg x \vee y \vee \neg z\ ) \wedge$$
$$(\ x \vee \neg y \vee \neg z\ ) \wedge (\ \neg x \vee \neg y \vee z\ ) \wedge$$
$$(\ \neg x \vee \neg y \vee \neg z\ ) \wedge (\ \neg x \vee y \vee z\ ) \wedge$$
$$(\ x \vee \neg y \vee z\ ) \wedge (\ x \vee y \vee \neg z\ )$$

History – As of 1970

Many of the above problems had been studied for decades.

All had real, practical applications.

None were known to be in P. Exponential algorithms were the best known.

It turns out they all have a very deep similarity under the skin. They all belong to a class of problems called NP.

NP: problems with efficient verifiers

Verification algorithm intuition:

- Verifier views things from "managerial" viewpoint.
- Verifier doesn't determine whether a problem instance is YES on its own.
- Rather, it checks a proposed proof (certificate) that an instance is YES.

NP stands for nondeterministic polynomial time

The complexity class NP

NP consists of all decision problems where

You can verify the YES answers efficiently (in polynomial time) given a short (polynomial-size) certificate

And

No certificate can fool your polynomial time verifier into saying YES for a NO instance

Precise Definition of NP

A decision problem is in NP iff there is a polynomial time procedure v(-,-), and an integer k such that

for every YES problem instance x there is a certificate h with $|h| \leq |x|^k$ such that v(x,h) = YES and

for every NO problem instance x there is no certificate h with $|h| \leq |x|^k$ such that v(x,h) = YES

Example: CLIQUE is in NP

procedure v(x,h)

if

x represents a graph G and h represents a set of vertices U.

and

there is an edge in G between each pair of vertices in U

then output YES

else output NOT CONVINCED

Is it correct?

For every x = (G,k) such that G contains a k-clique, there is a certificate h that will cause v(x,h) to say YES, namely h = a list of the vertices in such a k-clique

and

No certificate can fool v into saying yes if either x isn't well-formed (the uninteresting case) or if x = (G,k) but G does not have any cliques of size k (the interesting case)

Another example: SAT ∈ NP

Certificate: the satisfying assignment A

Verifier: v(F,A) = syntax(F,A) && satisfies(F,A)

- Syntax: True iff F is a well-formed formula & A is a truth-assignment to its variables
- Satisfies: plug A into F and evaluate

Correctness:

- If F is satisfiable, it has some satisfying assignment A, and we'll recognize it
- If F is unsatisfiable, it doesn't, and we won't be fooled

Keys to showing that a problem is in NP

What's the output? (must be YES/NO)

First, describe the certificate and verifier.

Second, for every YES instance, show that there is a certificate that would cause the verifier to output YES in polynomial time.

Third, for every NO instance, show that there is *no* certificate that would cause the verifier to output YES, i.e. the verifier can't be tricked.

Another Example: Hamiltonian Cycle $\in$ NP

Certificate: a list of the vertices in the cycle

Verifier: check that inputs are well-formed, that there is an edge between each of the vertices in certificate, and that every vertex appears exactly once.

Correctness:

If YES instance: then there is a certificate corresponding to a Hamiltonian cycle, and verifier outputs YES in polynomial time.

If NO instance: no certificate will fool the verifier.

Are all problems in NP?

No, think about Tautology: given a boolean formula, decide whether it is *always* true.

Not clear what a certificate would look like.

How would one efficiently show (or check) that all assignments evaluate to true?

Review

- Described move from optimization to decision problems
- Described complexity class P
- Described complexity class NP
- Showed that a bunch of problems are in NP
- Next up: NP – who cares?
- In the book: 8.3 -> 8.1 -> 8.2 -> 8.4

P vs. NP

If a problem is in P, then we can construct a verifier that ignores the certificate and just solves the problem.

This verifier satisfies our requirements for being in NP.

Thus,

$$P \subseteq NP$$

Example: Graph Connectivity

Graph-Connectivity: Is graph G = (V, E) connected?

Certificate: “”

Verifier: Ignore certificate. Run BFS (or DFS) to determine if graph is connected. If so, output YES. Else, output NOT CONVINCED.

Proving Connectivity is in NP

First, for every YES instance, the verifier outputs YES given the certificate “”

Second, for every NO instance, the verifier will never output YES, no matter what the certificate is.

So Graph-Connectivity is in NP.

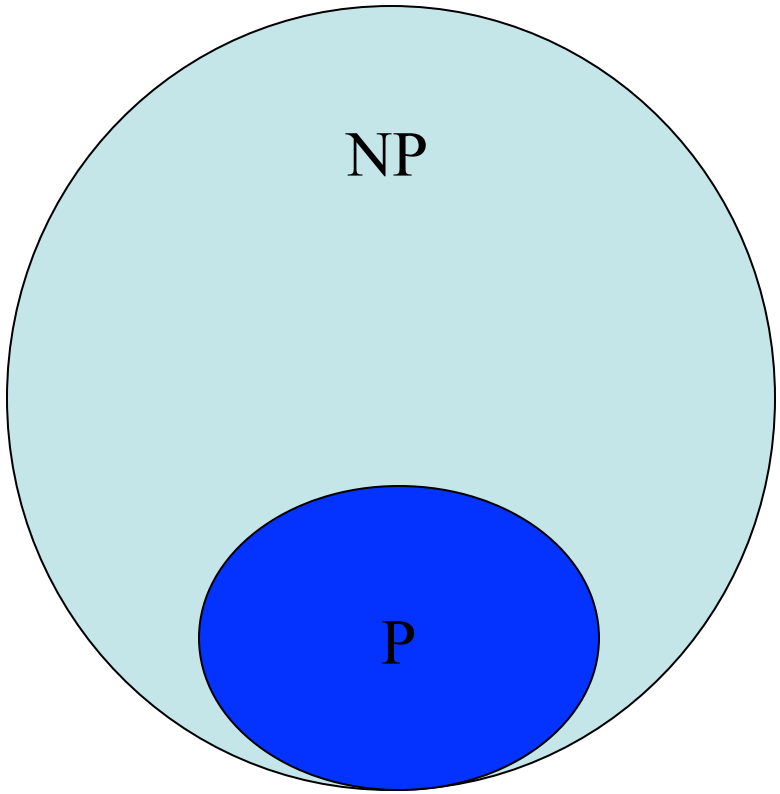
This works for any problem in P.

Hence, $P \subseteq NP$

Complexity Classes

NP = Polynomial-time
verifiable

P = Polynomial-time
solvable



$P \subseteq NP$

P vs. NP

But does P = NP?

How would we answer this question?

- Yes: provide a polynomial time algorithm for every problem in NP

- No: find just one problem in NP and prove there is no polynomial time algorithm for it

Doing either of these is worth $1M

But a beautiful theory was developed

NP-complete: the "hardest" problems in NP

As long as P ≠ NP (seems likely), there is no polynomial time algorithm for any NP-complete problem

We can show that lots of problems are NP-complete

SAT, Clique, Vertex Cover, Independent Set, TSP, etc.

