Satisfiability

A formula F is *satisfiable* if there exists an assignment to its Boolean variables such that F becomes true.

E.g., $x_1 \vee x_2$ is satisfiable;

$(x_1 \vee x_2) \wedge (\sim x_2)$ is satisfiable;

$x_2 \wedge (\sim x_2)$ is not satisfiable.

The Satisfiability problem (SAT): Given a formula F, determine whether F is satisfiable.

A restricted version: A formula F is in *conjunctive normal form* (CNF) if F is in the form of

$(x_1 \vee x_2 \vee \sim x_4) \wedge (\sim x_2 \vee x_5) \wedge (x_2 \vee x_8 \vee x_9) \wedge \ldots$

I.e., F comprises several **clauses** connected with $\wedge$s, where a clause comprises **literals** (i.e., variables or their negations) connected with $\vee$s.

3CNF-SAT (3SAT)

The CNF-SAT problem: Given a CNF-formula F, determine whether F is satisfiable.

3CNF-formula: is a CNF formula in which every clause contains exactly 3 literals.

The 3CNF-SAT problem: Given a 3CNF-formula F, determine whether F is satisfiable.

3CNF-SAT (3SAT)

Languages:

- SAT = { F | F is a satisfiable formula}.
- CNFSAT = { F | F is a satisfiable CNF-formula}.
- 3CNFSAT = { F | F is a satisfiable 3CNF-formula}.

Fact. (1) SAT, CNFSAT, 3CNFSAT $\in$ NP;

(2) 3CNFSAT $\leq_p$ CNFSAT $\leq_p$ SAT

The frist NP-complete problem

The Satisfiability problem (SAT): Given a formula F, determine whether F is satisfiable.

E.g., $x_1 \vee x_2$ is satisfiable;

$(x_1 \vee x_2) \wedge (\sim x_2)$ is satisfiable;

$x_2 \wedge (\sim x_2)$ is not satisfiable.

Theorem: SAT is NP-complete.

Last lecture: For any problem X in NP, $X \leq_p$ SAT.

More NP-Complete problems

Lemma. Let A, B be two problems in NP. Suppose that $A \leq_p B$ and A is NP-complete. Then B is NP-complete.

Example 1: SAT $\leq_p$ CNFSAT.

Conclusion: CNFSAT is NP-complete.

Lemma 1. Suppose $A \leq_p B$ and A is NP-complete.
Then B is NP-complete.

Proof.

B is in NP.

A is NP-complete $\Rightarrow$ For any problem L in NP, $L \leq_p A$.

Since $L \leq_p A$ and $A \leq_p B$, we conclude that $L \leq_p B$.

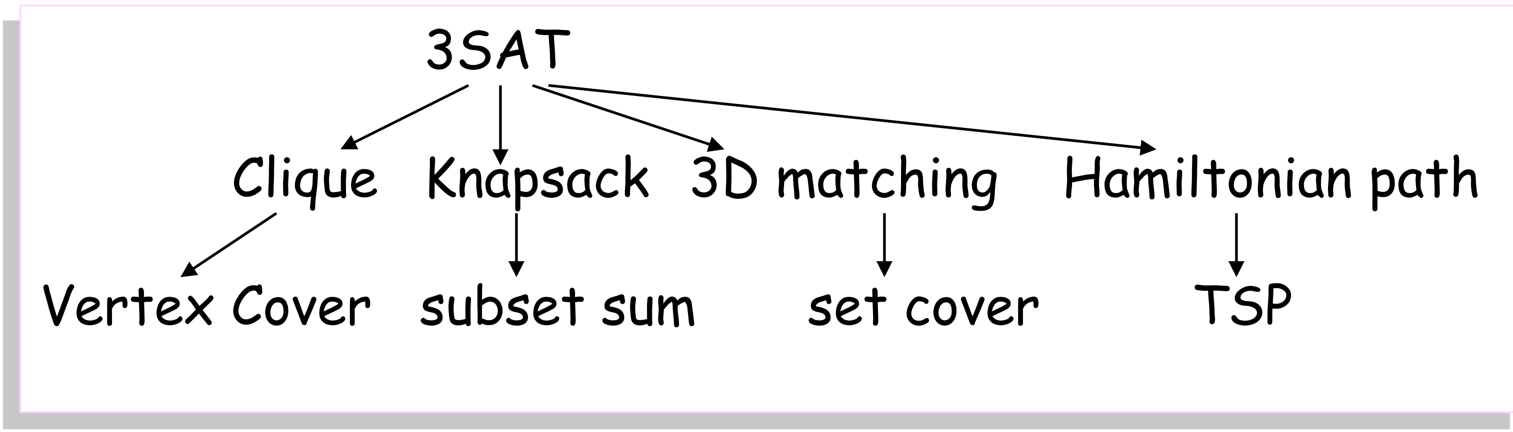
($\leq_p$ is transitive.)

Therefore, B is NP-complete.

More NP complete problems

SAT $\leq_p$ CNFSAT

CNFSAT $\leq_p$ 3CNFSAT (more commonly known as 3SAT)



Several reductions will be sketched in today's lecture.
You should fill in the details and proof of correctness.

SAT $\leq_p$ CNFSAT

SAT: to determine whether a formula F is satisfiable.

Example of a CNF-formula : $\sim(((x_1 \vee x_2) \wedge \sim x_4)) \wedge (\sim x_2 \vee x_5)) \wedge (x_2 \vee (x_8 \wedge x_9))$ ...

CNFSAT: to determine whether a CNF-formula F is satisfiable.

Example of a CNF-formula : $\underbrace{(x_1 \vee x_2 \vee \sim x_4)}_{\text{a clause}} \wedge (\sim x_2 \vee x_5) \wedge (x_2 \vee x_8 \vee x_9) \wedge \ldots$

A literal is a variable or its neagation

What to do ?

Show that any given formula F can be transformed in polynomial time to a CNF-formula F' such that

F is satisfiable if and only if F' is satisfiable.

SAT $\leq_p$ CNFSAT

Step 1: Transform F to an equivalent formula F_1 such that all ~ (negation) operators are applied to <u>variables</u> only.

Tools: $\sim(E \wedge E') \equiv \sim E \vee \sim E'$;

$\sim(E \vee E') \equiv \sim E \wedge \sim E'$;

$\sim\sim E \equiv E$

E.g., $\sim(\sim(x1 \wedge x2) \vee x3)$ is logically equivalent to

$(x1 \wedge x2) \wedge \sim x3$

Step 2: Transform F_1 to a CNF-formula F_2 recursively.

Note that F_1 can always be decomposed into the following three possible forms:

- **E** $\vee$ **E'** , where E and E' are formulas (but not necessarily CNF-formulas)
- **E** $\wedge$ **E'**
- a literal (i.e., x_i , $\sim x_i$)

NB. F_1 can't be of the form ~E, where E is a formula involving operators, i.e., more complicated than a variable.

Step 2: Recursive transformation of F_1

If F_1 is in the form of $E \vee E'$ (where E and E' are formulas)

(a) Transform E to a CNF formula $C_1 \wedge C_2 \wedge \ldots \wedge C_l$

(b) Transform E' to a CNF formula $D_1 \wedge D_2 \wedge \ldots \wedge D_k$

$$F_1 \equiv [C_1 \wedge C_2 \wedge \ldots \wedge C_l] \vee [D_1 \wedge D_2 \wedge \ldots \wedge D_k].$$

(c) Let F_2 be $[y \vee C_1] \wedge [y \vee C_2] \wedge \ldots \wedge [y \vee C_l] \wedge$
$[\sim y \vee D_1] \wedge [\sim y \vee D_2] \wedge \ldots \wedge [\sim y \vee D_k]$

where y is a new variable.

Claim. F_1 is satisfiable if and only if F_2 is satisfiable.

Proof ($\Rightarrow$)

Claim. F_1 is satisfiable $\Rightarrow$ F_2 is satisfiable.

Suppose $[C_1 \wedge C_2 \wedge \ldots \wedge C_l] \vee [D_1 \wedge D_2 \wedge \ldots \wedge D_k]$ is satisfiable.

