Proving relative hardness (by reduction)

Lecture 08.02

by Marina Barsky

The proof by reductions

$$Y \le_p X$$

Known problem   New problem

Suppose we can reduce problem Y to problem X in polynomial time

- If Y is in P then nothing is known about X - we can always encode the easy instance into a hard one
- If X is in P then Y is also in P: solve X in poly time + poly time of reduction
- If Y is not in P then X is not in P (contrapositive): suppose X is in P then Y should also be in P - but we know it is not
- If X is not in P then nothing known about Y - Y might still have a poly solution - we could have encoded an easy problem Y into a hard problem X

To prove that a new problem X is NP-complete, reduce a known NP complete problem Y to X

Proving NP-hardness

The new problem X is NP-complete if:

1. X is in NP (solution is verifiable in polynomial time)

2. A known NP-complete problem is polynomial-time reducible to X

Recap: 3-SAT problem

3-SAT. Given a CNF formula ϕ where each clause contains exactly 3 literals, does it have a satisfying truth assignment?

$$\phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Satisfying instance: $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$

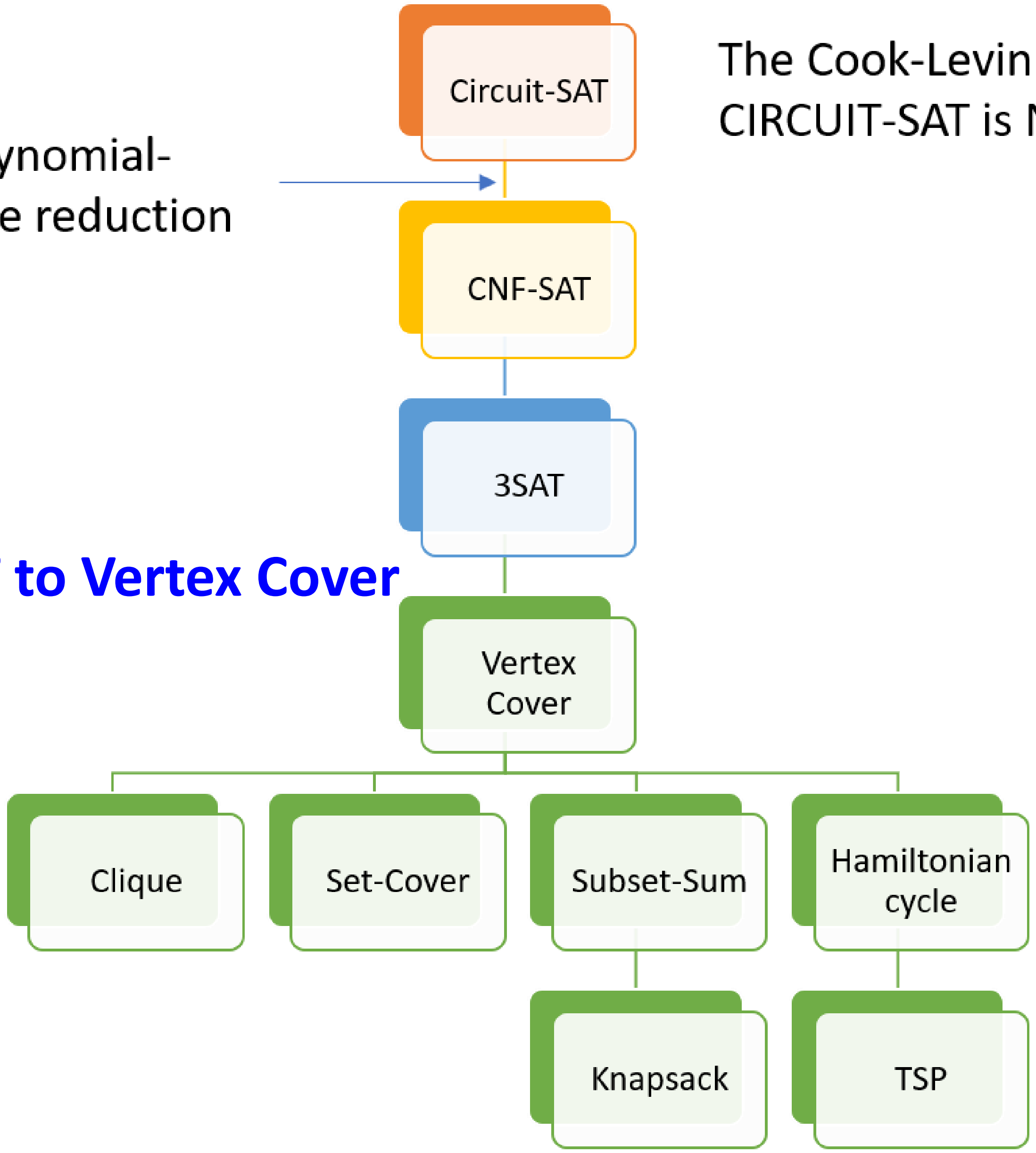
Let's assume that we know that 3-SAT is NP-hard

Sample reduction

The Cook-Levin Theorem:
CIRCUIT-SAT is NP-complete

Polynomial-
time reduction

Reduction of 3-SAT to Vertex Cover



New problem X: Vertex cover

VERTEX-COVER as a decision problem
Input: undirected graph G(V,E) and an integer k **Output**: Yes, if there is a subset C of k vertices such that, for every edge (v,w) of G, $v \in C$ or $w \in C$ (possibly both). No, otherwise.

So if we can show that $Y \leq_p X$ AND we know that Y is hard (NP-complete), then X must be NP-complete.

