

Revision: NP-completeness

- A language is in *NP* if it can be decided by a non-deterministic algorithm in polynomial time.
- This is equivalent to say that it can be verified by a deterministic algorithm in polynomial time.
- If every language in *NP* is polynomial-time reducible to a language, it is said to be *NP-hard*.
- *NP*-complete languages are those problems in *NP* that are *NP-hard*. One example is *CIRCUIT_SAT*, which we show is *NP* complete using this definition.
- To show other languages are *NP-hard*, we just need to reduce a known *NP*-complete problem to it. E.g. we *CIRCUIT_SAT* is polynomial-time reducible to SAT, so SAT is also *NP*-complete.

ALG/TITCS L06-1

3CNF-SAT

We already know that SAT is *NP*-complete. However, it turns out that even if we remove most of the capabilities of SAT, it is still *NP*-complete.

This is very useful in *NP*-hardness proofs, since then we just need to solve the much simpler problem using the new problem.

Problem 3CNF-SAT

Given a formula which is the conjunction (i.e. AND) of *n* clauses, with each being the disjunction (i.e. OR) of 3 different variables or their negations, determine whether it is satisfiable.

Example: Is $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$ satisfiable?

ALG/TITCS L06-2

Reducing 3CNF-SAT

By the same argument of SAT, 3CNF-SAT is in *NP*. (i.e. guess all variables.)

So it remains to show 3CNF-SAT is *NP-hard*. We will reduce SAT to it.

We will need a construction of 3CNF-SAT instance from a SAT instance, and then the equivalence of the two instances.

Construction: We have an SAT instance.

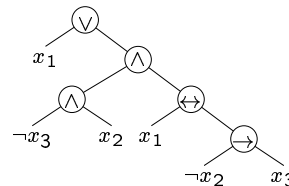
- Turn it to conjunction of clauses each containing at most 3 variables.
- Turn each clause to a conjunction of smaller clauses containing disjunctions of at most 3 terms.
- Pad the clauses so that they contains exactly 3 terms.

ALG/TITCS L06-3

Turning formulae to shallow formulae

Suppose we have a SAT instance, e.g. $(x_1 \vee (\neg x_3 \wedge x_2 \wedge (x_1 \leftrightarrow (\neg x_2 \rightarrow x_3))))$.

First of all, we "understand" the formula by turning it to a binary parse-tree. We can use associative-rules to turn things like $x_1 \vee x_2 \vee x_3$ to $(x_1 \vee x_2) \vee x_3$.



With this, we can create a new variable for the "output" of each operator, just like the reduction for *CIRCUIT_SAT* to SAT.

ALG/TITCS L06-4

Making CNF's from formulae

Now we get something like this. The size is larger, but only at most 3 times. The whole expression becomes very "shallow", like a 3CNF.

$$x_4 \wedge ((x_1 \vee x_5) \leftrightarrow x_4) \wedge ((x_6 \wedge x_7) \leftrightarrow x_5) \wedge ((\neg x_3 \wedge x_2) \leftrightarrow x_6) \wedge ((x_1 \leftrightarrow x_8) \leftrightarrow x_7) \wedge ((\neg x_2 \wedge x_3) \leftrightarrow x_8)$$

Now we use truth tables to convert each term to conjunctions of disjunctions. Since there are at most 3 variables, we get at most 3 terms in each clause. E.g.: for $((x_1 \vee x_5) \leftrightarrow x_4)$:

x_1	x_5	x_4	$\neg((x_1 \vee x_5) \leftrightarrow x_4)$
F	F	F	F
F	F	T	T
F	T	F	T
F	T	T	F
T	F	F	T
T	F	T	F
T	T	F	T
T	T	T	F

$$\neg((x_1 \vee x_5) \leftrightarrow x_4) = (\neg x_1 \wedge \neg x_5 \wedge x_4) \vee (x_1 \wedge \neg x_5 \wedge \neg x_4) \vee (\neg x_1 \wedge x_5 \wedge \neg x_4) \vee (x_1 \wedge x_5 \wedge \neg x_4)$$

$$((x_1 \vee x_5) \leftrightarrow x_4) = (x_1 \vee x_5 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_5 \vee x_4) \wedge (x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee \neg x_4)$$