There exists an assignment to the variables such that all C_i's are true **or** all D_i's are true.

If all C_i's are true, then further assigning y to false.

- $[\sim y \vee D_1] \wedge [\sim y \vee D_2] \wedge \ldots \wedge [\sim y \vee D_k]$: true.
- $F_2 = [y \vee C_1] \wedge \ldots \wedge [y \vee C_l] \wedge [\sim y \vee D_1] \wedge \wedge \ldots \wedge [\sim y \vee D_k]$: true.

If all D_i's are true, then assigning y to true.

- $F_2 = [y \vee C_1] \wedge \ldots \wedge [y \vee C_l] \wedge [\sim y \vee D_1] \wedge \ldots \wedge [\sim y \vee D_k]$: true.

Proof ($\Leftarrow$)

Suppose $F_2 = [y \vee C_1] \wedge \dots \wedge [y \vee C_l] \wedge [\sim y \vee D_1] \wedge \dots \wedge [\sim y \vee D_k]$ is satisfiable.

Then there is an assignment A to the variables such that F_2 becomes true. Note that y is assigned either true or false.

- If y is assigned true, then the assignment A makes all D_i's true.
- If y is assigned false, then the assignment A makes all C_i's true.

In either case, A makes $F_1 = [C_1 \wedge C_2 \wedge \dots \wedge C_l] \vee [D_1 \wedge D_2 \wedge \dots \wedge D_k]$ true.

If F_1 is in the form of $E \wedge E'$

(a) Transform E to a CNF formula $C_1 \wedge C_2 \wedge \ldots \wedge C_l$

(b) Transform E' to a CNF formula $D_1 \wedge D_2 \wedge \ldots \wedge D_k$

$$F_1 \equiv [C_1 \wedge C_2 \wedge \ldots \wedge C_l] \wedge [D_1 \wedge D_2 \wedge \ldots \wedge D_k]$$

(c) Let $F_2 = C_1 \wedge C_2 \wedge \ldots \wedge C_l \wedge D_1 \wedge D_2 \wedge \ldots \wedge D_k$

If F_1 is in the form of a literal, i.e., x , $\sim x$

$F_2 = F_1$ is already in CNF.

Fact: F is satisfiable $\Leftrightarrow$ F_1 is satisfiable $\Leftrightarrow$ F_2 is satisfiable.

Fact: If F_1 contains w OR operators, then we introduce at most w new variables in F_2.

Fact: Given F, F' = F_2 can be computed in polynomial time.

Example

$F_1 = ((x1 \land \sim x3) \lor x2) \lor (x1 \land \sim x3)$

$(x1) \land (\sim x3)$

$((y \lor x1) \land (y \lor \sim x3) \land (\sim y \lor x2)$

F' : $(y' \lor y \lor x1) \land (y' \lor y \lor \sim x3) \land (y' \lor \sim y \lor x2)$
$\land (\sim y' \lor x1) \land (\sim y' \lor \sim x3)$

CNFSAT $\leq_p$ 3CNFSAT (3SAT)

3SAT: to determine a 3CNF formula, in which every clause contains exactly 3 literals, is satisfiable.

What to do: Let F be any CNF-formula.

Show that, in polynomial time, F can be transformed to a 3CNF-formula F' such that

F is satisfiable if and only if F' is satisfiable.

Example:

$F = (x1 \vee \sim x2) \wedge (\sim x1 \vee x5 \vee x6 \vee x8)$

$F' = (x1 \vee \sim x2 \vee x1) \wedge (\sim x1 \vee x5 \vee y) \wedge (\sim y \vee x6 \vee x8)$

Transformation

F is already in CNF. Our concern is whether each clause in F has exactly 3 literals.

Let C be any clause in F containing $\neq$ 3 literals.

If $C = (l_1)$, replace with $(l_1 \vee l_1 \vee l_1)$.

If $C = (l_1 \vee l_2)$, replace with $(l_1 \vee l_2 \vee l_1)$.

If $C = (l_1 \vee l_2 \vee l_3 \vee l_4)$, replace with $(l_1 \vee l_2 \vee y)$ and $(\sim y \vee l_3 \vee l_4)$.

If $C = (l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5)$, replace with

$(l_1 \vee l_2 \vee y_1)$ and $(\sim y_1 \vee l_3 \vee y_2)$ and $(\sim y_2 \vee l_4 \vee l_5)$.

…

Remarks

Lemma. SAT $\leq_p$ CNFSAT.

Lemma. CNFSAT $\leq_p$ 3SAT

The above lemmas imply that CNFSAT and 3SAT are NP-complete.

What about 1SAT? (Every clause contains one literal.)

What about 2SAT? (Every clause contains two literals.)

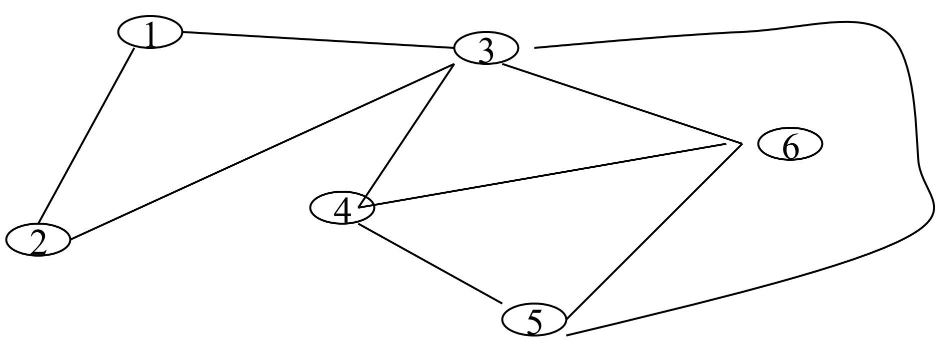
Can we prove 2SAT to be NP-complete by showing that 3SAT $\leq_p$ 2SAT? No.

Both 1SAT and 2SAT are in P.

The Clique Problem

Consider a set V of n cities with some pairs of cities connected by direct flight.

An international bank plans to set up offices in as many cities as possible; however, the management requires that the cities chosen ($V' \subseteq V$) must satisfy the property that every pair of cities in V' is connected by direct flight.



V' contains 3 cities?
4 cites? 5 cities

Graphs

We can model the cities and their connection as a **graph**.

A graph G contains a set V of nodes (vertices, points) with some pairs of nodes connected.

Formally speaking, the connection is specified by a set E of edges, i.e., $E \subseteq V \times V$.

E.g., E = { (1,2), (2,3), (1,3) …}

A clique of G is a subset V' of V such that every pair of nodes in V' is connected with respect to E.

The clique problem: Given a graph G and an integer k, find a clique with k nodes (or else report none).

The clique problem is in NP

The clique problem is in NP.

- Guess (non-deterministically choose) a subset W of k nodes in G;
- Check every pair of nodes in W are connected in G.

The clique problem is NP-complete

Lemma 3CNFSAT $\leq_p$ Clique [Why not SAT $\leq_p$ Clique ?]

What to do?

Let F be a 3CNF formula.

- How to transform F in polynomial time into the clique problem (G,k) such that
  - F has a satisfying assignment ➔ G has a clique with k nodes;
  - F has no satisfying assignment ➔ G has no clique with k nodes (equivalently, G has a clique with k nodes ➔ F has a satisfying assign.)