As an example, we will prove that Vertex-Cover is NP-complete

1. **Vertex-Cover is in NP**
2. **3-SAT $\leq_p$Vertex-Cover**

Y X

1. Vertex-Cover is in NP

We need to show that there exists a poly-size certificate and verification is in polynomial time

Let's number vertices of G from 1 to N.
If somebody hands us a collection C of k numbers each in interval from 1 to N, we can verify if this is a vertex cover in polynomial time

For this, we insert all the numbers of C into a dictionary, and then we examine each of the edges in G to make sure that, for each edge (v,w) in G, v is in C or w is in C.

- **If we ever find an edge with neither of its end-vertices in G, then we output "no"**
- **If we run through all the edges of G so that each has an end-vertex in C, then we output "yes"**

Such a verification runs in polynomial time O(m) = O(n^2).

Thus, VERTEX-COVER is in NP.

2. Reduction of 3-SAT to Vertex-Cover

We take a general instance of 3-SAT problem
Each 3-SAT instance contains n literals x_1, x_2, … x_n and m clauses

3 clauses

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$

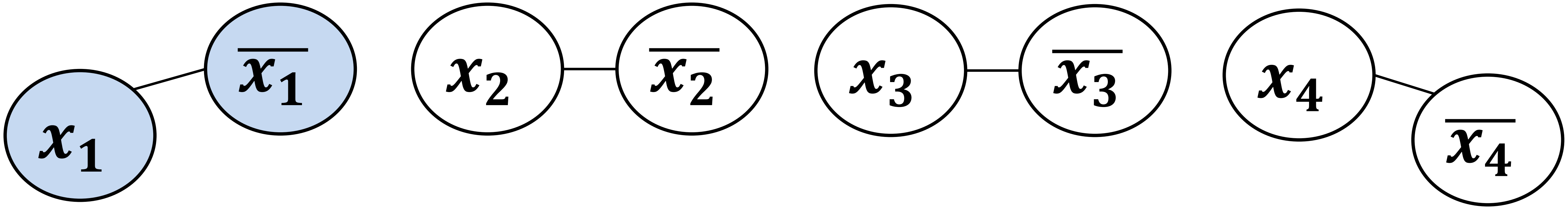
4 literals

We convert the instance into a graph as following:
For each literal i, we create 2 nodes x_i and $\overline{x_i}$ with an edge between them: truth-setting component

For each clause we create 3 nodes connected into a triangle. Each node has an additional edge to the corresponding literal: clause-satisfying component

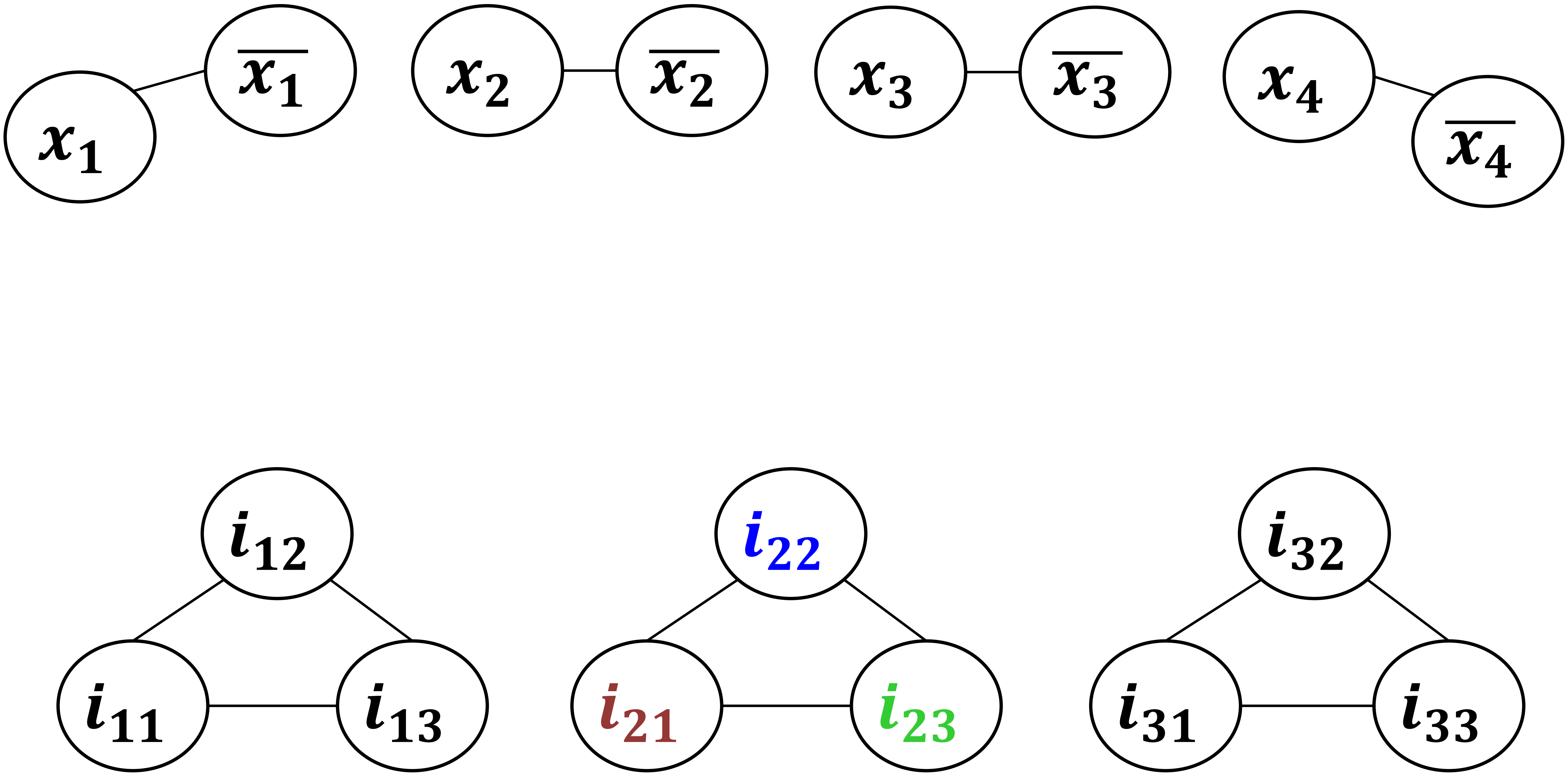
For each literal, create a pair of nodes

