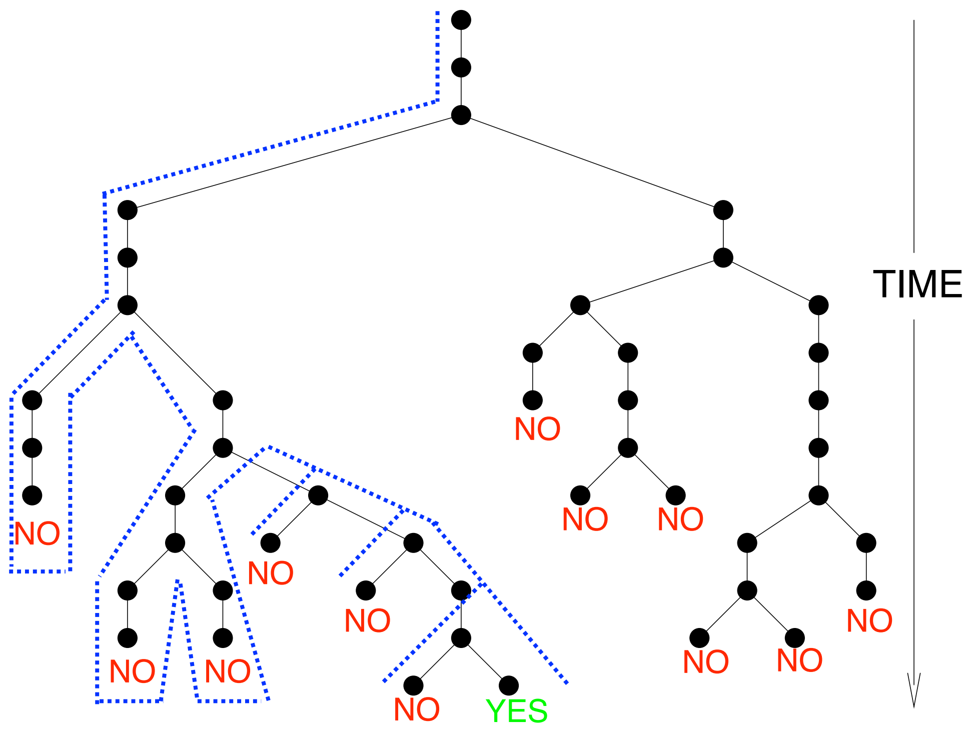
Running time of exact algorithms

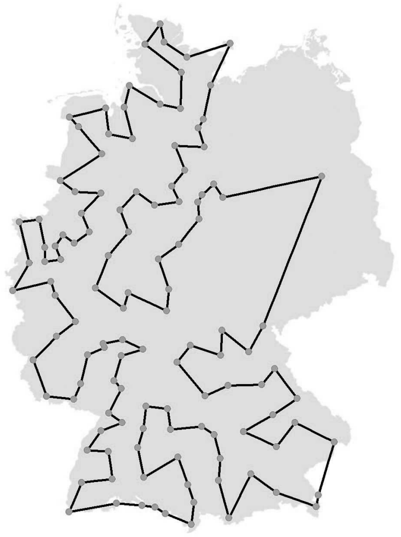


P: *polynomial* $t < \mathcal{P}(N)$

NP : *non-deterministic polynomial (check in polynomial time)*

NP-complete : *hardest in NP*

What if P=NP ?