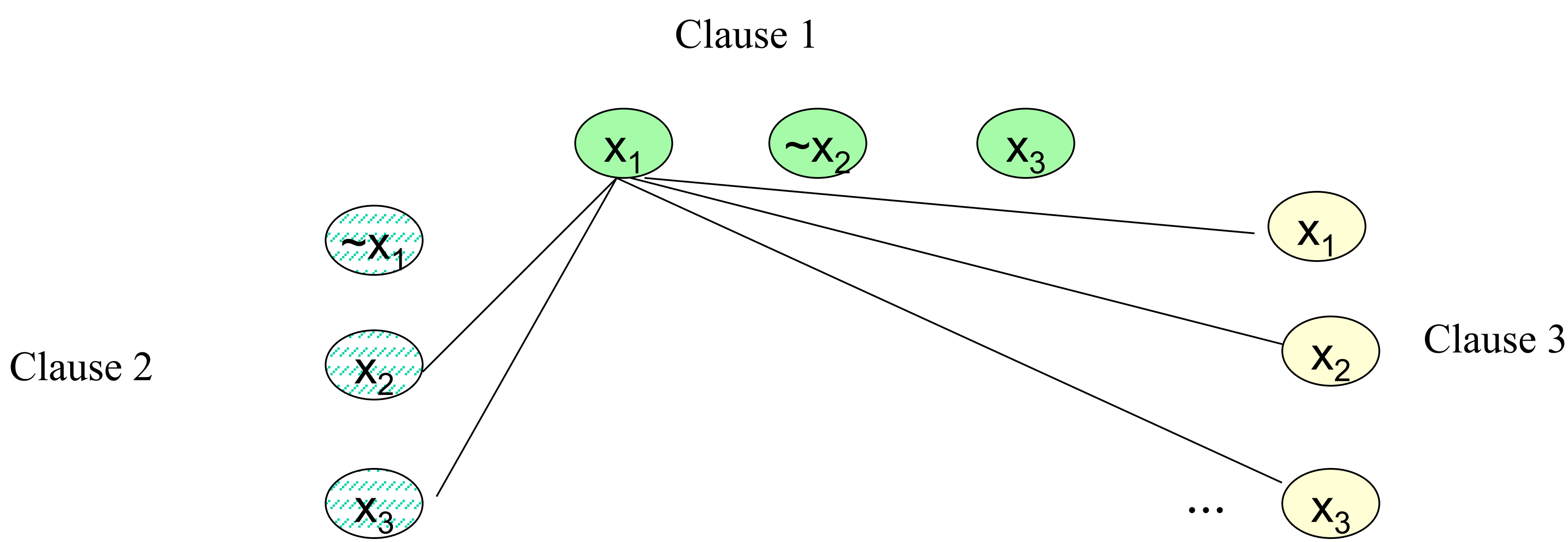
Does F have a satisfying assignment? $\xrightarrow{\text{polynomial-time}}$ Does G has a clique with k nodes?

Transforming F to (G,k).

E.g., $F = (x_1 \vee \sim x_2 \vee x_3) \wedge (\sim x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.

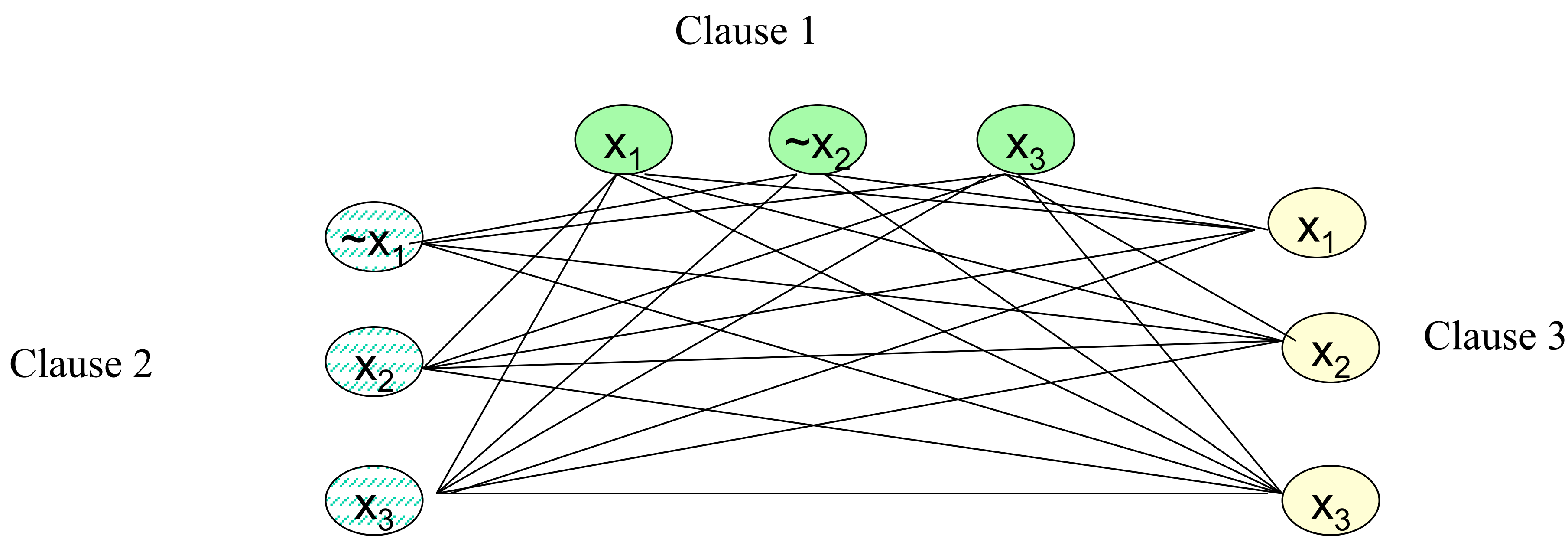
- k = 3 (the number of clauses in F).
- G has 9 nodes. Each node corresponds to a literal in F.
- Connect every two nodes originating from two different clauses, **except** when the corresponding literals are of the form x_i and $\sim x_i$.

Example



$F = (x_1 \vee \sim x_2 \vee x_3) \wedge (\sim x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.

