Suppose I construct a TM M_f which:

1. Preserves its input u.

2. Simulates a machine M_b on input ε.

3. If M_b hangs on input ε, force an infinite loop.

4. If M_b halts on input ε, then run a UTM on the original input u which forces an infinite loop if u is not "M" for any TM M, and otherwise it runs M (u= "M") on input ε. If M hangs on input ε, the simulating UTM also hangs.

What language does this machine M_f accept in each of two cases:

Case 1: When machine M_b does not halt on input ε.

Case 2: When machine M_b halts on input ε.

Announcements

Assignment #5 has been posted.
Due Friday Aug. 3 at the beginning of class.
We will have a tutorial this week.

Final exam tutorial:

Monday Aug. 6, 10am, ECS 116.

If the building is locked, I will prop open the back door to ECS (the one that opens on to the campus).

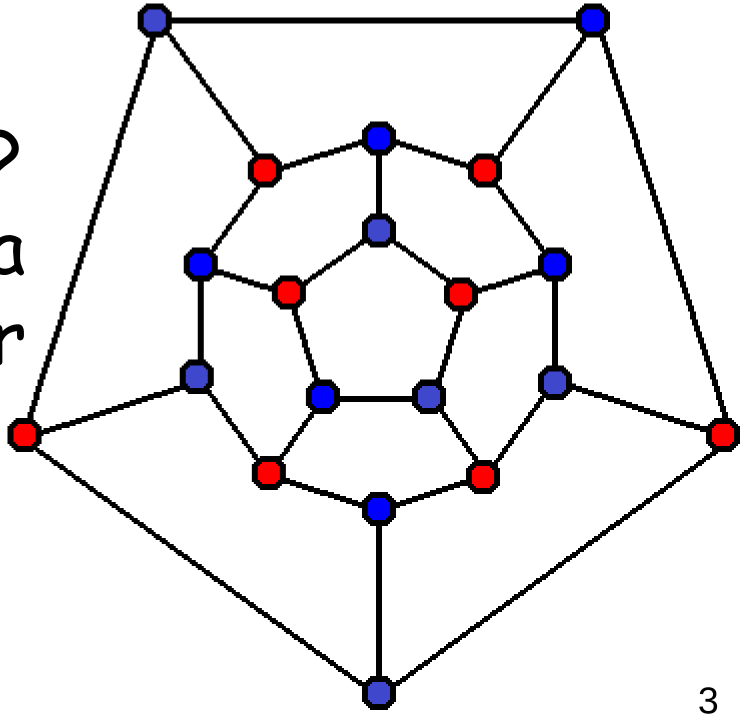
Old finals are available from the course web page.

Two vertices u and v are independent in G if edge (u,v) is not an edge of G.

INDEPENDENT SET: Input: Graph G, integer k

Question: Does G have an independent set of order k?

1. What could you use for a certificate for this problem?
2. Give the pseudo code for a polynomial time algorithm for checking your certificate.



SAT (Satisfiability)

Variables: $u_1, u_2, u_3, \ldots u_k$.

A **literal** is a variable u_i or the negation of a variable $\neg\, u_i$.

If u is set to *true* then ¬ u is *false* and if u is set to *false* then ¬ u is *true*.

A **clause** is a set of literals. A clause is *true* if at least one of the literals in the clause is *true*.

The input to SAT is a collection of clauses.

SAT (Satisfiability)

The output is the answer to: Is there an assignment of *true/false* to the variables so that every clause is satisfied (**satisfied** means the clause is *true*)?

If the answer is yes, such an assignment of the variables is called a **truth assignment**.

SAT is in NP: Certificate is true/false value for each variable in satisfying assignment.

If SAT is solvable in polynomial time, then so is HAMILTON CYCLE .

This is on p. 303 in text but there are numerous typos. Use my notes not text.

Graph G has n vertices 0, 1, 2, ..., n-1.

Variables: $x_{i,\,j}$: $0 \le i, j \le n-1$

Meaning: Node i is in position j in Hamilton cycle.

