NP-Completeness III

Chapter 34

Lecture Overview

- 3SAT, or 3-CNF-SAT
- CLIQUE
- VERTEX-COVER
- Other problems

3-Satisfiability

3-CNF-SAT, or 3-SAT, is the problem:

Instance: A boolean formula Φ in 3-CNF
Question: is there a setting of the variables in Φ that causes Φ to be true?

3-Conjuctive Normal Form is the conjunction (AND) of a finite set of clauses, where each clause is the disjunction (OR) of three variables or variable complements.

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_4 \vee x_5) \wedge \ldots \wedge (\neg x_{2417} \vee \neg x_{4280} \vee \neg x_{1589})$

Observation: a 3-SAT instance is an instance of SAT. Since SAT is in NP, we can conclude that 3-SAT is also in NP.

Transforming Formula Satisfiability

We will show that SAT $\leq_P$ 3-SAT. We transform an instance $I = \Phi$ of SAT to an instance $I' = \Phi_3$ of 3-SAT in 3 steps.

The first step relies on constructing a parse tree of the formula Φ. A parse tree is a tree that has operations (the boolean connectives) as nodes, with the arguments of the operations as children of the nodes. (For more detail, take CMPT 379!). We will allow variables or their complements as leaves of the tree.

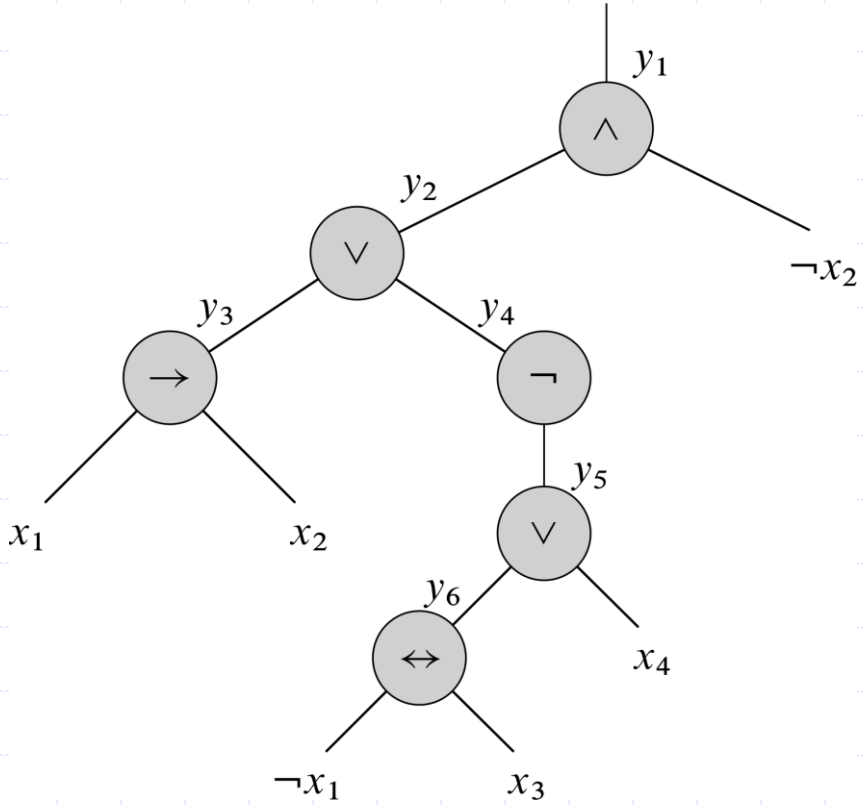
For example, consider the following formula:

$$\Phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

Transforming Formula Satisfiability: Step 1

$$\Phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

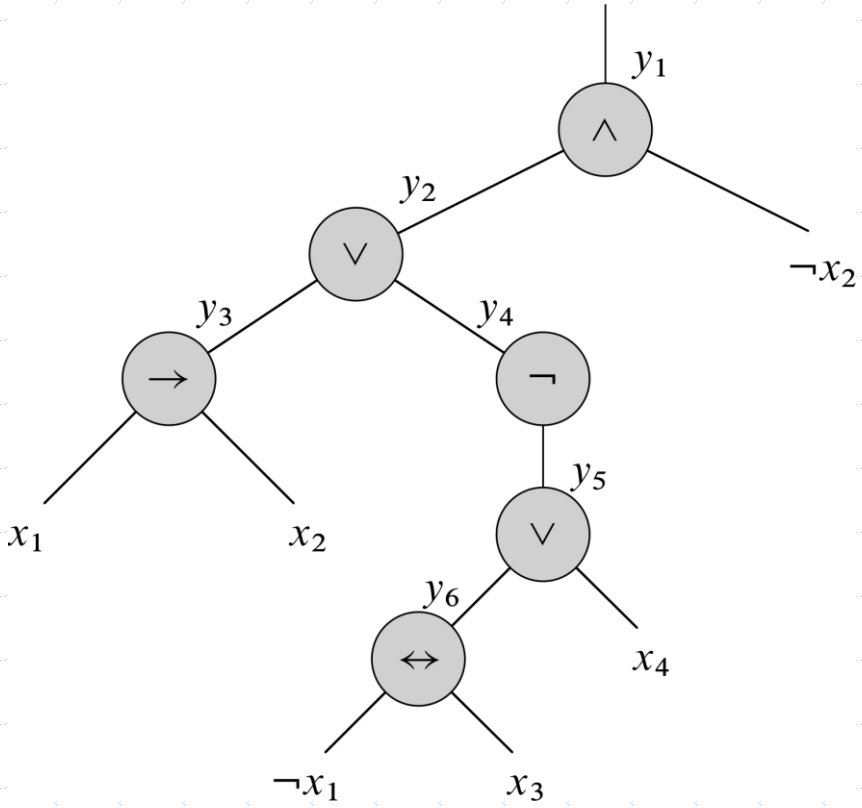
This formula has the parse tree:



We give each result a name consisting of y with a subscript.

Now we will make clauses in the same way as we did when transforming CIRCUIT-SAT to SAT.

Transforming Formula Satisfiability: Step 1



$$\Phi' = y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \wedge (y_4 \leftrightarrow \neg y_5) \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))$$

Because the operations in the tree have at most two operands, the clauses in Φ' each have at most three literals (variable or its complement).

Transforming Formula Satisfiability: Step 2

We will replace each clause in Φ' with an equivalent CNF clause on the same variables.

Let Φ'_i be a clause of Φ' and write out the truth table for Φ'_i.

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

From "0" entries in the final column of the truth table, we can pick out a DNF formula (an OR of ANDs) equivalent to $\neg\Phi'_i$:

$(y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$

To get a CNF formula for the clause Φ'_i, we then just apply DeMorgan's law:

$(\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$

Transforming Formula Satisfiability: Step 3

We let Φ'' denote the result of applying this transformation to each clause. Φ'' is in CNF, and its clauses each have length 1, 2, or 3.

We will add two variables p and q to the formula to construct Φ'''. For each clause Φ''_i:

If Φ''_i has 3 literals then we simply add it to Φ'''.
If Φ''_i has 2 literals then it is $(l_1 \vee l_2)$.
Add the two clauses $(l_1 \vee l_2 \vee p)$ and $(l_1 \vee l_2 \vee \neg p)$ to Φ'''.
If Φ''_i has 1 literal then it is (l_1).
Add the four clauses $(l_1 \vee p \vee q)$, $(l_1 \vee p \vee \neg q)$, $(l_1 \vee \neg p \vee q)$, and $(l_1 \vee \neg p \vee \neg q)$ to Φ'''.

Φ''' is then our desired instance Φ_3 of 3SAT.