But a beautiful theory was developed

NP-complete: the "hardest" problems in NP

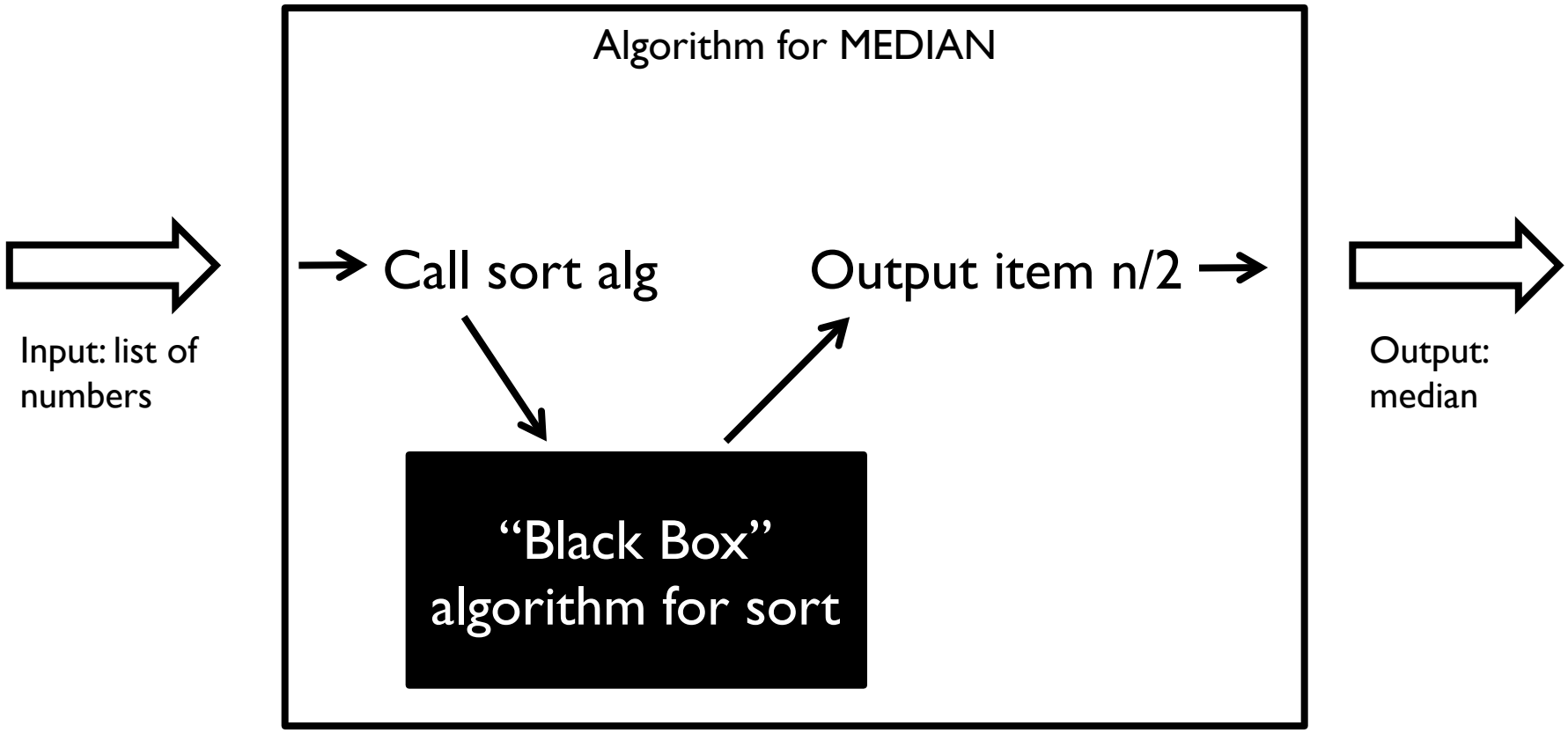
What does it mean for one problem to be harder than another? Before defining NP-complete, we need to define this:

Polynomial-Time Reductions

Reductions: a useful tool

Intuitive Definition: To reduce A to B means to solve A, given a "black box" subroutine solving B.

Reducing MEDIAN to SORT



Since we can use alg for SORT to solve MEDIAN, SORT is "at least as hard" as MEDIAN

More reductions

Example: reduce MEDIAN to SORT

Solution: sort, then select (n/2)nd

Example: reduce SORT to FIND_MAX

Solution: FIND_MAX, remove it, repeat

Example: reduce MEDIAN to FIND_MAX

Solution: transitivity: compose solutions above.

Interlude: some notation

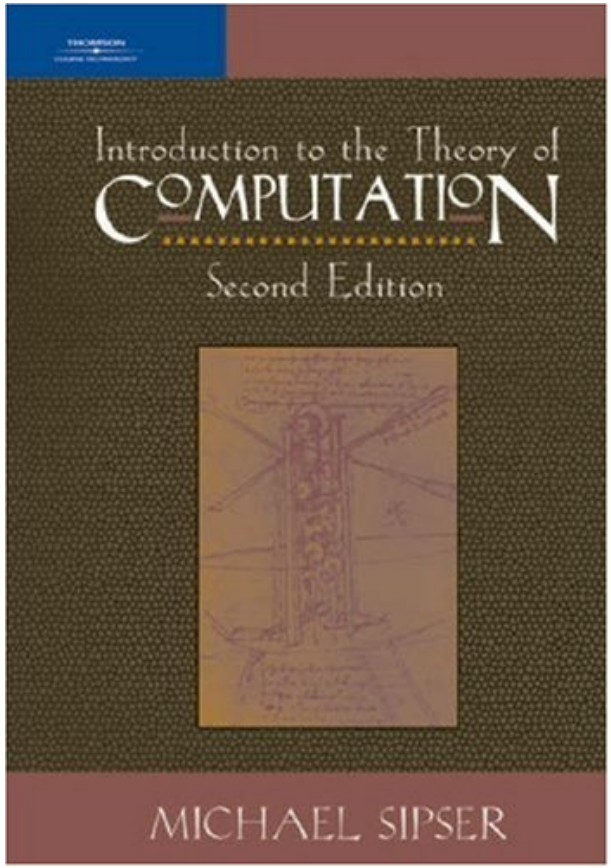
Let A be a problem and x be an input to A.

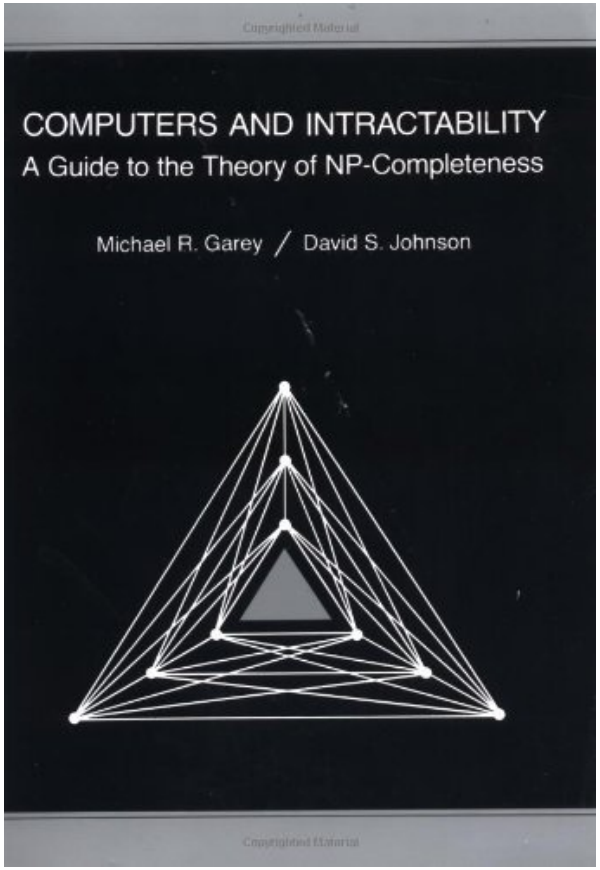
If x is a YES instance, we write $x \in A$

If x is a NO instance, we write $x \notin A$

Comes from a more formal treatment of this material, in which problems can be thought of as sets of strings.

If you want to learn more…





Polynomial-Time Reductions

Sometimes the direction of this inequality confuses people

Let A and B be two problems.

We say that A is *polynomially reducible* to B ($A \leq_p B$) if there exists a polynomial-time algorithm f that converts each instance x of problem A to an instance f(x) of B such that:

x is a YES instance of A iff f(x) is a YES instance of B

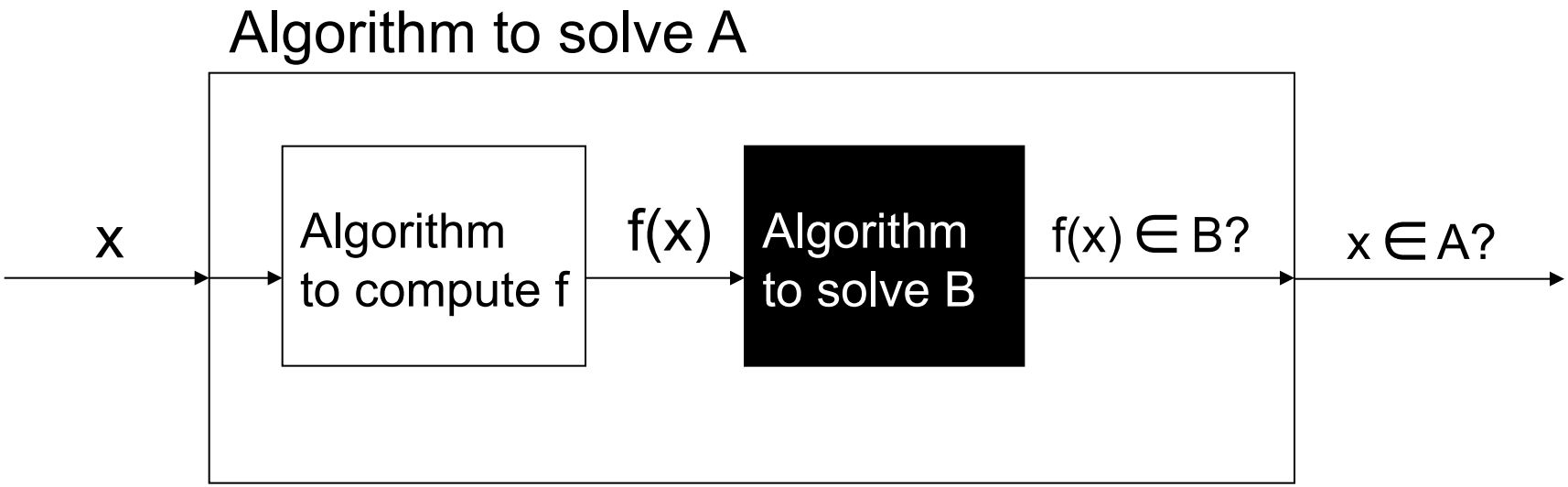
$$x \in A \quad \Leftrightarrow \quad f(x) \in B$$

Why $\leq_p$ notation?