Variables: $x_{i,j}$: $0 \le i, j \le n-1$ Meaning- Node i is in position j in Ham. cycle.

Conditions to ensure a Hamilton cycle:

1. Exactly one node appears in position j.

(a) At least one node appears in position j.

For each j, add a clause:

$(x_{0,j}$ OR $x_{1,j}$ OR $x_{2,j}$ OR … $x_{n-1,j})$

(b) At most one node appears in position j.

For each pair of vertices i,k add a clause

(not $x_{i,j}$ OR not $x_{k,j}$)

Variables: $x_{i,j} : 0 \le i, j \le n-1$ Meaning- Node i is in position j in Ham. cycle.

2. Vertex i occurs exactly once on the cycle.

(a) Vertex i occurs at least once.

For each i, add a clause:

$(x_{i,0}$ OR $x_{i,1}$ OR $x_{i,2}$ OR … $x_{i,n-1})$

(b) Vertex i occurs at most once.

For each vertex i and pair of positions j,k add a clause (not $x_{i,j}$ OR not $x_{i,k}$)

Variables: $x_{i,j}$: $0 \le i, j \le n-1$ Meaning- Node i is in position j in Ham. cycle.

3. Consecutive vertices of the cycle are connected by an edge of the graph.

For each edge (i, k) which is missing from the graph and for each j add a clause

(not $x_{i,j}$ OR not $x_{k,j+1 \bmod n}$)

Known: SAT is NP-complete.

We just proved that if SAT is solvable in polynomial time, then so is HAMILTON CYCLE.

Is this a proof that HAMILTON CYCLE is NP-complete?

Known: SAT is NP-complete.

We showed that if SAT is solvable in polynomial time, then so is HAMILTON CYCLE.

Is this a proof that HAMILTON CYCLE is NP-complete?

NO. Wrong direction. We would have to show that you can solve SAT in polynomial time using HAMILTON CYCLE instead.

Another example: proving you can solve 2-SAT using a SAT solver does not mean 2-SAT is hard.

A problem Q in NP is NP-complete if the existence of a polynomial time algorithm for Q implies the existence of a polynomial time algorithm for all problems in NP.

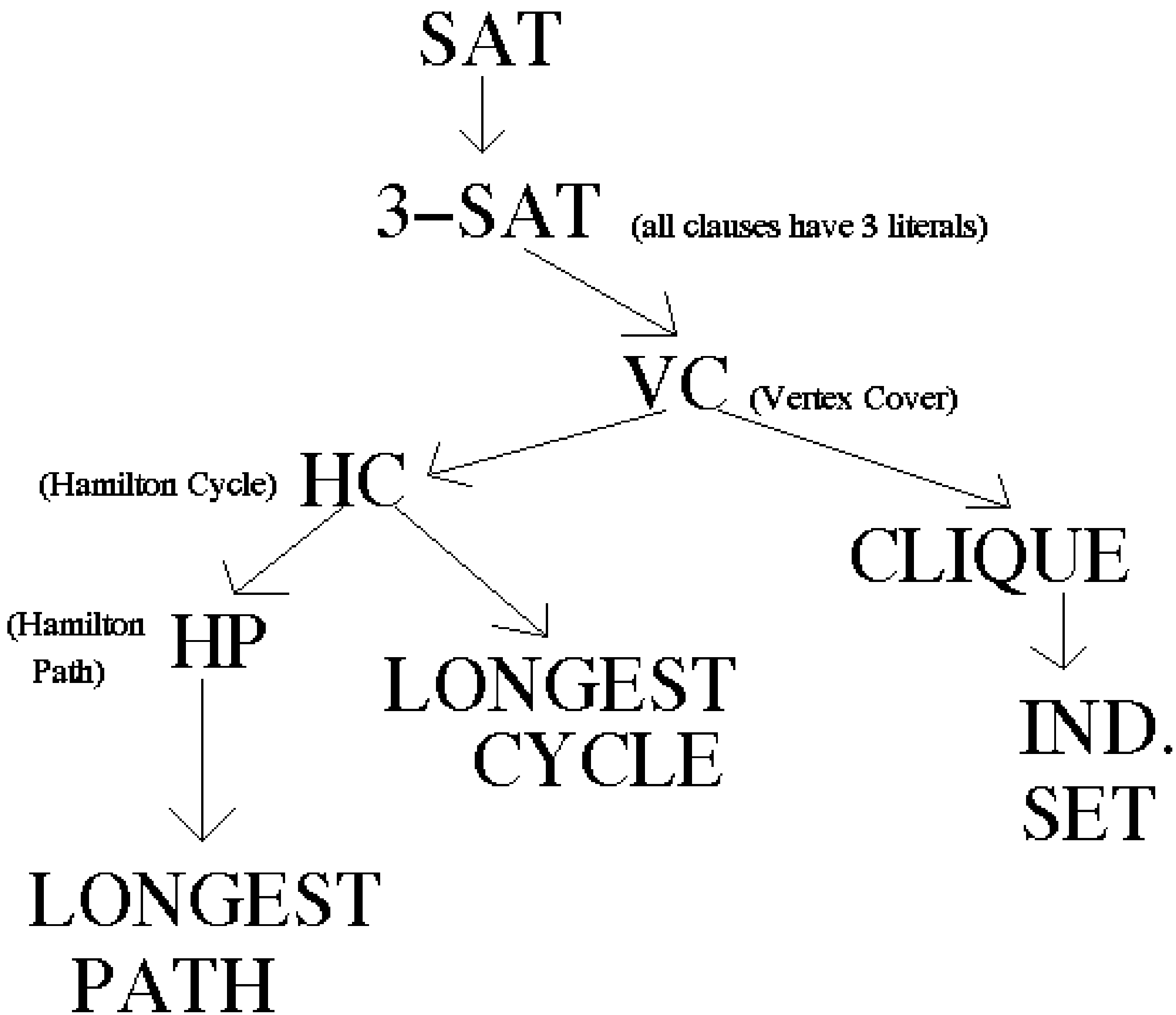
How do we prove SAT is NP-complete?

