Analysis of Algorithms

V. Adamchik

Lecture 11

CSCI 570

University of Southern California

NP-Completeness

Reading: chapter 9

In 1935 Alan Turing described a model of computation, known today as the Turing Machine (TM).

A problem P is *computable* (or *decidable*) if it can be solved by a Turing machine that halts on every input.

Alan Turing (1936, age 22)

We say that P has an algorithm.

Turing Machines were adopted in the 1960's, years after his death.

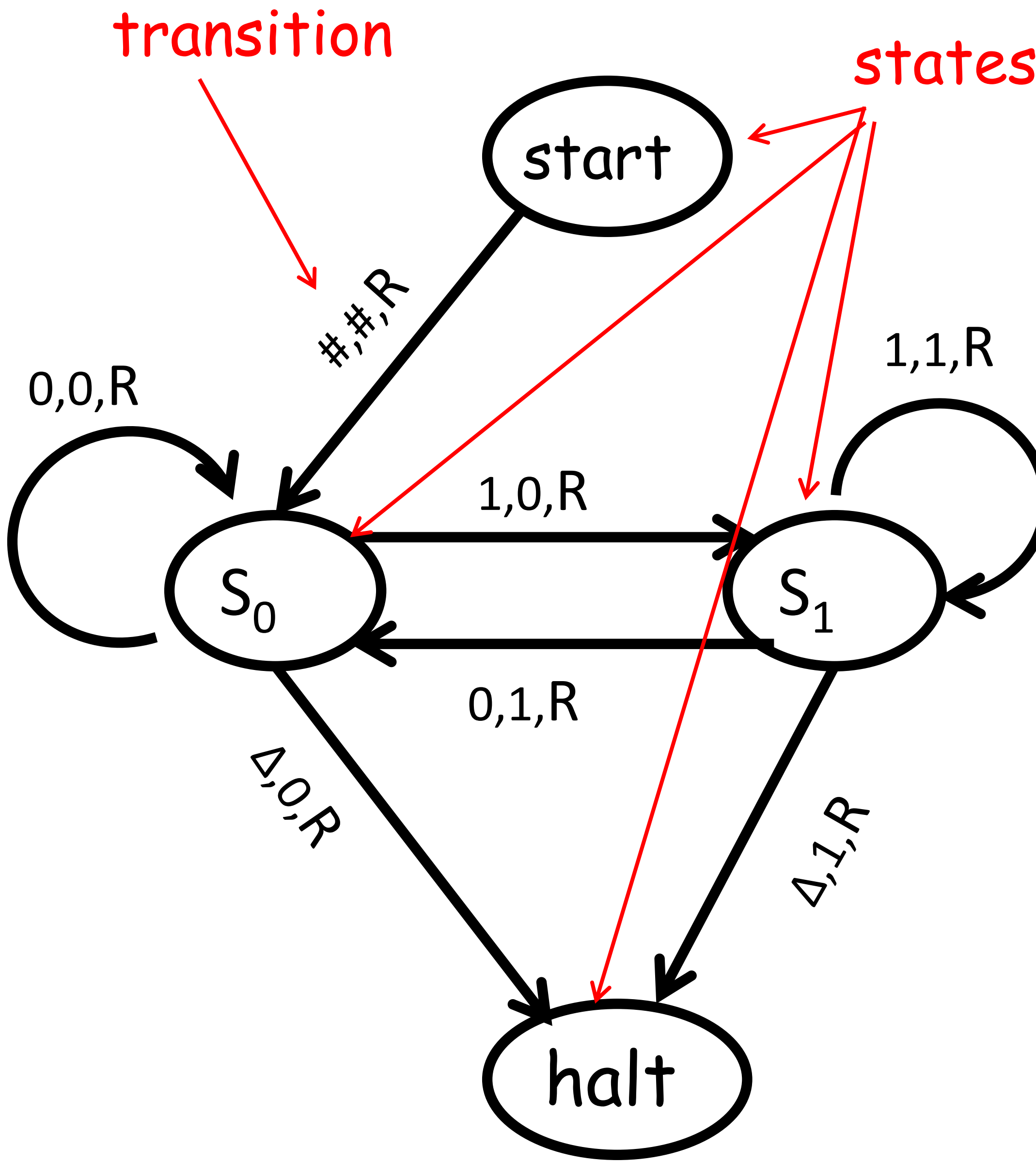
High Level Example of a Turing Machine

The machine that takes a binary string and appends 0 to the left side of the string.

Input: #10010Δ
Output: #**0**10010Δ

- leftmost char
Δ – rightmost char

Transition on each edge
read,write,move (L or R)



The Church-Turing Thesis

"*Any natural / reasonable notion of computation can be simulated by a TM.*"

This is not a theorem.

Is it… *…an observation?*

…a definition?

…a hypothesis?

…a law of nature?

…a philosophical statement?

Everyone believes it. No counterexample yet.

Undecidable Problems

Undecidable means that there is no computer program that always gives the correct answer: it may give the wrong answer or run forever without giving any answer.

paradox

"I am a liar"

The halting problem is the problem of deciding whether a given Turing machine halts when presented with a given input.

Turing's Theorem:

The Halting Problem is not decidable.

Super-Turing computation

In 1995 Hava Siegelmann proposed
Artificial Recurrent Neural Networks (ARNN).

She proved mathematically that ARNNs have computational powers that extend the TM.

She claims that ARNNs can "compute" Turing non-computable functions.

As of today the statement is not proven nor disproven.

Runtime Complexity

Let M be a Turing Machine that halts on all inputs.
Assume we compute the running time purely as a function of the length of the input string.

<u>Definition:</u> The running complexity is the function $f : N \to N$ such that f(n) is the maximum number of steps that M uses on any input of length n.

P and NP complexity classes

$P \subset NP$

P = set of problems that can be solved in polynomial time by a deterministic TM.

$TOTAL \leq K$

NP = set of problems for which solution can be verified in polynomial time by a deterministic TM.

Polynomial Reduction: $Y \leq_p X$

To reduce a decision problem Y to a decision problem X (we write $Y \leq_p X$) we want a function f that maps Y to X such that:

1) f is a polynomial time computable

2) $\forall y \in Y$ (y is instance of Y) is YES
 if and only if $f(y) \in X$ is YES.

If we cannot solve Y, we cannot solve X.

We use this to prove NP completeness: knowing that Y is hard, we prove that X is at least as hard as Y.

if - then

$$Y \le_p X$$

If we can solve X in polynomial time,
we can solve Y in polynomial time.

Examples:

Bipartite Matching $\le_p$ Max-Flow

Circulation $\le_p$ Max-Flow

$Y \leq_p X \leq_p Z \leq_p T$

know prove

If we can solve X, we can solve Y.

The *contrapositive* of the statement "if A, then B" is "if not B, then not A."

If we cannot solve Y, we cannot solve X.

Knowing that Y is hard, we prove that X is harder.

In plain form: X is at least as hard as Y.

Two ways of using $\boxed{Y \le_p X}$

if I know that

1) X is easy

If we can solve X in polynomial time, we can solve Y in polynomial time.