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



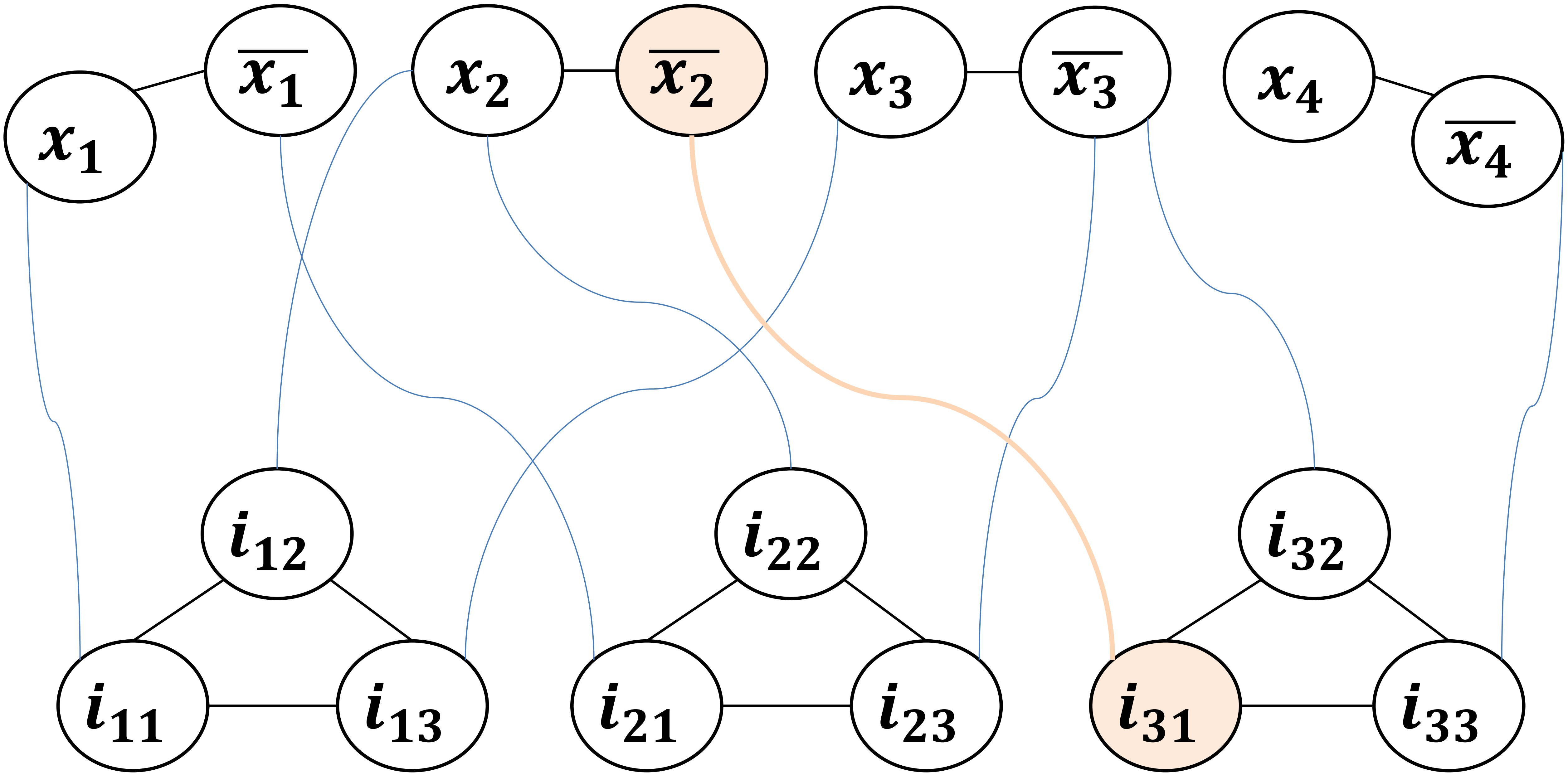
For each clause, create a triangle

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



Connect each variable in the clause to the corresponding literal

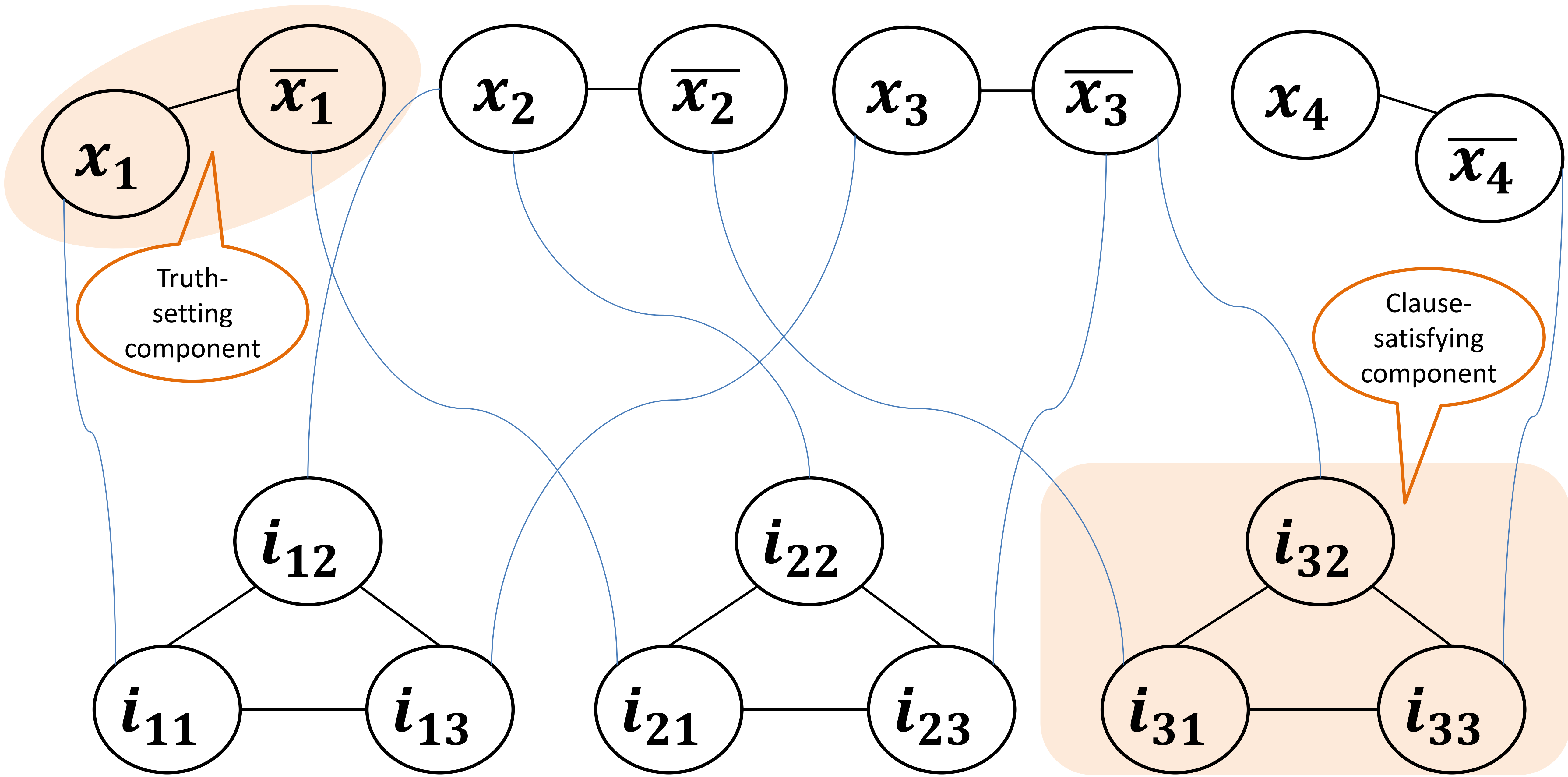
