Reductions and NP-Completeness – Contd.

$k-CNFSat$:

When $k=1$, the problem is easy: satisfy all literals and if for some variable x, the formula includes also $\bar{x}$, it will not be satisfied. For $k=2$, it is also easy to solve. For $k=3$, the problem becomes hard.

3CNF-SAT:

An input formula: $\phi(x_1,\dots,x_n,\overline{x_1},\dots,\overline{x_n}) = \wedge_{i=1}^{N} C_N$, where the clauses $C_k = (l_1 \vee l_2 \vee l_3)$ where $l_j \in \{x_1,\dots,x_n,\overline{x_1},\dots,\overline{x_n}\}$.

$3CNF \in NP$: a verifying algorithm simply takes a poly-length satisfying assignment $< b_1,\dots,b_n >$, plugs it in ϕ and tests whether the formula is satisfied.

$3CNF \in NPC$:

We will show a reduction $CNF-SAT \leq_P 3CNF-SAT$.

Given $B(x_1,\dots,x_n,\overline{x_1},\dots,\overline{x_n}) = C_1 \wedge \dots \wedge C_i \wedge \dots \wedge C_M$ where $C_i = (l_1 \vee \dots \vee l_k)$. We will create $\phi_B = K_1 \wedge \dots \wedge K_j \wedge \dots \wedge K_N$ where $K_j = (y_1 \vee y_2 \vee y_3)$. We will use $\lambda: C < l_1 \vee l_2 \vee \dots \vee l_k > \rightarrow \lambda(C) \in 3CNF$ – given a clause it creates a formula in 3CNF.

We then define: $\phi_B = \lambda(C_1) \wedge \lambda(C_2 \wedge \dots \wedge \lambda(C_M)$ – the conjunction of 3CNF formulas derives a 3CNF formula.

It is then sufficient that $\lambda(C)$ is satisfyable if and only if C is satisfyable. That would work only when $k \geq 3$.

The reduction λ:

For a given $C = (l_1 \vee l_2 \vee \dots \vee l_m)$ in a formula $B(z_1,\dots,z_n,\overline{z_1},\dots,z_n) = C_1 \wedge \dots \wedge C_M$, we convert it as follows:

$\lambda(C) = (l_1 \vee l_2 \vee x_1) \wedge (\overline{x_1} \vee l_3 \vee x_2) \wedge (\overline{x_2} \vee l_4 \vee x_3) \wedge \dots \wedge (x_{m-3} \vee l_{m-1} \vee l_m)$, where $x_1,\dots,x_{m-3}$ are new variables.

We claim that C is satisfyable iff $\lambda(C)$ is satisfyable:

Assume C is satisfyable, then at least one of $l_1,\dots,l_m$ is TRUE (and the other could be FALSE). W.L.O.G. assume $l_m = T$. We will assign the other variables such that all clauses are satisfied. In this case, assign $x_1 = T$ to satisfy the first clause; in the second clause $\overline{x_1}$, forcing use to assign $x_2 = T$. This goes all the way to the last clause, and so all clauses are satisfied.

Assume C is NOT satisfyable, then $l_1,\dots,l_m = F$. In this case we must assign $x_1 = T$ to satisfy the first clause, which forces us to assign all other $x_i = T$. But then the last clause is $(\overline{x_{m-3}} \vee l_{m-1} \vee l_m)$, and literal in it ends up being FALSE, thus $\lambda(C)$ is also not satisfyable.

The reduction is polynomial, as every (almost) literal maps to a 3-variable clause, and we the number of added variables per clause is $2m$, thus the total size / time is: $O(m \cdot M) + O(m \cdot M)$, which is polynomial in the input size. Therefore we have proven that $3CNF-SAT \in NP-Hard$, thus it is in NPC.

2CNF-SAT:

We will show that $2CNF-SAT \in P$. For a formula in $2CNF$ every clause has 2 variables. Notice that:

$(l_1 \vee l_2)$ is satisfyable iff $(\overline{l_1} \Rightarrow l_2) \vee (\overline{l_2} \Rightarrow l_1)$ is satisfyable.

