Lecture 23

Polynomial-Time Reductions (cont.)

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

Why the notation?

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

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)

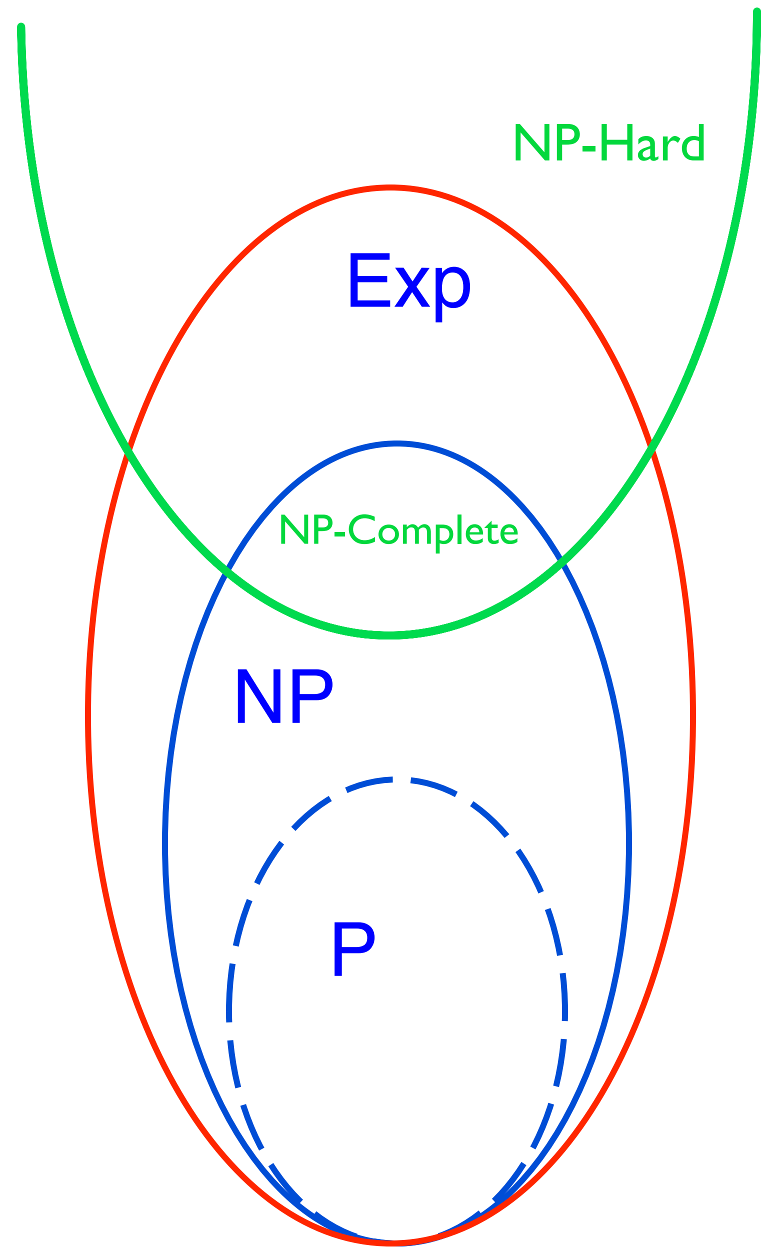
NP-Completeness

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.



“NP-completeness”

Cool concept, but are there
any such problems?

Yes!

Cook’s theorem: SAT is NP-complete

Why is SAT NP-complete?

Cook's proof is somewhat involved; details later.
But its essence is not so hard to grasp:

Generic "NP" problem:
is there a poly size "solution,"
verifiable by computer in poly time

"SAT":
is there a (poly size) assignment
satisfying the formula

Encode "solution" using Boolean variables. SAT mimics "is there a solution" via "is there an assignment". Digital computers just do Boolean logic, and "SAT" can mimic that, too, hence can verify that the assignment *actually* encodes a solution.

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.

Alt way to prove NP-completeness

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.

Ex: VertexCover is NP-complete

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

3-SAT $\leq_p$ VertexCover

VertexCover is in NP (we showed this earlier)

Therefore VertexCover is also NP-complete

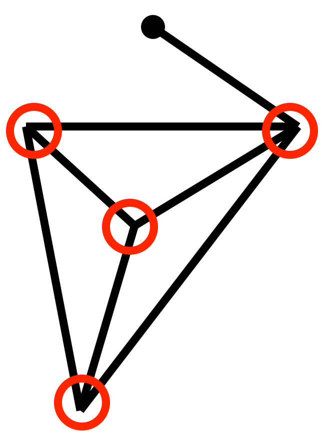
So, poly-time algorithm for VertexCover would give poly-time algs for everything in NP

NP-complete problem: Clique

Input: Undirected graph G = (V, E), integer k.