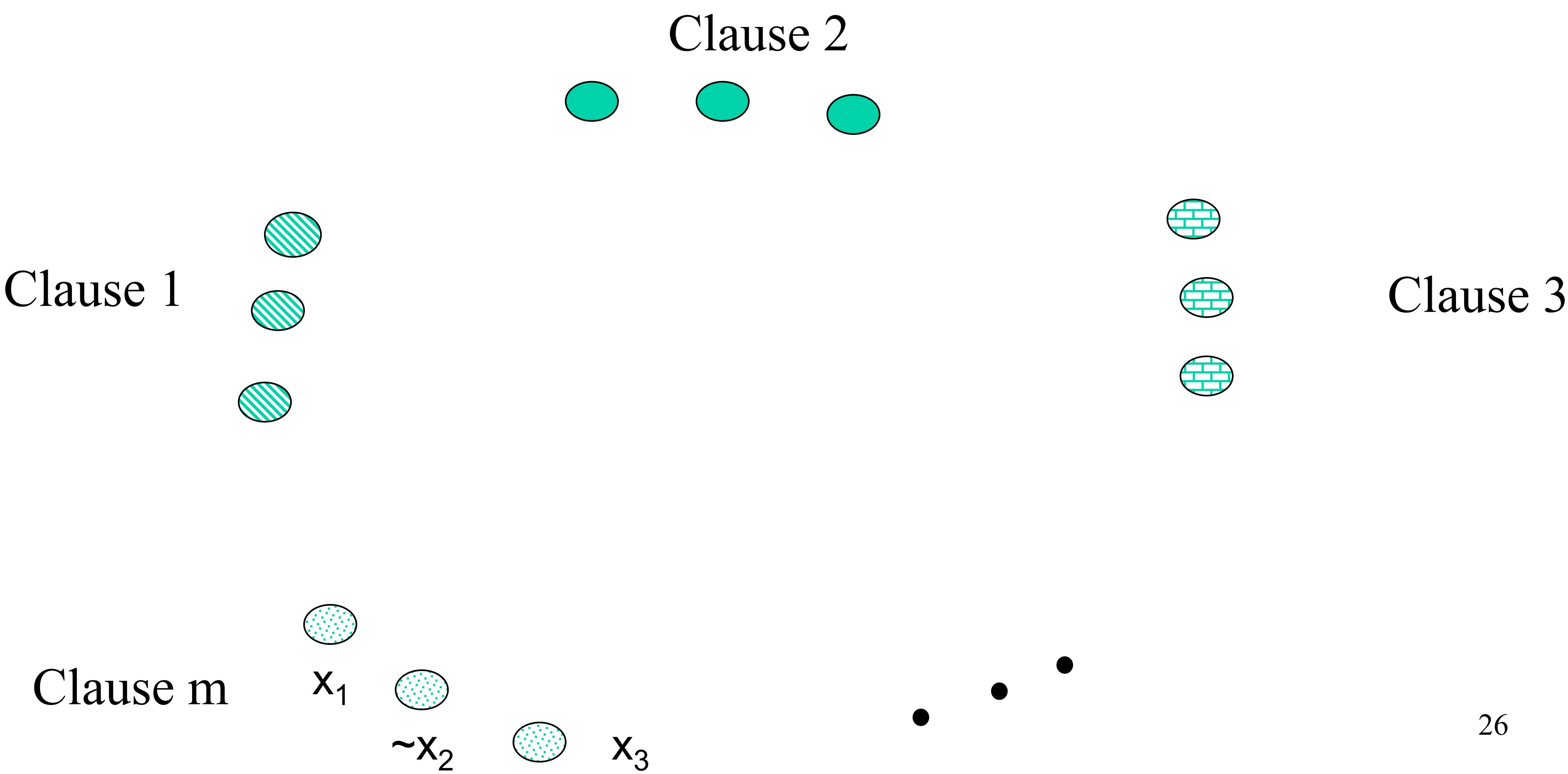
Example



$F = (x_1 \vee \sim x_2 \vee x_3) \wedge (\sim x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3).$

In general, if F has m clauses, then G has 3m nodes, divided into m groups, and k = m.

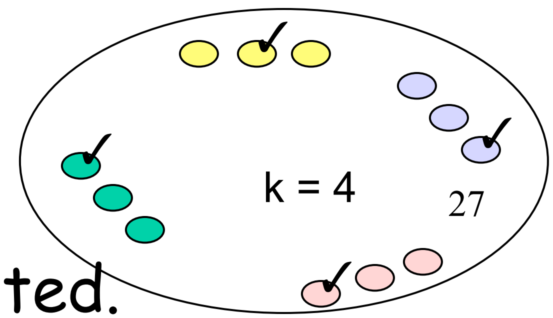
Every two nodes originating from different clauses are connected unless they are in the form of x_i and $\sim x_i$.



Correctness

F has a satisfying assignment **if and only if** G has a k-clique (i.e., a clique with k nodes).

Lemma (⇐). If G has a k-clique V', then F has a satisfying assignment.



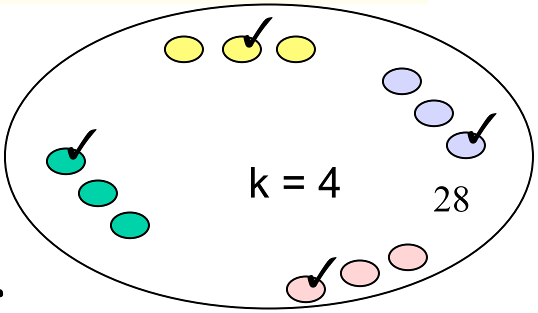
|V'| = k, and all nodes inside V' are fully connected.

Consider any two nodes in the same clause (group).

- They are not connected by an edge and cannot be in V' simultaneously.

As |V'| = k = m, every clause of F has exactly one node in V'.

Clique -> Satisfying assignment



Consider the following assignment *A* for F.

- If V′ contains a literal x_i, then x_i is assigned true;
- if V′ contains a literal $\sim x_i$, then x_i is assigned false.

Note that x_i and $\sim x_i$ are not connected in G, and can't be included simultaneously in V′.

Thus, the assignment *A* gives a unique truth value to a variable.

A makes every clause to receive at least one "**true**". In other words, *A* is a satisfying assignment.

Lemma (⇒). If F has a satisfying assignment, then G contains a k-clique.

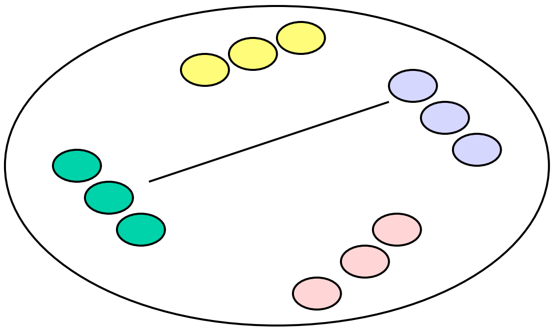
Proof: Suppose that F has a satisfying assignment *A*.

Based on *A*, we **construct a clique V'** as follows:

For each clause C_i in F, say, $C_i = x_5 \vee \sim x_8 \vee x_{12}$, one of the three literals must be true with respect to the assignment *A* (which makes F true).

V' includes the node in G corresponding to one such literal.

Note that |V'| = m = k.



Question: Are every two nodes in V' connected by an edge?

Question: Are every two nodes in V' connected by an edge?
Answer: Yes.

For any 2 nodes u and w in V', u and w come from different clauses.

Moreover, u and w cannot be labeled with x and ~x (or ~x and x). Otherwise, *A* cannot make x and ~x both true.

Thus, u and w are connected by an edge.

In other words, V' is a k-clique.

In conclusion, we have shown that
given a 3CNF formula F,
we can construct a graph G and an integer k such that
F has a satisfying assignment if and only if G has a k-clique.

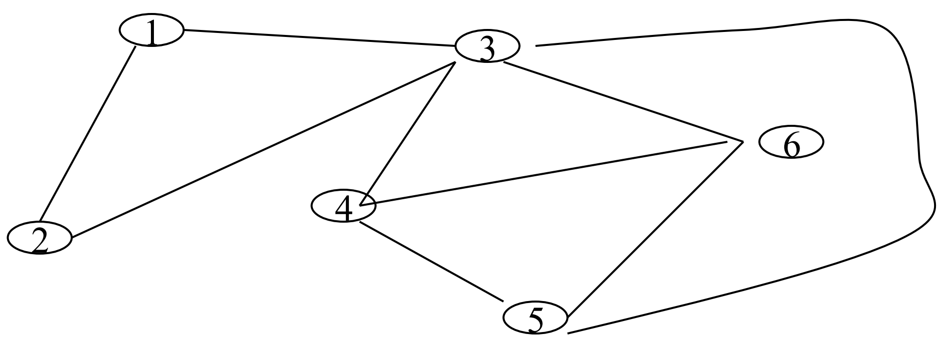
How much time does it take to construct G and k?

Nodes of G: $O(n)$ time, where n is the length of F
Edges of G: $O(n^2)$ time
Therefore, 3SAT is polynomial-time reducible to the clique problem.

The Node (Vertex) Cover Problem

Given an graph G = (V,E), a node cover U is a subset of V where every edge in E connects to at least one of the nodes of U.

Example:



{1,2,3,4,5,6} is a node cover.

{2,3,4,5,6} is a node cover.

{3,4,5,6} isn't a node cover.

{2,3,5,6} is a node cover.

- A node cover with 3 nodes?
- What is the size of the smallest node cover?

Fact: Let n be the number of nodes in G. A node cover with h < n nodes implies a node cover with h +1 nodes.

The Node (Vertex) Cover Problem

Find the smallest node cover is difficult.

Node cover (NC) problem: Given a graph G and an integer h, find a node cover with h nodes.

The NC problem is in NP.

- Given a candidate solution (a subset **U** of nodes), a certifier checks whether $|U|=h$ and every edge of G has an endpoint in **U**.