Cook's theorem: SAT is NP-complete.

Proving problems are NP-complete.

Assuming SAT is NP-complete, we prove that 3-SAT is NP-complete.

We use the fact that 3-SAT is NP-complete to prove that VERTEX COVER is NP-complete.



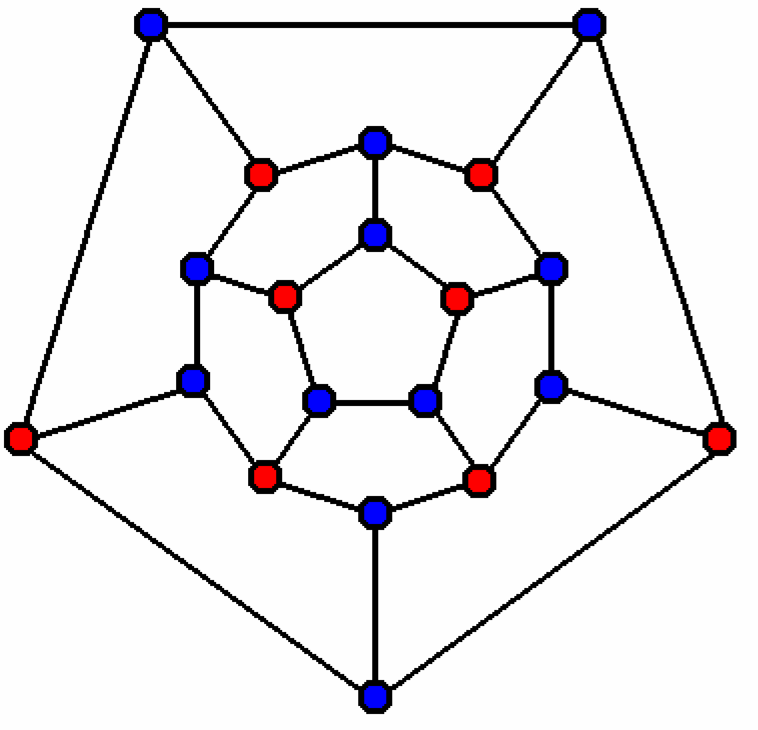
History of NP–completeness Reductions

A set $S \subseteq V(G)$ is a vertex cover if every edge of G has at least one vertex in S.