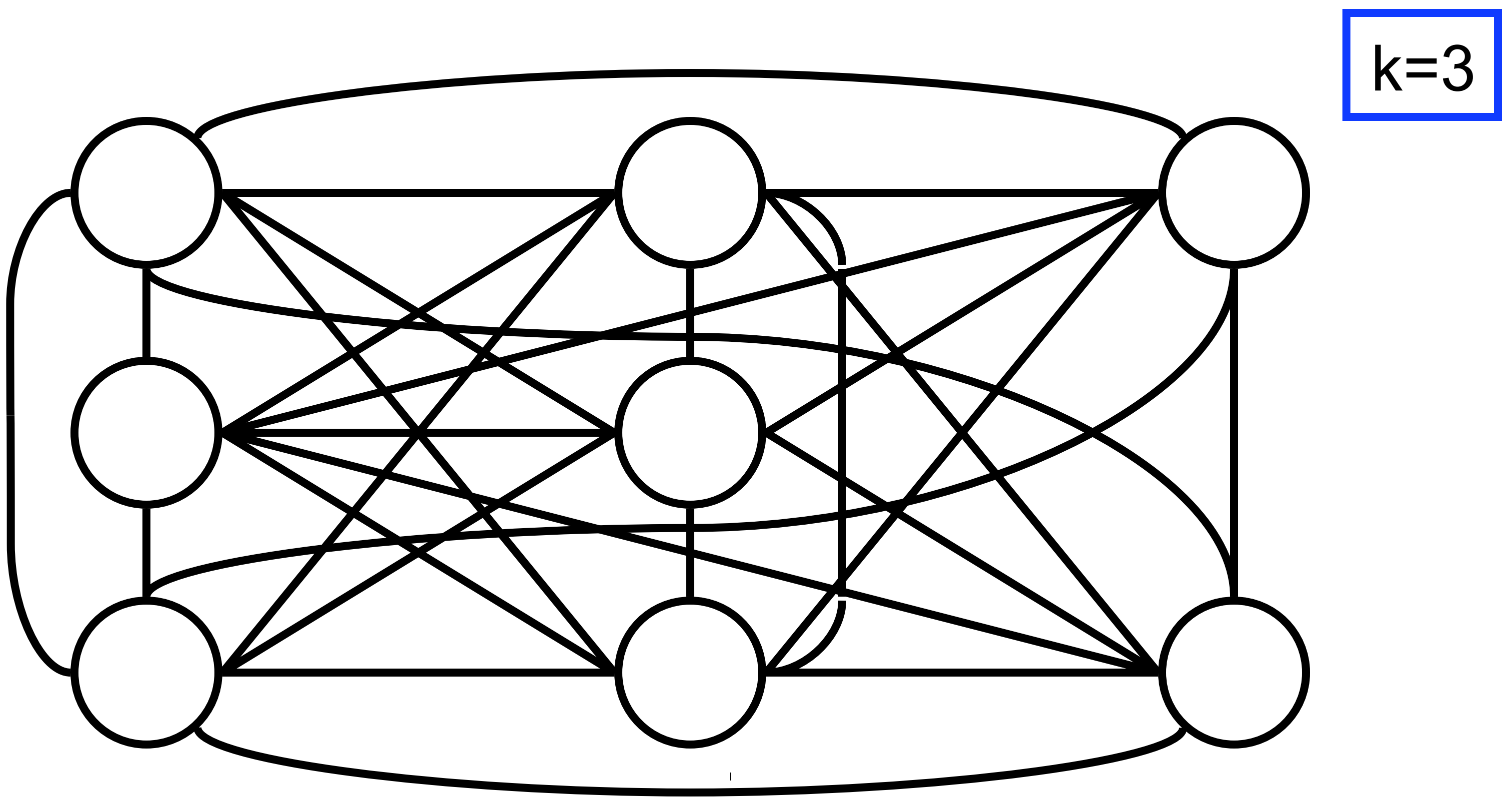
Output: True iff there is a subset C of V of size ≥ k such that all vertices in C are connected to all other vertices in C.