NF, LP

today

2) Y is hard

Then X is at least as hard as Y

From a hard problem TO your problem X

$Y \rightarrow X$

NP-Hard and NP-Complete

optimization

X is *NP-Hard*, if $\forall Y \in NP$ and $Y \leq_p X$.

decision

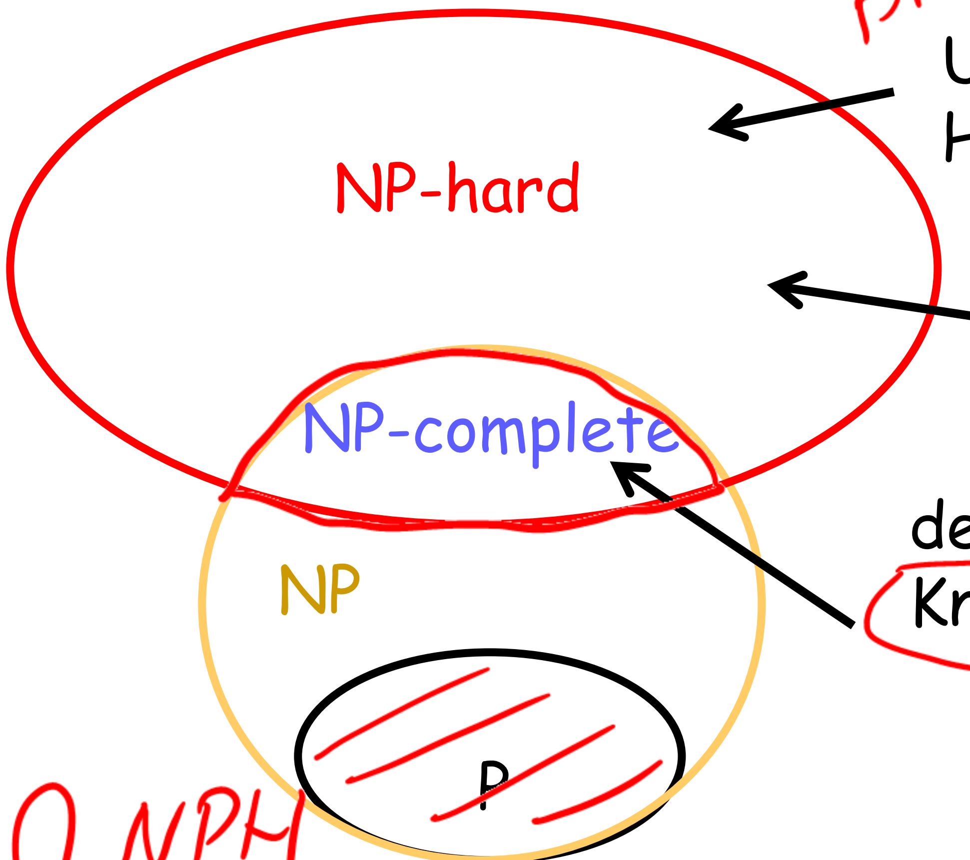
X is *NP-Complete*, if X is NP-Hard and $X \in NP$.

decision: $\exists$ cycle $\in$

opt: $\exists$ min length cycle

Venn Diagram (P ≠ NP)

NPH problems do not have to be in NP.



prove it!

Undecidable:
Halting Problem

optimization

decision
Knapsack

NPC problems are the most difficult NP problems.

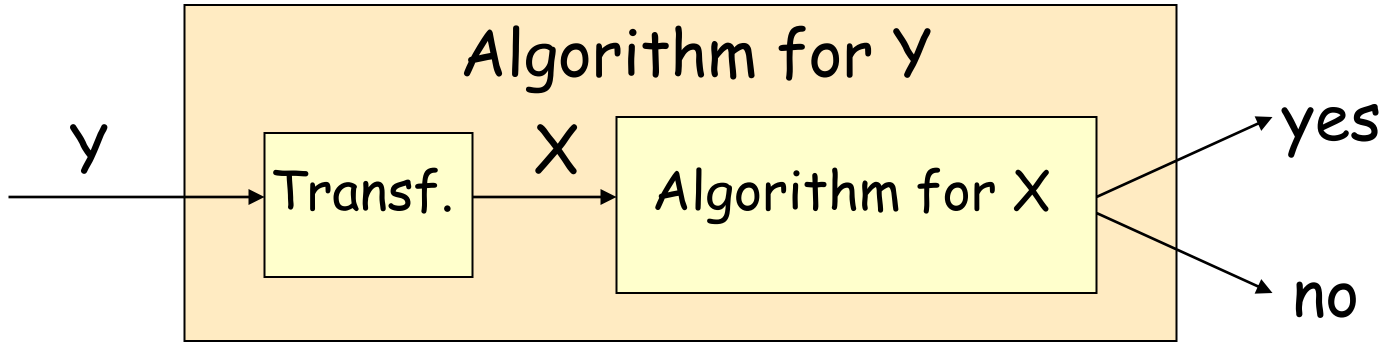
$NPC = NP \cap NPH$

It's not known if NPC problems can be solved by a *deterministic* TM in polynomial time.

NP-Completeness Proof Method

To show that X is NP-Complete:

1) Show that X is in NP
2) Pick a problem Y, known to be an NP-Complete
3) Prove $Y \leq_p X$ (reduce Y to X)



Boolean Satisfiability Problem (SAT)

A propositional logic formula is built from variables, operators AND (conjunction, $\wedge$), OR (disjunction, $\vee$), NOT (negation, $\neg$), and parentheses:

0/1 AND OR

$$(X_1 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_4 \vee X_5) \wedge \ldots \quad = \text{TRUE}$$

clause clause

A formula is said to be satisfiable if it can be made TRUE by assigning appropriate logical values (TRUE, FALSE) to its variables.

A formula is in conjunctive normal form $\boxed{\text{(CNF)}}$ if it is a conjunction of clauses.
A literal is a variable or its negation.
A clause is a disjunction of literals.

Cook-Levin Theorem (1971)

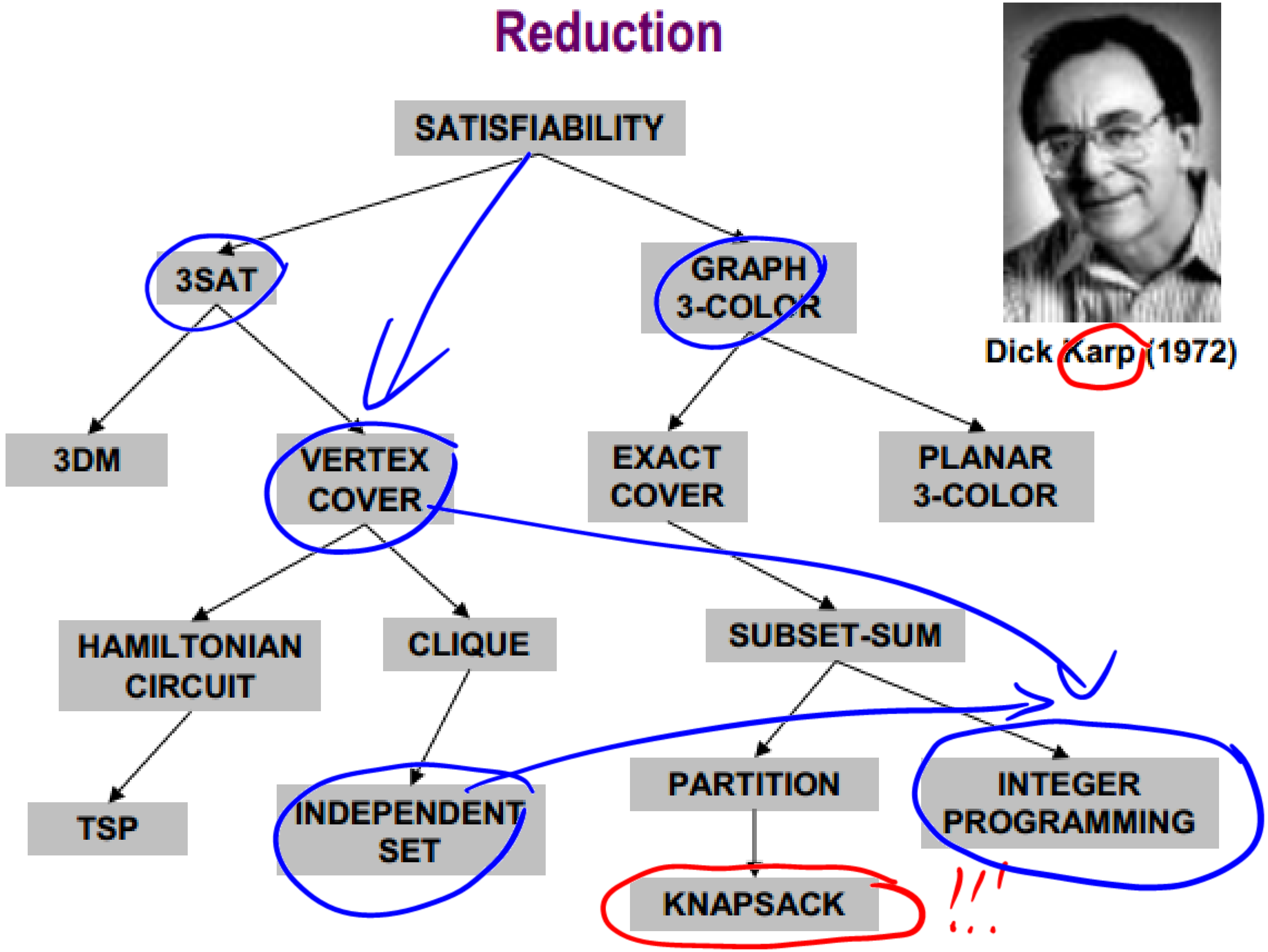
Theorem. CNF SAT is NP-complete.