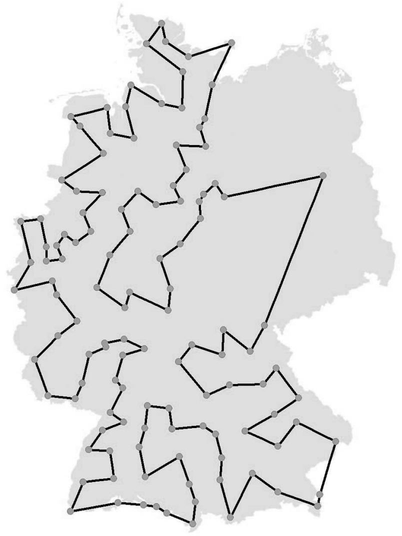


Optimization
shorter tours

Cryptography

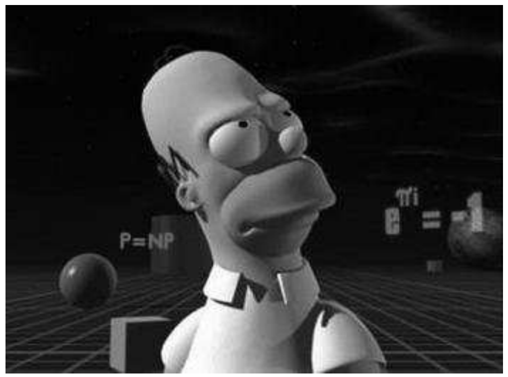
Decrypt: Does encrypted message M correspond to clear text T?

Decrypt $\in$ NP

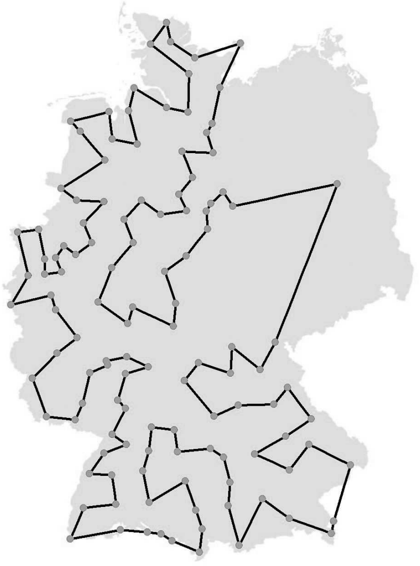


Optimization
shorter tours

Cryptography
disappears



What if P=NP ?