Define: $A \leq_p B$ "A is polynomial-time reducible to B", iff there is a polynomial-time computable function f such that: $x \in A \Leftrightarrow f(x) \in B$

"complexity of A" ≤ "complexity of B" + "complexity of f"

$A \leq_p B$ pictorially



Example: Vertex Cover $\leq_P$ 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 (i.e. that cover U)?

Ex:

U = { 1, 2, 3, 4, 5, 6, 7, 8, 9 }

k = 3

S_1 = {1, 7, 8, 9}

S_2 = {3, 4, 5, 6}

S_3 = {1}

S_4 = {2, 4, 9}

S_5 = {5, 8}

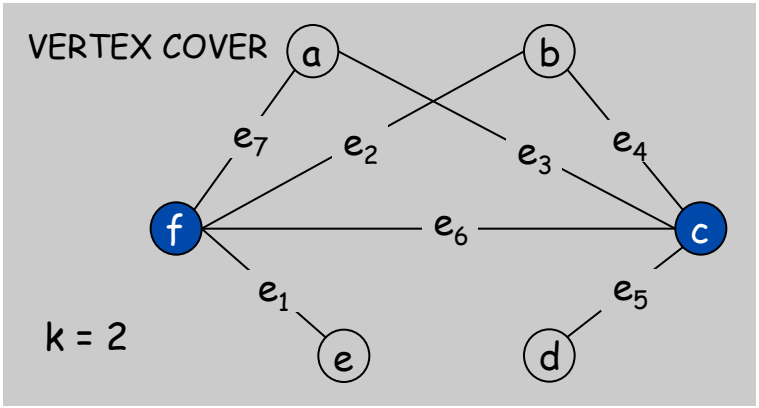
S_6 = {1, 2, 6, 7}

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.

Example: Vertex Cover $\leq_P$ Set Cover

Vertex Cover $\leq_P$ Set Cover because Set Cover is a generalization of Vertex Cover



f(x)

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}

Example: Vertex Cover $\leq_P$ Set Cover

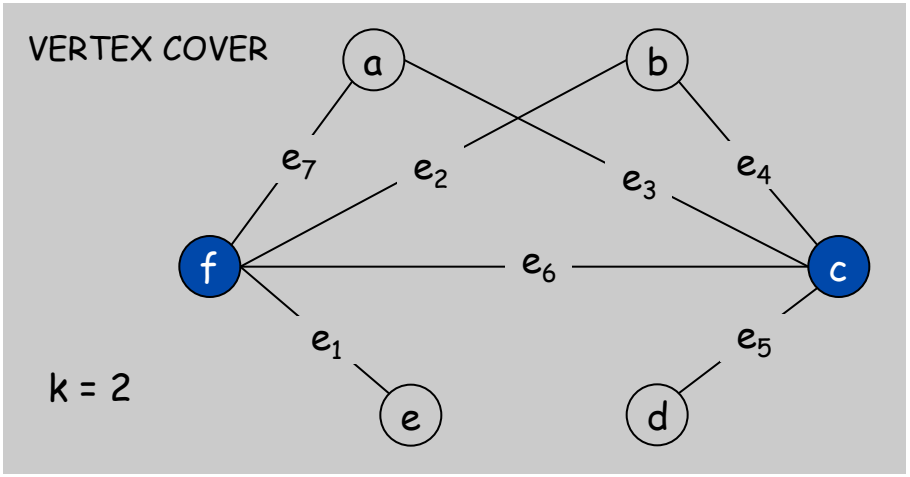
Claim. Vertex Cover $\leq_P$ Set Cover.

Pf. Given a Vertex Cover instance $x = (G = (V, E), k)$, we construct a Set Cover instance $f(x)$ as follows:

- $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$

$f(x)$ can be computed in polynomial time.

$x \in$ Vertex Cover iff $f(x) \in$ Set Cover, because there is set cover of size $\leq k$ iff vertex cover of size $\leq k$. ▪



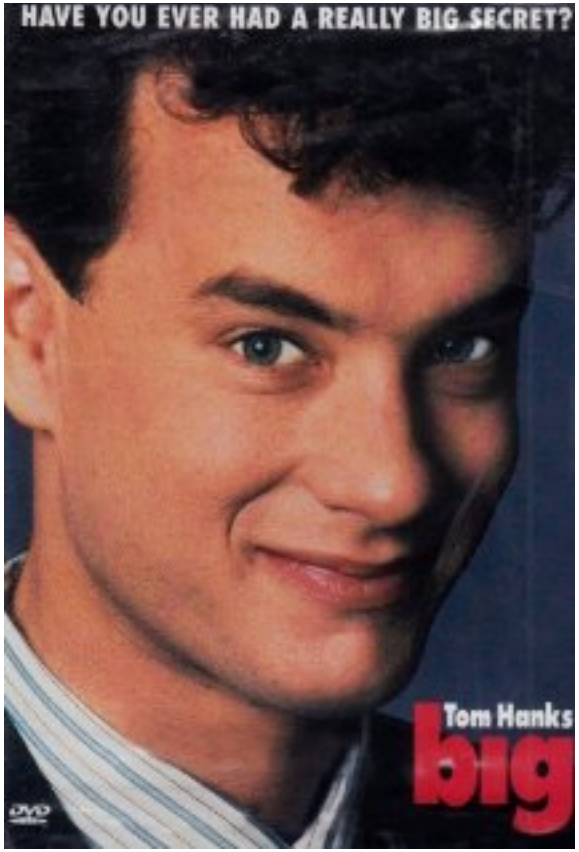
SET COVER

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

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

Why do we care about reductions?

We'll see *plenty* more reductions, but let's come back to the big picture first.



What does a reduction tell us?

Observation: p(x) and q(x) polynomials, then

p(x) + q(x) is polynomial

(1) $A \leq_p B$ and $B \in P \Rightarrow A \in P$

(2) $A \leq_p B$ and $A \notin P \Rightarrow B \notin P$

(3) $A \leq_p B$ and $B \leq_p C \Rightarrow A \leq_p C$ (transitivity)

Definition of NP-Completeness

Definition: Problem B is *NP-hard* if every problem in NP is polynomially reducible to B.

Definition: Problem B is *NP-complete* if:

(1) B belongs to NP, and

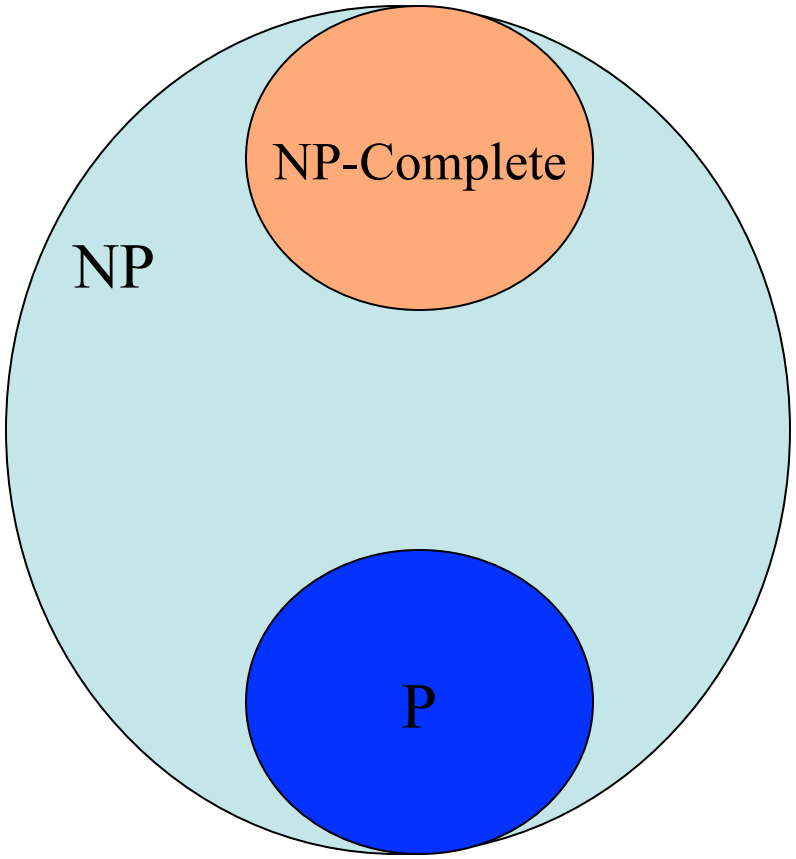
(2) B is NP-hard.