Example: Clique of size ≥ 4

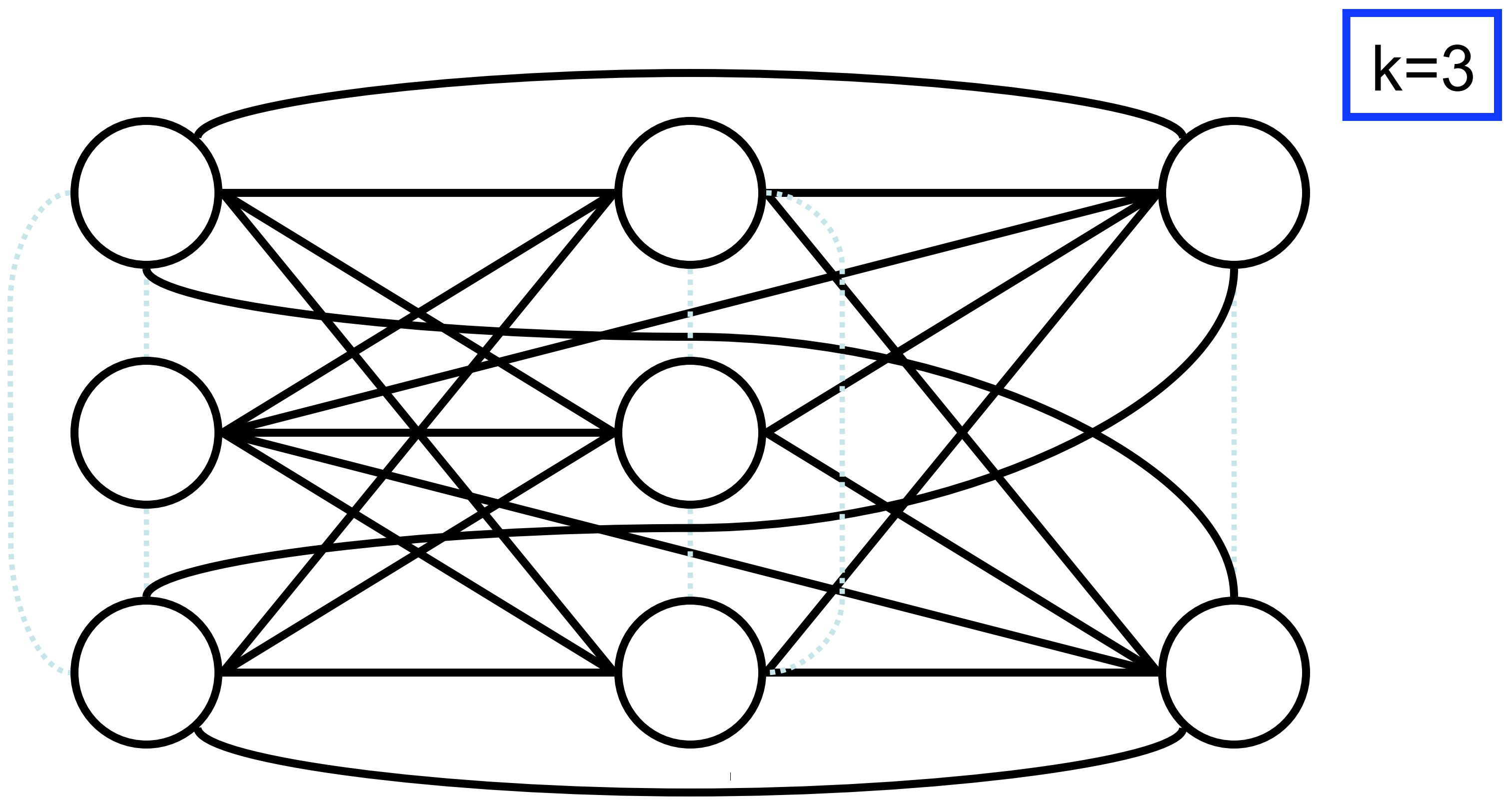


In NP? Exercise

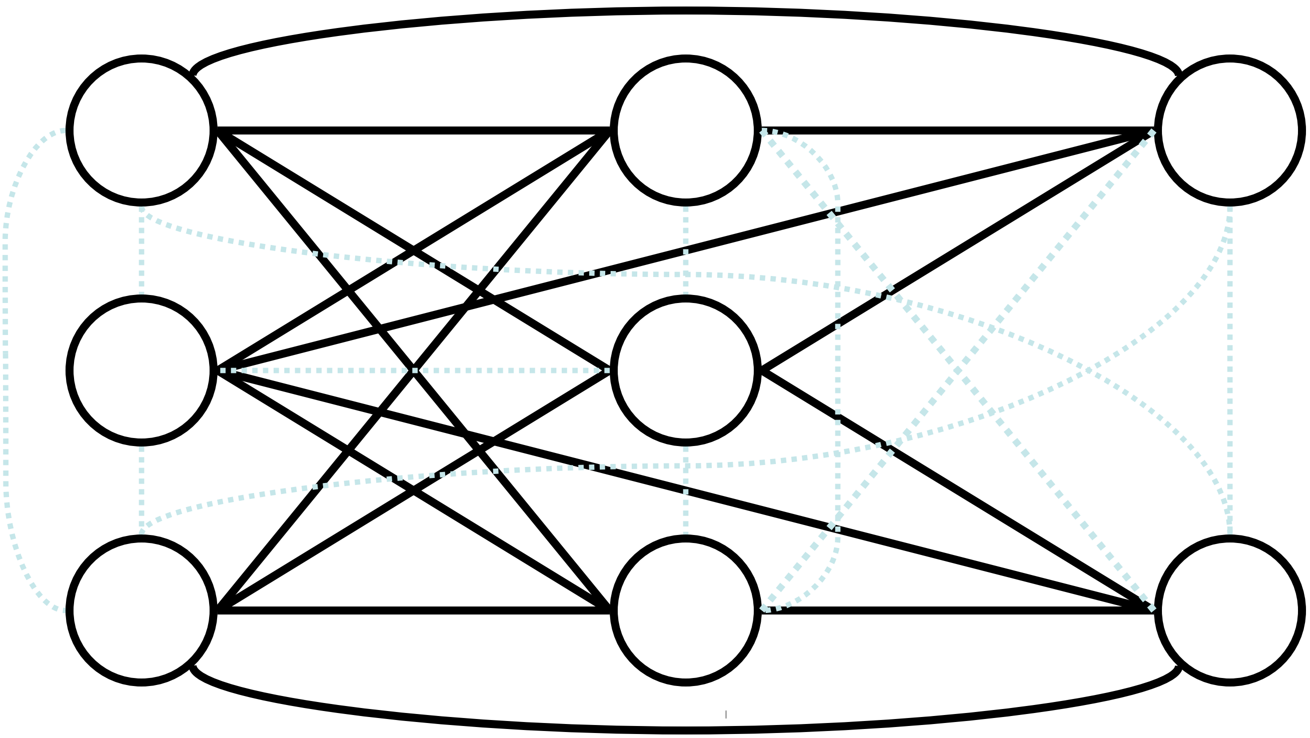
3SAT $\leq_p$ Clique



3SAT $\leq_p$ Clique

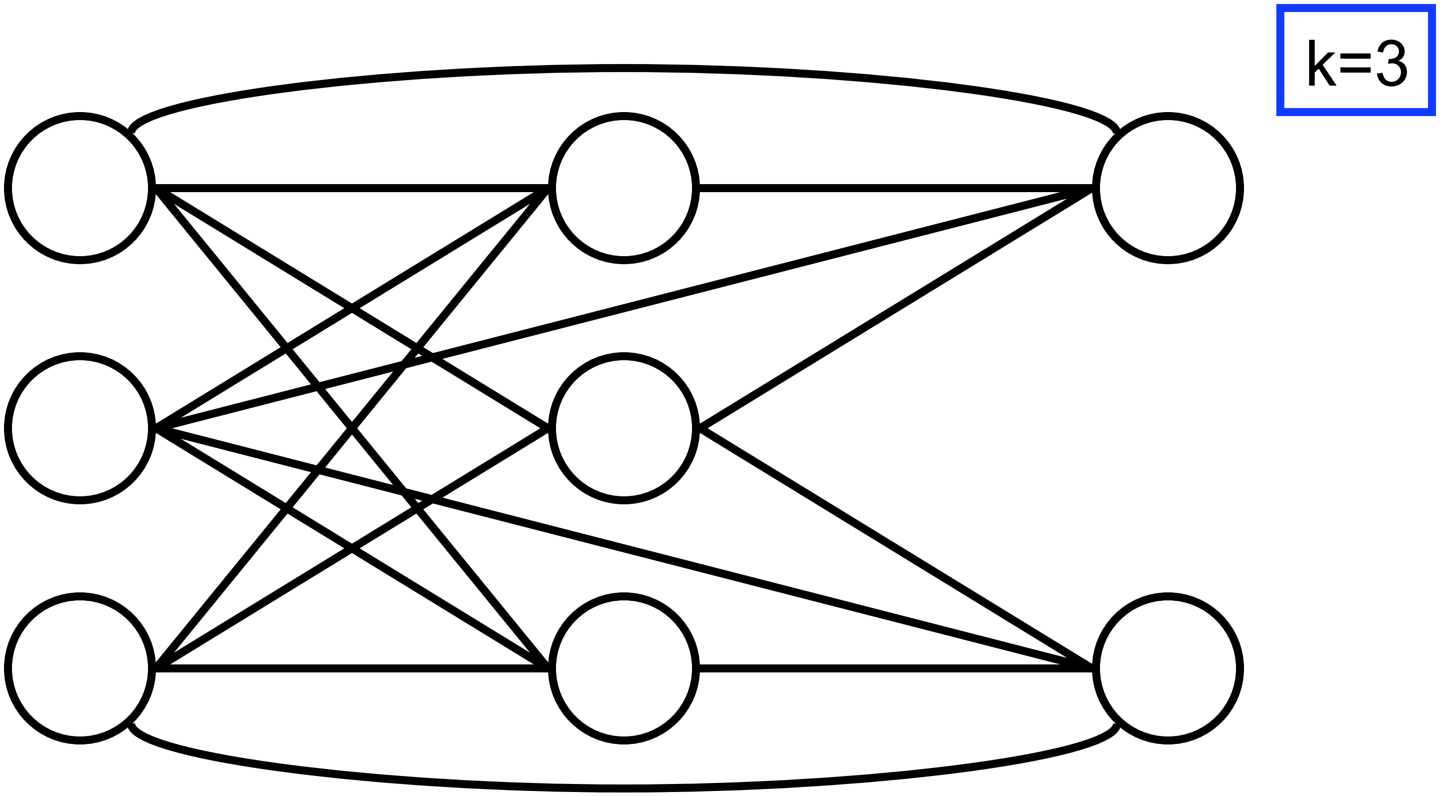


3SAT $\leq_p$ Clique



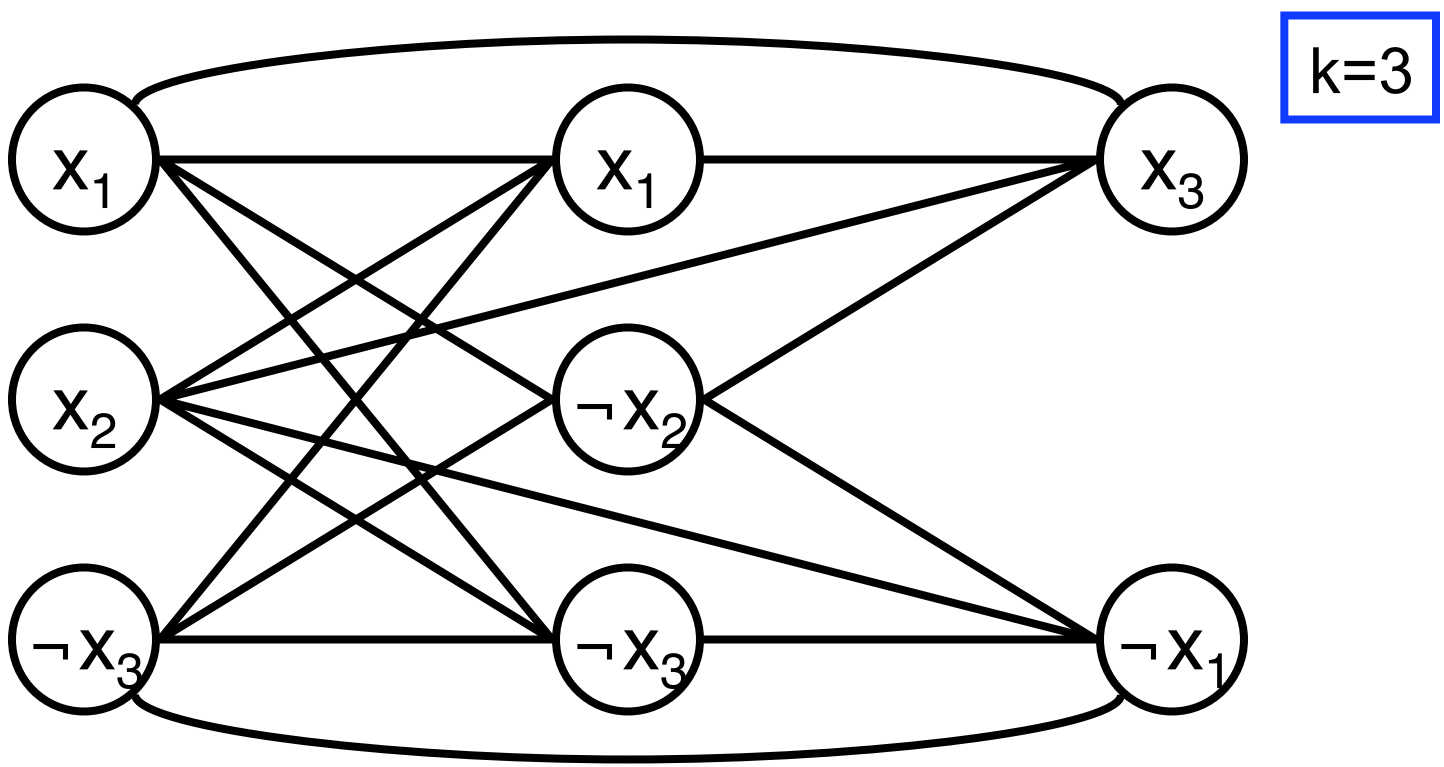
k=3

3SAT $\leq_p$ Clique



3SAT $\leq_p$ Clique

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



3SAT $\leq_p$ Clique

f (

3-SAT Instance:

– Variables: $x_1, x_2, \ldots$

– Literals: $y_{i,j}$, $1 \leq i \leq q$, $1 \leq j \leq 3$

– Clauses: $c_i = y_{i1} \vee y_{i2} \vee y_{i3}$, $1 \leq I \leq q$

– Formula: $c = c_1 \wedge c_2 \wedge \ldots \wedge c_q$

) =

Clique Instance:

– $K = q$

– $G = (V, E)$

– $V = \{ [i,j] \mid 1 \leq i \leq q,\ 1 \leq j \leq 3 \}$

– $E = \{ ([i,j], [k,l]) \mid i \neq k \text{ and } y_{ij} \neq \neg y_{kl} \}$

Correctness of “3-SAT $\leq_p$ Clique”

Summary of reduction function f:
Given formula, make graph G with column of nodes per clause, one node per literal. Connect each to all nodes in other columns, except complementary literals (x, ¬x). Output graph G plus integer k = number of clauses. *Note: f does not know whether formula is satisfiable or not; does not know if G has k-clique; does not try to find satisfying assignment or clique.*

