Proving NP-Completeness

Nikhil Sardana

February 2020

1 SAT

A conjunctive normal form (cnf) **boolean formula** is a conjunction of clauses. Each clause consists of literals (boolean variables or their negations) and OR operators.

From now on, we will refer to a cnf boolean formula as simply a boolean formula.

For example, consider the following boolean formula:

$$\phi_1 = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee \neg x_1 \vee \neg x_2)$$

x_1, x_2, x_3, and x_4 are variables. Each variable can be set to either True or False. A literal is a variable or complement of a variable, so the literals in the above formula are $x_1, x_2, x_3, x_4, \neg x_1$, and $\neg x_2$. $\neg x_1$ is the negation of x_1; when x_1 is set to True, $\neg x_1$ is False. $\vee$ is the OR operator, so the first clause (expression within the first parentheses) reads "x_1 or x_2 or x_3." $\wedge$ is the AND operator. In a boolean formula, clauses are joined by ANDs.

The formula below reads "(x_1 or not x_1 or x_2) and (x_4 or not x_3) and x_2."

$$\phi_2 = (x_1 \vee \neg x_1 \vee x_3) \wedge (x_4 \vee \neg x_3) \wedge (x_2)$$

A boolean formula is **satisfiable** if there exists a variable setting that makes the entire statement true. A variable setting that satisfies a boolean formula is called a *satisfying assignment* for the formula. Because each clause is joined by an AND, and within each clause every literal is joined by ORs, a satisfying assignment must ensure every single clause has at least one true literal.

Example 1.1. A satisfying assignment for the above formula ϕ_2 is

Variable	Assignment
x_1	True
x_2	True
x_3	False
x_4	False

Under this assignment, ϕ_2 becomes

$$\phi_2 = (1 \vee 0 \vee 0) \wedge (0 \vee 1) \wedge (1) = 1 \wedge 1 \wedge 1 = 1$$

where 1 denotes a True assignment and 0 a False assignment.

A formula can have multiple satisfying assignments. It needs only 1 to be considered satisfiable.

Exercise 1.1. Does the following boolean formula have a satisfying assignment? If, so, provide one.

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

Exercise 1.2. Does the following boolean formula have a satisfying assignment? If, so, provide one.

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

It's tough, isn't it! There are a maximum of 2^n variable settings, where n is the number of variables.

Don't worry, it's tough for computers too. We don't have a polynomial time algorithm for solving satisfiability. (Time complexity is always in terms of the input length, which in this case is the length of the formula $|\phi|$, or the number of literals.)

However, we do know that the problem of satisfiability is in NP. If I tell you that a formula is satisfiable, and give you a certificate, i.e. a valid satisfying assignment, you can easily verify in polynomial time that I am correct. For example, a satisfying assignment for our first example $\phi_1 = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee \neg x_1 \vee \neg x_2)$ above is:

Variable	Assignment
x_1	True
x_2	True
x_3	True
x_4	True

Since the number of variables is at most the number of literals $|\phi|$, you can easily verify that each clause has at least one True literal given the above setting. Even if you're being inefficient, and loop over x_1, x_2, x_3 and x_4 for each term in ϕ_1, you still only take $O(n^2)$ time to verify that I have indeed provided you with a correct satisfying assignment, and thus ϕ_1 is in SAT, the language of satisfiable boolean formulas.

$$SAT = \{\phi \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Now, let's prove that SAT is NP-complete.

2 NP-completeness of SAT

Theorem 1 (*Cook-Levin*). *SAT is NP-complete.*

Proof. To prove SAT is NP-complete, we must prove

1. $SAT \in NP$.

2. For all languages $A \in NP$, $A \leq_p SAT$.

We have already proven (1). Now, for part (2).

For some problem A, if $A \leq_p SAT$, there is a polynomial-time computable function $f(w) = \phi$ such that $w \in A \iff \phi \in SAT$.