Complexity Classes

P = Poly-time **solvable**

NP = Poly-time **verifiable**

NP-Complete = **"Hardest"** problems in **NP**



NP-completeness

Cool concept, but are there any such problems?

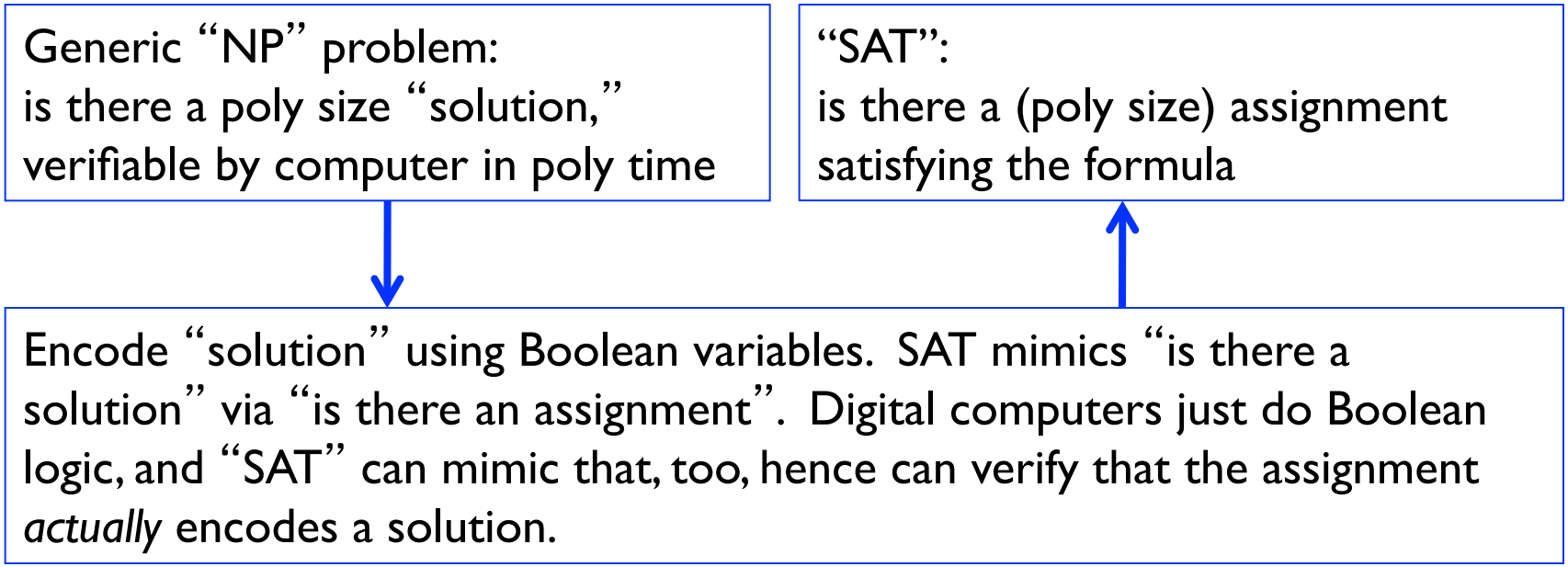
Yes!

Cook-Levin theorem (1971):

SAT is NP-complete

Why is SAT NP-complete?

Proof of Cook-Levin is somewhat involved; I won't show it. But its essence is not so hard to grasp:



Proving a problem is NP-complete

Technically, for condition (2) we have to show that every problem in NP is reducible to B.
(Yikes! Sounds like a lot of work.)

For the very first NP-complete problem (SAT) this had to be proved directly.

However, once we have one NP-complete problem, then we don't have to do this every time.

Why? Transitivity.

Re-stated Definition

Lemma: Problem B is NP-complete if:

(1) B belongs to NP, and

(2') A is polynomial-time reducible to B, for some problem A that is NP-complete.

That is, to show (2') given a new problem B, it is sufficient to show that SAT or any other NP-complete problem is polynomial-time reducible to B.

Usefulness of Transitivity

In order to show that P is NP-hard, we only have to show P' $\leq_p$ P for some NP-hard problem P', Why?

1) Since P' is NP-hard,

$$\forall\ P'' \in NP, \text{ we have } P'' \leq_p P'$$

2) If we show P' $\leq_p$ P, then by transitivity we know that: $\forall\ P'' \in NP$, we have P'' $\leq_p$ P.

Thus P is NP-hard.

Ex: VertexCover is NP-complete

SAT is NP-complete (shown by S. Cook)

SAT $\leq_p$ VertexCover (we'll show this later)

VertexCover is in NP (why?)

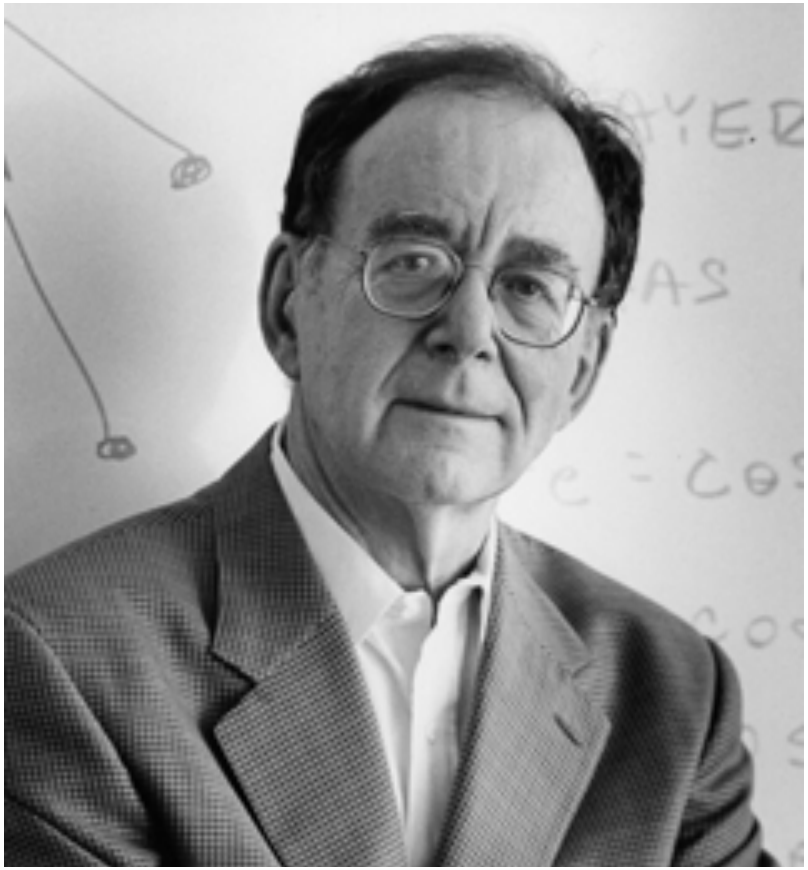
Therefore VertexCover is also NP-complete

So, poly-time algorithm for VertexCover would give poly-time algs for everything in NP

NP-completeness

Karp (1972):
$SAT \leq_p$ Clique,
$SAT \leq_p$ Vertex Cover,
$SAT \leq_p$ Ham Path, …

Since, then, thousands more problems proved NP-complete



NP-completeness

If there was a polynomial time algorithm for any NP-complete problem, then P = NP.

If at least one cannot be solved in polynomial time, then none could.

So either all NP-complete problems have polynomial time algorithms, or none do. Since no one has ever found a polynomial time algorithm for an NP-complete problem, they are "probably" intractable.

What's next?

Use polynomial time reductions to show that a number of problems we care about are NP-complete.

Important to know how to do this, in order to determine whether you should try to solve a new problem.