VERTEX COVER:

Given: G, k

Question: Does G have a vertex cover of order k?



Blue: vertex cover

Red: independent set

Theorem: Vertex Cover is NP-complete.

Proof: Certificate: vertex numbers of vertices in the vertex cover. To check:

```
for (i=0; i < n; i++) cover[i]= 0;
for  (i=0; i < k; i++)
{    scanf("%d", &t);
     if (t < 0 || t >= n)
          {printf("Bad cover.\n"); exit(0);}
     else  cover[t]= 1;
}
```

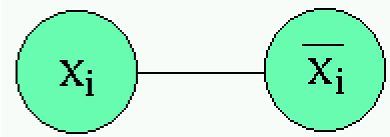
Read in certificate.

```
for (i=0; i < n; i++) {
   for (j=i+1; j< n; j++) {
      if (A[i][j]){
         if (cover[i]==0 && cover[j]==0)
         {  printf("Bad cover.\n"); exit(0); }
      }
   }
}
printf("Good cover\n");
```

Make sure each edge is covered.

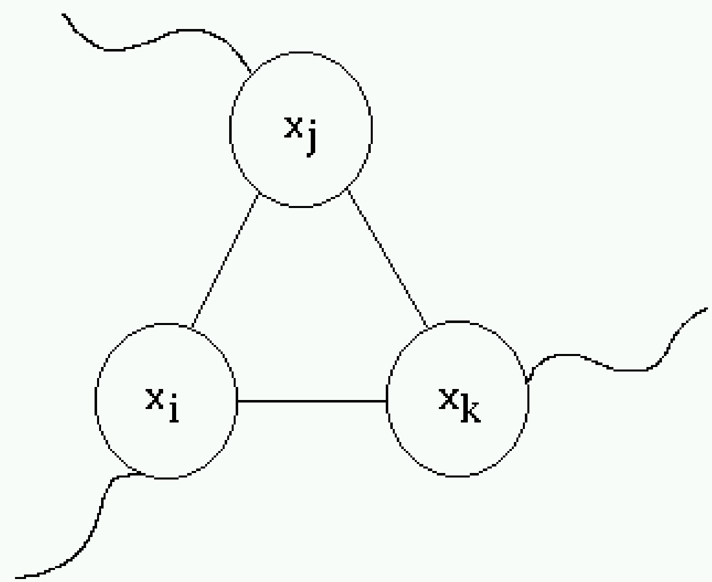
To solve 3-SAT using vertex cover:

1. For each literal x_i, include:



2. For each clause (x_i, x_j, x_k) use a gadget:

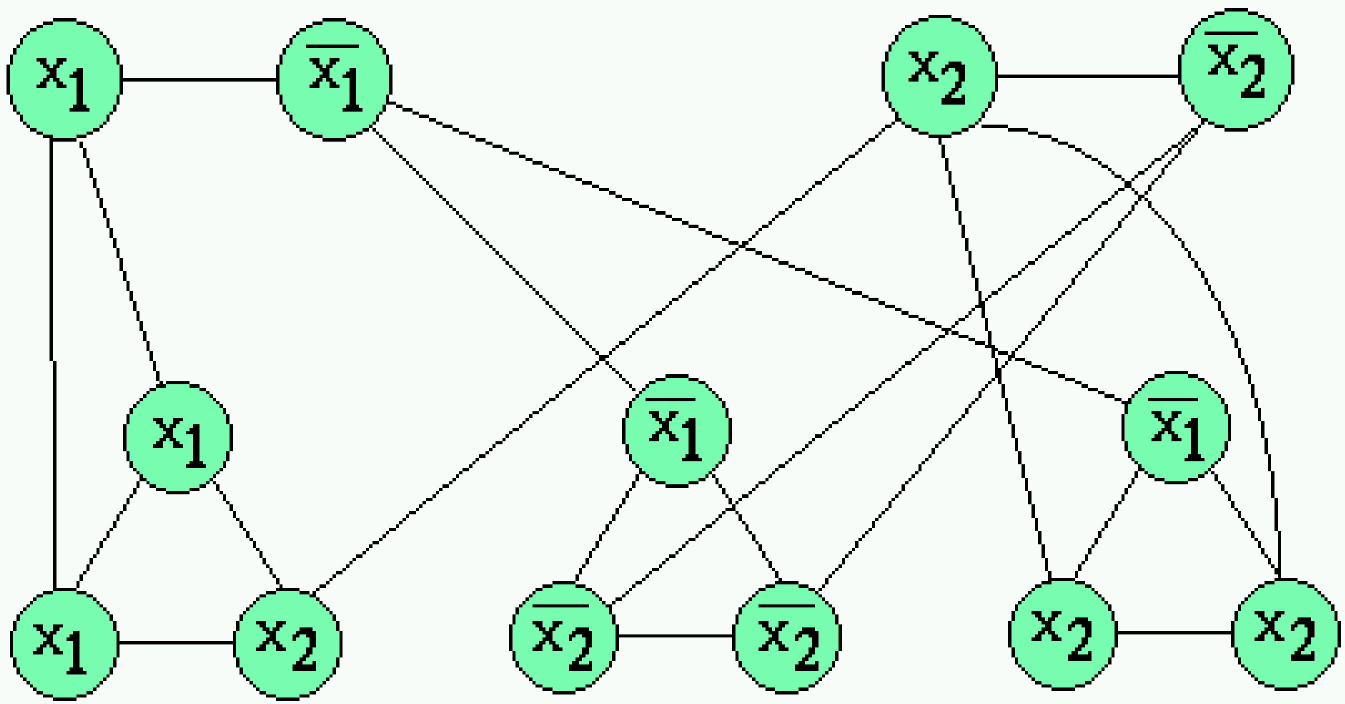
Each white vertex connects to the corresponding green one.



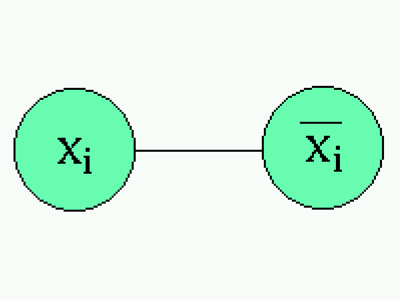
Pictures from: http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2001/CW/npproof.html

3-SAT Problem:

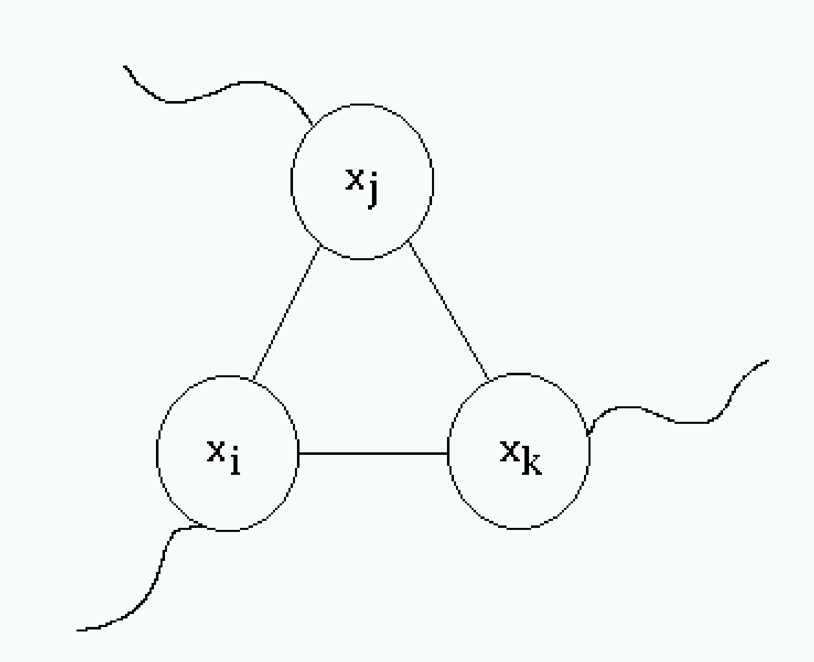
(x1 or x1 or x2) AND (¬x1 or ¬ x2 or ¬ x2) AND (¬x1 or x2 or x2)



At least one vertex from is in the vertex cover.