3SAT is NP-Complete

We've just shown that SAT can be reduced to 3SAT. Furthermore, each of the three steps is straightforward and can be accomplished in polynomial time:

- Step 1 introduces m variables and m clauses of at most 3 literals each, giving a formula with m+n variables and m clauses.
- Step 2 replaces each clause in the formula from step one with at most 8 clauses (as each truth table is on at most 3 variables and therefore has at most 8 rows).
- Step 3 replaces each clause with at most 4 clauses.

The steps are designed so that the resulting formulae are equivalent to the input formula Φ, so Φ is a yes instance of SAT iff Φ_3 is a yes instance of 3SAT.

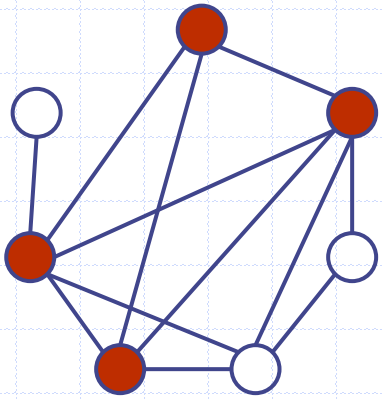
Thus, SAT $\leq_P$ 3SAT, so 3SAT is NP-Hard. As it is in NP, it is also NP-Complete.

Clique

3SAT is a relatively easy problem to transform from. There are a lot of proofs that show 3SAT $\leq_P$ X, for some problem X.

We'll look at the Clique Problem.

Given a graph G, a clique of G is a subset V' of its vertices such that for all distinct x, y $\in$ V', (x, y) $\in$ E. In other words, a clique is a subgraph that is a complete graph.



The size of a clique is the number of vertices it has. The Clique Problem (CLIQUE) is then:

Instance: A graph G and integer k.
Question: Does G have a clique of size k?

Clique

To solve CLIQUE, we can look at all $\binom{n}{k}$ different sets of size k and check each in k^2 time to see if it is a clique. If k is fixed, this is a polynomial ($O(n^k k^2)$). However, if k is not fixed (say it is around n/2, for example) this approach runs in superpolynomial time.

Theorem. CLIQUE is NP-Complete.

Proof. CLIQUE is in NP, as a certificate could consist of a list of k vertices that form a clique in G. Verification would take $O(k^2)$ time.

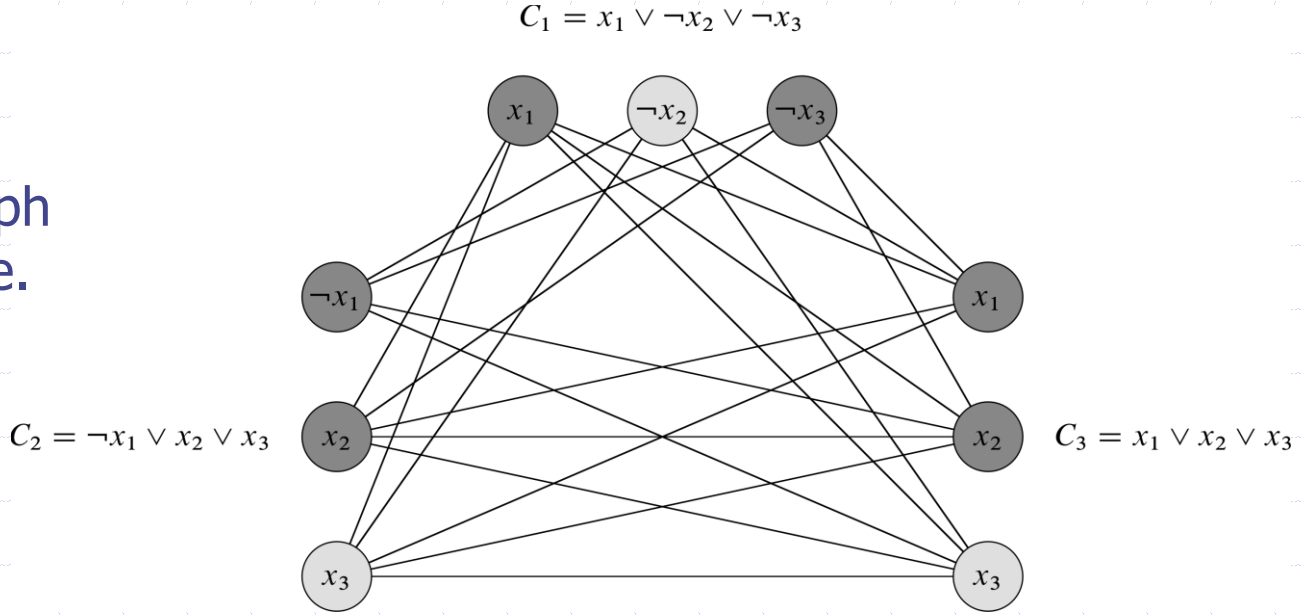
Now we show 3SAT $\leq_P$ CLIQUE. Let I be an instance of 3SAT with formula $\Phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$. Each clause C_r has exactly three distinct literals l_1^r, l_2^r, and l_3^r. We shall construct a graph G such that Φ is satisfiable iff G has a clique of size k.