Optimization

shorter tours

Cryptography

disappears

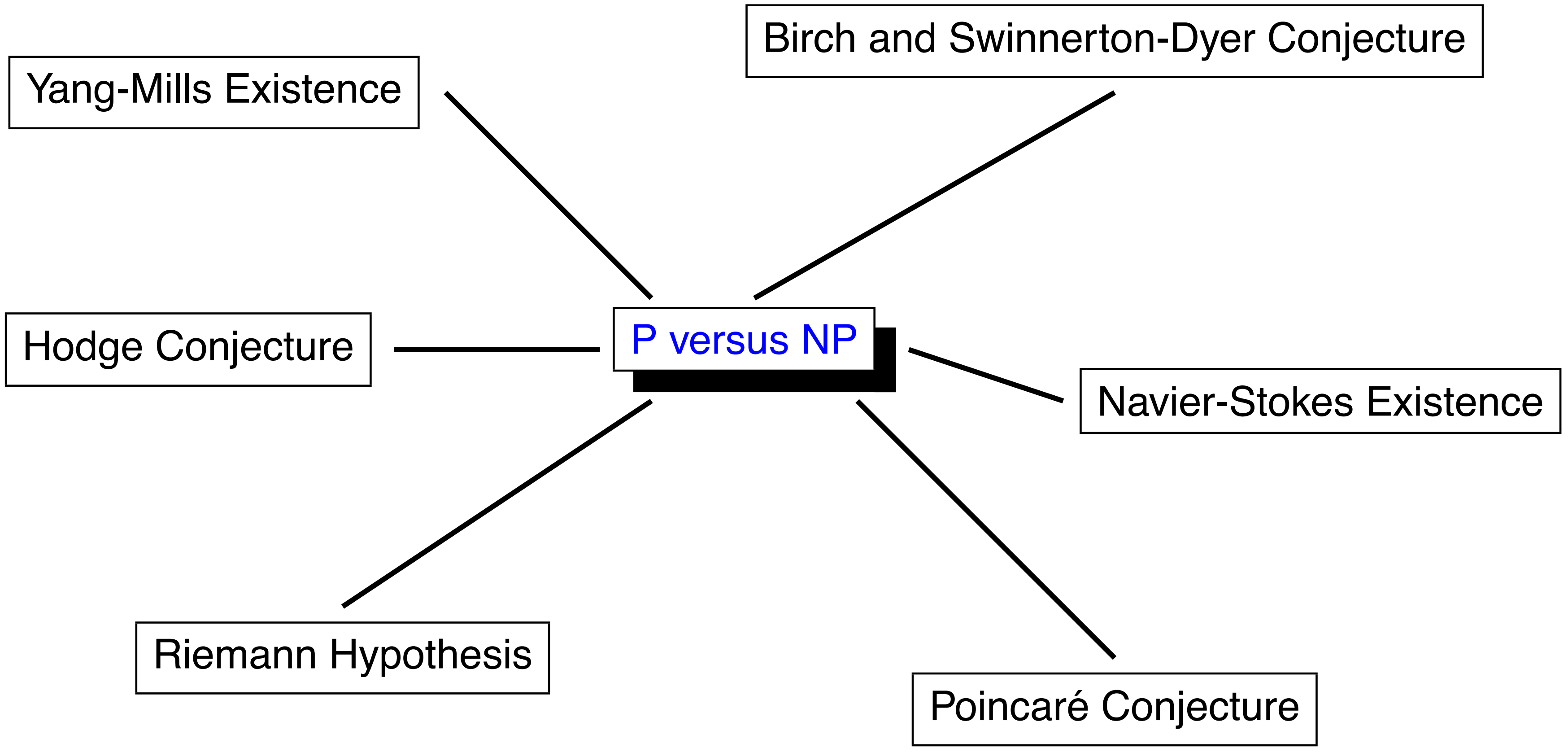
Mathematics

Short-Proof-Existence: Does Theorem T have a proof with less than n lines?