No proof…

Cook received a Turing Award for his work.

You are not responsible for knowing the proof.

Reduction



Dick Karp (1972)

Karp introduced the now standard methodology for proving problems to be NP-Complete.

He received a Turing Award for his work (1985).

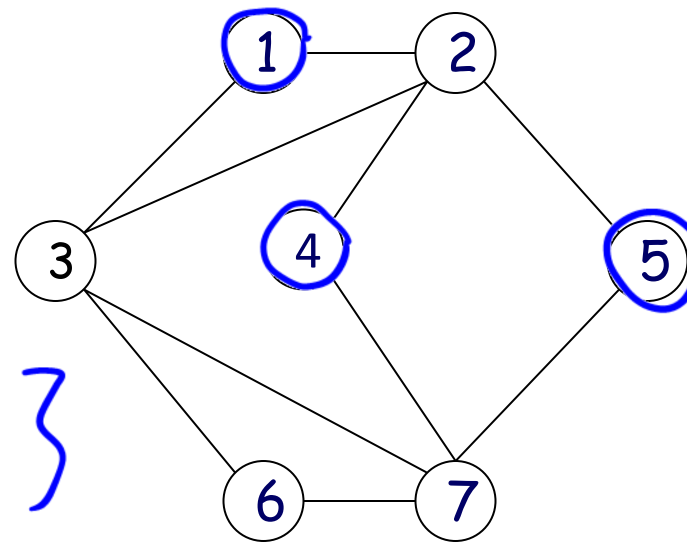
Independent Set (IS)

Given a graph, we say that a subset of vertices is "independent" if no two of them are joined by an edge.

$IS = \{1\}$

$= \{3, 4\}$

$= \{1, 4, 5\}$



Max size IS

Independent Set

Optimization Version:

Given a graph, find the largest independent set.

Decision Version:

Given a graph and a number k, does the graph contains an independent set of size k?

NP Hard NPC

Optimization vs. Decision

Optimization vs. Decision Problems

OPT → Decision

If one can solve an optimization problem (in polynomial time), then one can answer the decision version (in polynomial time)

Decision → OPT

Conversely, by doing binary search on the bound b, one can transform a polynomial time answer to a decision version into a polynomial time algorithm for the corresponding optimization problem

In that sense, these are essentially equivalent.

However, they belong to two different complexity classes.

Independent Set is NP Complete

Given a graph and a number k, does the graph contains an independent set of size k? (at ~~most~~ least k)

Is it in NP?

We need to show we can verify a solution in polynomial time.

Given a set of vertices, we can easily count them and then verify that any two of them are not joined by an edge.

Independent Set is NP Complete

Given a graph and a number k, does the graph contains an independent set of size at least k?

$3SAT \le_p IS$

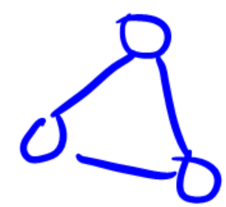
logic graph

Is it in NP-hard?

We need to pick Y such that Y $\le_p$ IndSet for $\forall$ Y $\in$ NP

$(x \vee y \vee z) \wedge (\bar{x} \vee y) \wedge \ldots$

Reduce from 3-SAT.



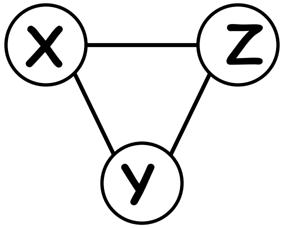
clause

3-SAT is SAT where each clause has at most 3 literals.

3SAT $\leq_p$ IndSet

We construct a graph G that will have an independent set of size k iff the 3-SAT instance with k clauses is satisfiable.

For each clause $(X \vee Y \vee Z)$ we will be using a special gadget:



Next, we need to connect gadgets.

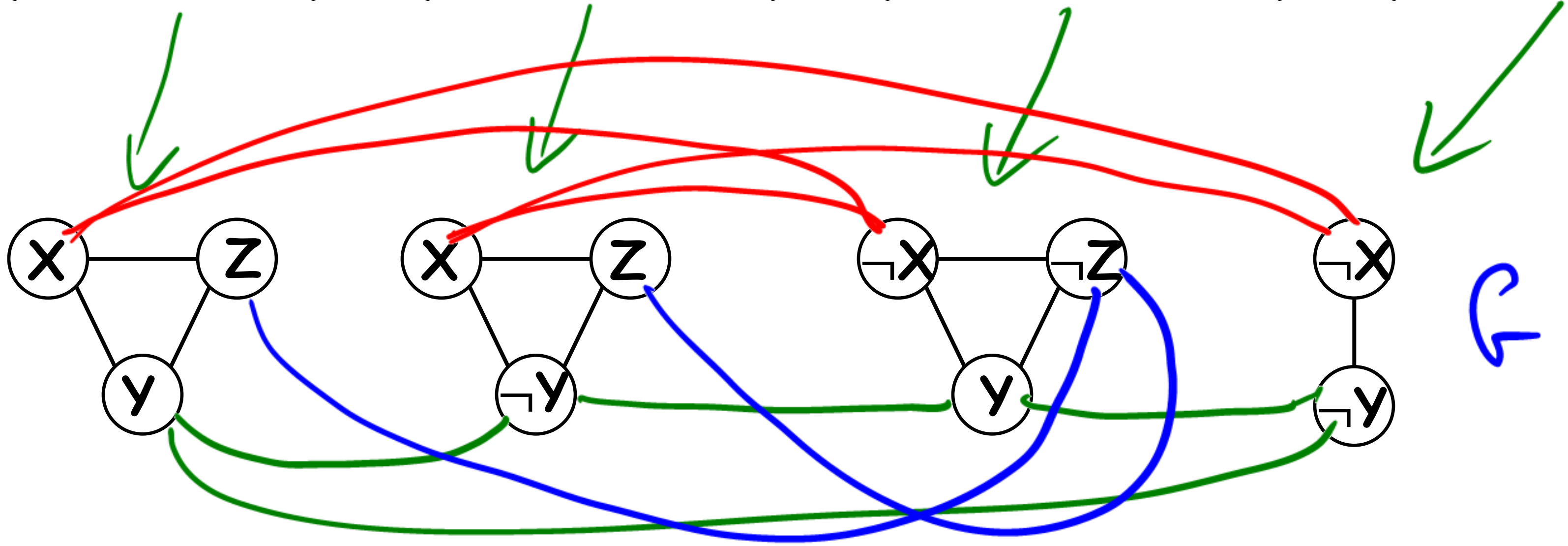
As an example, consider the following instance:

$$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y)$$

$\forall$ 3SAT $\leq_p$ IndSet

general specific

$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y)$



G

How do we connect gadgets?