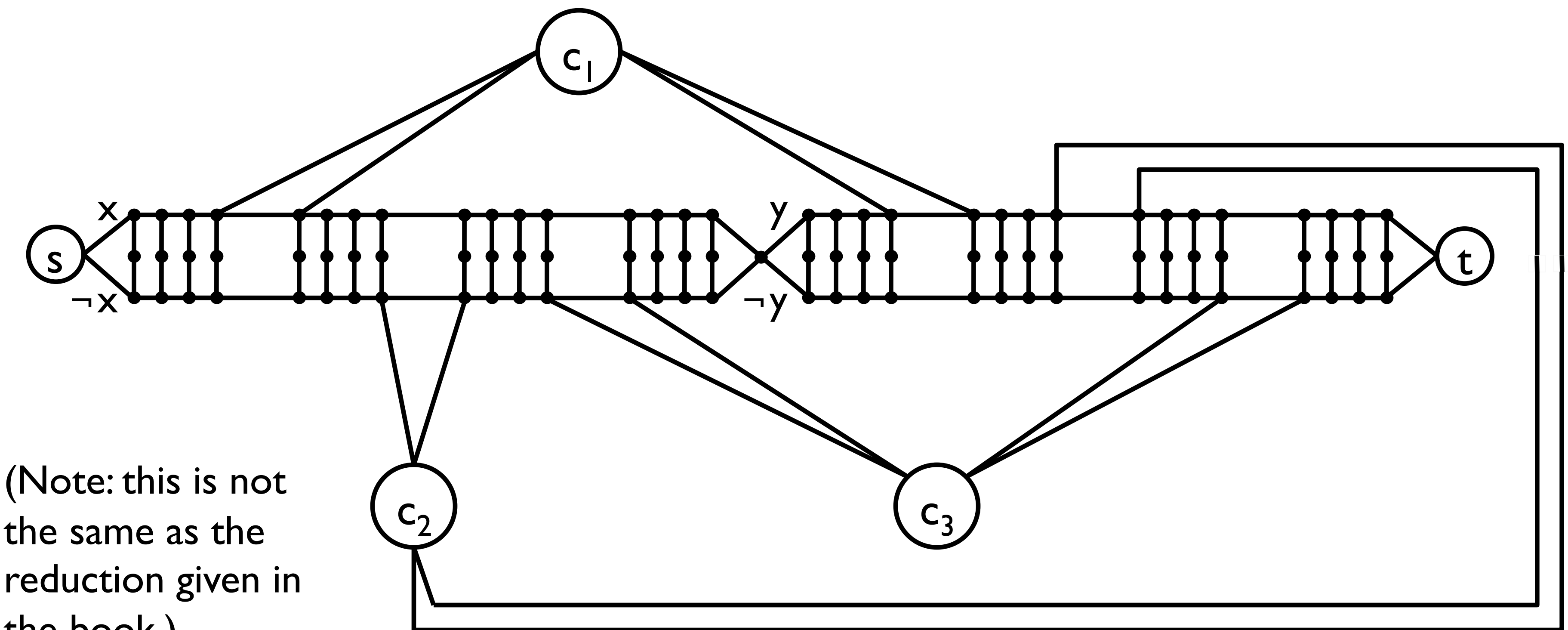
Correctness:

Show f poly time computable: A key point is that graph size is polynomial in formula size; mapping basically straightforward.

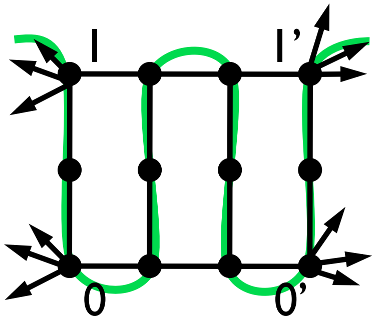
Show c in 3-SAT iff f(c)=(G,k) in Clique:
(⇒) Given an assignment satisfying c, pick one true literal per clause. Show corresponding nodes in G are k-clique.
(⇐) Given a k-clique in G, clique labels define a truth assignment; show it satisfies c. Note: literals in a clique are a valid truth assignment [no “(x, ¬x)” edges] & k nodes must be 1 per column, [no edges within columns].

3-SAT $\leq_P$ UndirectedHamPath

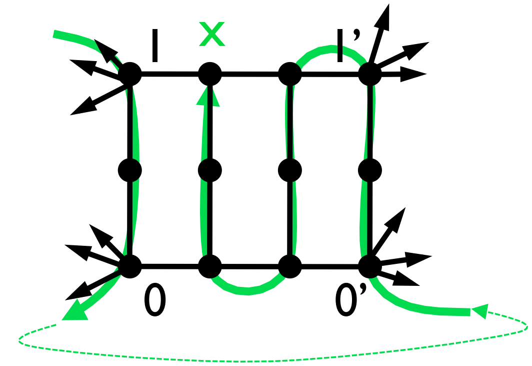
Example: $(x \vee y) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$



(Note: this is not the same as the reduction given in the book.)



Ham Path Gadget



Many copies of this 12-node gadget, each with one or more edges connecting each of the 4 corners to other nodes or gadgets (but no other edges to the 8 “internal” nodes).

Claim: There are only 2 Ham paths – one entering at 1, exiting at 1’ (as shown); the other (by symmetry) 0→0’

Pf: Note *: at 1st visit to any column, must next go to *middle* node in column, else it will subsequently become an untraversable “dead end.”
WLOG, suppose enter at 1. By *, must then go down to 0. 2 cases:

Case a: (top left) If next move is to right, then * forces path up, left is blocked, so right again, etc; out at 1’.

Case b: (top rt) if exit at 0, then path must eventually reenter at 0’ or 1’. * forces next move to be up/down to the other of 0’/1’. Must then go left to reach the 2 middle columns, but there’s *no exit* from them. So case b is impossible.