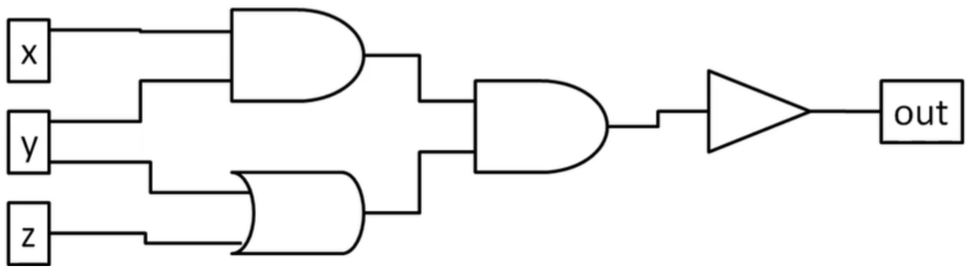
1 NP-Completeness

We continue our discussion on NP-Completeness.

1.1 Problem: Circuit SAT

Problem Statement: Given a circuit C with several Boolean inputs and one Boolean output, determine if there is a set of inputs that give the output `1`.

For example, the following circuit can be satisfied with inputs $x = y = z = 0$.



1.1.1 Important Reduction

Any NP-decision problem can be reduced to Circuit SAT. This is because any NP-decision problem is always of the same form; something is a NP-decision problem if it asks if there is some X that satisfies a polynomial-time checkable property.

Essentially, for some polynomial-time computable function F, is there an X such that $F(X) = 1$? We can create a circuit C that computes F. Then, the problem is equivalent to asking if there is an input for which C outputs 1. So, any NP-decision problem can be encoded as a Circuit SAT problem by taking the function that checks whether or not something satisfies your condition and encoding that as a Boolean circuit.

1.1.2 NP-Complete

Circuit-SAT is our first example of an **NP-Complete** problem. This is a problem in NP that is at *least* as hard as any other problem in NP. Therefore:

- If we can find a polynomial time algorithm that can solve Circuit SAT, we have a polynomial time algorithm for *all* NP problems.
- However, if any problem in NP is hard, then Circuit SAT is *hard*.

Remark: Decision problems can be NP-Complete. For optimization problems, we call them NP-Hard.

1.1.3 Other NP-Complete/Hard Problems

We note that the following problems are all NP-Complete or NP-Hard:

- Formula SAT.
- Maximum Independent Set.
- Traveling Salesman Problem.
- Hamiltonian Cycle.
- Generalized Knapsack.

We can show this by finding reductions from other NP-Hard or NP-Complete problems.

1.1.4 3-SAT

3-SAT is a special case of formula-SAT where the formula is an `AND` of clauses and each clause is an `OR` of at most 3 variables or their negations. The following is an example of a 3-SAT problem:

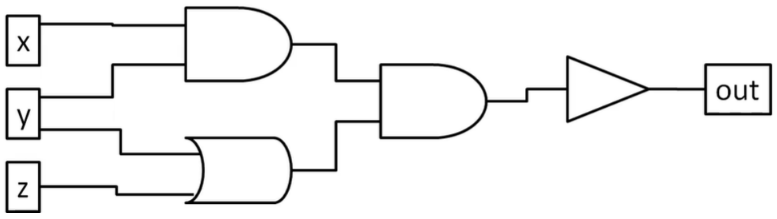
$$(x \vee y \vee z) \wedge (\overline{x} \vee u) \wedge (w \vee \overline{z} \vee u) \wedge (\overline{u} \vee w \vee \overline{z}) \wedge (\overline{y})$$

Here, each clause has *at most three* variables or their negations. Each clause only has `OR` operations, and each clause is separated by `AND` operations.

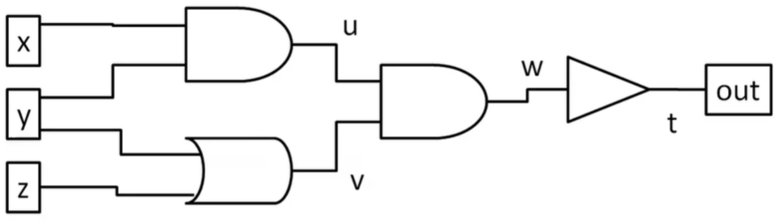
1.1.5 Showing that 3-SAT is NP-Complete

We will show that 3-SAT is NP-Complete. This is done by reducing Circuit-SAT to 3-SAT.

- Let's suppose we start with a circuit from a circuit-SAT problem.



- Next, we'll create new variables for each outgoing wire.



- For each variable, we'll create a formula for each gate and for the output. We know that output t must be true. Then, each clause is defined by its variables and the output after performing some operation (e.g. x and y are the inputs to the `AND` gate and u is the output). Thus, we can write the formula out like so:
$$(v \iff y \vee z) \wedge (u \iff x \wedge y) \wedge (w \iff u \wedge v) \wedge (t \iff \overline{w}) \wedge t$$

- Note that the formula above is *not* a 3-SAT formula. Effectively, we have 3-variable clauses that aren't 3-SAT clauses, so the goal is to write it in terms of them.