$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



This construction clearly runs in polynomial time

Instance of Vertex Cover constructed from instance of 3-SAT

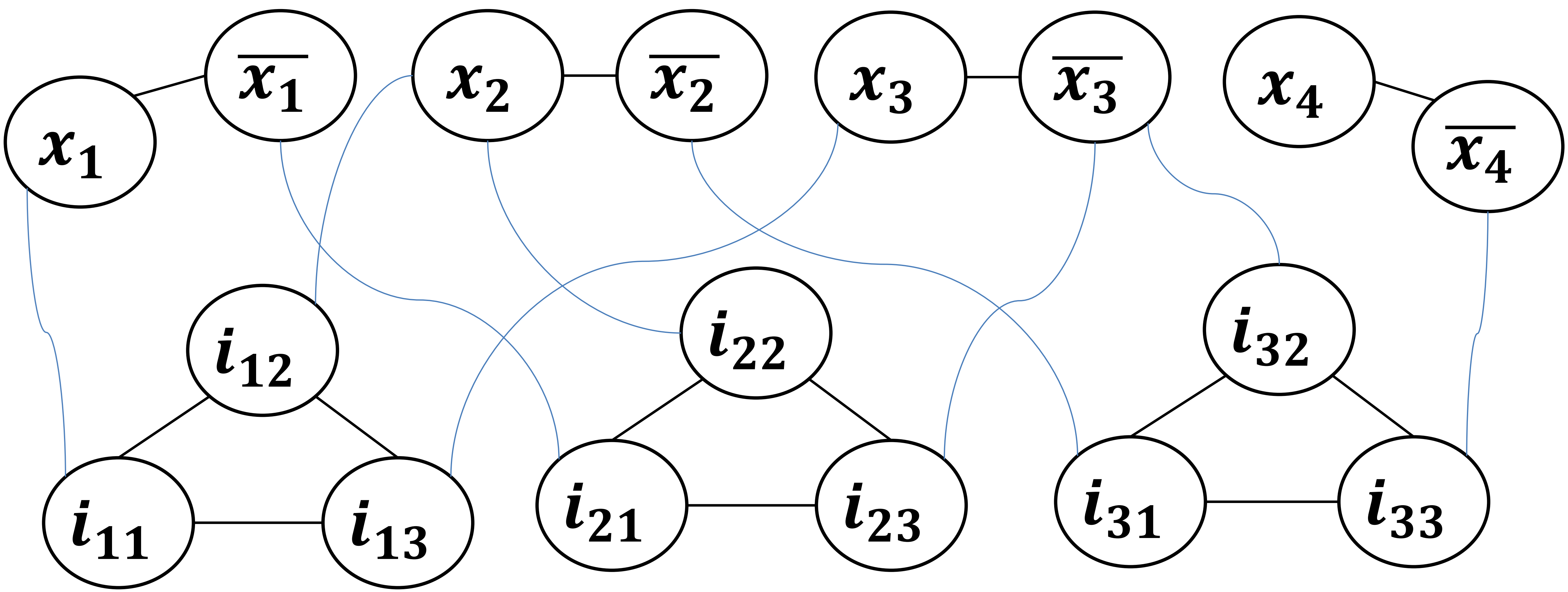
$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