ALG/TITCS L06-5

Getting a 3CNF

If we do this for all the clauses, we get something really similar to a 3CNF.

$$x_4 \wedge (x_1 \vee x_5 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_5 \vee x_4) \wedge (x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee \neg x_5 \vee x_4) \wedge (x_6 \vee x_7 \vee \neg x_8) \wedge (\neg x_6 \vee \neg x_7 \vee \neg x_8) \wedge (\neg x_6 \vee x_7 \vee \neg x_8) \wedge (\neg x_6 \vee \neg x_7 \vee x_8) \wedge (x_3 \vee x_2 \vee \neg x_6) \wedge (\neg x_3 \vee \neg x_2 \vee x_6) \wedge (\neg x_3 \vee x_2 \vee \neg x_6) \wedge (\neg x_3 \vee \neg x_2 \vee x_6) \wedge (x_1 \vee x_8 \vee x_7) \wedge (\neg x_1 \vee \neg x_8 \vee \neg x_7) \wedge (\neg x_1 \vee x_8 \vee \neg x_7) \wedge (\neg x_1 \vee \neg x_8 \vee x_7) \wedge (x_2 \vee x_3 \vee x_8) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_8) \wedge (\neg x_2 \vee x_3 \vee \neg x_8) \wedge (\neg x_2 \vee \neg x_3 \vee x_8)$$

Apart from the leading term, everything is in 3CNF. But we can always change the leading term to

$$x_4 \vee F = x_4 \vee (x_9 \wedge \neg x_9) \vee (x_{10} \wedge \neg x_{10}) = (x_4 \vee x_9 \vee x_{10}) \wedge (x_4 \vee \neg x_9 \vee \neg x_{10}) \wedge (x_4 \vee \neg x_9 \vee x_{10}) \wedge (x_4 \vee x_9 \vee \neg x_{10})$$

Now we have a 3CNF. Also, it need only polynomial time to compute. The question is, is this 3CNF satisfiable if and only if the original formula is satisfiable? (If so, 3CNF-SAT is *NP*-complete.)

ALG/TITCS L06-6

Equivalence of the 3CNF with the original formula

Suppose the original formula is satisfiable.

- There is a way to assign values to the original variables to make the formula true.
- For that particular set of values, each of the variables have a value, and that must make the 3CNF true. Thus the 3CNF is satisfiable.

Suppose the 3CNF is satisfiable.

- There is a way to assign values to the variables in the 3CNF to make it true.
- We extract the values of the original variables from the 3CNF and put it to the formula. Each subexpression evaluates to the value of the corresponding variable in the 3CNF.
- Since the output variable must be true, the formula is satisfied.

ALG/TITCS L06-7

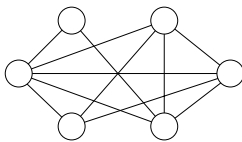
CLIQUE

- All the above problems concern about the satisfiability of something that can either be true or false.
- Now we turn to a problem that does not deal with satisfiability. Instead it is a graph problem.
- A clique of size k in graph G is a complete sub-graph in G . I.e. a set of k vertices so that all are connected by an edge in G .
- Problem CLIQUE:

Given a graph $G = (V, E)$ and a number k , is there any clique of size k in G ?

ALG/TITCS L06-8

Example



- Clearly, it is in NP (we guess k vertices and check that it is a clique.)
- To prove that it is NP -hard, we reduce 3CNF-SAT to CLIQUE.
- I.e. there is a clique of a certain size if and only if the corresponding 3CNF is satisfiable.
- This time it really need some imagination, since the problems are not exactly similar.