To write thes 3-variable clauses in a way that represents 3-SAT clauses, we write out a truth table for each clause. Note that this is just an example of how you would do this.

For example, consider a clause $(z \iff x \vee y)$. There are 8 different possibilities for all combinations of x, y, and z. For each combination of the original statement, we fill out the table like so.

x	y	z	$z \iff x \vee y$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

We note that when we impose a 3-SAT clause on three variables, it excludes exactly *one* combination of the inputs. We now describe how this works specifically. The idea is, given the original truth table of each clause:

- Find all rows where the result is `0`. In our initial example, we have

x	y	z	$z \iff x \vee y$
0	0	1	0
0	1	0	0
1	0	0	0
1	1	0	0

- Then, for each row, negate the correct inputs so that the result would be `0` in the parameters; then, `OR` each of them. So, for example, with the first row, you would have $x \vee y \vee \overline{z}$. This is because x's original value was `0`, so you would use x in the 3-SAT instance to "keep" the `0`. Likewise, since z's original value is `1`, you would use $\overline{z}$ to negate the `1` to a `0`.
- For each of the remaining rows, do the same thing. Combine the clauses with an `AND` operator. For example, with the second row, we would have $x \vee \overline{y} \vee z$. Then, we can combine this with the previous step by using the `AND` operator like so:

$$(x \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee z)$$

Eventually, you'll end up with the 3-SAT-equivalent of the above expression:

$$\underbrace{(x \vee y \vee \overline{z})}_{\text{1st Row}} \wedge \underbrace{(x \vee \overline{y} \vee z)}_{\text{2nd Row}} \wedge \underbrace{(\overline{x} \vee y \vee z)}_{\text{3rd Row}} \wedge \underbrace{(\overline{x} \vee \overline{y} \vee z)}_{\text{4th Row}} = (z \iff x \vee y)$$

- Repeat this process for each of the clauses that needs to be converted.

- This means that 3-SAT is also NP-Complete since any problem in NP can be reduced to Circuit-SAT, which in turn can be reduced to 3-SAT.

1.1.6 Another Look at 3-SAT

Lemma 1.1

A 3-SAT instance is satisfiable if and only if it is possible to select one term from each clause without selecting both a variable and its negation.

For example, suppose we have the 3-SAT instance

$$(x \vee y \vee z) \wedge (\overline{x} \vee y) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee z)$$

For each clause, suppose we select one of the terms like so.

$$(x \vee \boxed{y} \vee z) \wedge (\boxed{\overline{x}} \vee y) \wedge (\overline{y} \vee \boxed{\overline{z}}) \wedge (\boxed{\overline{x}} \vee z)$$

We note that we did not select both a variable and its negation (e.g. we didn't select an x and $\overline{x}$). This shows that this 3-SAT instance is satisfiable. For this example, if x = False, y = True, and z = False, then this statement will be True.

Proof. Suppose the instance was satisfiable. Then, at least one term in each clause must be true since each clause consists of `OR` of some number of terms. We select one such term from each clause. We note that they cannot contradict each other since all of those terms are consistent with the same satisfying assignment (it can't be true that x was true in one clause and $\overline{x}$ was true in another clause because x had to either be true or false).

Suppose we have a 3-SAT instance where we can select one term from each clause without selecting both a variable and its negation. Suppose we set those clauses to be true (for example, we can set x to be True and y to be False or whatever so each clause is true). Then, we set the other variables arbitrarily since we know that at least one variable in each clause is true. This causes the whole statement to be true. □

1.1.7 Reducing 3-SAT to Maximum Independent Set

This reduction is interesting especially since these problems appear to be unrelated. So, we start with a new formulation of 3-SAT. We want to select one term from each clause. We note that:

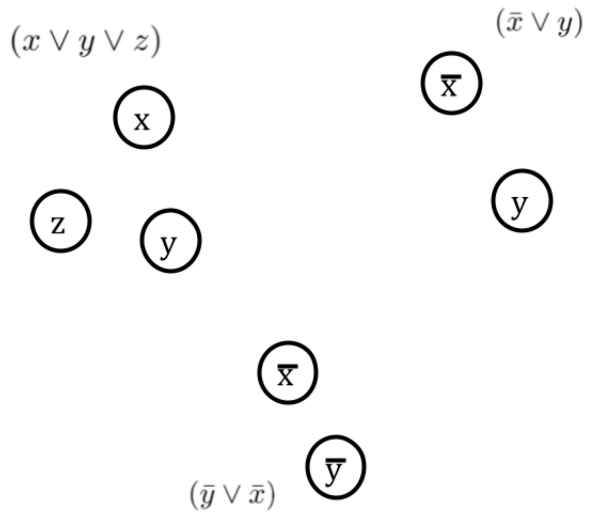
- For 3-SAT, we have a bunch of terms and we want to select some of them.
- For MIS, we have a bunch of vertices and we want to select some of them.

Suppose we want to reduce the following 3-SAT instance to a Maximum Independent Set instance.