Coming up with this function is going to be tricky. We don't know much about A. All we know about a generic language $A \in NP$ is the following:

1. A can be verified in polynomial time by a deterministic Turing machine.

2. A can be decided in polynomial time by a nondeterministic Turing machine.

We're going to use the second definition of A to construct our reduction.

For a given $A \in NP$, let N be a nondeterministic Turing machine that decides A in $O(n^k)$ time.

We can write down a single computation state of N using an array. For example, our starting computation state. We begin with only the input $(w_1 w_2 \dots w_n)$ on the tape, and our head starts at state q_0, reading the first character.

q_0	w_1	w_2	$\dots$	w_n

Depending on the transition function, N might move left or right and write different characters. Say N reads w_1, writes w_2, moves right, and transitions to q_4. After this step, our machine can be written as.

w_2	q_4	w_2	$\dots$	w_n

Because N is nondeterministic, it has many branches of computation, so there can be exponentially many different "computation arrays", each representing a different state of N. However, each of these branches has at most length $O(n^k)$, so every one of these "computation arrays" can have at most $O(n^k)$ cells. This is because we can add at most one character to the tape at every step.

Instead of writing down all the computational branches of N for a given input w, let's only write down a single computation path. We can write down a single computation path of N in an $n^k \times n^k$ *tableau.*

#	q_0	w_1	w_2	$\dots$	w_n	$\square$	$\dots$	#
#								#
#								#
#								#

Each row represents a different computation state of N. The first row is the initial state, with only q_0 and the input w on the tape. The second row shows N one step later, and the third one step after that. Since there are at most $O(n^k)$ steps in any computation path, every tableau has at most $O(n^k)$ rows. As we said earlier, each "computation array," or row, has length at most $O(n^k)$, so every tableau is of size $O(n^k) \times O(n^k)$. For simplicity, we'll just say the tableau has size $n^k \times n^k$.

Every cell of the tableau contains a single element of $Q \cup \Gamma \cup \{\#\}$. Q is the set of states $\{q_0, q_1, \dots\}$. Γ is tape alphabet (which contains the alphabet of A). The special # character marks the beginning and end of each row. We will call the symbol at the cell in the ith row and jth column $\mathit{cell}[i, j]$.

We call a tableau *accepting* if it corresponds to an accepting computation path. A tableau is accepting if and only if it contains an accept state q_A somewhere in the final row.

2.1 The Reduction

So, how does this tableau help us with our polynomial-time reduction from A to SAT?

If $w \in A$, then there exists an accepting tableau for N on w. If $w \notin A$, there does not exist an accepting tableau. Our reduction will proceed as follows. We will construct a computable polynomial-time function $f(w) = \phi$ that uses the tableau's information and structure.

ϕ will be a formula that encodes the tableau. The variable settings of ϕ will correspond the possible tableaus for N on w. We want satisfying assignments of ϕ to correspond to accepting tableaus of N on w, and unsatisfying assignments to correspond to rejecting tableaus. That way, there exists a satisfing assignment of ϕ iff $w \in A$.

ϕ needs to encode the contents of the tableau (i.e. the symbols in each cell) as boolean variables. Recall that every cell contains a single element of $Q \cup \Gamma \cup \{\#\}$. So, we create a boolean variable $x_{i,j,s}$ for every single $s \in Q \cup \Gamma \cup \{\#\}$. This gives us $n^k \times n^k \times |Q \cup \Gamma \cup \{\#\}|$ total variables.

We will encode the information $\mathit{cell}[i, j] = s$ as setting the variable $x_{i,j,s} = 1$. We can do this for every single (i, j) cell in the tableau, so collectively, these variables can store the information in the entire tableau.

ϕ also needs to encode the constraints of a tableau. Not every setting of these variables with $x_{n^k,j,q_A} = 1$ corresponds to an accepting tableau—some variable settings will not correspond to a valid tableau at all! (x_{n^k,j,q_A} is a variable corresponding to a cell on the last row.)