Example graph G as an instance of the VERTEX-COVER problem constructed from the formula ϕ

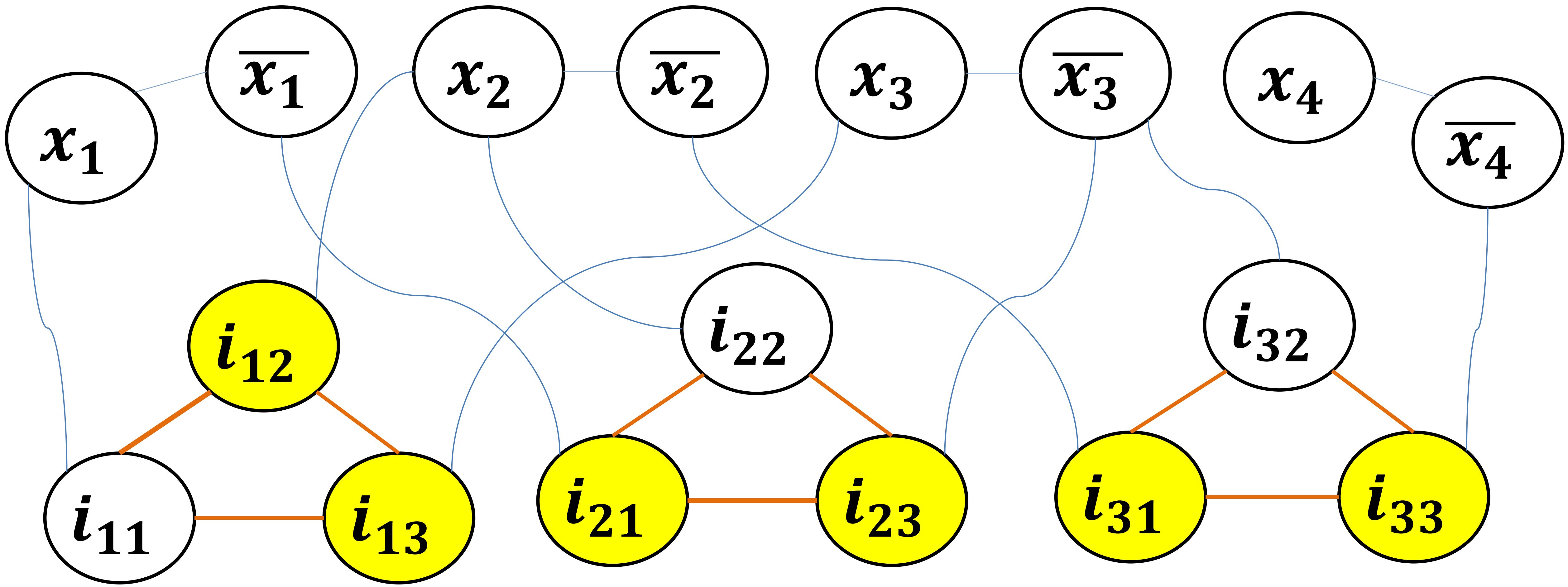
Claim: If we can find a vertex cover of size at most k=n + 2m (n-number of literals, m-number of clauses), then this vertex cover represents a truth assignment for 3-SAT problem

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



Proof: 1/4

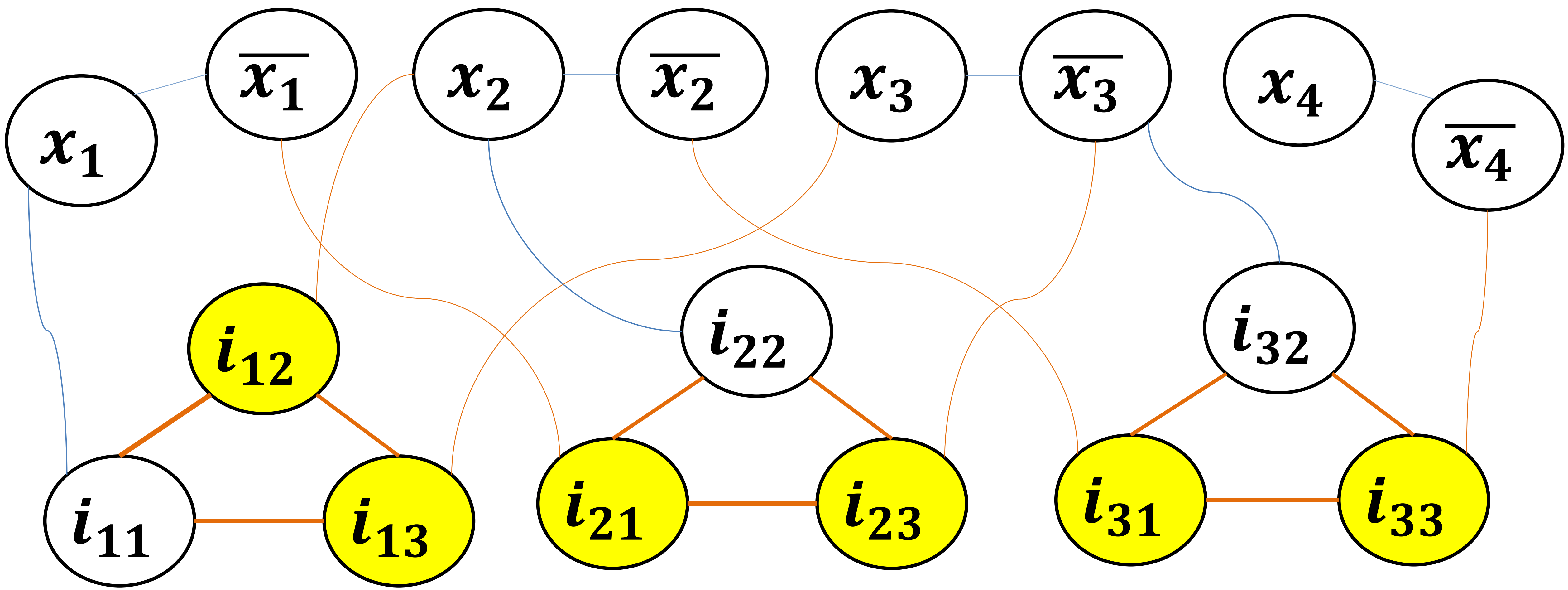
$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