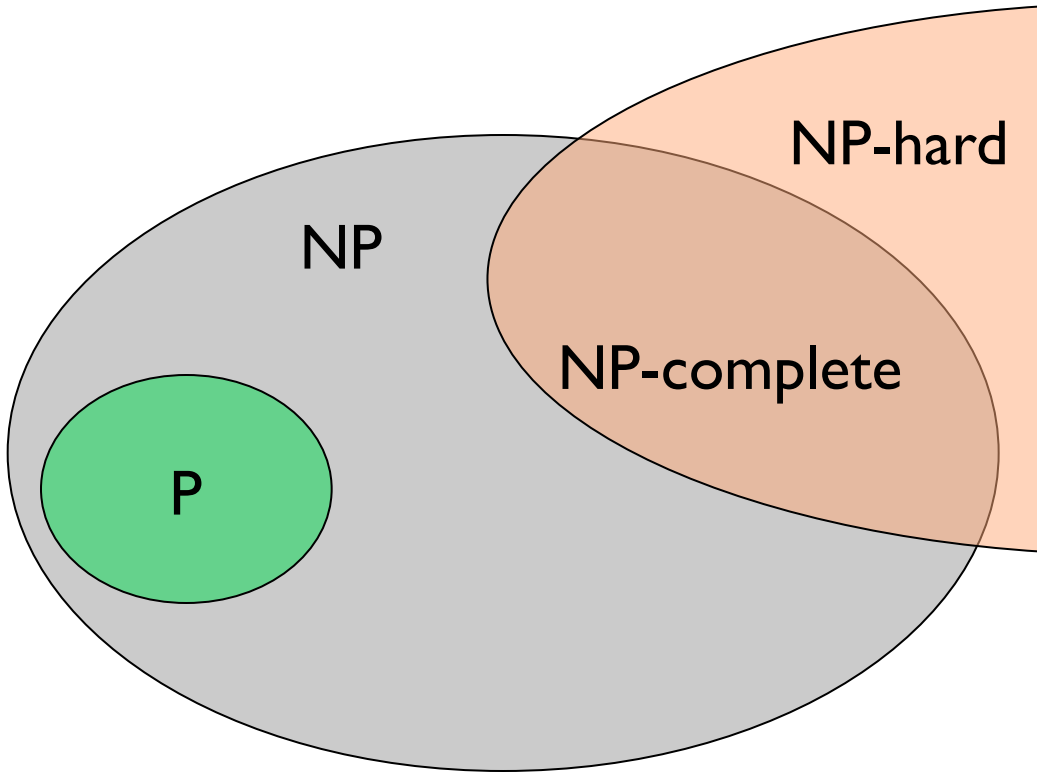
Later, we ask what do we do with all these NP-complete problems?

Review

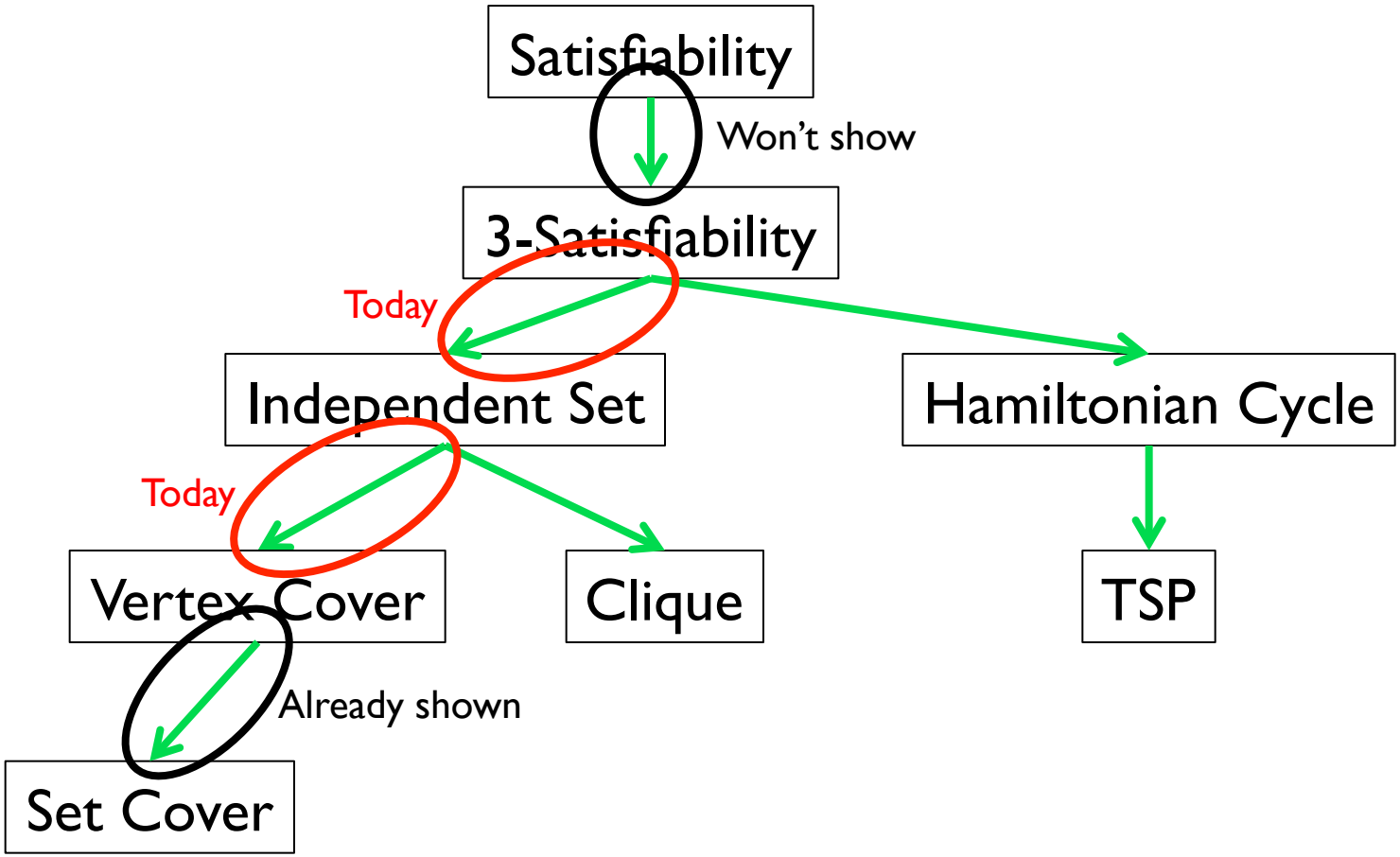
We defined some useful complexity classes.

Cook-Levin Theorem: SAT is NP-complete

To prove a new problem is NP-complete, we need to show a chain of reductions from SAT



Reduction Tree



NP-completeness proof outline

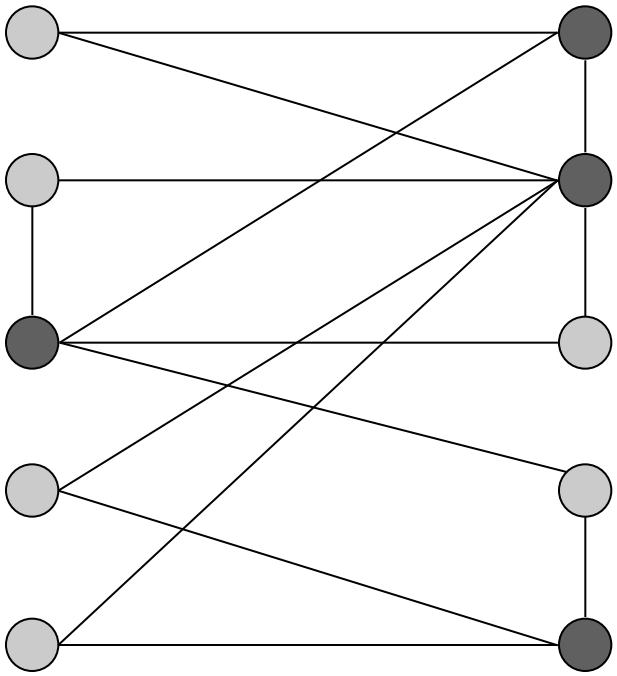
To show a problem P is NP-complete:

1. Show it is NP (usually easy).
2. For a problem P' known to be NP-complete, show that $P' \leq_p P$.
   a. Provide an algorithm (function) f for transforming input of P' to input of P.
   b. Argue that f can be computed in polynomial time (usually easy).
   c. Show that $x \in P' \Leftrightarrow f(x) \in P$.
      i. Show that $x \in P' \Rightarrow f(x) \in P$.
      ii. Show that $f(x) \in P \Rightarrow x \in P'$.

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.

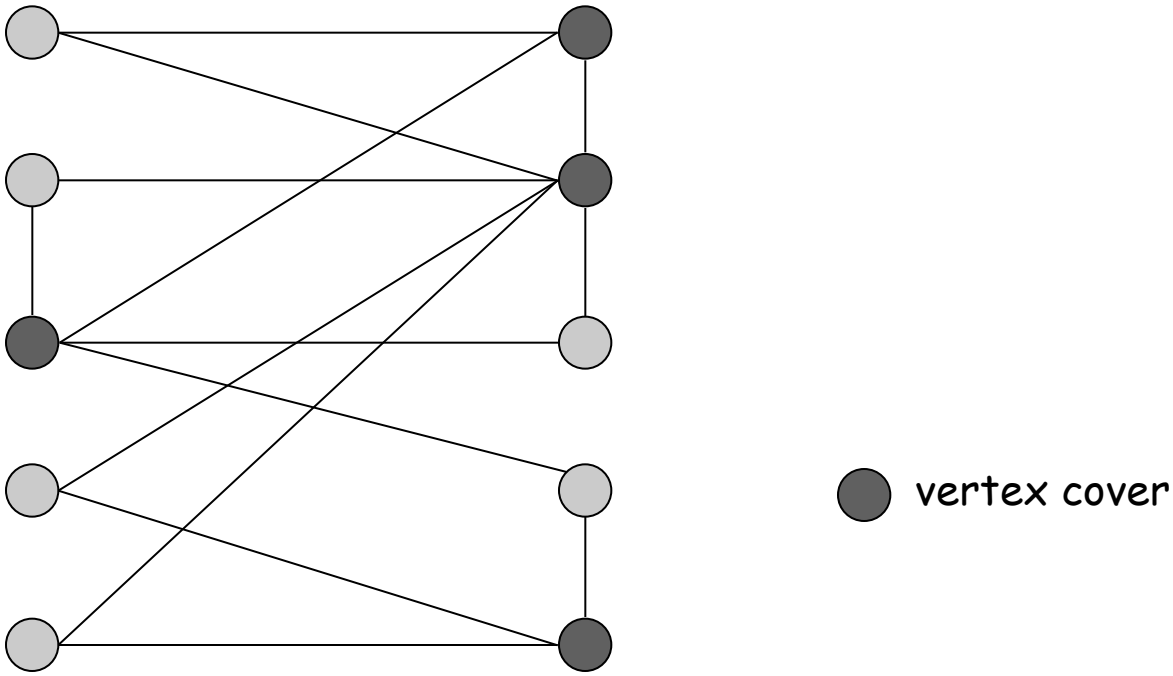


independent set

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.