Clique

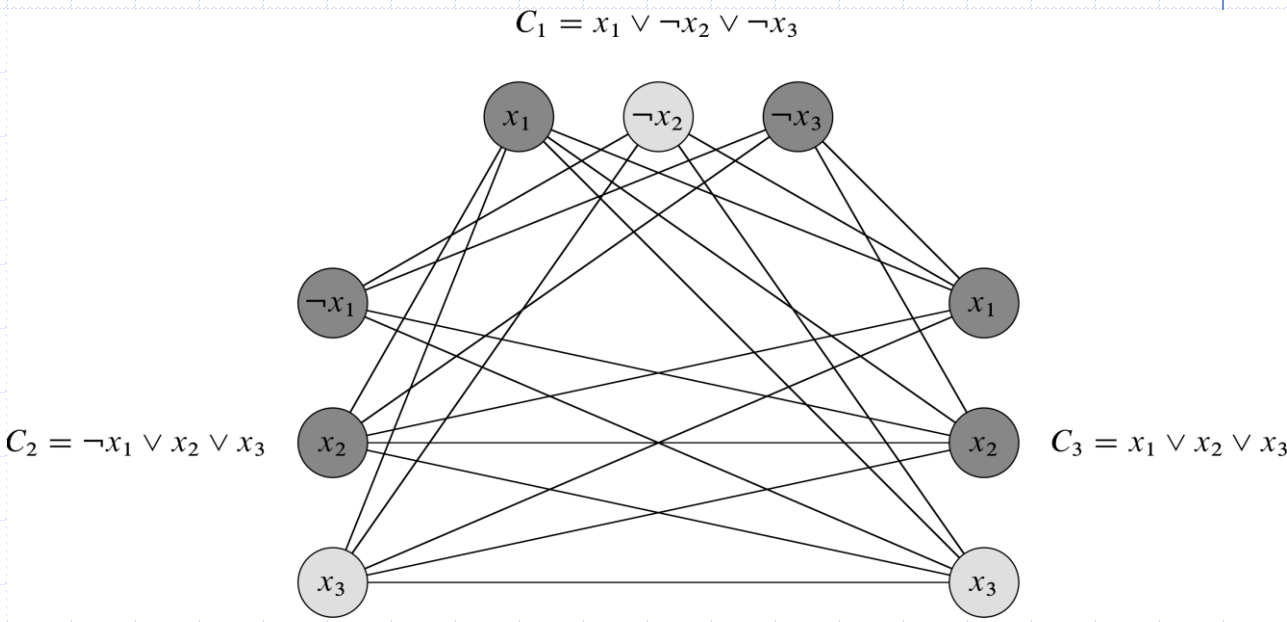
We construct G = (V, E). For each clause $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ we add three vertices v_1^r, v_2^r, and v_3^r to V. We put an edge between two vertices v_i^r and v_j^s if

- v_i^r and v_j^s are in different triples (i.e., $r \neq s$), and
- the corresponding literals l_i^r and l_j^s are consistent – they are not negations of each other.