Short-Proof-Existence $\in$ NP

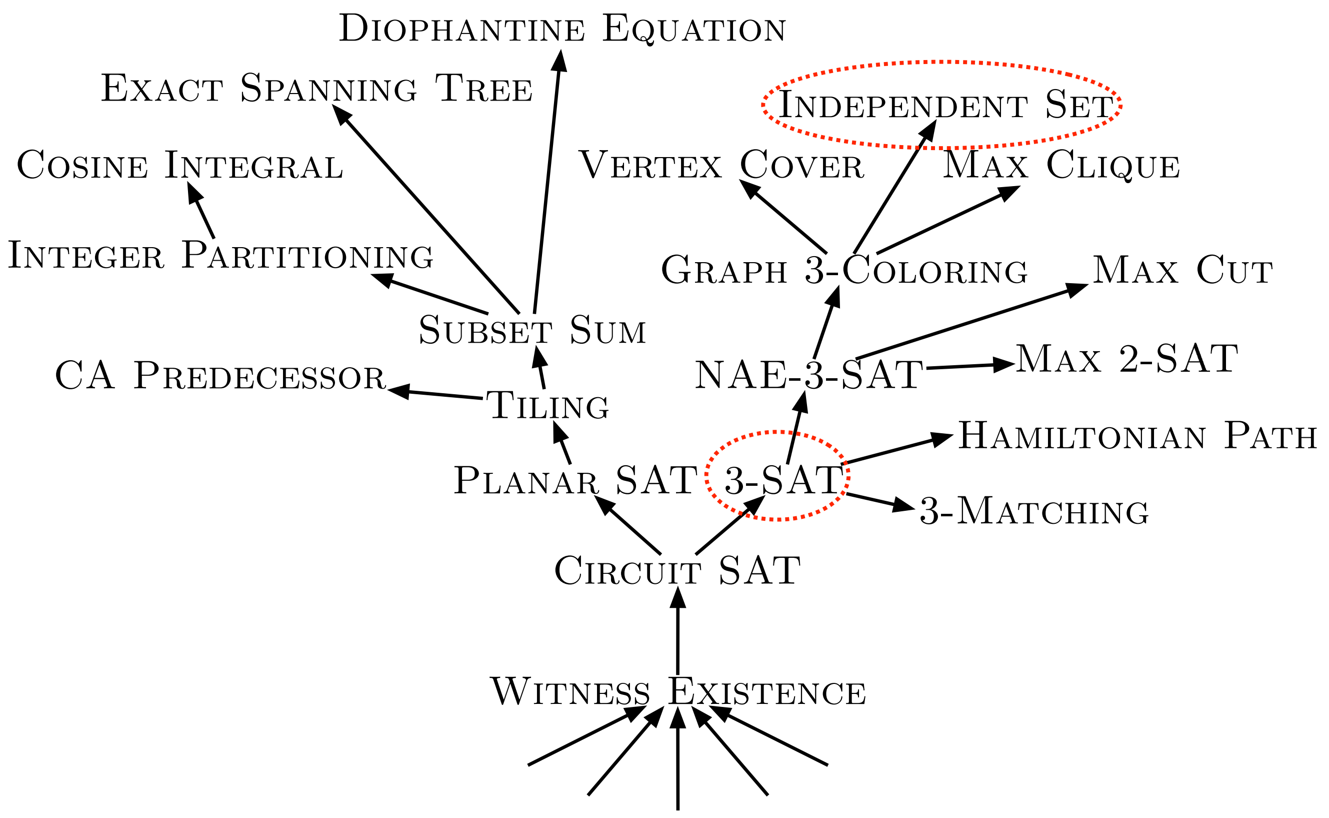
Clay Millenium Problems

P versus NP—a gift to mathematics from computer science
Steve Smale

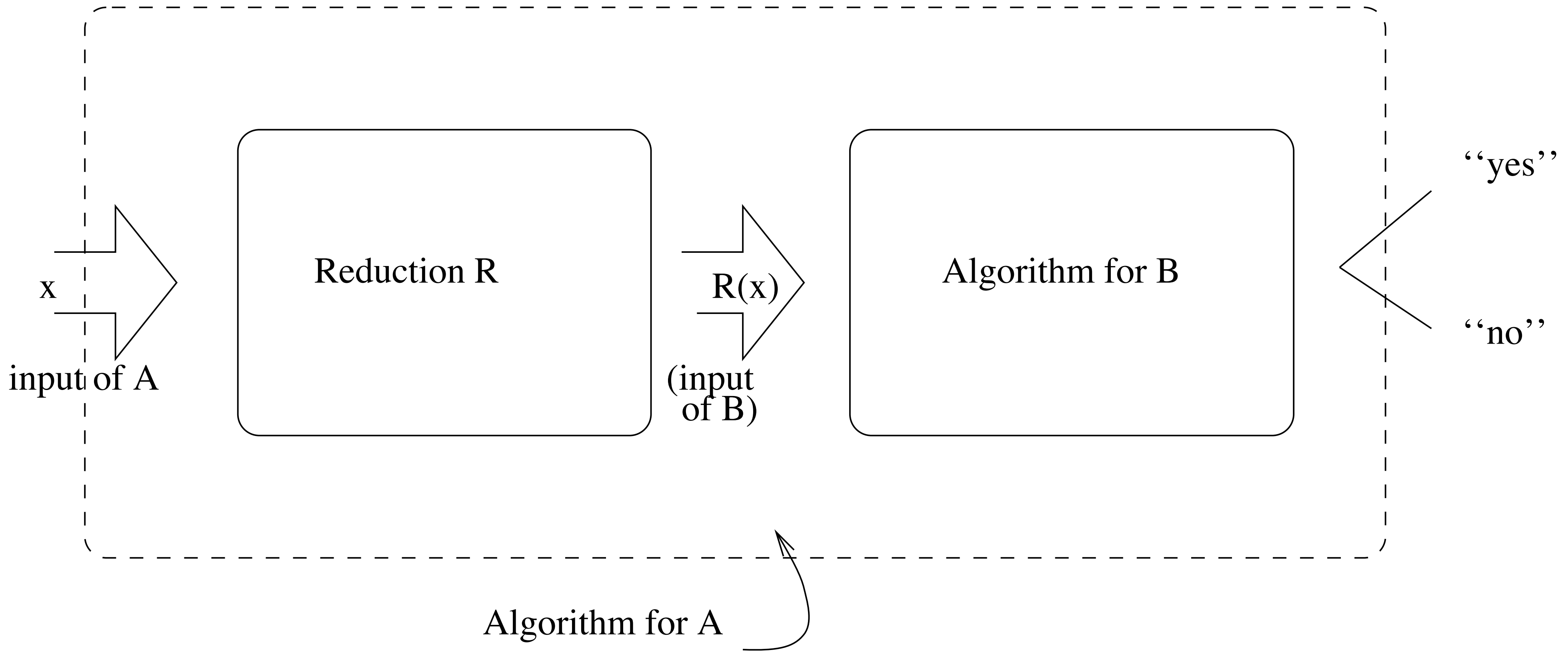


Cook–Levin theorem

In computational complexity theory, the **Cook–Levin theorem, also known as Cook's theorem**, states that the Boolean satisfiability problem is NP-complete. That is, any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of determining whether a Boolean formula is satisfiable.