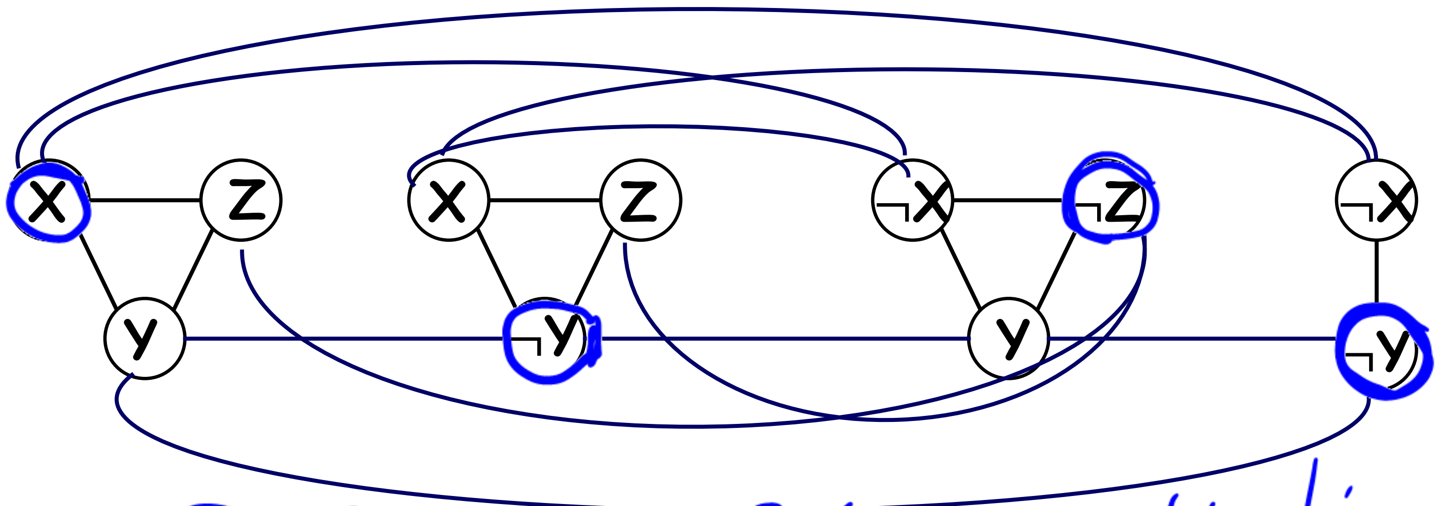
$\forall$ 3-SAT $\Longrightarrow$ a specific graph

Claim: 3-SAT with k clauses is satisfiable iff G has an IS of size k.

3SAT $\le_p$ IndSet

T T T T

$(X \lor Y \lor Z) \land (X \lor \neg Y \lor Z) \land (\neg X \lor Y \lor \neg Z) \land (\neg X \lor \neg Y)$



Proof. ⇒) Given 3-SAT satisfiable

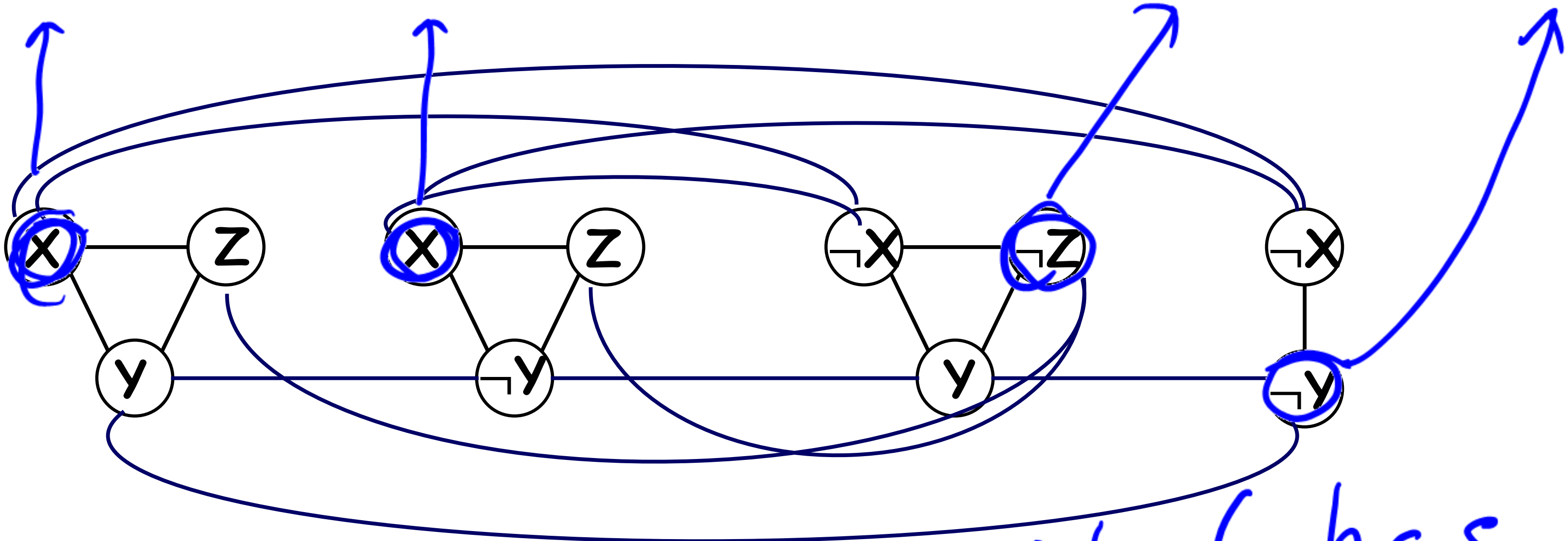
$x = \bar{y} = \bar{z} = True$ given

True literals in f make a IS

3SAT $\le_p$ IndSet

$$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y)$$

T T T T



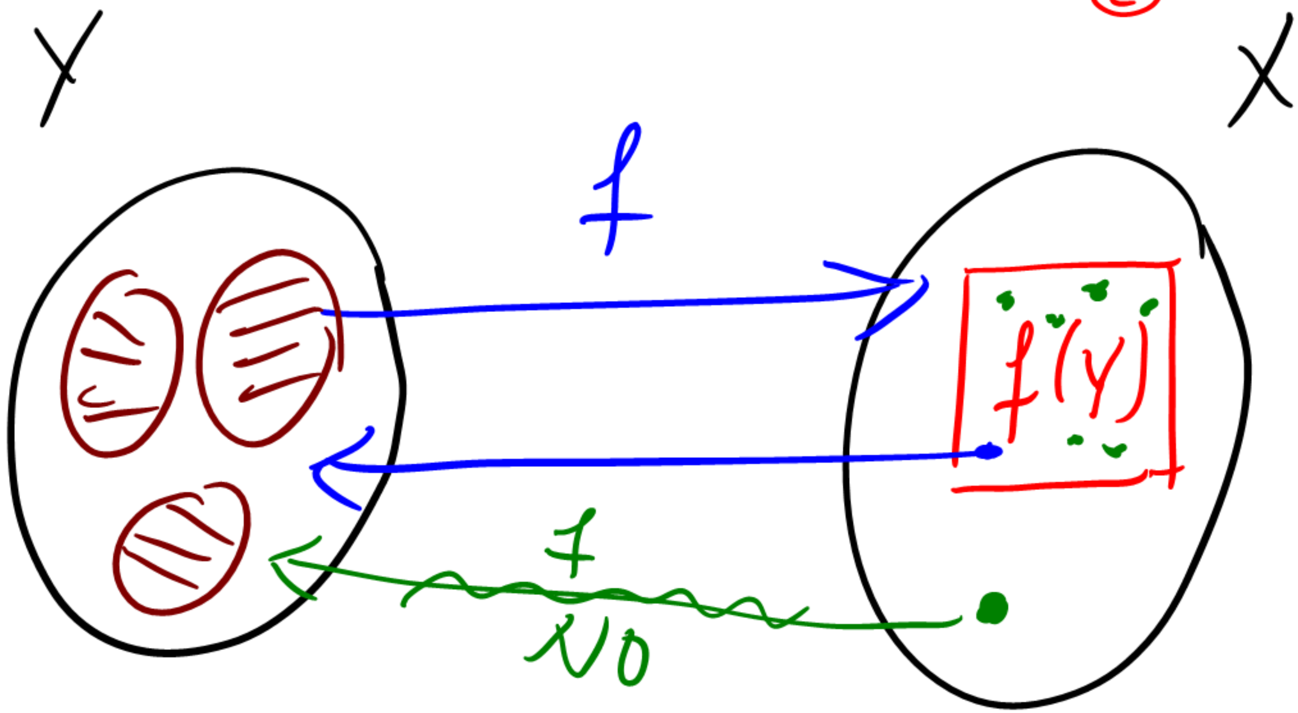
Proof. $(\Leftarrow)$ Triangular graph G has