ALG/TITCS L06-9

Constructing a CLIQUE problem from a 3CNF: idea

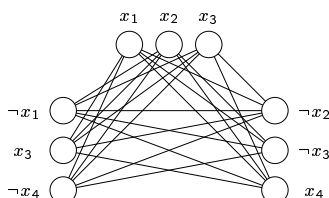
- We represent each of the terms of a 3CNF by a vertice in G .
- Each clause is then represented by a group of 3 vertices.
- In the 3CNF problem, we want all the clauses to have at least one term to be true.
- The only problem: if a variable x_i is true, then $\neg x_i$ cannot be true.
- We can imagine what is the clique: it contains a vertice in each group that is true.
- If two terms cannot be true at the same time, we do not have an edge between the corresponding vertices.

ALG/TITCS L06-10

The actual construction

- For each pair of vertices (v_i, v_j) in two different groups, there is an edge in G if and only if the corresponding terms are not negation of each other.
- The vertices of the same group are never connected.

Example: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$



ALG/TITCS L06-11

The equivalence between the CLIQUE and 3CNF-SAT instances

Claim: for the CLIQUE problem we constructed for a 3CNF-SAT instances with n clauses, there is a clique of size n if and only if the original 3CNF is satisfiable.

Proof:

Part 1: if the 3CNF is satisfiable, then there is a clique of size n .

- If the 3CNF is satisfiable, then there is a way to assign variables so that at least one term of each clause is true.
- Select one such term in each clause, and their corresponding vertices must form a clique of size n in the graph we constructed.

ALG/TITCS L06-12

CLIQUE and 3CNF-SAT instances cont'd

Part 2: suppose the graph has a clique of size n .

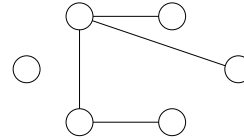
- Note that none of the vertices in the clique comes from the same vertice group, since there is no edges between two nodes in the same group.
- So if we assign true to the corresponding term, we have one true term in each clause. The 3CNF is thus satisfied. The only catch is: is it a valid assignment? I.e., will we assign both x_i and $\neg x_i$ to be true?
- Note that if the vertice corresponding to a term x_i is in the clique, any vertice corresponding to a term $\neg x_i$ is not in the clique.
- Thus we can never assign true to both x_i and $\neg x_i$. The assignment is thus valid, completing the proof that CLIQUE is *NP*-complete.

ALG/TITCS L06-13

Vertice cover

- Not all problems require such complicated analysis to know that it is *NP*-hard. If the new problem is similar enough to one known problem, we just requires a small observation.
- A vertice cover of a graph is a set of vertices, such that all edges are adjacent to at least one vertice.
- Problem VERTICE-COVER:

Given a graph $G = (V, E)$. Is there a vertice cover of at most k nodes?



ALG/TITCS L06-14

VERTICE-COVER is NP-complete

It is clear that VERTICE-COVER is in *NP*, since we can always guess a set of k vertices and then check whether it is a vertice cover (i.e. whether each edge is adjacent to at least one vertice).

Look at the VERTICE-COVER problem in a different angle, we are asked to drop $|V| - k$ vertices so that no pair of them has an edge between them.

But then it is very similar to CLIQUE. CLIQUE asks us to find a set of k vertices such that each pair has an edge between them. VERTICE-COVER asks us to find a set of $|V| - k$ vertices such that each pair has no edge between them.

They are similar enough that a very simple construction suffices.

ALG/TITCS L06-15

VERTICE-COVER is NP-complete (cont'd)

We reduce CLIQUE to VERTICE-COVER as follows. For a CLIQUE instance $G = (V, E)$, k , we define

- $G' = (V, E')$, where $e \in E'$ iff $e \notin E$;
- $k' = |V| - k$.

Clearly, this can be constructed in polynomial time.