For example, if $x_{i,j,q_A} = 1$ and $x_{i,j,t} = 1$ for $q_A, t \in Q \cup \Gamma \cup \{\#\}$, $q_A \neq t$, then this corresponds to $\mathit{cell}[i, j] = q_A$ and $\mathit{cell}[i, j] = t$. However, since $\mathit{cell}[i, j]$ can only contain a single character, this variable setting does not correspond to an accepting tableau.

So, what are the constraints that make any old $n^k \times n^k$ matrix a valid accepting tableau for N on w?

- Each cell contains only elements in the set $Q \cup \Gamma \cup \{\#\}$.
- Each cell contains exactly one element.
- The first row corresponds to a valid starting initial state for N on w (i.e. $q_0 w_1 w_2 w_3 \dots w_n$).
- If the tableau is accepting, the final row contains a cell with the accepting state q_A.
- The contents of row i correspond a state reachable in one Turing machine step from the contents of row $i - 1$.

Our first constraint is handled by the fact that we are only working with variables $x_{i,j,s}$ that correspond to elements in $Q \cup \Gamma \cup \{\#\}$. For simiplicity, let us call $Q \cup \Gamma \cup \{\#\} = C$.

Our formula must be satisfied only when the other four constraints are all satisfied.

$$\phi = \phi_1 \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{transition}$$

ϕ_1 is a formula satisfied when each cell contains exactly one element. Writing this constraint down in a formula results in this complicated-looking expression:

$$\phi_1 = \bigwedge_{1 \leq i,j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t,\in C, s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right]$$

But, this is a fairly simple formula once we break it down:

$$\phi_1 = \underbrace{\bigwedge_{1 \leq i,j \leq n^k} \left[\underbrace{\left(\bigvee_{s \in C} x_{i,j,s} \right)}_{\text{At least one variable true}} \wedge \underbrace{\left(\bigwedge_{s,t,\in C, s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right)}_{\text{No two variables true}} \right]}_{\text{Conditions must hold for every cell}}$$

ϕ_{start} is even simpler. We know our first row has to look like:

#	q_0	w_1	w_2	...	w_n	$\square$	...	#

So, we can just write this explicitly in a formula.

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge \ldots \wedge x_{1,n+1,w_n} \wedge x_{1,n+2,\square} \wedge \ldots \wedge x_{1,n^k,\#}$$

ϕ_{accept} is the simplest of the four. Our only constraint is that q_A appears somewhere in the final row.

$$\phi_{accept} = \bigvee_{1 \leq j \leq n^k} x_{n^k,j,q_A}$$

$\phi_{transition}$ is the hard one. Given a row that corresponds to a valid state of N on w, how do we know the next row represents a valid state of N on w that is reachable from the prior row's state in a single step? This has to be true for our variables to encode a proper tableau, because each row represents a successive step in a computation branch.

Here, we're going to use the specific properties of our nondeterministic Turing machine N. We know N has some transition function. We can use the transition function to determine exactly what makes one row valid, given the previous row.

For example, suppose $Q = \{q_0, q_1\}$, and we have the following transition function:

$$(\text{State}, \text{Read}) \rightarrow (\text{Next State}, \text{Write}, \text{Direction})$$

$$(q_0, 0) \rightarrow (q_1, 0, L)$$

$$(q_0, 1) \rightarrow (q_1, 0, R)$$

$$(q_1, 0) \to (q_0, 1, L)$$
$$(q_1, 1) \to (q_0, 0, L)$$

Then, if we see the following cells in a row of a tableau

...	0	q_1	1	...

We know that directly below these three cells, in a valid tableau, we should see

...	q_0	0	0	...

because we read the 1, write a 0, and move left. On the other hand, in a valid tableau for N on w, we cannot not see

...	0	q_1	1	...
...	0	0	q_1	...

because the transition function does not allow it.