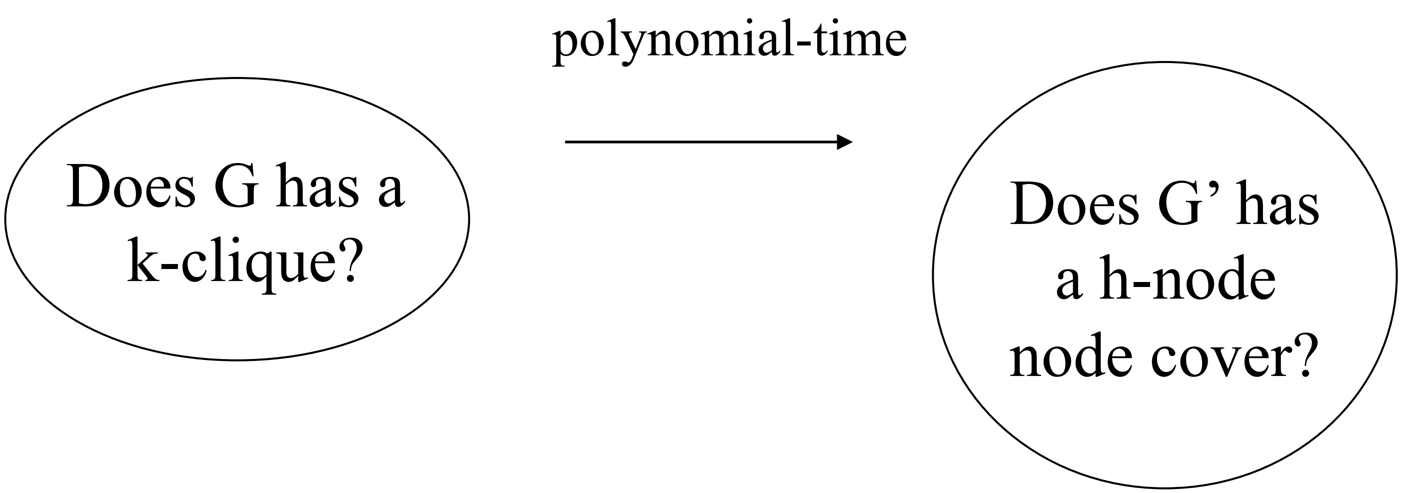
The NC problem is NP-complete.

Clique $\leq_p$ Node Cover

Given an instance of the clique problem: (G,k); does G contain a k-clique?

We want to construct an instance of the NC problem:

f(G,k) = (G', h); does G' has a node cover with h nodes ?



Reduction requirement: G has a k-clique if and only if G' has a node cover with h nodes.

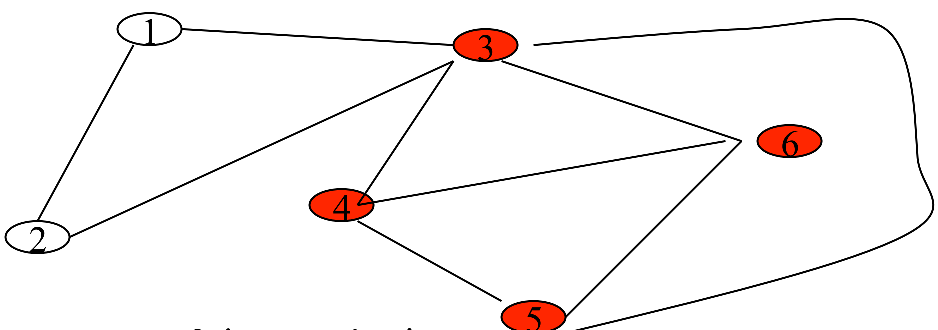
Polynomial time transformation

Given G= (V,E) and k, construct G' and h as follows:

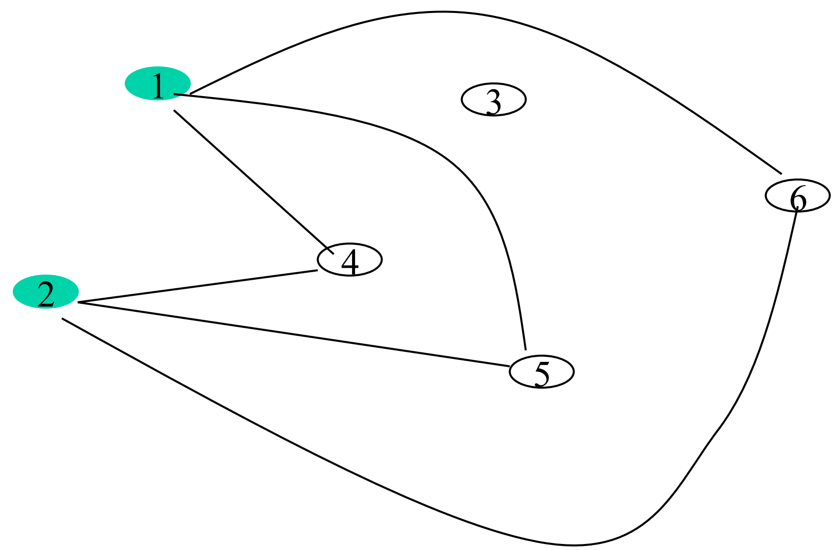
- G' has the same set of nodes as G;
- For any two nodes u, v,
  - if G contains an edge between u and v, then G' doesn't;
  - if G doesn't contains an edge between u and v, then G' does.

What is h? h = n - k.

E.g.,



G has a 4-clique.



G has a 2-NC.

Correctness

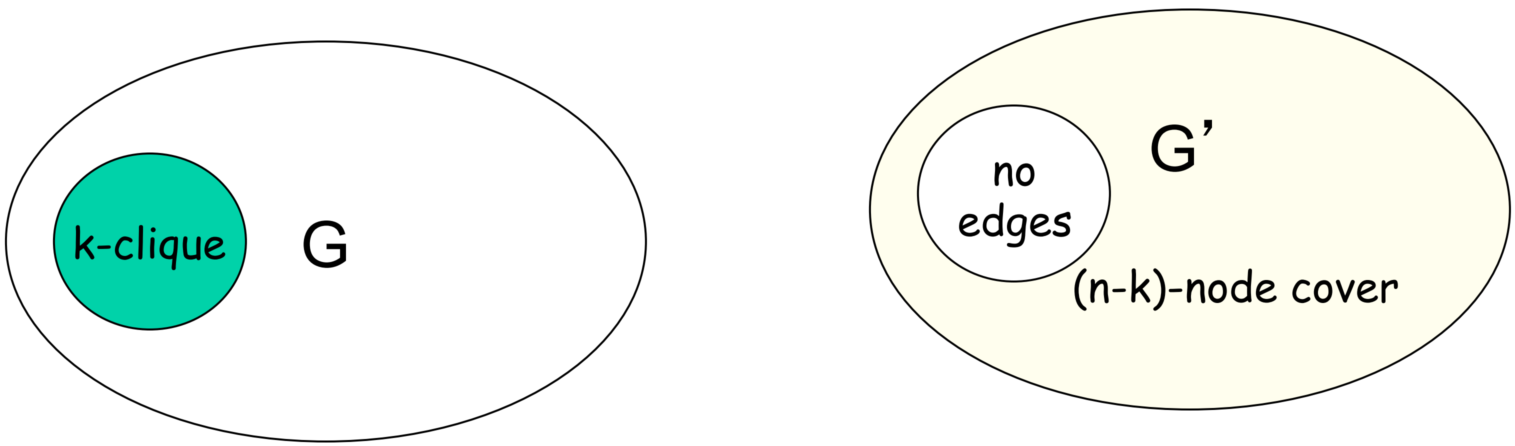
Claim: If G has a k-clique W, then **U** = V – W is a node cover of G'. Note that |U| = n - k = h.

Claim: If G' has a node cover U with h nodes, then W = V – U is a k-clique of G.

Correctness

Claim: If G has a k-clique W, then **U** = V – W is a node cover of G'. Note that |U| = n - k = h.

Claim: If G' has a node cover U with h nodes, then W = V – U is a k-clique of G.



Correctness: If G has a k-clique, …

If G has a k-clique W, then G contains an edge between any pair of nodes in W in G.

By definition of G′, G′ doesn't contain an edge between any pair of nodes in W.

Define **U** = V - W. Then |U| = n - k = h.

U is a node cover of G′. Why?

Consider any edge (**x**,**y**) in G′.

- **x** and **y** can't be both in W.
- That means, at least one of **x**, **y** is in U = V - W.

Thus, U is a node cover of G′.

Correctness: If G' has a h-node node cover, …

Let U be a node cover of G' with h = n - k nodes.

Define W = V - U. Obviously, |W| = k.

W is a clique of G. Why?

Consider any two nodes a, b in W. Both a, b are not in U.

G' can't contain the edge (a,b). Otherwise, U is not a node cover.