Reduction of A on B



Reduction theorems:

B is NP-c. If B can be reduced to A then A is NP-c

Composing Reductions

Polynomial time reductions are clearly closed under composition.

By def.:

$$\text{If } L_1 \leq_P L_2 \text{ and } L_2 \in P, \text{ then} L_1 \in P$$

So, if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then we also have $L_1 \leq_P L_3$.

The usefulness of reductions is that they allow us to establish the relative complexity of problems, even when we cannot prove absolute lower bounds.

If we show, for some problem A in NP that

$$\mathrm{SAT} \le_P A$$

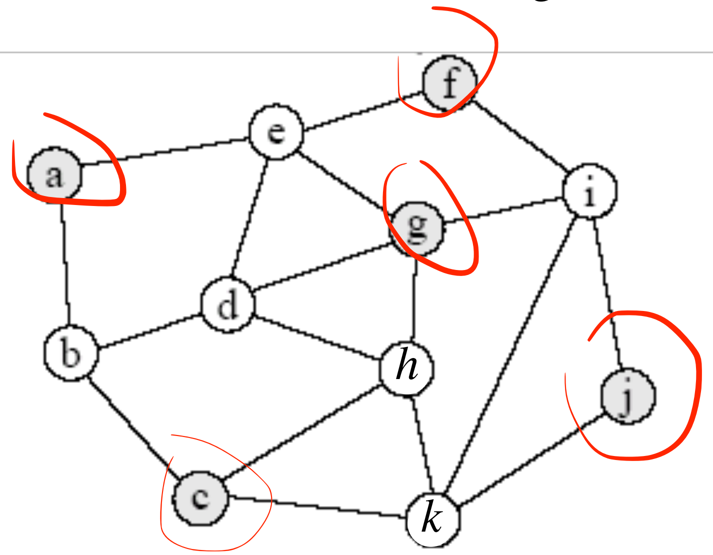
or

$$\mathrm{3SAT} \le_P A$$

It follows that A is also NP-complete.

Independent Set

Given a graph G = (V,E), a subset $X \subseteq V$ of the vertices is said to be an independent set, if there are no edges (u, v) for $u, v \in X$.



{a, d, i, c} is an independent set

{a, c, j, f, g} is a maximal independent set

The natural algorithmic problem is, given a graph, find the largest independent set.

To turn this optimisation problem into a decision problem, we define IND as:

The set of pairs (G,K), where G is a graph, and K is an integer, such that G contains an independent set with K or more vertices.

IND is in NP. We now show it is NP-complete.

Reduction

We can construct a reduction from $\underline{\text{3SAT}}$ to $\underline{\text{IND}}$.

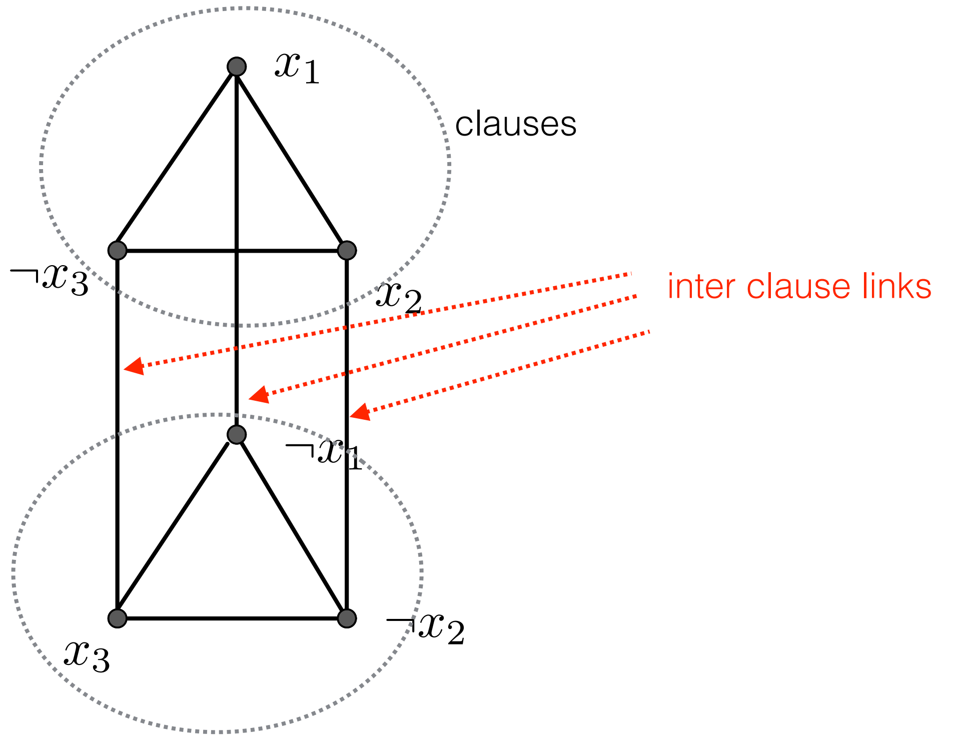
A Boolean expression ϕ in 3CNF with m clauses is mapped by the reduction to the pair (G, m), where G is the graph obtained from ϕ as follows:

G contains m triangles, one for each clause of ϕ, with each node representing one of the literals in the clause.

Additionally, there is an edge between two nodes in different triangles if they represent literals where one is the negation of the other.

Example

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

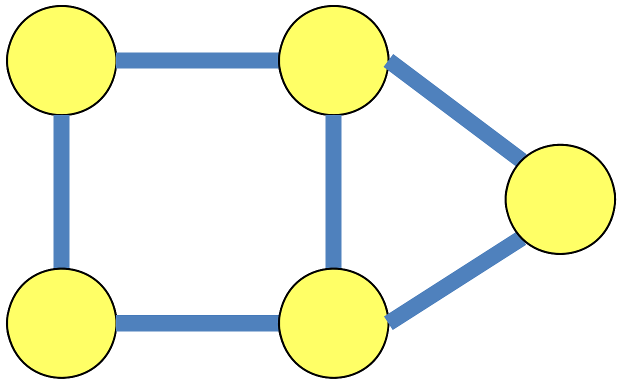


If this graph has and independent set with m (or more) vertices then the such set would provide a solution for the associated 3-SAT problem.

Vertex Cover

- Given an undirected graph $G = (V, E)$
- Find a minimum-cardinality set S of vertices containing at least one endpoint of every edge
  - Equivalently, find a minimum set of guards for a building of corridors, or (unaligned) streets in city

Example:

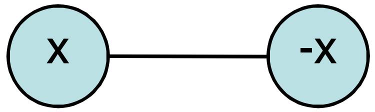


3-sat —> vertex cover

Suppose our boolean formula for 3-SAT, ϕ, contains m variables and l clauses.

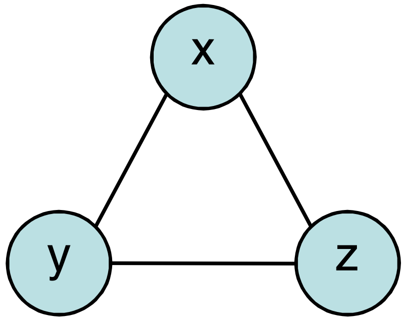
For our vertex cover we will use k = m + 2l.

For each variable in ϕ we will use the following variable gadget:



The variable x in a ϕ for 3-SAT is replaced by two nodes in a graph. One node represents the literal x, while the other represents the literal ¬x.

For each clause in ϕ we will use the clause gadget :