Whatever the transition function for N, every step results in changes at a local level. In fact, it's sufficient for us to loop over every 2×3 set of cells between two rows to determine if the second row is reachable from the first. Most of these 2×3 tiles won't be very interesting, since they aren't near the head, but if we check all the ones near the head, it will tell us if the second row is valid given the first. We're not going to show this rigorously, but you can do the math to prove 2×3 is sufficiently large to prove or disprove correctness. It comes intuitively from the fact that the head can move left or right one cell, and writes to it's right.

Since the transition function of N is finite, the number of possible 2×3 cells is finite. We can make a list and write down all of them, and assign them "valid" or "invalid." Then, we can check over every 2×3 group of cells in the tableau, and check if they are valid. We show this in the following formula:

$$\phi_{transition} = \bigwedge_{\substack{1 \le i \le n^k - 1 \\ 1 \le j \le n^k - 2}} \bigvee_{a_1, \ldots, a_6 \text{ is valid}} (x_{i,j,a_1} \wedge x_{i,j+1,a_2} \wedge x_{i,j+2,a_3} \wedge x_{i+1,j,a_4} \wedge x_{i+1,j+1,a_5} \wedge x_{i+1,j+2,a_5})$$

This formula breaks into two parts. The first part ensures that the inner clause is true for all 2×3 sets of cell. The inner clause ensures that all six cells in a particular 2×3 group correspond to some valid group $a_1, \ldots, a_6$ in the set of all valid 2×3 groups.

$$\phi_{transition} = \underbrace{\bigwedge_{\substack{1 \le i \le n^k - 1 \\ 1 \le j \le n^k - 2}}}_{\text{For all 2x3 cells}} \underbrace{\bigvee_{a_1, \ldots, a_6 \text{ is valid}} (x_{i,j,a_1} \wedge x_{i,j+1,a_2} \wedge x_{i,j+2,a_3} \wedge x_{i+1,j,a_4} \wedge x_{i+1,j+1,a_5} \wedge x_{i+1,j+2,a_5})}_{\text{Cell at (i, j) is valid}}$$

And thus, our final formula

$$\phi = \phi_1 \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{transition}$$

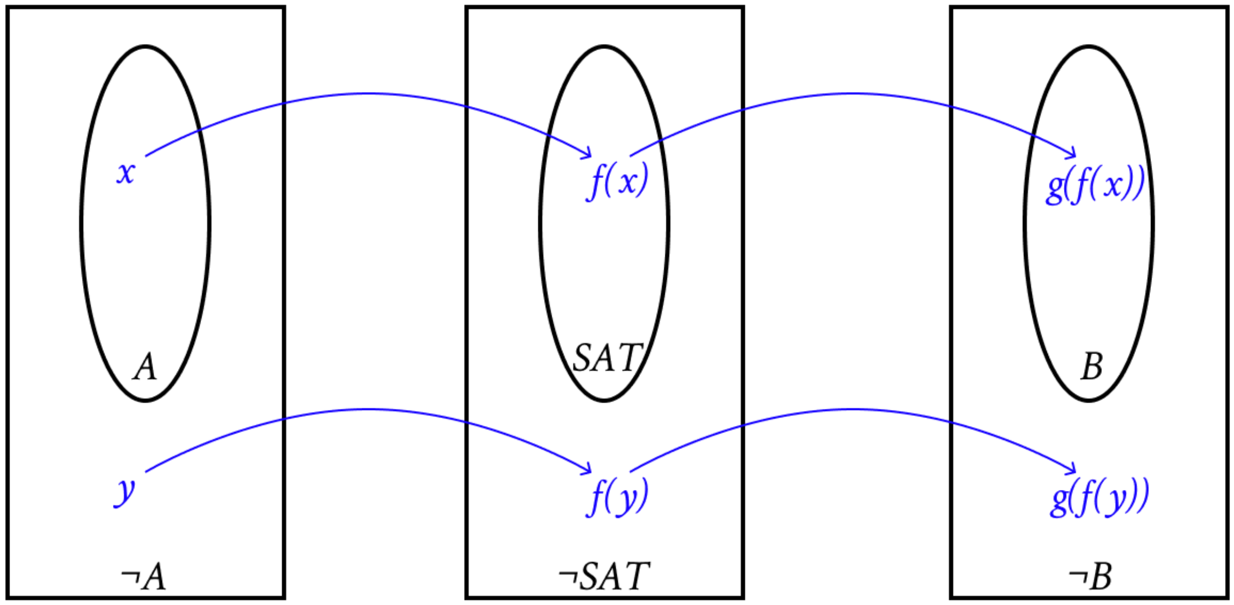
encodes all four constraints, and so, $f(w) = \phi$, where ϕ is the formula above. ϕ is satisfied iff there exists an accepting tableau for N on w, so $\phi \in SAT \iff w \in A$. Thus, $A \le SAT$.