By definition, G contains the edge (a,b).

In summary, any two nodes in W are connected by an edge in G, or equivalently, W is a clique of G.

In conclusion, we have shown that
given a graph $G=(V,E)$ and an integer k,
we can construct another graph G' and an integer h
such that
G has a k-clique if and only if G' has a node cover with h nodes.

How much time does it take to construct G' and h?

Linear time.

The Knapsack problem is NP-complete

Given n integers $a_1, a_2, \ldots, a_n$ together with a target number t, find a collection of a_i's adding up to t.

Lemma: 3SAT $\leq_p$ the knapsack problem

We want to transform a 3CNF formula F to a set of numbers.

E.g., $F = (x_1 \text{ or } x_2 \text{ or } \sim x_3)$ and $(\sim x_1 \text{ or } x_2 \text{ or } x_3)$

What are the a_i's ? t ?

Assume that F has N variables and m clauses.

- $a_1, a_2, \ldots, a_n$ where $n = 2(N + m)$, each with (N+m) digits;
- t is also a number with N+m digits.

True/false → pick 1 out of 2 numbers

For each variable x_i in the formula F, we create two numbers a_j and a_{j+1}.

The idea is that if x_i is true, we must choose a_j; otherwise we must choose a_{j+1}. And we can never choose both.

How to enforce such a relationship? Make use of "t".

N+m digits

a_1 (x_1)	1	0	0	…
a_2 $(\sim x_1)$	1	0	0	…
a_3 (x_2)	0	1	0	…
a_4 $(\sim x_2)$	0	1	0	…
t	1	1		

Clause

For each variable C in the formula F, we want at least one of the tree literals are assigned to true.

That is, we want at least one of the three corresponding three numbers are picked.

How ? Each clause defines a column of digits.

Example: $C = x_1$ or $\sim x_2$ or x_5

	N+m digits			Clause C	
a_1 (x_1)	1	0	0	1	…
a_2 ($\sim x_1$)	1	0	0	0	…
a_3 (x_2)	0	1	0	0	…
a_4 ($\sim x_2$)	0	1	0	1	…
t	1	1		1	(2, or 3)

F = (x1 or x2 or ~x3) and (~x1 or x2 or x3)

	3 variables			Clause 1	Clause 2
a_1 (x_1)	1	0	0	1	0
a_2 $(\sim x_1)$	1	0	0	0	1
a_3 (x_2)	0	1	0	1	1
a_4 $(\sim x_2)$	0	1	0	0	0
a_5 (x_3)	0	0	1	0	1
a_6 $(\sim x_3)$	0	0	1	1	0
a_7	0	0	0	1	0
a_8	0	0	0	1	0
a_9	0	0	0	0	1
a_{10}	0	0	0	0	1
t	1	1	1	3	3

Clause 1

Clause 2

Each column contains 2 or 5 ones.

Why it works ?

If there exists a collection B whose sum is exactly t, then

- for each variable x_i, B includes either the number associated with x_i or the number associated with $\sim x_i$;
- for each clause, B includes at least one literal of the clause.

I.e., B defines a satisfying assignment for the formula F.

If F has a satisfying assignment A, define B as follows:

1. Include the number associated with x_i if A sets x_i true; otherwise include the number associate with $\sim x_i$.
2. For each column associated with a clause, the numbers chosen in step 2 must contain $x \geq 1$ in this column; include another 3-x numbers associated with this clause.

The sum of these numbers is exactly t.

Partition Problem

Knapsack: Given n integers $a_1, a_2, \ldots, a_n$ together with a target number **t**, find a collection of a_i's adding up to **t**.

Subset sum: Given n integers $a_1, a_2, \ldots, a_n$, find a collection B of a_i's such that B and the rest have equal sum.

Lemma: Knapsack $\leq_p$ partition

Let sum = $a_1 + a_2 + \ldots + a_n$

If t = sum/2, then a solution to the subset sum of C is a solution to knapsack of C.

If t < sum/2, then $A = C \cup \{ a_{n+1} = \text{sum} - 2t \}$, …

If t > sum/2, then $A = C \cup \{a_{n+1} = 2t - \text{sum} \}$, …

General Knapsack

Knapsack: Given n integers $a_1, a_2, \ldots, a_n$ together with a target number **t**, find a collection of a_i's adding up to **t**.

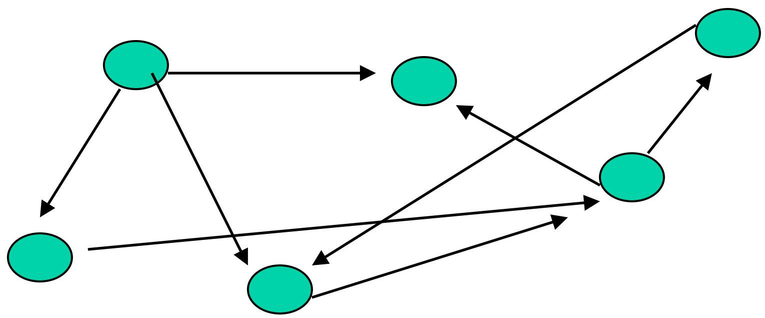
General knapsack: Given n items with (integer) weights $w_1, w_2, \ldots, w_n$ and (integer) values $v_1 \ldots v_n$, and a knapsack of capacity W and a goal *g*, find a collection of items such that their total weight is W and their total values is at least *g*.

Lemma: Knapsack $\leq_p$ general knapsack

The idea: $w_i = v_i = a_i$. $W = g = t$.

Hamiltonian path : directed case

Let G be a directed graph. A Hamiltonian path is a simple path visiting every vertex of G exactly once.



Finding a Hamiltonian path is NP-complete.