Proving Vertex Cover NP-complete

Theorem: If Independent Set is NP-complete, then Vertex Cover is NP-complete.

Proof: Vertex Cover is in NP. A certificate consists of the set of vertices in the cover. It can be verified in polynomial time that such a set of vertices has the required size and does cover all edges.

To finish the proof, we will show that Independent Set $\leq_P$ Vertex Cover.

Proving Vertex Cover NP-complete

Observation: 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. ▪

Proving Vertex Cover NP-complete

Theorem: If Independent Set is NP-complete, then Vertex Cover is NP-complete.

Proof (continued): Given an input $x = (G = (V, E), k)$ to Independent Set, let $f(x)$ be the Vertex Cover input $G = (V, E), n - k$. Clearly, f can be computed in polynomial time.

By our observation, x has an independent set of size $\geq k$ iff $f(x)$ has a vertex cover of size $\leq n - k$. Hence $x \in$ Independent Set $\Leftrightarrow f(x) \in$ Vertex Cover.

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

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

Independent Set is NP-complete

Theorem. 3-SAT is NP-complete.

Pf. We won't show this, but there is a reduction from SAT.

Theorem. Independent Set is NP-compete.

Pf. First, Independent Set is clearly in NP. A certificate would consist of the list of vertices. It could be easily verified in polynomial time that no edge has both endpoints in this list of vertices.

We now show that 3-SAT $\leq_P$ Independent Set.

3 Satisfiability Reduces to Independent Set

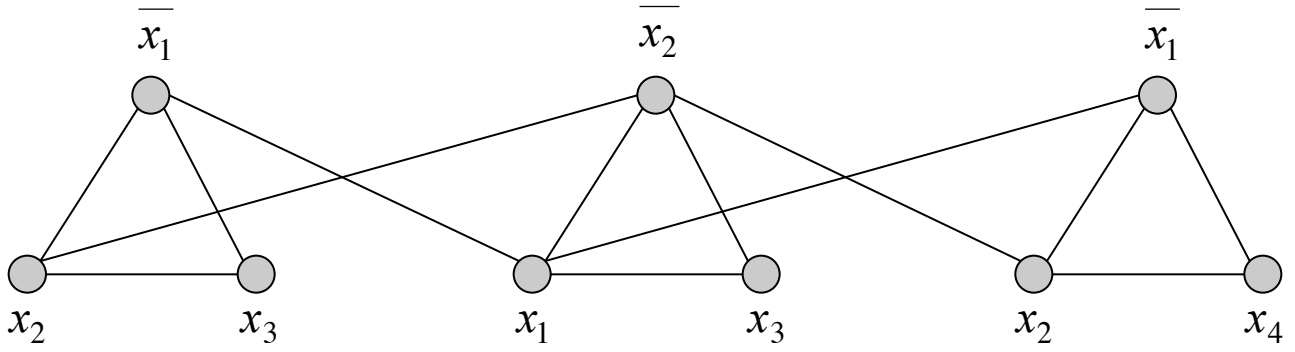
Claim. 3-SAT $\leq_P$ INDEPENDENT-SET.

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

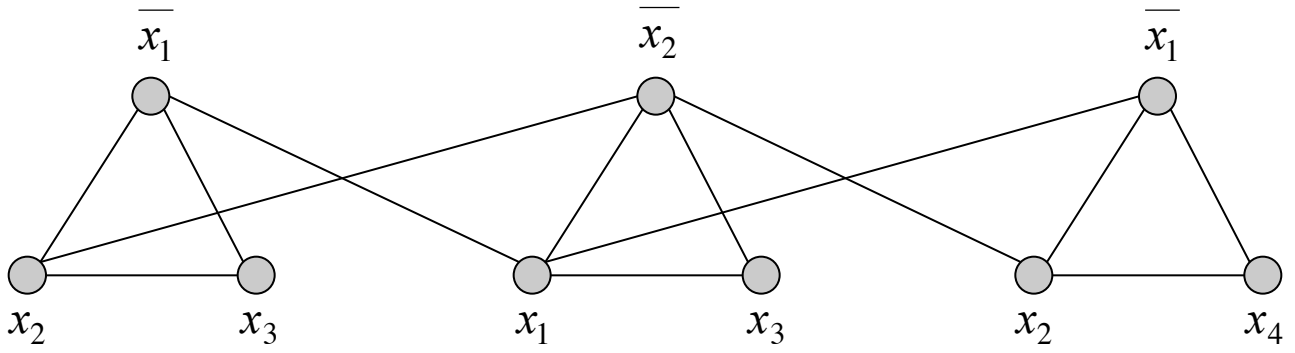
Claim. G contains independent set of size k = $|\Phi|$ iff Φ is satisfiable.