2.2 Polynomial Time

To prove $A \leq_p SAT$, we need to show our formula ϕ is polynomial in length. ϕ_{start} is of size $O(n^k)$, since it encodes a single row. ϕ_{accept} is also length $O(n^k)$, since it looks only at the final row. ϕ_1 looks at every single cell on the table, and does constant computation (the size of C) in each cell. So, $\phi_1 \in O((n^k)^2) = O(n^{2k})$. Finally, $\phi_{transition}$ looks at a constant sized 2×3 grid over the entire tableau, so $\phi_{transition}$ is also in $O(n^{2k})$. Thus, $|\phi| = O(n^k) + O(n^{2k}) + O(n^k) + O(n^{2k}) = O(n^{2k})$. Since our input is of length $|w| = n$, $\phi \in O(n^{2k}) \in P$. Thus, f is a polynomial-time reduction from A to SAT. So, $A \leq_p SAT$. $\square$

3 Reductions to NP-complete Problems

Now, let's prove some other problems are NP-complete. Luckily, we'll never have to do another Cook-Levin style proof again. Now that we've proved that SAT is NP-complete, we can prove another problem B is NP-complete by simply finding a polynomial time reduction to B from SAT. This is because if any $A \in NP$ has a polynomial time reduction $A \leq_p SAT$, and $SAT \leq_p B$, then $A \leq_p B$.



To see why, suppose that $f, g, \in P$ and $x \in A \iff f(x) \in SAT$, and $f(x) \in SAT \iff g(f(x)) \in B$. Then, clearly, $x \in A \iff g(f(x)) \in B$. Since f and g are both in P, $f \circ g \in P$ as well. So, $A \leq_p B$ for all $A \in NP$ if $SAT \leq_p B$.

Let's practice constructing reductions from NP complete problems.

Example 3.1. Claim: $3SAT = \{\phi \mid \phi \text{ is a satisfiable 3cnf formula}\}$ is NP-complete.

A 3cnf formula is boolean formula where every clause has three literals. For example, the formula below is in $3SAT$, because it is satisfied by $x_1 =$ True, $x_2 =$ True, $x_3 =$ True.

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

We'll prove $3SAT$ is NP-complete by constructing a polynomial time reduction to show $SAT \leq_p 3SAT$.

First, we need to show $3SAT \in NP$. This is very similar to the proof of $SAT \in NP$. Given any certificate, or satisfying assignment of a 3cnf-formula, we can in constant time check each clause is satisfiable, and thus in $O(n)$ time check if the formula is in $3SAT$.

Now, to prove the reduction. Consider any satisfiable formula with clauses $C_1, \dots, C_n$.

$$\phi = C_1 \wedge C_2 \wedge C_3 \wedge \dots \wedge C_n$$

If some $C_1 = (x_1 \vee x_2 \vee \dots \vee x_m)$ has more than three literals, we can add a few variables and split this clause into many clauses of length 3, such that the below formula is satsifiable iff the above clause is satsifiable.

$$\tilde{C_1} = (x_1 \vee x_2 \vee z_1) \wedge (\neg z_1 \vee x_3 \vee z_2) \wedge \dots \wedge (z_{m-3} \vee x_{m-1} \vee x_m)$$

We see that whether $z_1, \dots, z_{m-3}$ are true or false, our new $\tilde{C_1}$ won't be satisfiable unless one of $x_1, \dots x_m$ is true, in which case C_1 is satisfiable.