From the above discussion, VERTICE-COVER for $G' = (V, E')$, k' asks for a set of $|V| - k' = k$ vertices so that none of them are in E' , i.e. all of them are in E . So this is equivalent to CLIQUE.

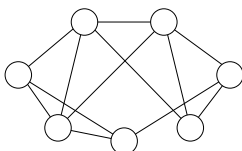
ALG/TITCS L06-16

3-Coloring is NP-complete

However, most *NP*-complete proofs require some clever ideas.

Problem 3-COLORING

Given a graph $G = (V, E)$, is there a function $c(v)$ from V to $\{1, 2, 3\}$, so that any edges is connecting two vertices with different $c(v)$ values?



Again, we can always guess all the colors of the vertices and check it in polynomial time, so the problem is clearly in *NP*.

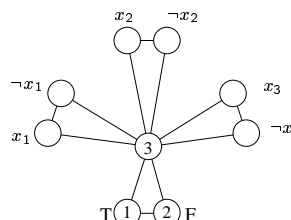
ALG/TITCS L06-17

Represent variables with 3-COLORING

Since this problem is unlike any of the problems that we know, it is best to try reducing 3CNF-SAT to it. I.e., we try to solve 3CNF-SAT with 3-COLORING as a routine.

We need two mechanisms, one to represent a variable or term, and the other to actually connect them to clauses.

For the first mechanism, consider the following "gadget":

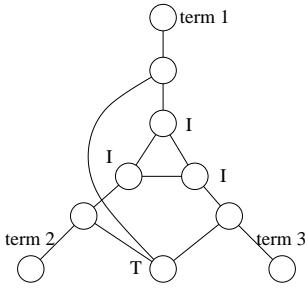


For each variable x_i , there are two vertices representing x_i and $\neg x_i$. Clearly, only one can be colored like the T vertice, and the other must be colored like the F vertice.

ALG/TITCS L06-18

Represent clauses with 3-COLORING

The remaining problem is to represent a clause with 3-COLORING. Basically we want a part to be 3-colorable only if at least one of the terms is the same of color 1. Consider the following "disjunction gadget":



The three *I* vertices must be colored by 3 different colors.

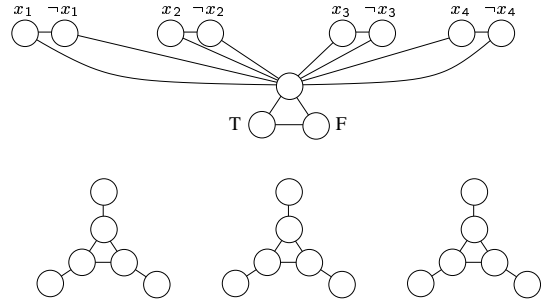
This pose no problem for the colors 1 and 2.

But for color 3, the corresponding term must be of color 1.

So we get the OR of three terms.

Combining gadgets

Example: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$



3-COLORING is NP-complete

This construction takes of course polynomial time.

If the formula is satisfiable, then there is a way to put 1 and 2 into the vertices corresponding to the variables. As discussed above, we can color the disjunction gadgets correctly.

If the graph is 3-colorable, we can look at x_1, x_2, \dots, x_n to find whether they have the same color as *T* or *F*. Thus we have an assignment. As discussed above, each of the "disjunction gadget" ensures that at least one of the terms has the same color as *T*, i.e. is assigned true. Thus the 3CNF is satisfied.

Exercise

- Read textbook section 36.5.4 (Hamiltonian-cycle is NP-complete).
- Prove that if $NP = P$, then all languages in P , except the null language and the universal language, are NP-complete. (Hint: recall the definition.)
- Prove that the following problem, DOMINATING-SET, is NP-complete. (Hint: reduce VERTEX-COVER to it.)

Given an undirected graph $G = (V, E)$ and an integer k , is there a set S of k vertices such that all vertices are either in S or is connected to a vertice in S by an edge?

- Problem 36.1 (a) in the textbook.