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. ▪

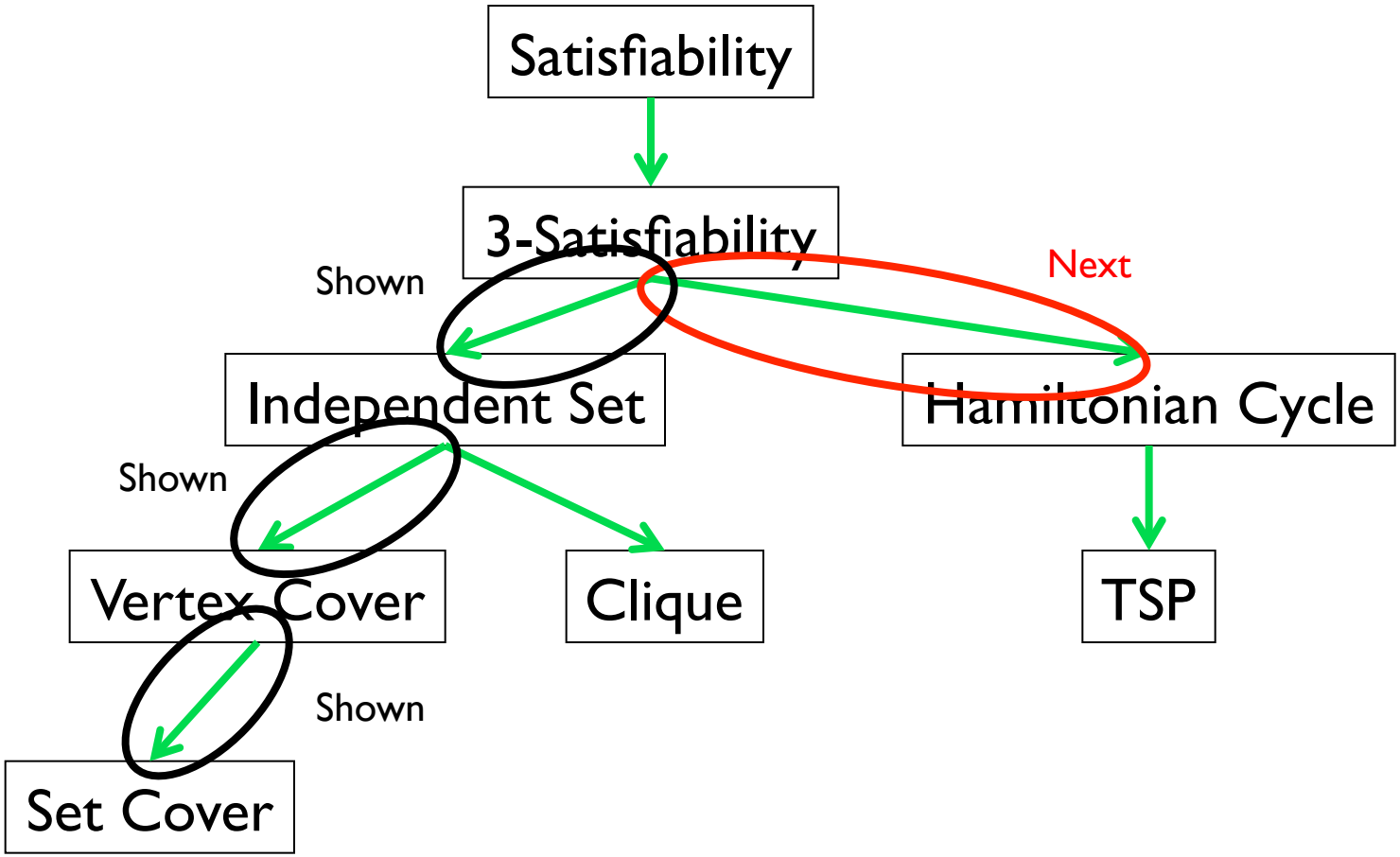
G



k = 3

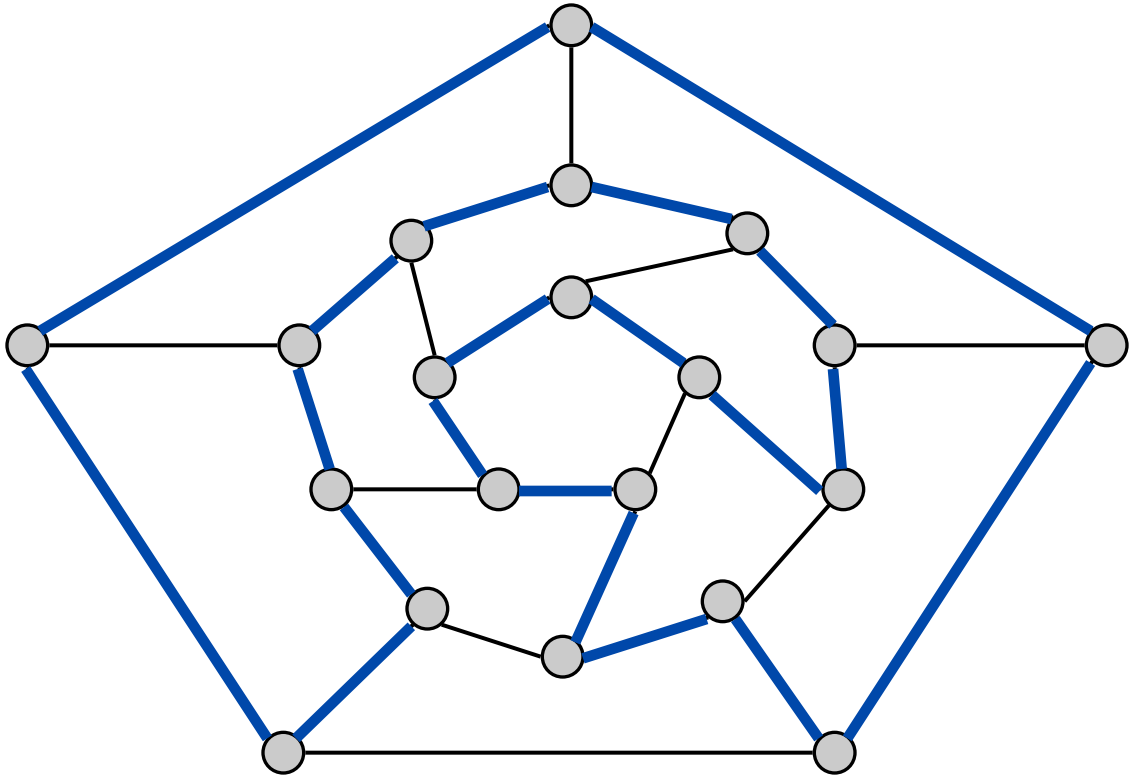
$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3 \right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4 \right)$$

Reduction Tree



Hamiltonian Cycle

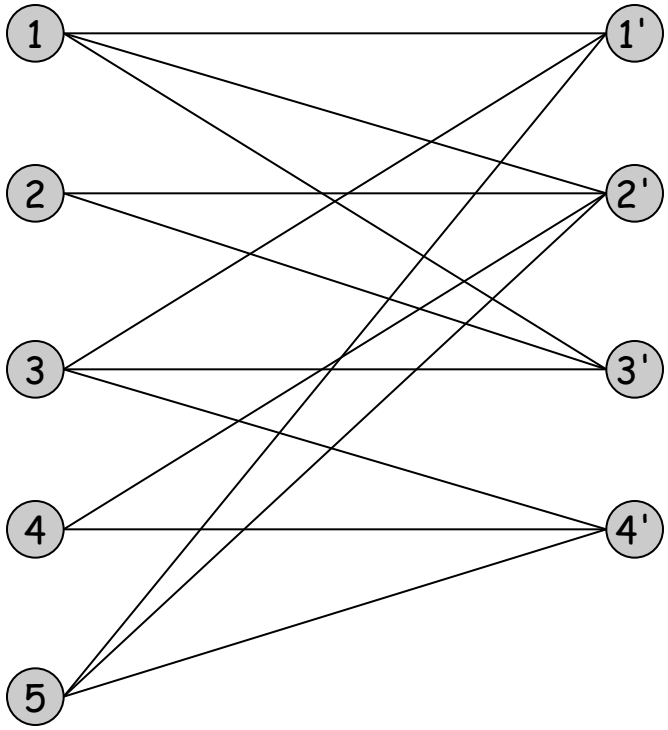
HAM-CYCLE: given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V.



YES

Hamiltonian Cycle

HAM-CYCLE: given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V.



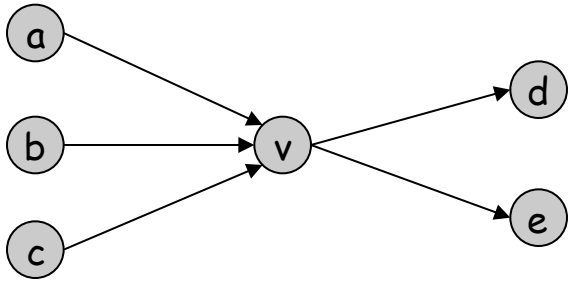
NO

Directed Hamiltonian Cycle

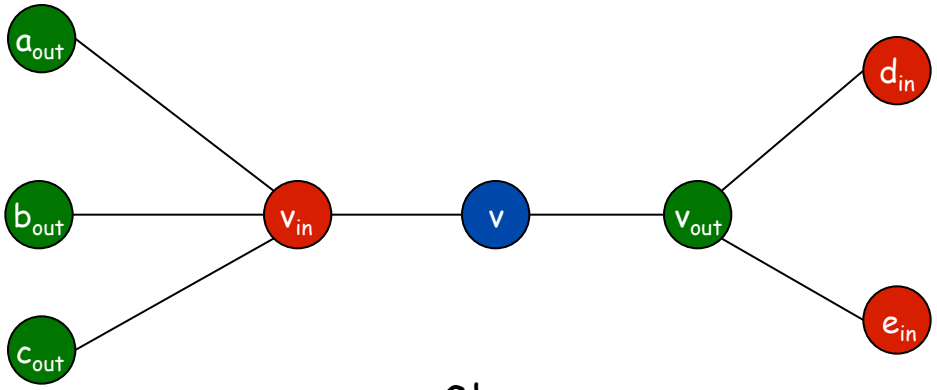
DIR-HAM-CYCLE: given a digraph $G = (V, E)$, does there exists a simple directed cycle Γ that contains every node in V?

Claim. DIR-HAM-CYCLE $\leq_p$ HAM-CYCLE.

Pf. Given a directed graph $G = (V, E)$, construct an undirected graph G' with 3n nodes.



G



G'

3-SAT Reduces to Directed Hamiltonian Cycle

Claim. 3-SAT $\leq_P$ DIR-HAM-CYCLE.