The vertex cover must contain at least two vertices from each clause-satisfying component (to cover all three edges of a triangle).

Proof: 2/4

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$

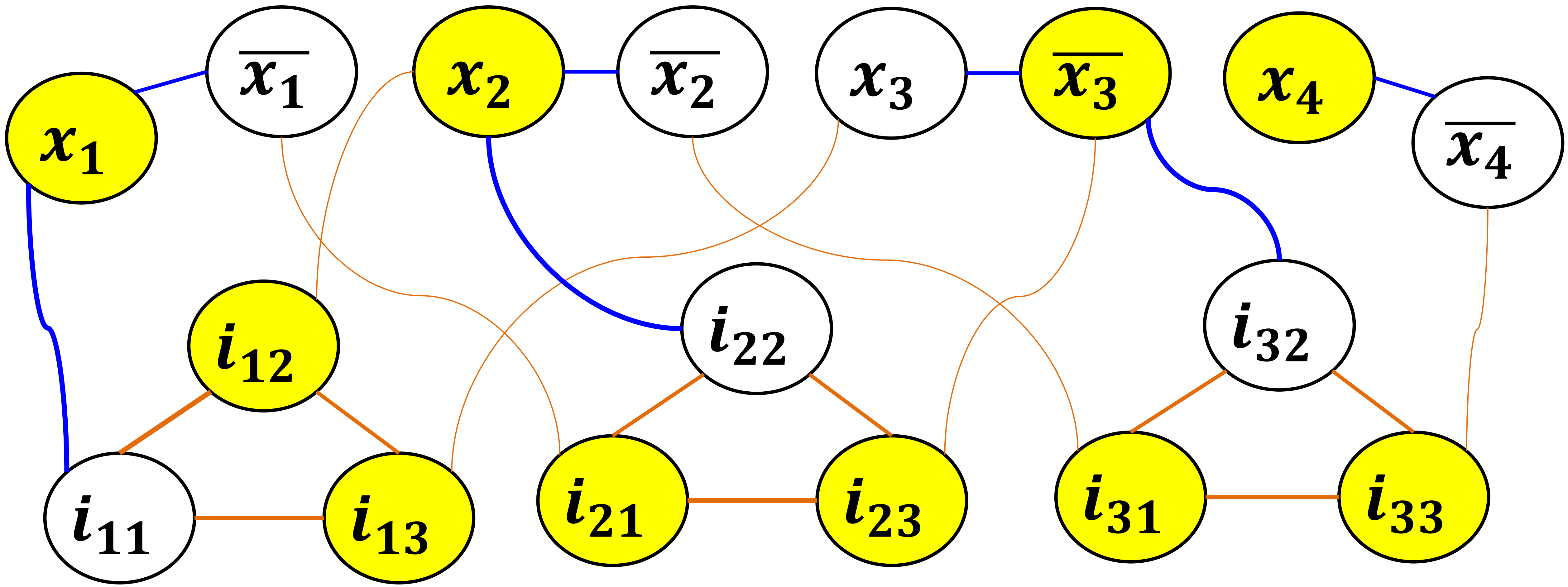


The vertex cover must contain two vertices from each clause-satisfying component (to cover all edges of a triangle).

Now all the outgoing edges from yellow vertices are covered automatically.