Since we require fewer than m literals to split a clause of size m into equivalent clauses of size 3, this reduction takes polynomial time. Thus,

$$\phi = C_1 \wedge C_2 \wedge C_3 \wedge \dots \wedge C_n \in SAT$$

if and only if

$$\tilde{\phi} = \tilde{C_1} \wedge \tilde{C_2} \wedge \tilde{C_3} \wedge \dots \wedge \tilde{C_n} \in 3SAT.$$

As $f(\phi) = \tilde{\phi} \in P$, $SAT \leq_p 3SAT$. Thus, $3SAT$ is NP-complete.

Let's try a less obvious example.

Example 3.2. $CLIQUE$ is NP complete.

Recall that a (graph, integer) tuple (G, k) is in $CLIQUE$ if and only if G contains a clique (complete subgraph) of k vertices.

We need to show that

- $CLIQUE \in NP$.
- $CLIQUE$ is NP-hard.

We have already shown the first point in earlier lectures. To show the second, we will show the reduction $3SAT \leq_p CLIQUE$. Then, $A \leq_p SAT \leq_p 3SAT \leq_p CLIQUE$ for all $A \in NP$, so we will know $CLIQUE$ is NP-hard.

In other words, we will construct a polynomial time function f such that given a 3cnf formula ϕ with $|\phi| = n$, $f(\phi) = (G, k)$, where G is a graph and k is a number. This function f will be such that if $\phi \in 3SAT$, $(G, k) \in CLIQUE$ (i.e., G has a clique of size k), and if $\phi \notin 3SAT$, $(G, k) \notin CLIQUE$.

$$\phi \in 3SAT \implies f(\phi) = (G, k) \in CLIQUE$$

$$\phi \notin 3SAT \implies f(\phi) = (G, k) \notin CLIQUE$$

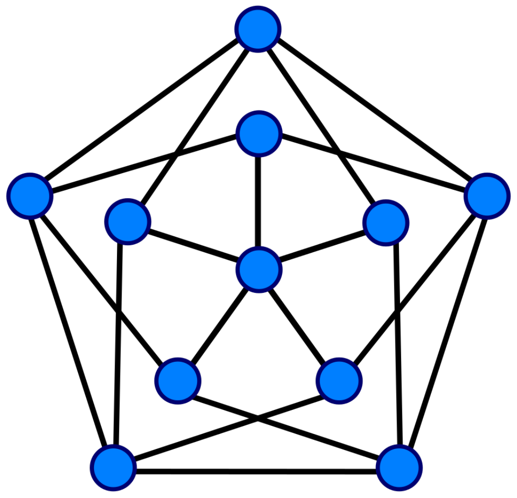


Figure 1: What is the largest clique in this graph?

We can also show this with an "if and only if" statement, which is stronger:

$$\phi \notin 3SAT \iff f(\phi) = (G, k) \notin CLIQUE$$

Our function f from ϕ to (G, k) is as follows:

Say $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ is a boolean 3cnf formula with m clauses. $f(\phi) = (G, k)$, where G is a graph with m groups of 3 vertices, where each literal $x_i \in \phi$ corresponds to a vertex. G contains an edge between all pairs of vertices in different groups, except between pairs $(x_i, \neg x_i)$, for all variables in ϕ. Draw no edges between vertices in the same group. Finally, $k = m$.