We can easily construct this graph in polynomial time.



Clique



Suppose Φ has a satisfying assignment. Then each clause has at least one literal assigned to "1". We form a subset V' of V by choosing the vertex corresponding to one such literal for each clause. Then V' must form a clique; any two vertices of V' are in different clauses, and since they were both assigned "1" by the solution of Φ, they must be consistent. V' is of size k, so G has a clique of size k.

Conversely, if G has a clique of size k, it must be one vertex from each set of three. Since no pair of these vertices are inconsistent, we can set the truth assignment of the corresponding variable to "1" (or "0" if the variable is negated). This gives a "1" in each clause.

Clique

So we have just shown 3SAT $\leq_P$ CLIQUE. So CLIQUE is NP-Hard. Because it is also in NP, it is NP-Complete. ■

Note that we've shown an instance of 3SAT can be transformed to a specially-structured instance of CLIQUE. This means both that the specially-structured CLIQUE instances are themselves NP-Hard and CLIQUE in general is NP-hard.

This does not work the other way; we cannot take specially-structured 3SAT instances and transform them. (The instances we choose might be the easy ones in 3SAT.) We have to transform all instances of 3SAT into CLIQUE.

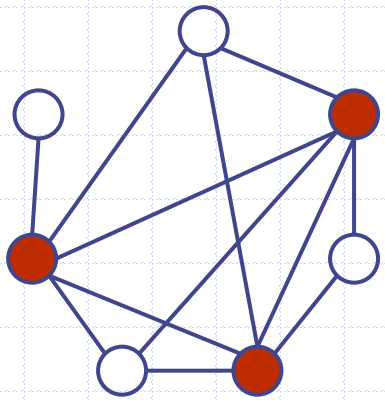
Also note that the reduction used the instance of 3SAT but not its solution. We cannot use the existence (or not) of a solution as an input to the reduction, as this cannot be determined in polynomial time, and the reduction must be accomplished in polynomial time.

Vertex Cover

Given an undirected graph G=(V, E), a vertex cover of G is a subset V' of V such that for all (u, v) in E, either u or v (or both) are in V'. That is, a vertex "covers" all incident edges and the subset V' must cover all edges in E. The size of a vertex cover V' is |V'|.

We define VERTEX-COVER as:

Instance: Graph G and integer k.
Question: Does G have a vertex cover of size k?



Theorem. VERTEX-COVER is NP-Complete.

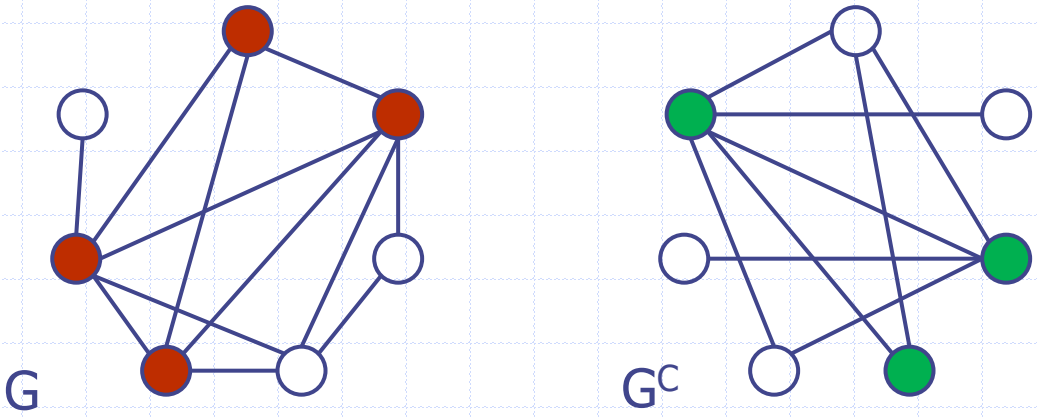
Proof. VERTEX-COVER is in NP; simply use the cover V' as the certificate, and have the verification algorithm check each edge.