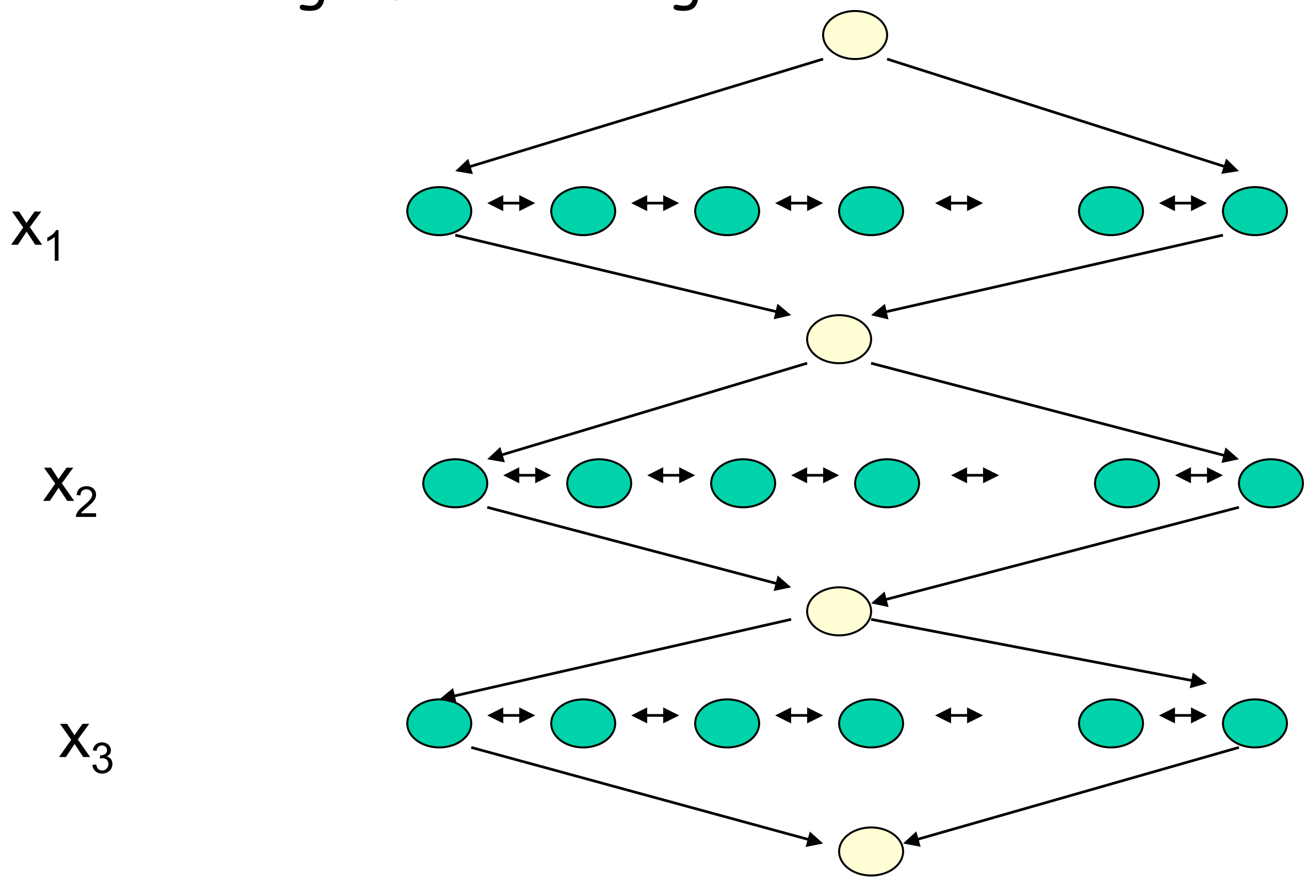
Lemma. 3SAT $\leq_p$ Hamiltonian Path

Reduction: $F \rightarrow G$

Let F be a 3CNF formula with n variables and m clauses.

Basic structure of the graph G: one row for each variable

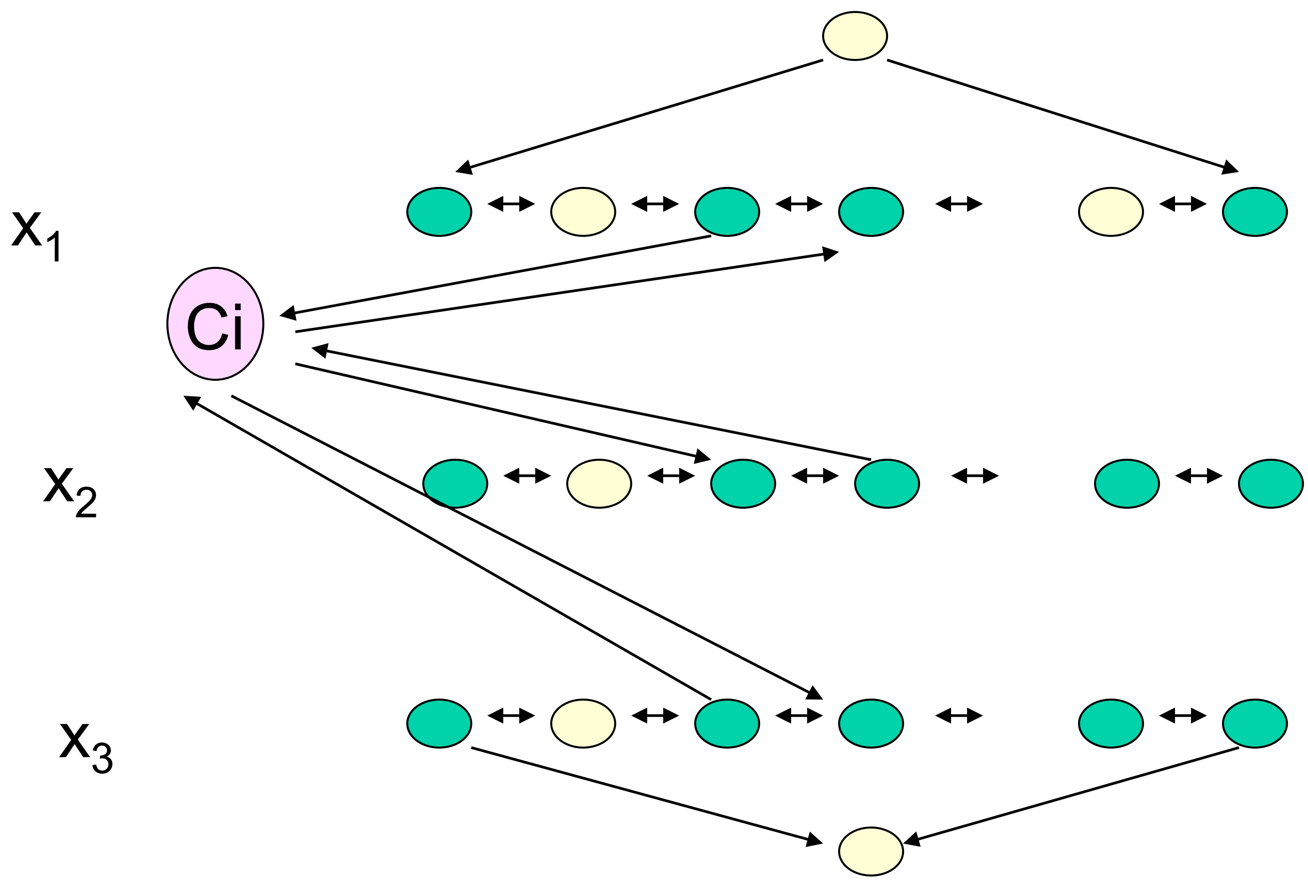
- 2^n different assignment to x_i's $\Leftrightarrow$ 2^n different Hamiltonian paths.
- True = left to right; false = right to left.



m clauses

Each row contains 3m+3 vertices.