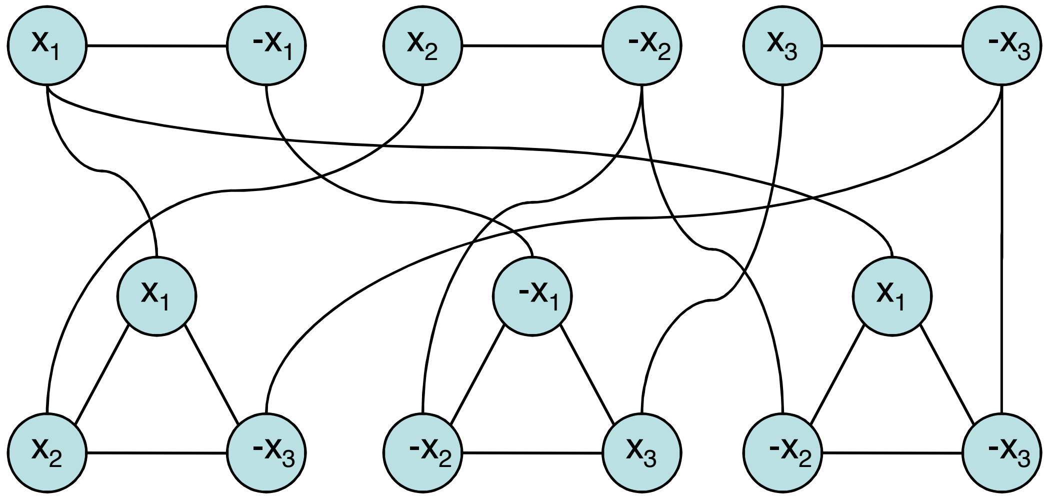
The 3-SAT clause is represented as a clique of 3 nodes, where each node represents a literal in the clause. The clause represented here is $(x \vee y \vee z)$.

Connect the literals in the variable gadgets to the matching literals in the clause gadgets with an edge.

For example, if we have

$\phi = (x1 \vee x2 \vee \neg x3) \wedge (\neg x1 \vee \neg x2 \vee x3) \wedge (x1 \vee \neg x2 \vee \geq x3)$

then we are looking for a vertex cover of size k = 3 + 2(3) = 9 on the graph in following figure:

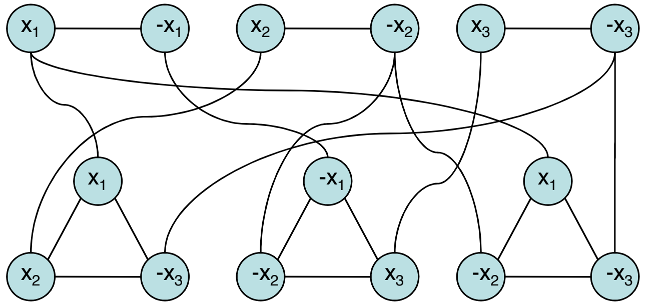


The boolean formula for this example is $\phi = (x1 \vee x2 \vee \neg x3) \wedge (\neg x1 \vee \neg x2 \vee x3) \wedge (x1 \vee \neg x2 \vee \geq x3)$. The variable and clause gadgets for this boolean formula are visible in this graph, as well as the edges connecting nodes from the literals in the variable gadgets to matching literals in the clause gadgets.

At this point the reduction is complete.

We still have to prove that this reduction is correct, **meaning that a vertex cover of size k = m + 2l exists if and only if the given ϕ is satisfiable.**

(1) sat -> cover



1) Suppose we have a satisfying assignment for ϕ.
2) For each variable, add the node corresponding to the true version of the literal for that variable to the vertex cover.
3) Then, for each clause, select one true literal in the clause and add the remaining two literals to the vertex cover.
4) We have used 2 vertices in each clause gadget and 1 vertex in each variable gadget for this cover, which meets the size requirement for the cover.
5) Each edge in a variable gadget is covered by the node selected from that gadget. All three edges in each clause are covered by the two nodes we selected in that clause gadget. One true literal in each clause gadget is left out of the cover, but because it is a true literal it is connected by an edge to the node corresponding to the true literal in a variable clause. **Therefore these nodes cover all of the edges in the graph, and we have a valid vertex cover.**

(2) cover ->sat

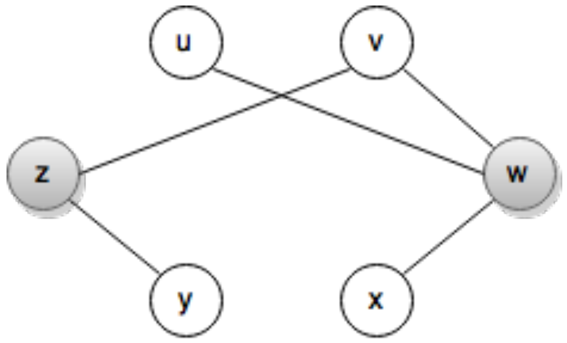
1) We must still prove the other direction of this implication. If we have a vertex cover of size k for this graph, that cover must contain one node in each variable gadget and two nodes in each clause gadget to cover the edges in those gadgets.
2) This requires exactly k = m+2l nodes.
3) Suppose we take literals corresponding to a covered node in a variable gadget to be true.
4) This assignment satisfies ϕ because, **for each clause gadget**, each of the three edges connecting the clause gadget to the variable gadgets is covered, and only two nodes in the clause gadget are in the cover.
5) Therefore one of these clause-gadget edges must be covered by a node in a variable gadget, and so the assignment of that covered variable gadget literal to true in ϕ will satisfy the clause. This holds for every clause gadget and clause, so this assignment satisfies ϕ.