Proof: 3/4

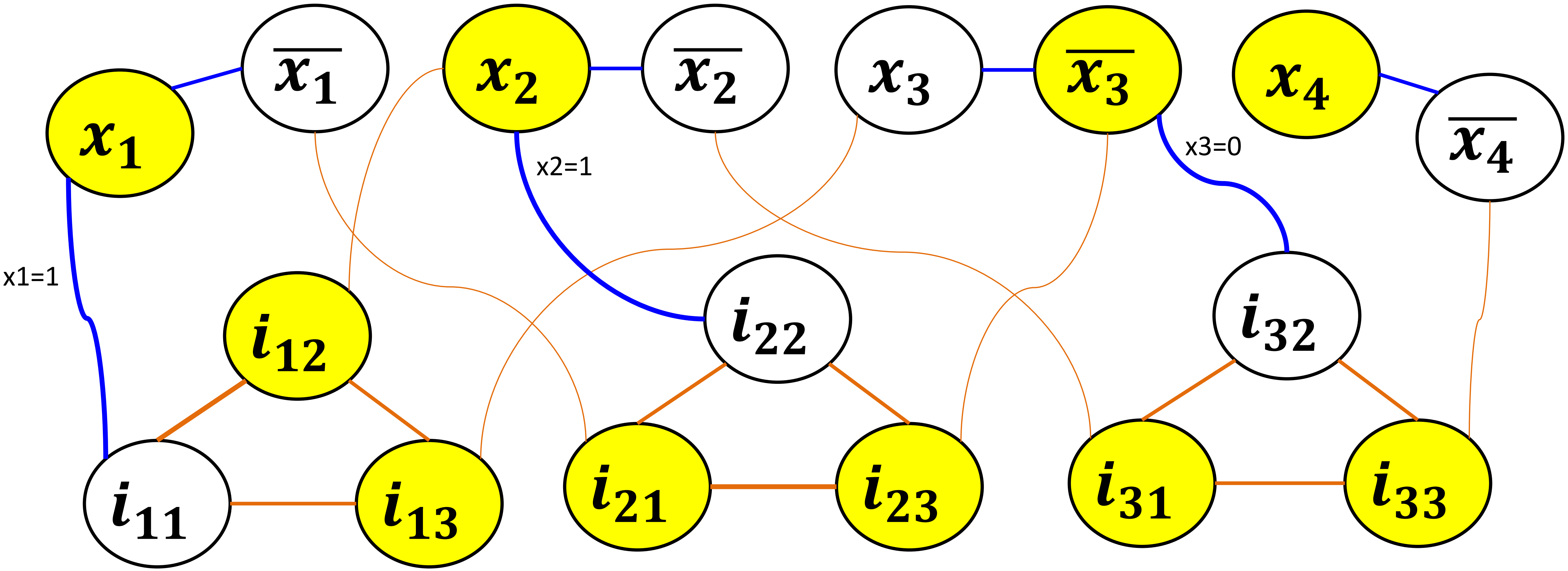
$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



This leaves one edge incident to a clause-satisfying component that is not covered by a vertex in the clause-satisfying component (colored blue).
Hence, each blue edge must be covered by the other endpoint, which is labeled with a literal.
We select this literal node from each pair of literals. This literal node will also cover an edge in the corresponding Truth-setting component

Proof: 4/4

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$

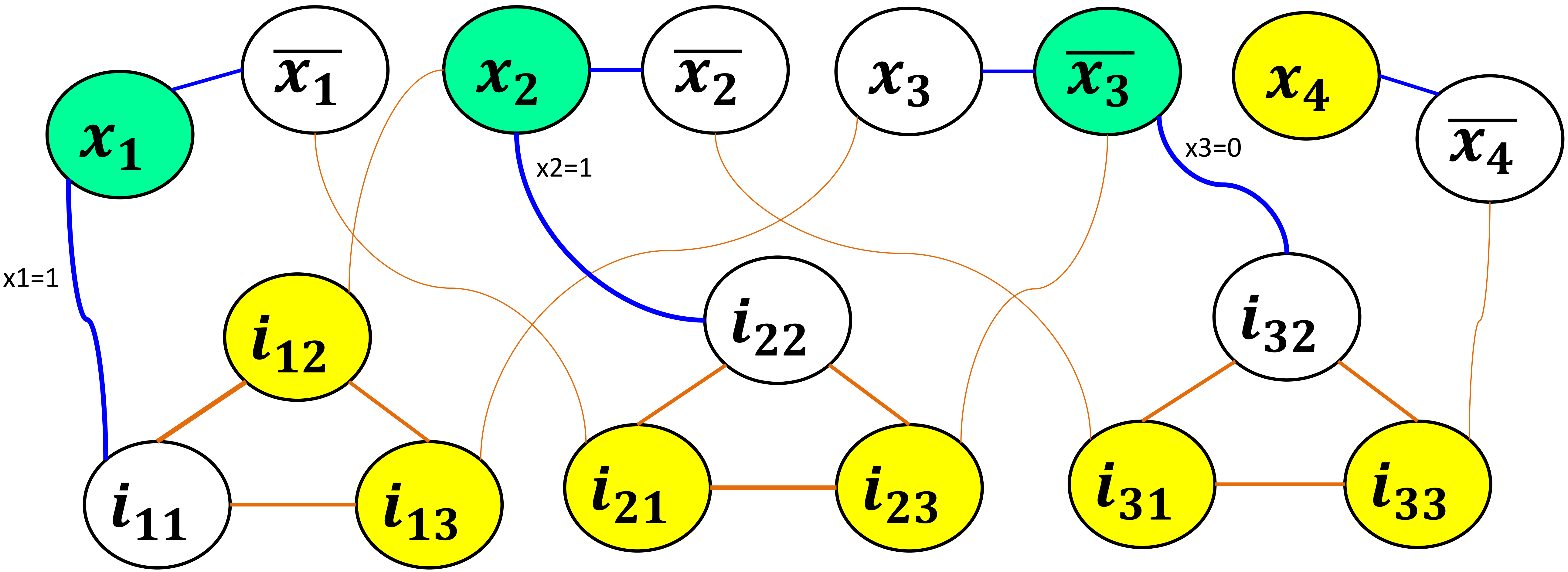


Now all edges are covered by 2m nodes from clause components plus n nodes – one from each truth component.