$|IS| = K$

BREAK!

$Y \le_p X$

① direction from Y to X
② $f(Y) \neq X$



$\forall Y$ to X

$f(Y) \subset X$

$$(Y \le_p X) \overset{NO}{\Longrightarrow} (X \le_p Y)$$

Vertex Cover (VC)

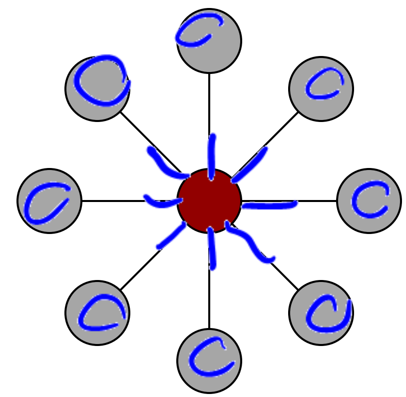
Given G=(V,E), find the smallest S⊂V s.t. every edge is incident to vertices in S.

OPT.

$IS \cup VC = V$

$VC = V - IS$

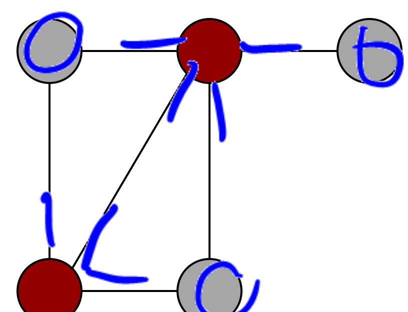
IS



Decision Problem.

Given G and K, does G contain a VC of size $|VC| = K$ (at most k)

IS



Vertex Cover

<u>Theorem</u>: *for a graph G=(V,E), S is an independent set if and only if V-S is a vertex cover*

Proof. $\Rightarrow$) S is an IS

1) $x \in IS$, then $y \notin IS$,
then $y \in VC$

2) $y \in IS \Rightarrow x \notin IS \Rightarrow x \in VC$

3) $x, y \notin IS \Rightarrow x, y \in VC$

Vertex Cover

<u>Theorem</u>: *for a graph G=(V,E), S is a independent set if and only if V-S is a vertex cover*

Proof. $\Leftarrow$) Given a VC

Pick $\forall x, y$ such that $x \notin VC$, $y \notin VC$

edge(x,y) - ? does not exist
proof by contradiction

It follows,

$x, y \in IS$

Vertex Cover in NP-Complete

<u>Claim</u>: *a graph G=(V,E) has an independent set of size at least k if and only if G has a vertex cover of size at most V-k.*

Ind. Set $\leq_p$ Vertex Cover

Vertex Cover $\leq_p$ Ind. Set

$Y \leq_p X \leq_p Z \; P$

$\Downarrow$

$Y \leq Z \; P$

In general

$Y \leq_p X \not\Rightarrow X \leq_p Y$

Discussion Problem 1

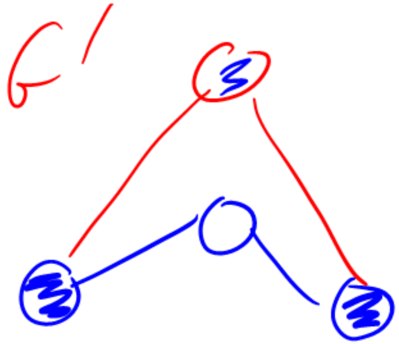
Show that vertex cover remains NP-Complete even if the instances are restricted to graphs with only <u>even</u> degree vertices. Let us call this problem VC-even.

Prove: VC $\leq_p$ VC-even

VC-even: Given G (with even degree vertices) and number k

① VC-even ∈ NP, easy

② VC-even ∈ NP-Hard



VC $\leq_p$ VC-even

in any graph

construct a graph with ALL even degrees.

The number of odd degree vertices is even!

Claim. $VC(G) = K$ iff $VC(G') = K+1$

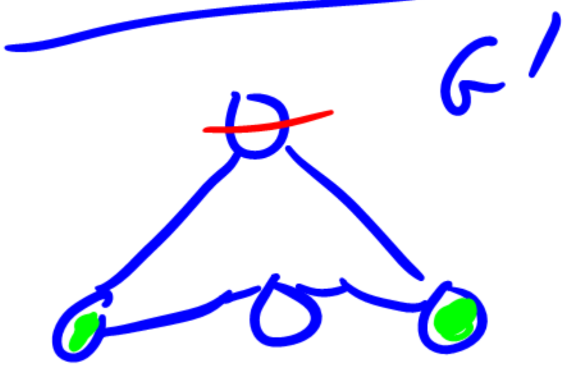
proof. by construction, $VC(G') = VC(G) +$ red vertex

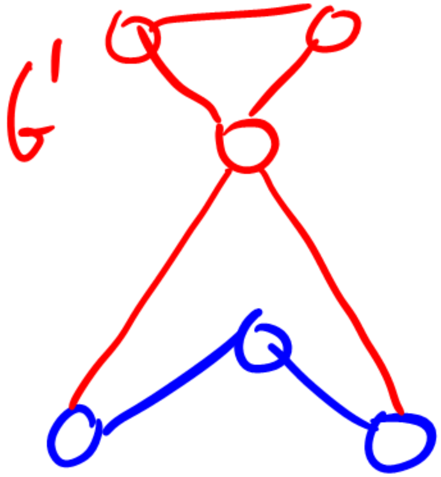
$\Rightarrow$ by construction,

$\Leftarrow$ Given $VC(G') = K+1$

$VC(G)$, remove what?

construction is wrong





at most k

VC $\leq_p$ VC-even

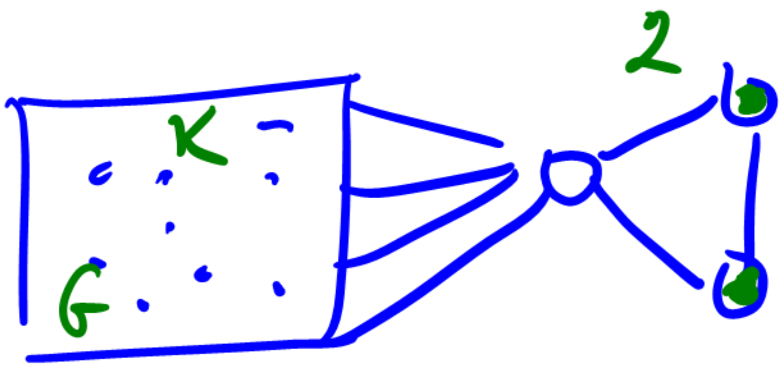
claim. $VC(G) = k$ iff $VC(G') = k+2$

Proof.