Therefore a k-covering of this graph corresponds to a satisfying assignment for ϕ, while a satisfying assignment for ϕ correspon to a k-covering of this graph, and this reduction is correct.

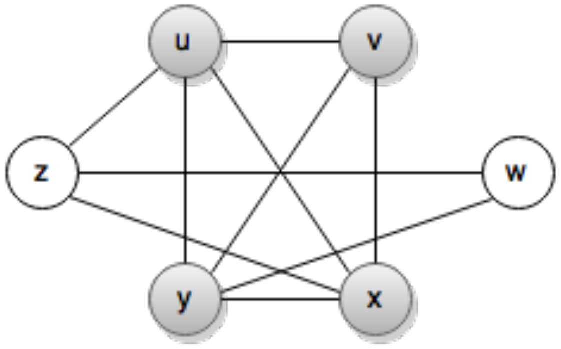
VERTEX-COVER → CLIQUE

Given an undirected graph G = (V,E), we define the **complement** of G as Ḡ = (V,Ē), where Ē = {(u,v) : u,v ∈ V, u≠ v, (u,v) ∉ E}. Essentially, Ḡ has the same set of vertices as G, none of the edges in G, and all the edges that do not exist in G. **The point is: if there exist a clique V ′ in Ḡ, then V – V ′ is a vertex cover in G.** The algorithm for VERTEX-COVER(G, k) by reduction to CLIQUE goes as follows:

1. Given a graph G = (V, E) and integer k
2. Generate the complement graph Ḡ = (V, Ē)
3. Solve the problem CLIQUE(Ḡ, |V | – k)
4. If there is a solution, return Yes, else return No



G = (V,E) with vertex cover V –V′ = {w,z}



complement G with clique V ′ = {u, v, x, y}

To prove this reduction is correct, we need to show two things:

1. If there is a solution to VERTEX-COVER(G, k), then there must be a solution to CLIQUE($\bar{G}$, $|V| - k$)

2. If there is a solution to CLIQUE($\bar{G}$, $|V| - k$), then there must be a solution to VERTEX-COVER(G, k)

1. Suppose that G has a vertex cover $V' \subseteq V$, where $|V|' = k$. Then, for all $u,v \in V$, if $(u,v) \in E$, then $u \in V'$ or $v \in V'$ or both, since the vertex cover must cover all edges. The contrapositive of this is that for all $u,v \in V$, if $u \notin V'$ and $u \in /V'$, then $(u,v) \notin E$ and thus $(u,v) \in E$. This is saying that for any pair of vertices that are both not in the vertex cover V' of G, there is an edge between them in $\bar{G}$. The union of all pairs of vertices that are all not in V' is simply $V - V'$. Thus, $V - V'$ is a clique in $\bar{G}$, by definition, and $V - V'$ has size $|V| - k$.

2. Conversely, suppose that $\bar{G}$ has a clique $V' \subseteq V$, with $|V'| = |V| - k$. Let (u,v) be any edge in E. Then, $(u, v) \notin \bar{E}$, which implies that at least one of u or v does not belong to V', since every pair of vertices in V' is connected by an edge of $\bar{E}$. Consequently, at least one of u or v is in $V - V'$. This causes the edge (u, v) to be covered by $V - V'$. Since we chose (u, v) arbitrarily from E, every edge in E is covered by $V - V'$. Therefore, the set $V - V'$ is a vertex cover of G, with size k.

We have proven that VERTEX-COVER can be reduced to CLIQUE. However, can the reduction be done in polynomial time? Analyse the size of the gadgets!