We constructed a vertex cover of size 2m +n

Proof: 4/4

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$

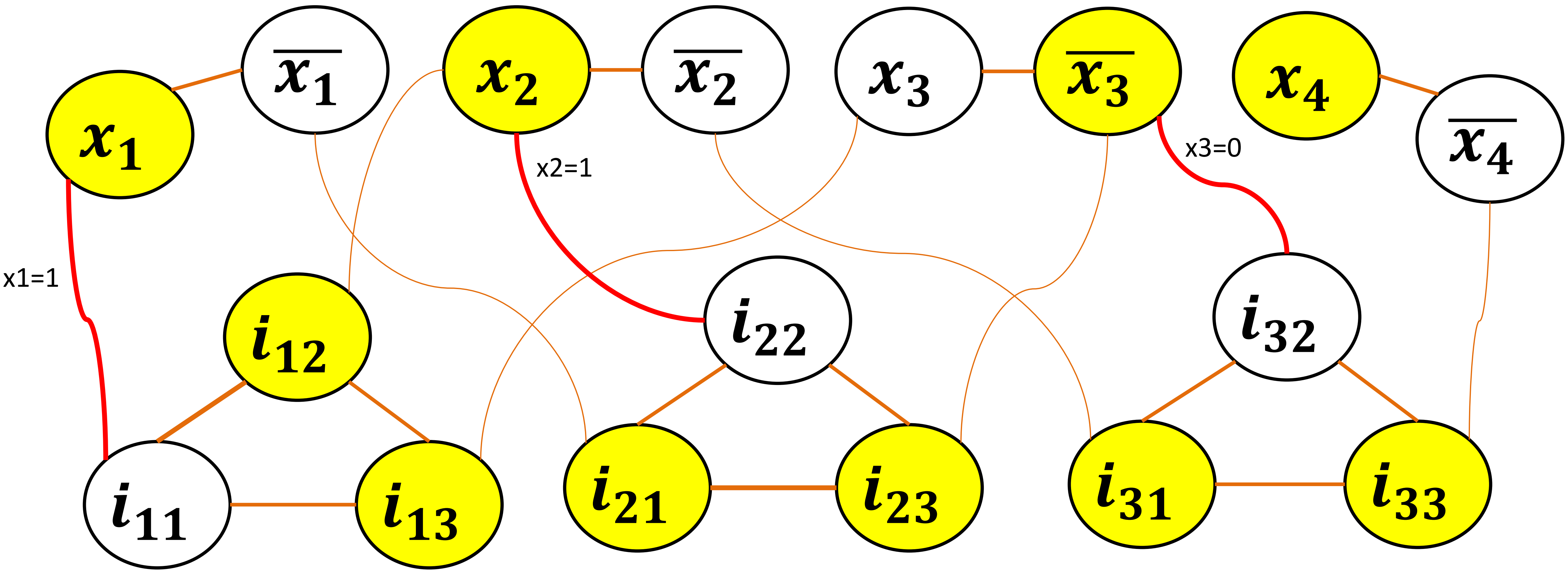


But if we could find a vertex cover in polynomial time for any instance of the problem, then we would also solve a 3SAT problem:
Having this cover, we can now assign the literal in ϕ associated with each green node 1, and each clause in ϕ will be satisfied (because each clause is a disjunction, and it is enough that at least one of the literals is True).

In this example: x1=1, x2=1, x3=0, x4=1. All of ϕ becomes satisfied.

Conclusion

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



Thus, if we can find a vertex cover of size at most k=n+2m in this graph, then we can find a set of variables that satisfy the original 3-SAT formula.

If we knew how to solve vertex-cover, we would be able to solve 3-SAT.
But 3-Sat is NP-complete – that means vertex-cover cannot be solved in polynomial time – or it would make the solution to 3-SAT polynomial too

We have shown that:

1. **Vertex-Cover is in NP**
2. **3-SAT $\leq_p$ Vertex-Cover**

$$\Downarrow$$

VERTEX-COVER is NP-complete

This reduction uses gadgets (components)

Constructing them is a skill which requires a lot of practice.
You will get this practice in the course on the Theory of Computation

Programmer’s Guide to NP-complete problems

- **If you suspect a new problem X is NP-complete, the first step is to look for it in the <u>catalogue of known NP-complete problems.</u>**

- **If it is not there - find there a known NP-complete problem Y similar to X, and prove a reduction showing that a similar known NP-complete problem is reducible to the one you want to solve.**

- **If neither of these works, you probably should continue to try to find an efficient algorithm…**

Dealing with NP-complete problems

Algorithmic approaches to NP-complete problems: <u>video</u>

- **Special case of NP-complete problem may have an efficient solution.** **Maybe the real-life problem you are trying to solve can be modeled differently.**
- **Solve the problem approximately** **instead of exactly. Often it is possible to come up with a provably fast algorithm, that doesn't solve the problem exactly but gives a solution within some error from optimal.**
- **Use an exponential time solution anyway.** **If you really have to solve the problem exactly, you can implement an exponential algorithm. In many cases you can design an exponential algorithm which is still better than the Brute-Force.**

Sample algorithms

The videos are from the second course on Algorithms by Tim Roughgarden (Stanford)

- **Solve the problem approximately.**

<u>Example</u>: approximate solution to knapsack problem using greedy and dynamic programming approaches:

  - <u>Video 1</u>
  - <u>Video 2</u>
  - <u>Video 3</u>
  - <u>Video 4</u>
  - <u>Video 5</u>
  - <u>Video 6</u>

- **Improve exponential-time solution.**

<u>Example</u>: better algorithm for Vertex Cover:

  - <u>Video 1</u>
  - <u>Video 2</u>
  - <u>Video 3</u>