$\Rightarrow$ by construction $VC(G') = VC(G) + 2$ red vertices

$\Leftarrow$ Given $VC(G') = k+2$

G':



Remove that Δ

Hamiltonian Cycle Problem

A Hamiltonian cycle (HC) in a graph is a cycle that visits each vertex exactly once.

<u>Problem Statement:</u>

Given a directed or undirected graph G = (V,E). Find if the graph contains a Hamiltonian cycle.

We can prove it that HC problem is NP-complete by reduction from SAT, but we won't.

Discussion Problem 2

Assuming that finding a Hamiltonian Cycle (HC) in a graph is NP-complete, prove that finding a Hamiltonian Path is also NP-complete. HP is a path that visits each vertex exactly once and isn't required to return to its starting point.

1 $HP \in NP$

$HC \leq_p HP$

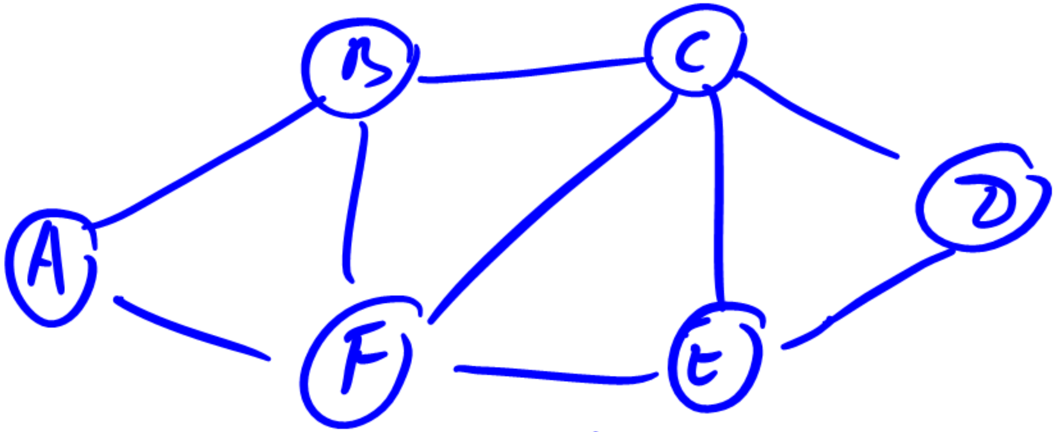
2 $HP \in NP$-hard

a) construct a polynomial mapping

b) make a claim

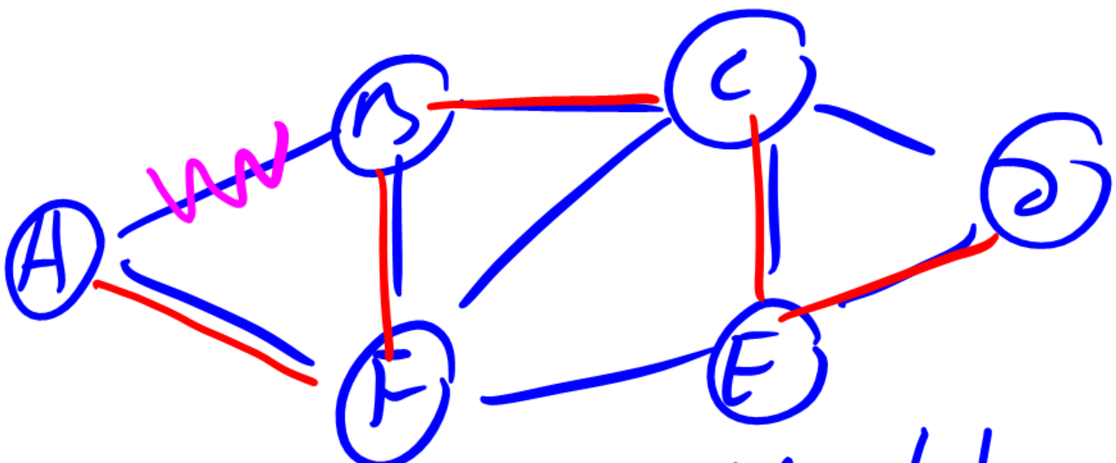
c) prove the claim in both directions.

G HC: AFEDCBA



Construct G'

G'



claim. G has a HC iff G' has a HP

Proof.

$\Rightarrow$ Given $HC(f)$

$HP(f')$ by construction

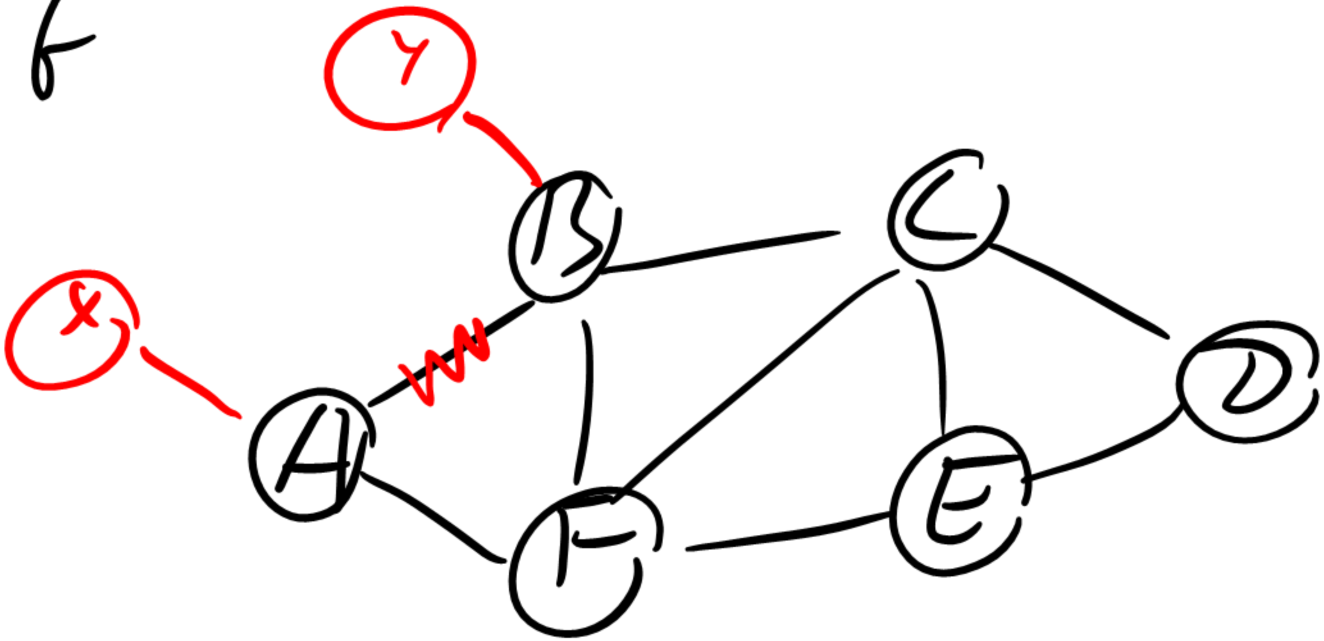
$\Leftarrow$ Given $HP(f')$

$HP(f') = AFBCED$

How do you get a cycle ?