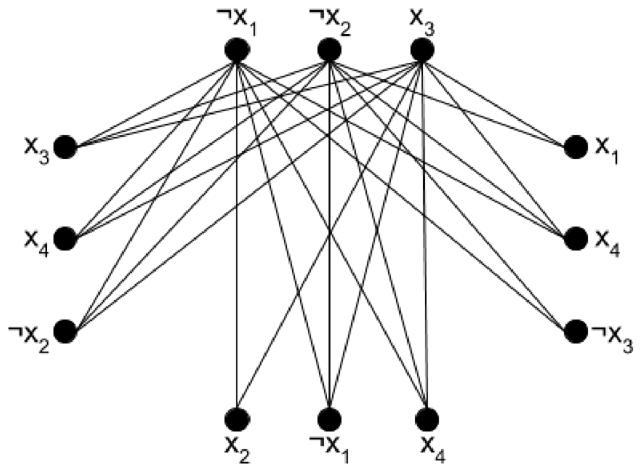


Figure 2: Given the boolean 3cnf formula $\phi = (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee \neg x_3) \wedge (x_2 \vee \neg x_1 \vee x_4) \wedge (x_3 \vee x_4 \vee \neg x_2)$, the above figure shows the corresponding graph $f(\phi) = G$, with only the edges from the first clause shown for clarity. In the true graph G, edges exist from all four clauses.

We claim if $\phi \in 3SAT$, then G has a clique of size $k = m$. If ϕ in $3SAT$, there exists a

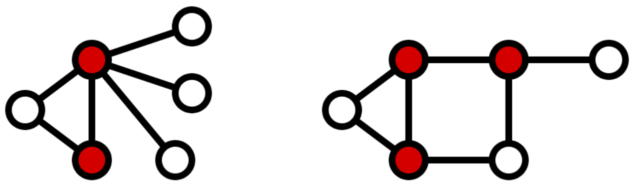


Figure 3: Minimum vertex covers in two graphs.

satisfying assignment, some variable assignment such that at least one literal in every clause C_i is true. Let v_i be a vertex in group i that corresponds to the true literal in C_i for the satisfying assignment. Then, we claim that the set of v_i's form a clique.

We see that if (v_i, v_j) is not an edge for $i \neq j$, then v_i and v_j must be opposite literals since they are in different groups (one must be x_k while the other is $\neg x_k$). But then, both v_i and v_j cannot correspond to true literals in the satisfying assignment. So, they both cannot be in our set. Hence, every pair of vertices in our set of v_i's must have an edge between them. So, our set of v_i's form a clique of size m. So, $\phi \in 3SAT \implies (G, m) \in CLIQUE$.

Now, we will prove the other direction off the "if and only if." We will show $(G, m) \in CLIQUE \implies \phi \in CLIQUE$.

Suppose we are given a graph created by the function f above from a ϕ. We don't yet know if $\phi \in 3SAT$, that is, if ϕ is satisfiable. We claim that if G has an m-clique, then there exists a satisfying assignment of ϕ. If there exists an m-clique in G, then there is a set of m vertices which have edges between every pair. These vertices must all be from different groups, because there are no edges within groups. Hence, there exists a set of m literals, each from a different one of the m groups, and all pairs of literals can be true at the same time. The only way a pair of literals cannot be true at the same time is if the pair is $(x_j, \neg x_j)$, but there are no edges between these pairs in G. But if m literals, each from different clauses can be true all at the same time, then this is a satisfying assignment of ϕ. Hence, if $(G, m) \in CLIQUE \implies \phi \in 3SAT$.

Hence,

$$\phi \notin 3SAT \iff f(\phi) = (G, k) \notin CLIQUE$$

Our function f which converts a formula ϕ to a (graph, integer) tuple (G, k), although not a typical $f(x) = y$ function, is indeed a computable function. In fact, it's also fairly quick—if there are n literals in ϕ, then there are n vertices in G, and a graph with n vertices has at most $\binom{n}{2} \approx n^2$ edges. Hence, $f \in O(n^2)$, which is in polynomial time.