For each gadget, at least 2 vertices are in the vertex cover:

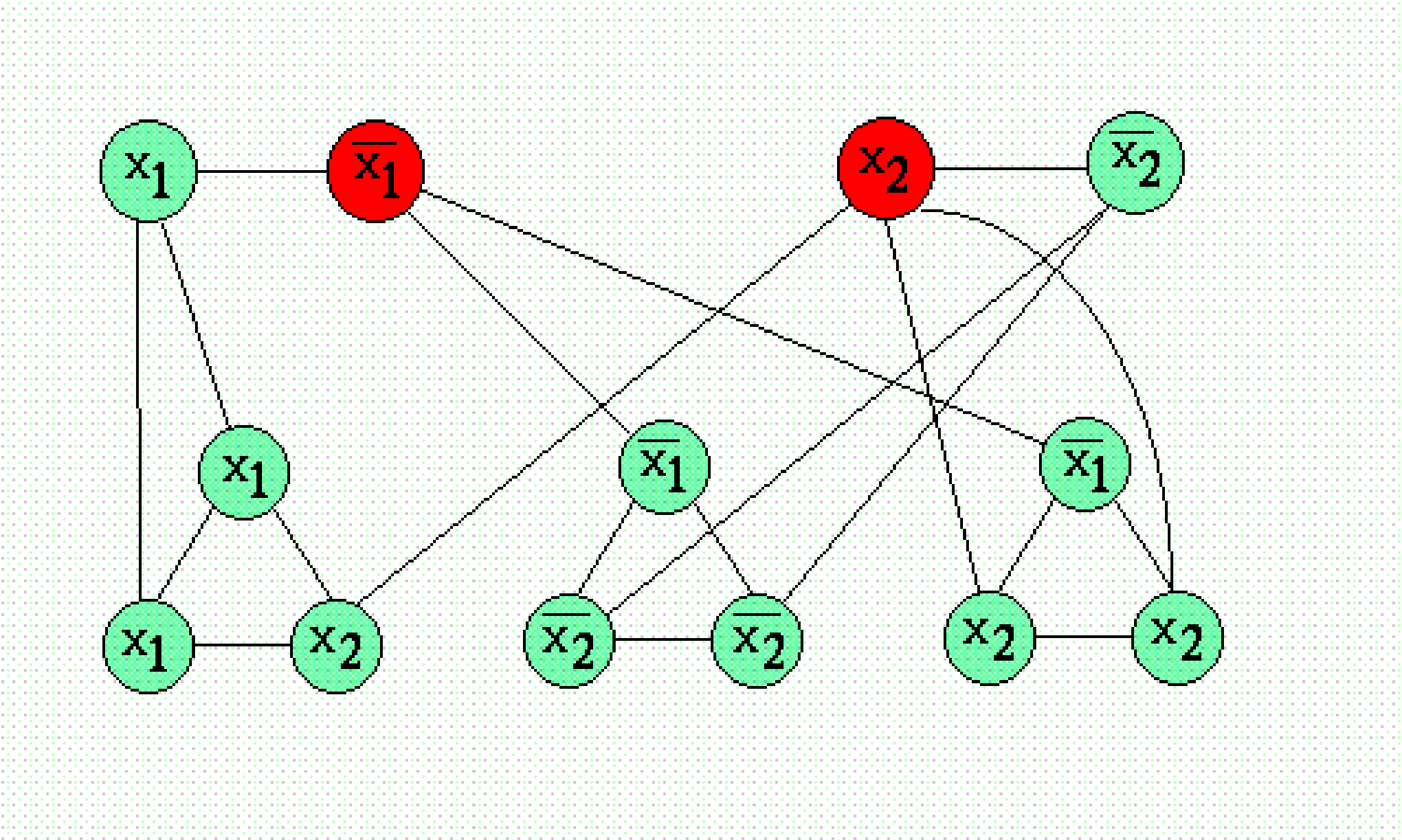


Number of variables: n

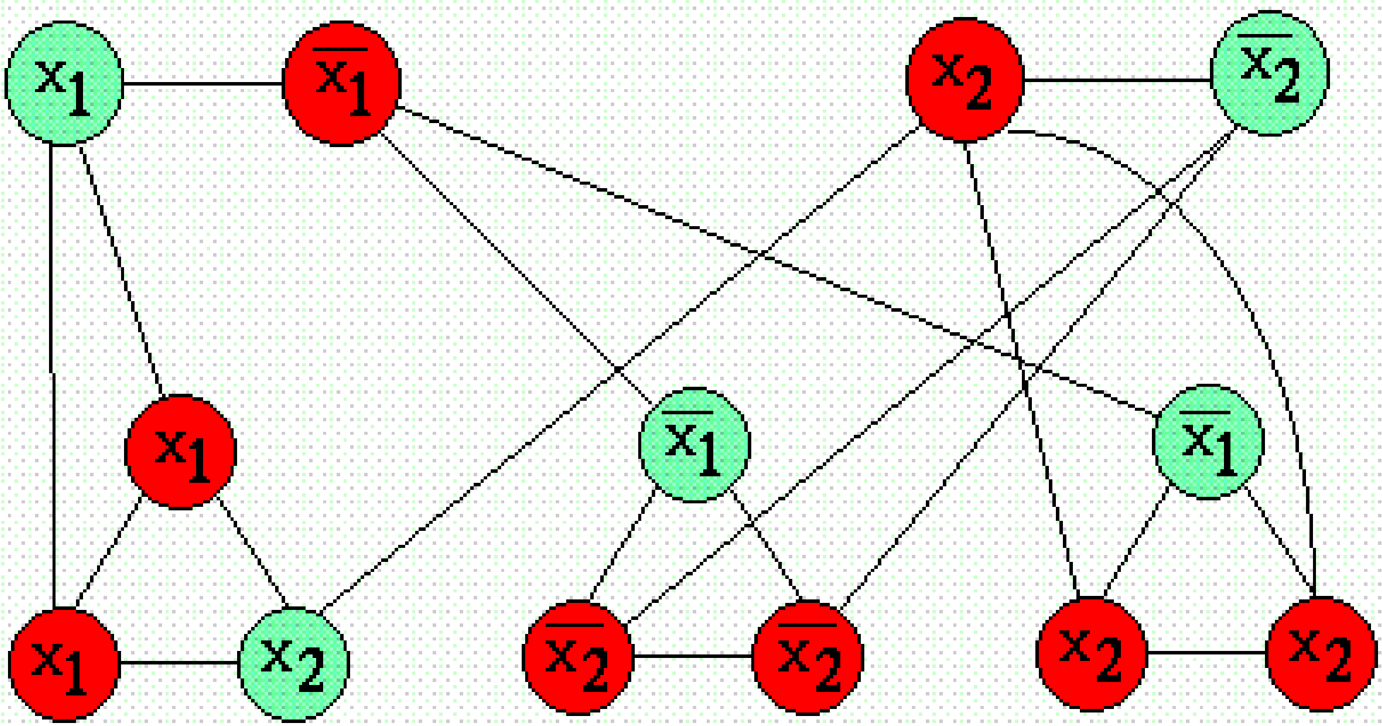
Number of clauses: m

When is there a vertex cover of order n + 2m?

Put vertices corresponding to true variables in the vertex cover.