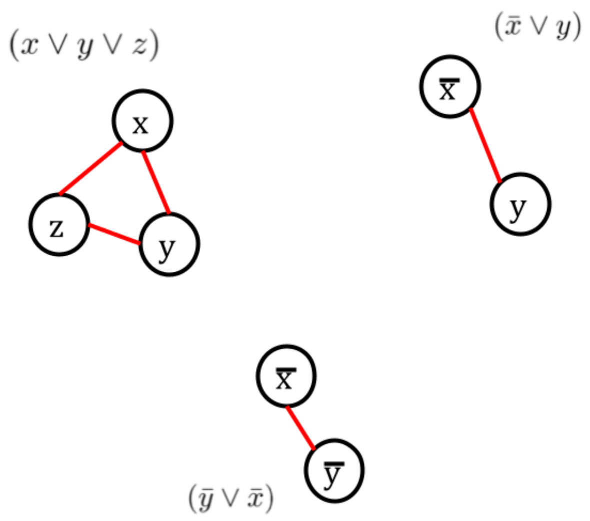
$$(x \vee y \vee z) \wedge (\overline{x} \vee y) \wedge (\overline{y} \vee \overline{x})$$

This is how you would go about it.

- First, for each term in each clause, create a vertex.

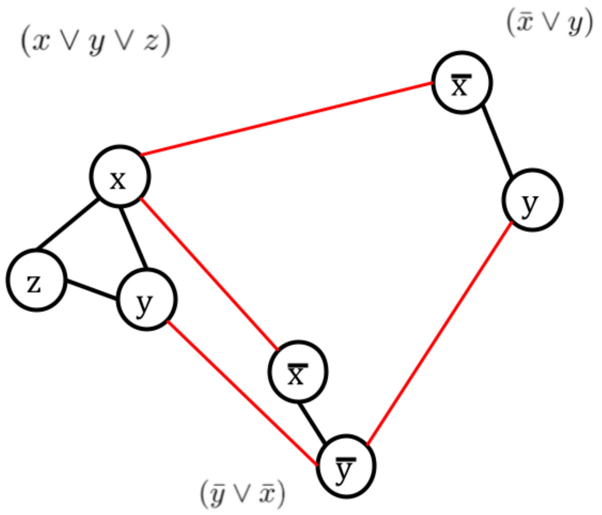


- We now need to add edges in the following way.
  - First, we want to add edges between terms in the same clause.



These edges ensure that, if we pick an independent set, we can only pick one term from each clause.

– We add edges between contradictory terms.



For example, we add an edge between the vertex x and a vertex $\overline{x}$.

This gives us the graph that we can use for the Maximum Independent Set instance.

We note that an independent set in this graph has:

- At most one vertex from each clause.
- No pair of vertices corresponding to contradictory terms.

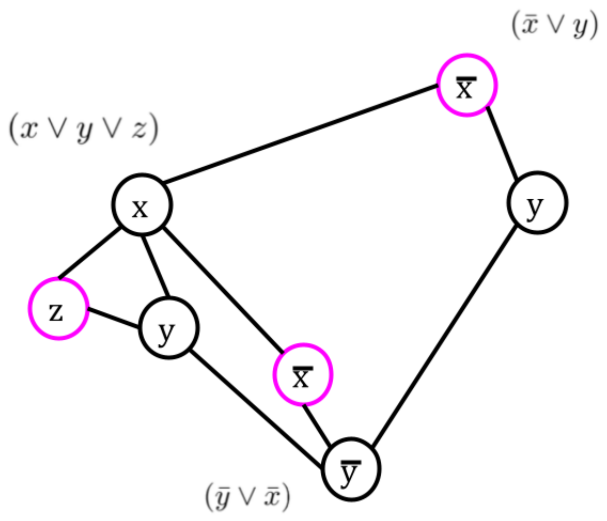
This is because of the edges; an independent set is not allowed to have two vertices connected by an edge. Since two things in the same clause are connected by an edge, it can't have two things in the same clause. And since two contradictory things are connected by an edge, it cannot have two contradictory things. Other than that, it can have whatever vertices it wants.

In particular, can we find an independent set such that its size is the same as the number of clauses in the original 3-SAT expression? If we can, then the only way to do that, since we can't select more than one vertex per clause, is if we pick exactly one vertex per clause. Additionally, because of the second contradict, we had to have done this without any pair that contradicts each other. This is exactly what we need to find a satisfying assignment for the 3-SAT. The opposite direction works as well; if we can satisfy the 3-SAT, we can pick one vertex from each clause and that will give us an independent set of the corresponding graph of the appropriate size.

So, to summarize the previous paragraph, we note that we have an independent set of size c (where c is the number of clauses) if and only if you can select one term from each clause without a contradicton. Therefore, the size of the maximum independent set is equal to the number of clauses if and only if the 3-SAT has a solution.

1.1.8 Example of Reducing 3-SAT to MIS

Consider the graph we created in the previous section. If we select the following vertices (the pink vertices):



Then, as long as $x =$ False, $z =$ True, and y is arbitrary, then we found a satisfying assignment.