Each clause defines an extra node. E.g., Ci = $(x_1 \vee \sim x_2 \vee x_3)$

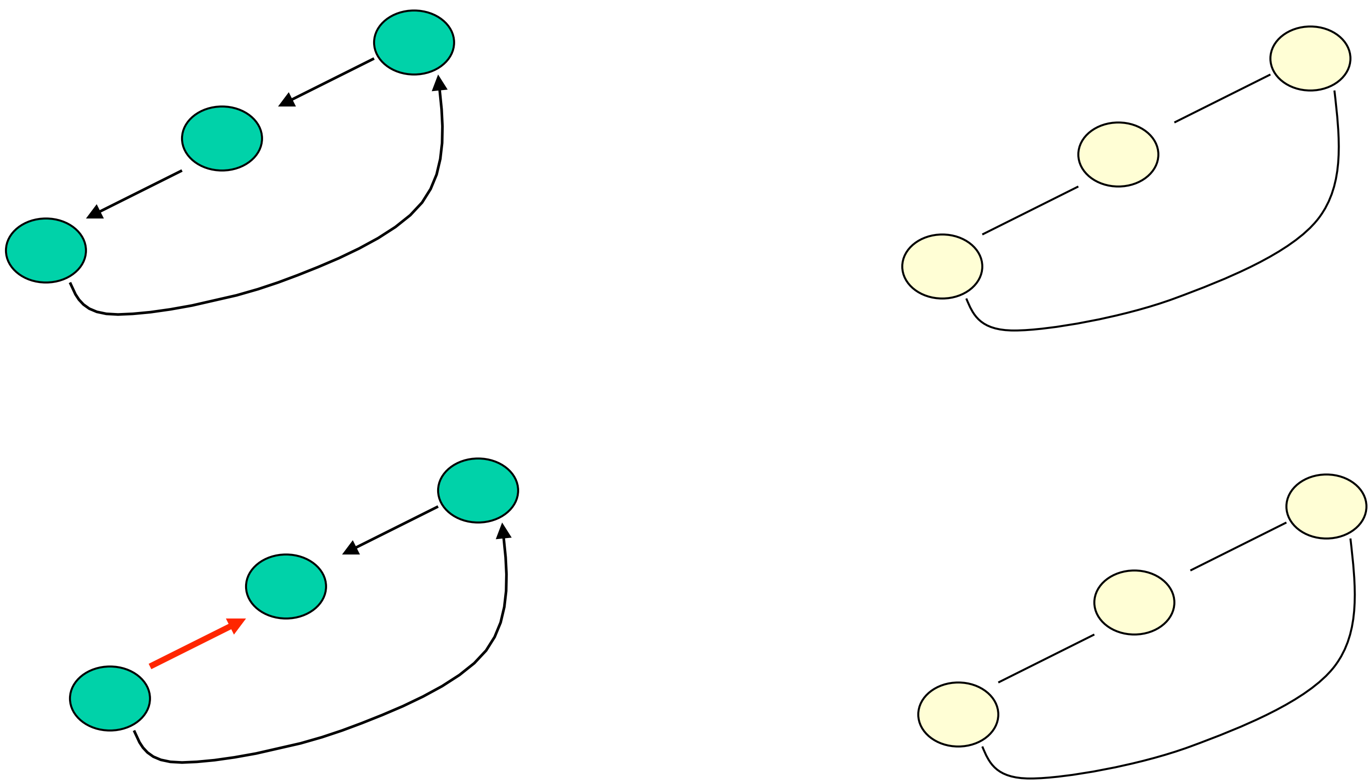


Other related problems

Hamiltonian path $\leq_p$ Hamiltonian cycle

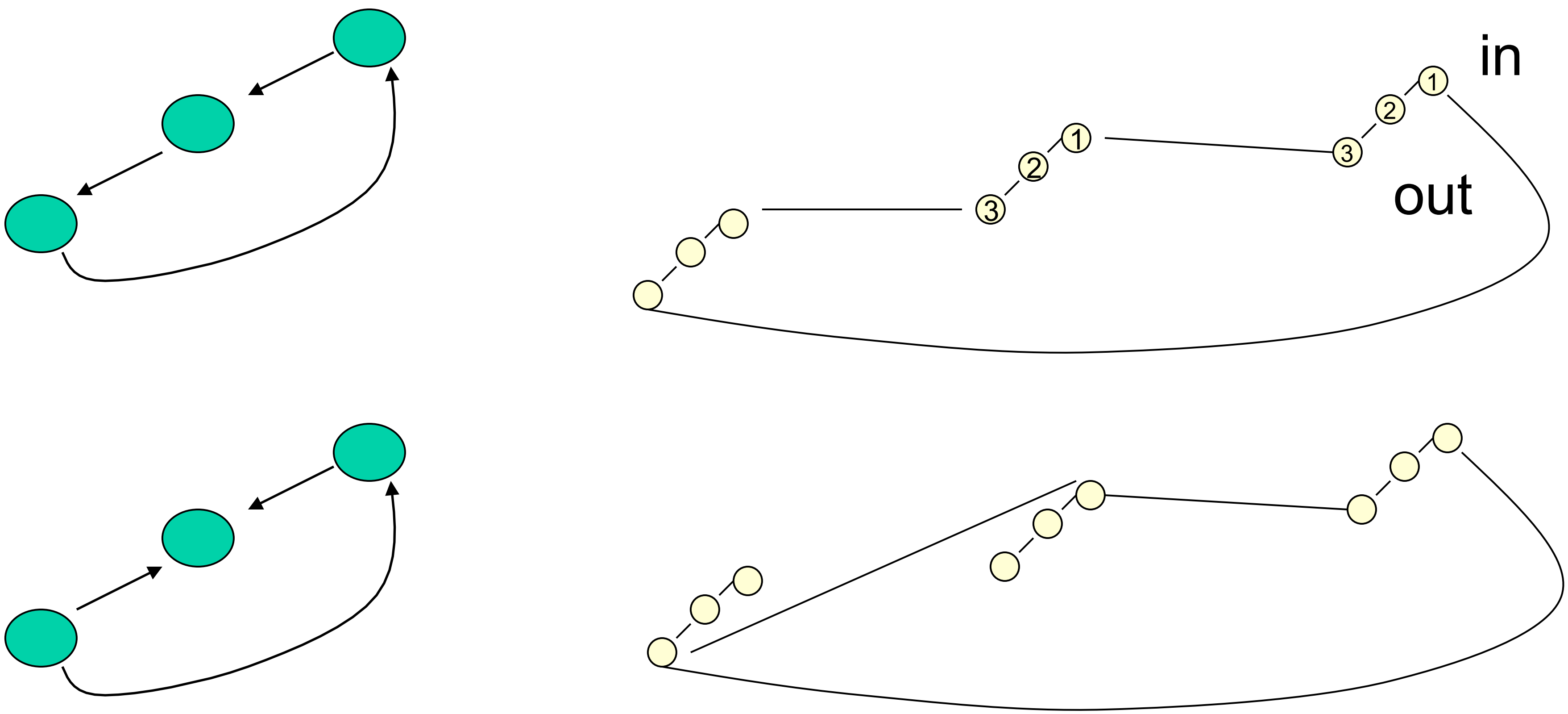
Other related problems

Hamiltonian path $\leq_p$ undirected Hamiltonian path



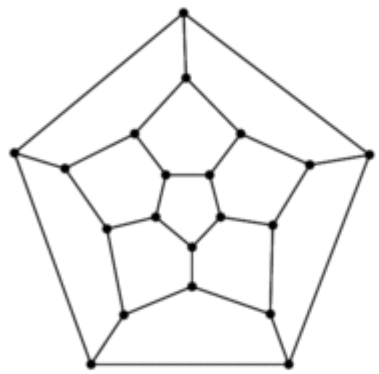
Directed cycles → undireted cycles

Hamiltonian path $\leq_p$ undirected Hamiltonian path



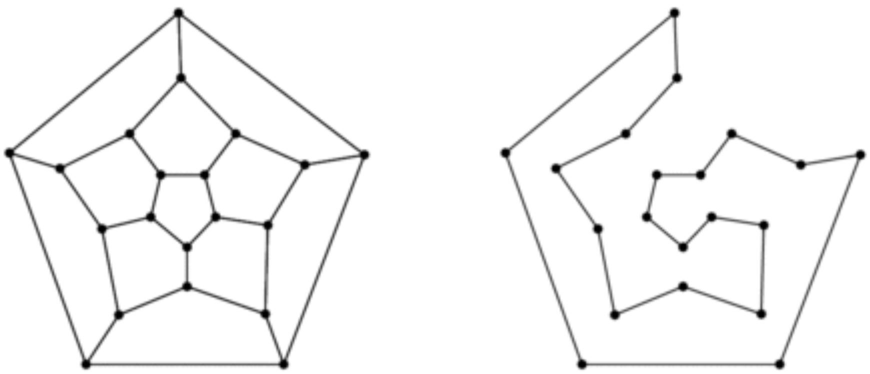
Hamiltonian cycles for undirected graphs

Classic example.



Hamiltonian path/cycles for undirected graphs

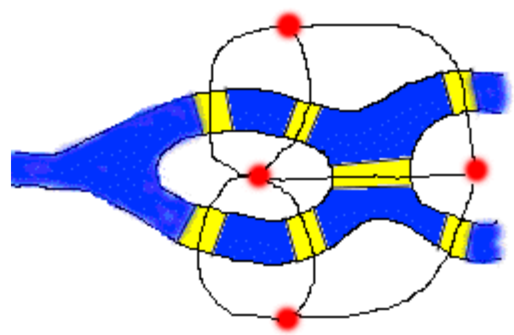
Classic example.



Hamiltonian path (cycle) problem is NP-complete.

Euler path/cycle

Let G be an undirected graph. An Euler path (cycle) is a path (cycle) visiting every edge exactly once.



An Euler cycle exists if and only if every vertex has even degree.

It takes linear time to determine whether an Euler cycle exists.

Hoe about Euler path? Except two vertices, all have even degree.

Longest path & TSP

Longest path: Given a weighted directed graph G, distinct vertices s & t, and a threshold h, find a simple path from s to t with total weight $\geq$ h.

(directed) Hamiltonian path $\leq_p$ Longest path

Edge weight=1; threshold = |V|-1. A Hamiltonian path exists if and only if a path with total weight at least |V|-1 exists.

TSP: Given an integer k and a complete undirected graph G that has a non-negative cost c(u,v) associated with each edge, find a Hamiltonian cycle (a tour visiting every node exactly once before returning to the starting vertex) with total cost at most k.

(undirected) Hamiltonian cycle $\leq_p$ travelling salesman problem