In general, the two steps of proving a problem is NP-complete is figuring out

1. Which well-known NP-complete problem to use in your reduction.

2. How to construct the reduction function.

Example 3.3. Vertex Cover (VC) is NP complete.

Recall that $(G, k) \in VC$ if there exists a set of at most k vertices in G that are adjacent to every edge.

We need to show that

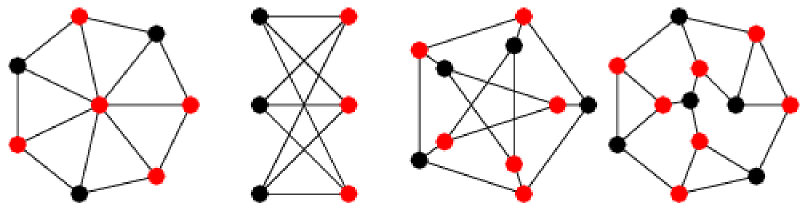


Figure 4: More vertex covers.

- $VC \in NP$.
- VC is NP-hard.

We have already shown the first point in earlier lectures. To show the second, we will show the reduction $CLIQUE \leq_p VC$. Then, $A \leq_p SAT \leq_p 3SAT \leq_p CLIQUE \leq_p VC$ for all $A \in NP$, so we will know VC is NP-hard.

In other words, we will construct a polynomial time function f such that given a graph G with a clique of size m, $f((G, m)) = (G', k)$, where G' is a graph with a vertex cover of size k. If G does not contain an m-clique, then G' does not contain a vertex cover of size k.

Our function is as follows:

$f(G, m) = (G', |V| - m)$, where G' is a graph with the edge complement of G.

Hence, in G', there are no edges between the m clique vertices of G. Clearly, all edges in G' must then touch one of the $|V| - m$ vertices not part of the original clique in G. Hence, G' has a vertex cover of size $|V| - m$. Thus, $(G, m) \in CLIQUE \implies (G', |V| - m) \in VC$.

Now we will show the other direction. Suppose under this construction that G' has a vertex cover of size at most $|V| - m$. This means there is a set of at least m vertices that do not have edges between them. Hence, in the complement of G', or G, there is a clique of m vertices. Thus, $(G', |V| - m) \in VC \implies (G, m) \in CLIQUE$.

Finally, our formula f runs in polynomial time. So, $CLIQUE \leq_p VC$, and thus VC is NP-complete.

Exercise 3.1. Prove IS (independent set) is NP-complete. A (graph, integer) tuple (G, k) is in the language IS if G contains a set of at least k vertices with no edges between any pair.

4 Practice Questions

1. Show Subset-Sum is NP-complete. The Subset-Sum problem is as follows:

   Given a set $S = \{a_1, \ldots, a_n\}$ of positive integers and a positive integer t, is there an $A = \{a_1, a_2, \ldots, a_m\} \subseteq \{1, \ldots, n\}$ such that

   $$t = \sum_{i \in A} a_i$$

   Hint: Show $VC \leq_p$ Subset-Sum.

2. Show HAMPATH $= \{(G, s, t) \mid G$ is a directed graph with a Hamiltonian path from$s$ to $t\}$ is NP-complete. Hint: Show $3SAT \leq_p$ HAMPATH.

3. Show Longest-Path $= \{(G, s, t, k) \mid G$ has a simple path of length $> k$ from s to $t\}$ is NP-complete. Hint: Reduce HAMPATH to Longest-Path.

5 Resources

Next week, we'll begin a completely new topic. I haven't quite decided what we will cover, but it will be much closer to practical algorithms. I hope you enjoyed our survey and dive into computer science theory. If you want to learn more, you can check out these resources.

- Stanford's CS 154 class is a great resource for learning more about Turing machines and theory.
  `https://cs154.stanford.edu`

- "Introduction to the Theory of Computation" by Michael Sipser is a great textbook if you're looking to learn more. You can find the problems online.
  `http://www-math.mit.edu/~sipser/book.html`