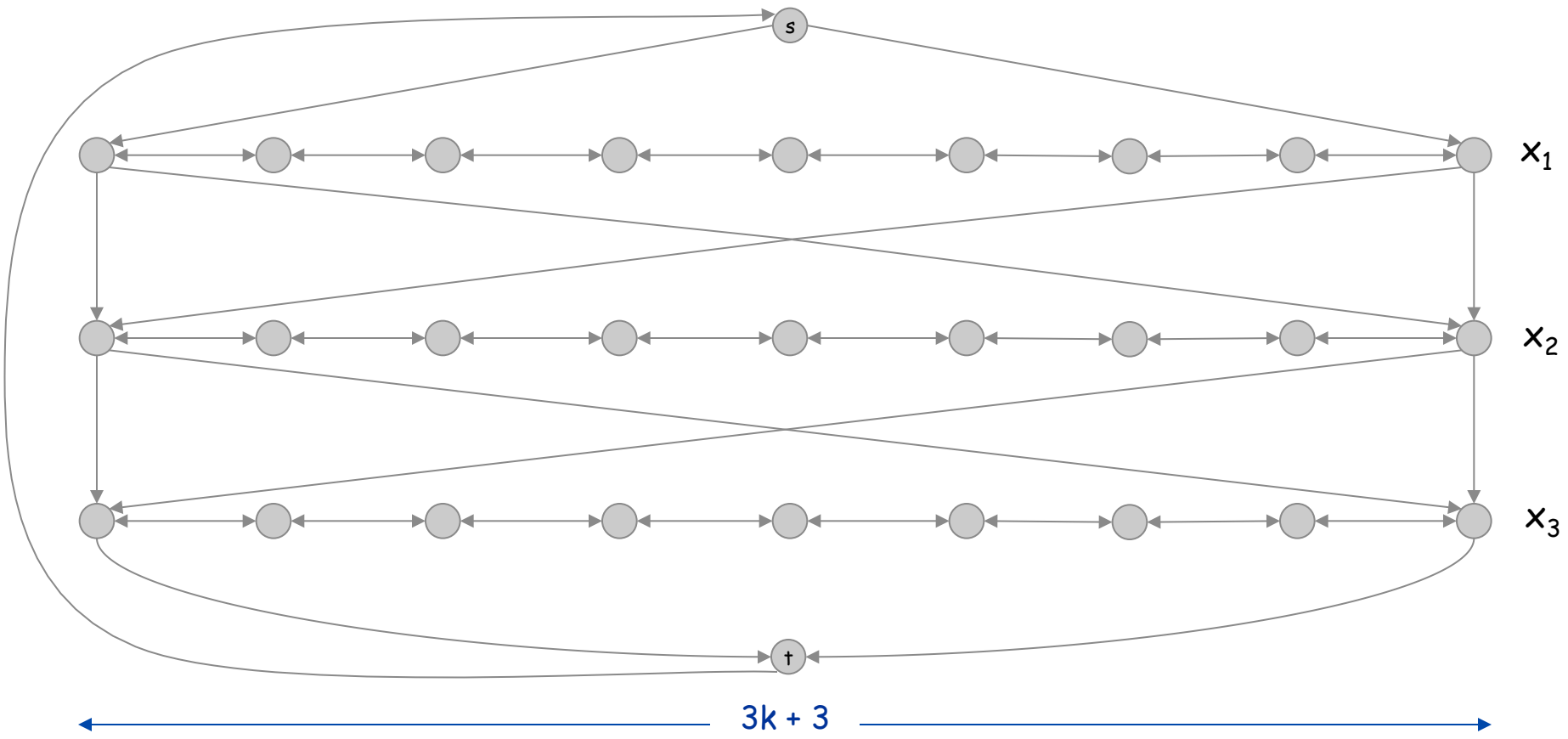
Pf. Given an instance Φ of 3-SAT, we construct an instance of DIR-HAM-CYCLE that has a Hamiltonian cycle iff Φ is satisfiable.

Construction. First, create graph that has 2^n Hamiltonian cycles which correspond in a natural way to 2^n possible truth assignments.

3-SAT Reduces to Directed Hamiltonian Cycle

Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.

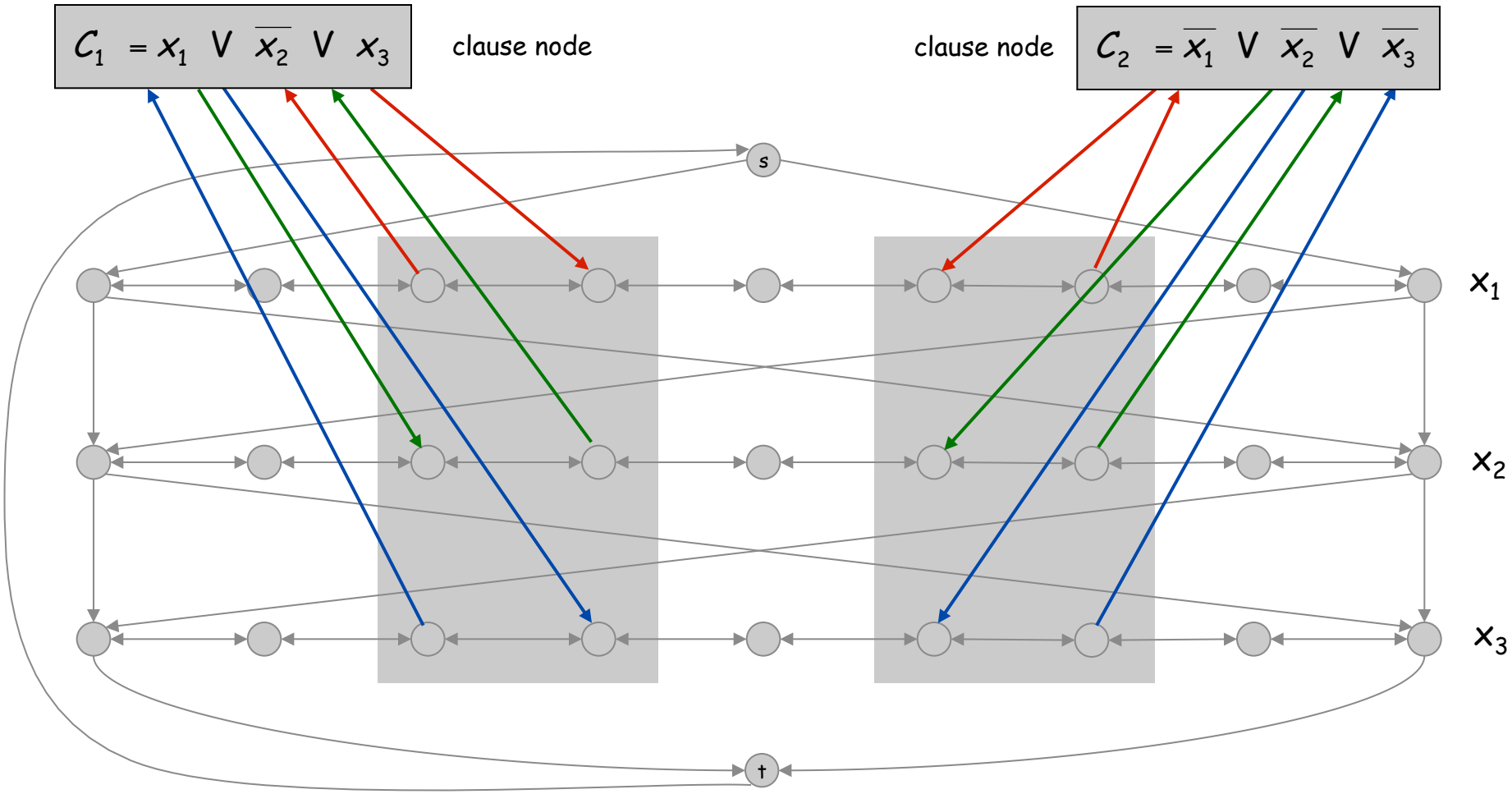
- Construct G to have 2^n Hamiltonian cycles.
- Intuition: traverse path i from left to right $\Leftrightarrow$ set variable $x_i = 1$.



3-SAT Reduces to Directed Hamiltonian Cycle

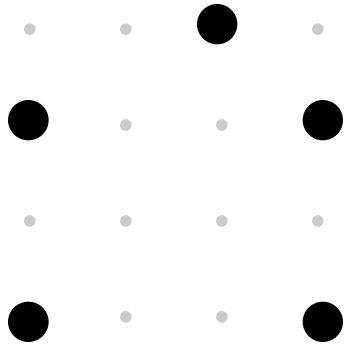
Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.

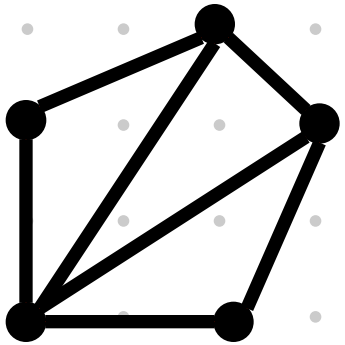
- For each clause: add a node and 6 edges.



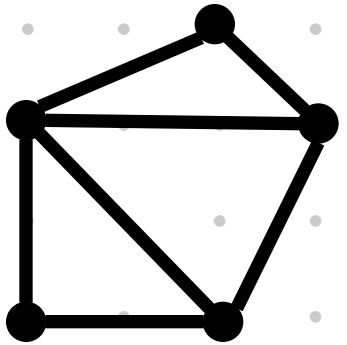
Minimum-Weight Triangulation

Two triangulations of a set of five points:





Weight = 22.9



Weight = 15.7

Minimum Weight Triangulation Problem: Given a set of n points in the plane, find the triangulation of minimum total weight. (Decision version: is there a triangulation of weight <= k?)

Problem first posed in 1970s. Until 2006, "the most longstanding open problem in computational geometry."

Mulzer and Rote (2006): MWT is NP-hard. (Reduction from 3-SAT.)