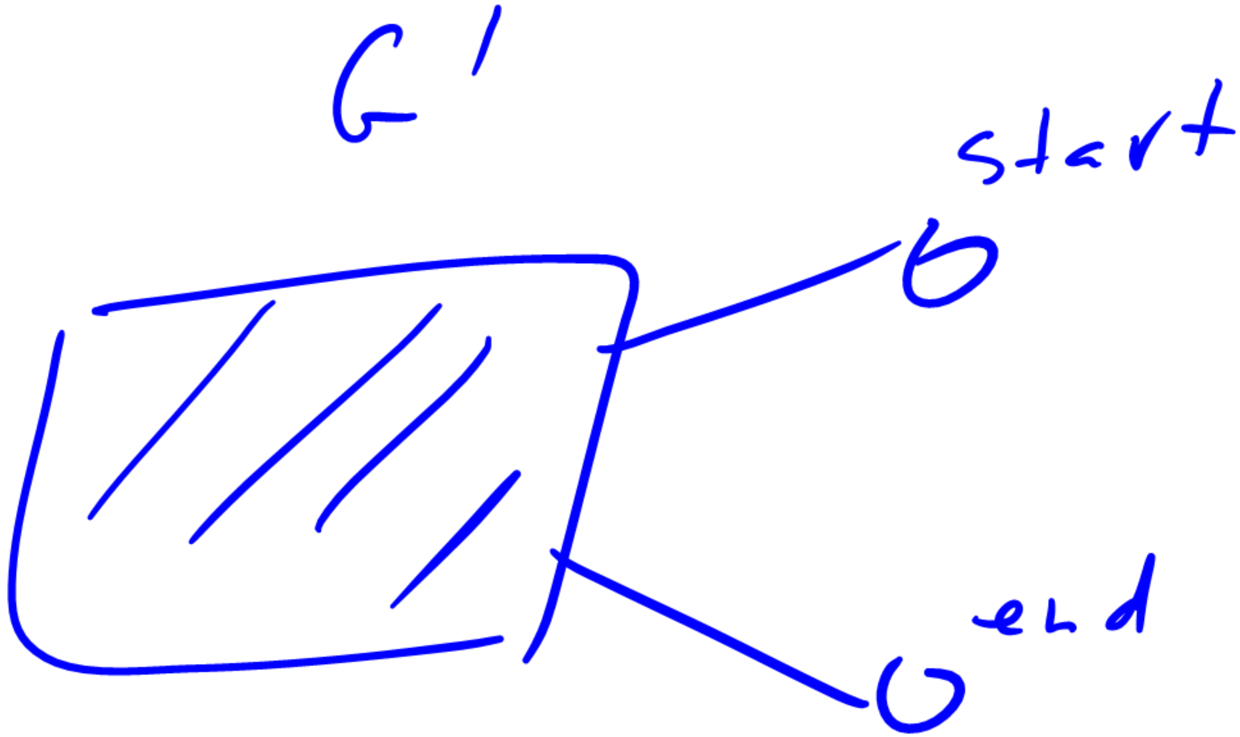
Wrong construction.

G'



G': remove any edge on a cycle
add two vertices, and connect
them to two eetices of that
edge

HP(G'): X-A-....-B-Y


G'
start
end

Discussion Problem 3

You are given an undirected graph $G = (V, E)$ and for each vertex v, you are given a number $p(v)$ that denotes the number of pebbles placed on v. We will now play a game where the following move is the only move allowed. You can pick a vertex u that contains at least two pebbles, and remove two pebbles from u and add one pebble to an adjacent vertex. The objective of the game is to perform a sequence of moves such that we are left with exactly one pebble in the whole graph. Show that the problem of deciding if we can reach the objective is NP-complete. Reduce from the Hamiltonian Path problem.

Discussion Problem 4

Given SAT in Conjunctive Normal Form (CNF)

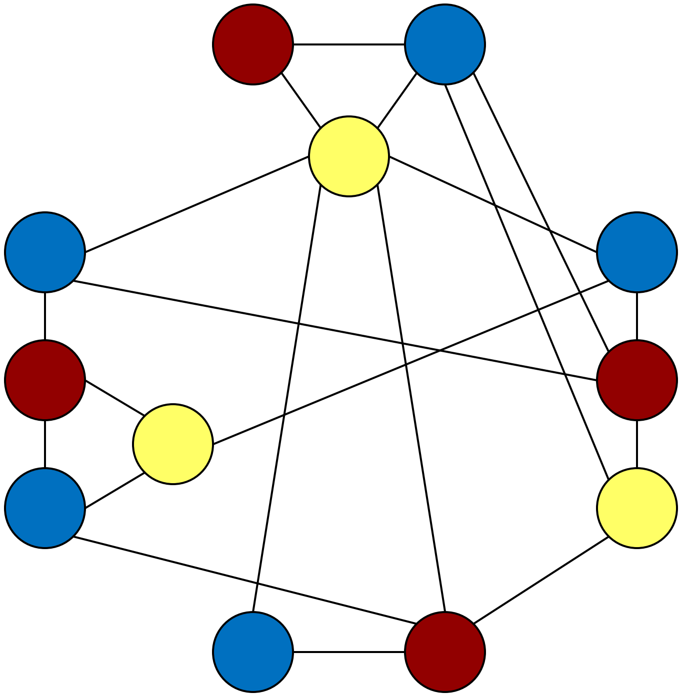
$$(X_1 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_4 \vee X_5) \wedge \ldots$$

with any number of clauses and any number of literals in each clause. Prove that SAT is polynomial time reducible to 3SAT.

Graph Coloring

Given a graph, can you color the nodes with $\leq k$ colors such that the endpoints of every edge are colored differently?

Theorem. (k>2)
k-Coloring is NP-complete.



Graph Coloring: $k = 2$

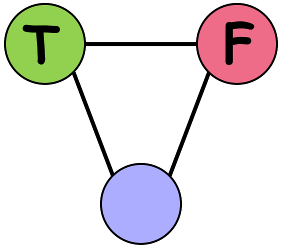
How can we test if a graph has a 2-coloring?

3-SAT $\leq_p$ 3-colorable

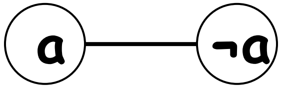
We construct a graph G that will be 3-colorable iff the 3-SAT instance is satisfiable.

Graph G consists of the following gadgets.

A truth gadget:



A gadget for each variable:



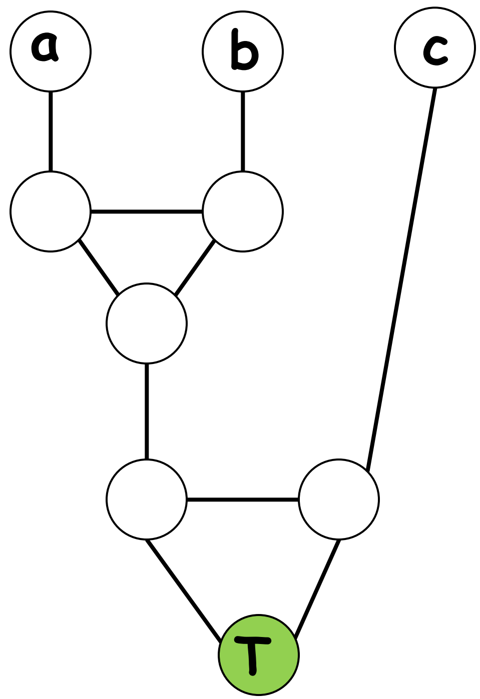
3-SAT $\leq_p$ 3-colorable

Combining those gadgets together (for three literals)

3-SAT $\leq_p$ 3-colorable

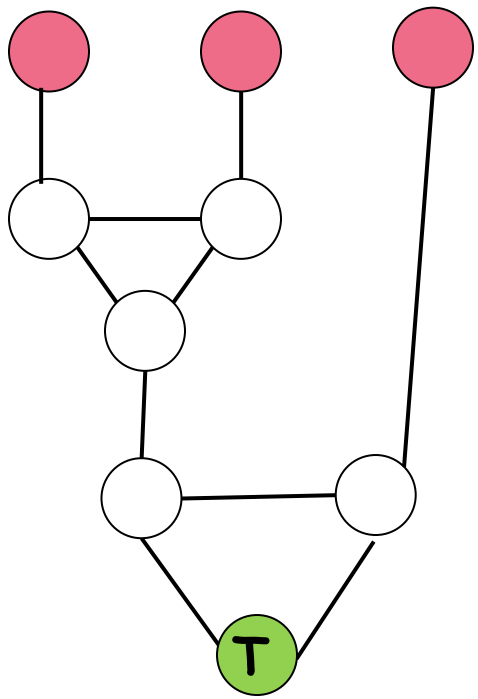
A special gadget for each clause

This gadget connects a truth gadget with variable gadgets.