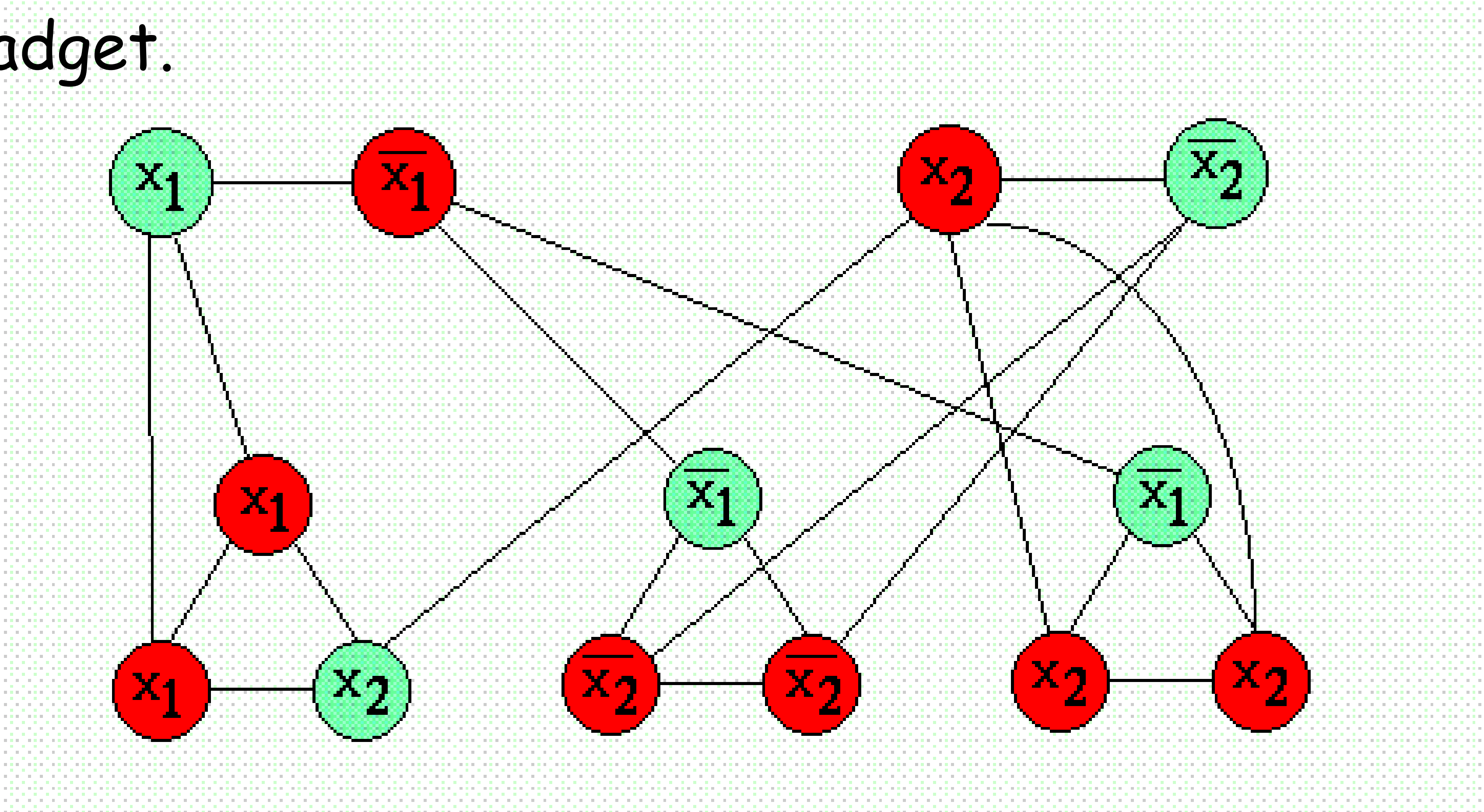


Satisfying assignment: Each clause has at least one true variable. Put two other vertices into the vertex cover:



So each truth assignment corresponds to a vertex cover of order n + 2m.

Any vertex cover of order n + 2m corresponds to a satisfying assignment because we can only select at most one of x and ¬x (these are the true variables). The true variables must satisfy each clause since at most 2 vertices can be selected from each clause gadget.



A problem Q in NP is NP-complete if the existence of a polynomial time algorithm for Q implies the existence of a polynomial time algorithm for all problems in NP.

How do we prove SAT is NP-complete?

Cook's theorem: SAT is NP-complete.