...

Vertex Cover

We will show that CLIQUE $\leq_P$ VERTEX-COVER. Given an instance I = (G, k) of CLIQUE, we form an instance I' = (G^C, |V| - k) of VERTEX-COVER.

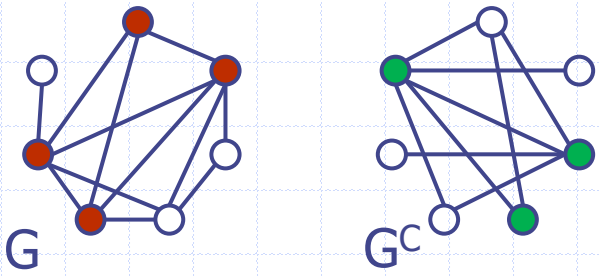
This reduction can easily be done in polynomial time. We need only show that G has a clique of size k iff G^C has a vertex cover of size |V| - k.



Vertex Cover

Suppose G has a clique V' of size k. Let V" = V – V' be all vertices of G not in V'. V" has a size of |V| – k. Furthermore, V" covers all edges of G^C. (Suppose not. Then an edge exists in G^C between two vertices of V', which means there wasn't an edge in G between these two vertices of V', a contradiction.) Thus G^C has a vertex cover of size |V| – k.

Similarly, suppose G^C has a vertex cover V" of size |V| - k; let V' = V – V" and note that V' has size k. V' is also a clique in G, because if it was missing an edge then there would be an edge between vertices of V' in G^C, meaning V" was not a vertex cover. ■



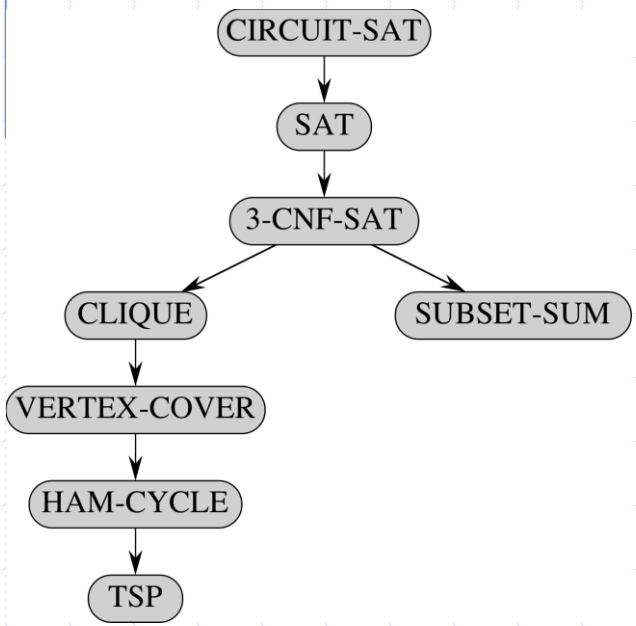
Vertex Cover

Vertex Cover is NP-Complete, but it has an approximation algorithm. This algorithm will not produce a minimum-size vertex cover for a graph, but it will produce a vertex cover that has size at most twice the minimum size.

So NP-Completeness is never the end of the story for a problem. We can always seek polynomial solutions for a restricted version of the problem or an approximation algorithm for the general problem.

Other Problems

Many problems have been proven to be NP-Complete. We've shown but a few, but they are basic problems that are all good candidates for the known NP-Hard problem when we are doing polynomial reductions to prove NP-Hardness / NP-Completeness.



The text shows all the problems on the left are NP-Complete. Please read the remaining proofs there.

HAM-CYCLE we've seen before. TSP is the Travelling Salesman Problem – finding a minimum-length route for a salesman to visit every vertex on a graph. SUBSET-SUM is a problem of finding a subset of a set of numbers that sum to a specific target.