3-SAT $\leq_p$ 3-colorable

Suppose all a, b and c are all False (red).

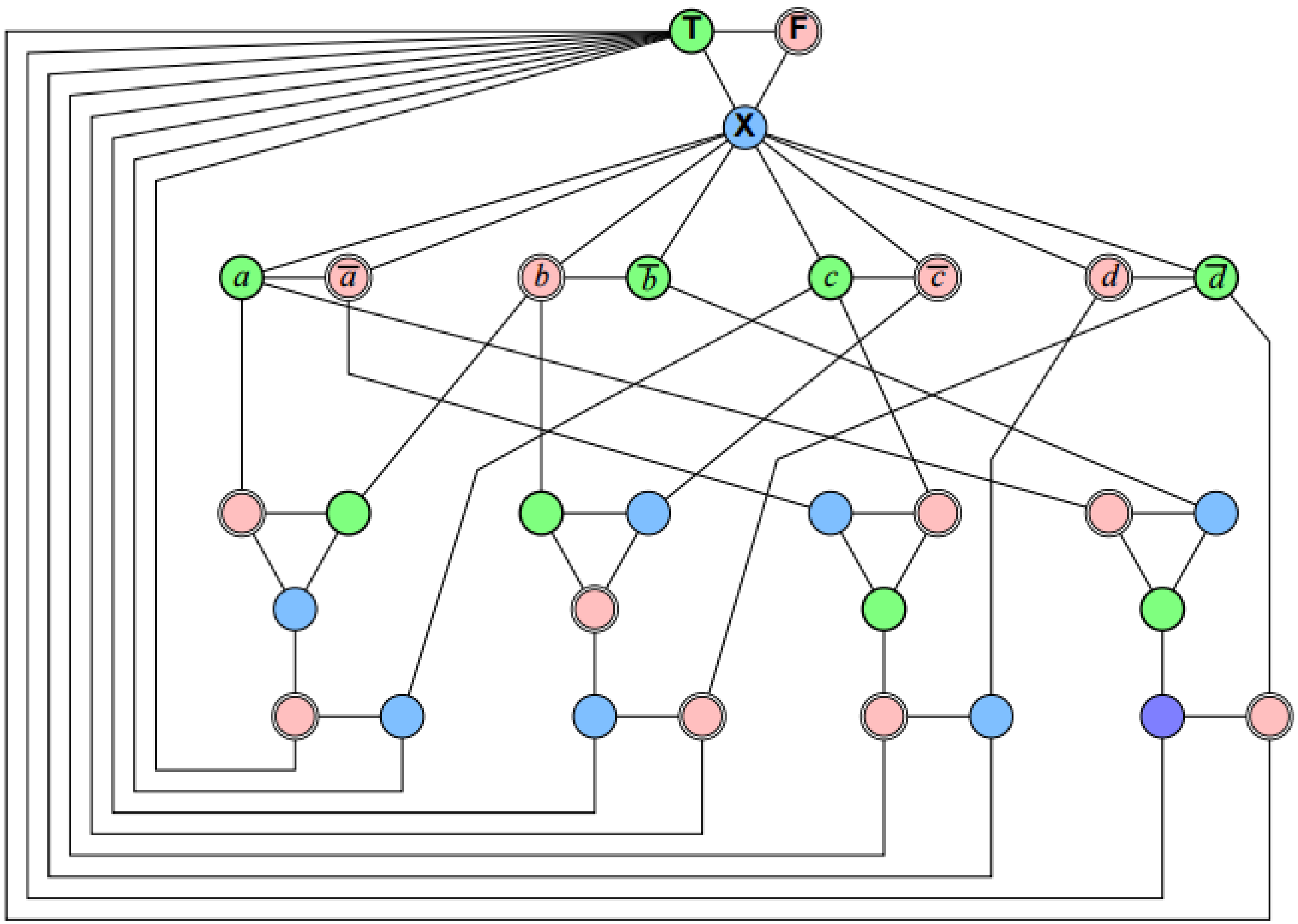


3-SAT $\leq_p$ 3-colorable

We have showed that if all the variables in a clause are false, the gadget cannot be 3-colored.

Example: $a \vee \neg b \vee c$

Example with four clauses



a=c=T

b=d=F

A 3-colorable graph derived from a satisfiable 3CNF formula.

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$

3-SAT $\leq_p$ 3-colorable

Claim: *3-SAT instance is satisfiable if and only if G is 3-colorable.*

Proof: $\Rightarrow$)

3-SAT $\leq_p$ 3-colorable

Claim: *3-SAT instance is satisfiable if and only if G is 3-colorable.*

Proof: $\Leftarrow$)

Sudoku: $n^2 \times n^2$

NP-?

NP-hard?

2			3		8		5	
		3		4	5	9	8	
		8			9	7	3	4
6		7		9				
9	8						1	7
				5		6		9
3	1	9	7			2		
	4	6	5	2		8		
	2		9		3			1

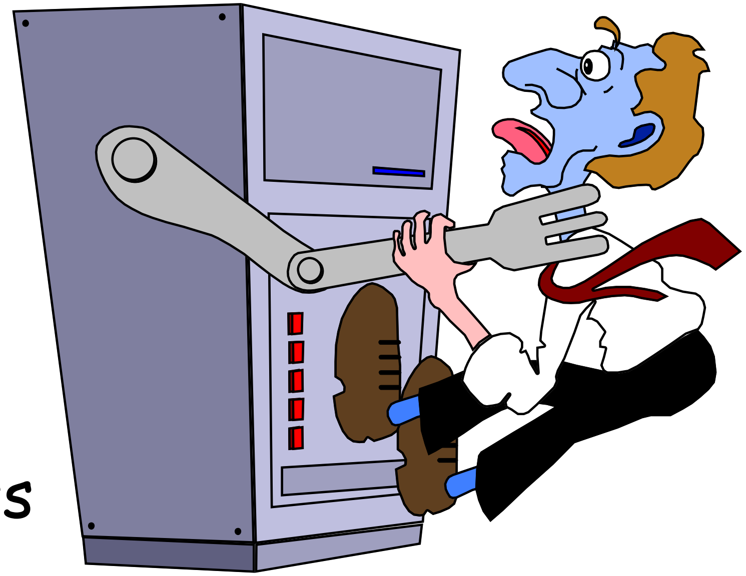
Sudoku Graph

2			3		8		5	
		3		4	5	9	8	
		8			9	7	3	4
6		7		9				
9	8						1	7
				5		6		9
3	1	9	7			2		
	4	6	5	2		8		
	2		9		3			1

Sudoku

Constructing a Sudoku graph we have proved:

Don't be afraid of NP-hard problems.



Many reasonable instances (of practical interest) of problems in class NP can be solved!

The largest solved TSP an 85,900-vertex route calculated in 2006. The graph corresponds to the design of a customized computer chip created at Bell Laboratories, and the solution exhibits the shortest path for a laser to follow as it sculpts the chip.