$\phi(x_1,\dots,x_n,\overline{x_1},\dots,\overline{x_n}) = \wedge_{i=1}^{M} C_i$ where $C_i = (\overline{l_1} \Rightarrow l_2) \vee (\overline{l_2} \Rightarrow l_1)$. We will create a graph with every variable as a node, and every $\Rightarrow$ will correspond to a directed edge in the graph. This construct is polynomial time, as it has $2N$ vertices and $2N$ edges. When will the formula not be satisfyable? IF the graph contains a cycle with both some variable and its complement.

So after constructing the graph we will identify the SCC graph, and make sure we have more than 1 component (otherwise it's one cycle with all variables and their complements) and that every component doesn't contain both a variable and its complement. Since the SCC graph is a DAG we can satisfy it – we find a topological sort, satisfy the last component and assign the rest accordingly. The algorithm is actually linear, which is polynomial.

K-Colorability:

An input of the problem: set of colors $C = \{c_1, \dots, c_k\}$ and an undirected graph $G = (V, E)$. We say G is k-colorable if there exists a map $X: V \to C$ such that $\forall (u, v) \in E: X(u) \neq X(v)$.

An easy problem is 2COL: a graph is 2-colorable iff a graph is bipartite (and that's easily verified by looking for odd cycles).

K-COL is NP: given a coloring, we color every edge and check the condition is satisfied. That's actually linear.

We will show $k - CNF - SAT \leq_P 3COL$, thus showing it is $NP - hard$ (and thus NPC).

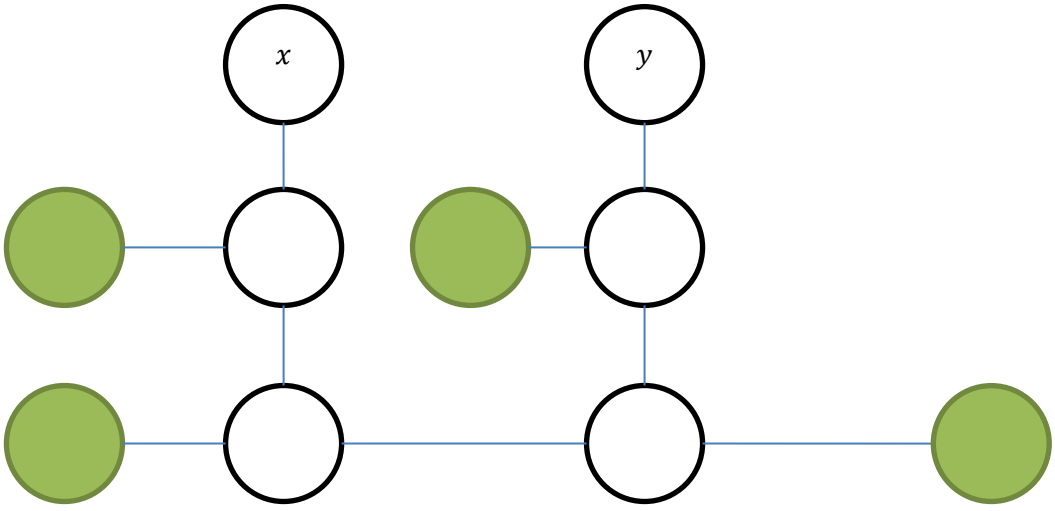
Given a formula $\phi \in k - CNF$ we will create a pair $< G, C >$ of a graph and set of colors such that $\phi \in k - CNF - SAT \Leftrightarrow < G, C > \in 3COL$. To ensure we will need 3 colors, our graph will contain a triangle (odd cycle of size 3), such that it requires 3 colors. We will map TRUE = GREEN and FALSE = RED.

We create a vertex $v \in V$ for each $x_1, \dots, x_n, \overline{x_1}, \dots, \overline{x_n}$. To make sure x_i and $\overline{x_i}$ won't get the same color, we connect them. In addition, we connect every one of them to the dummy BLUE node from the triangle. This way every one of $x_i, \overline{x_i}$ must be either RED or GREEN, and no variable and its complement can have the same coloring.

In addition, for every clause we create a gadget in the graph such that the clause is satisfyable iff the gadget is 3-colorable.

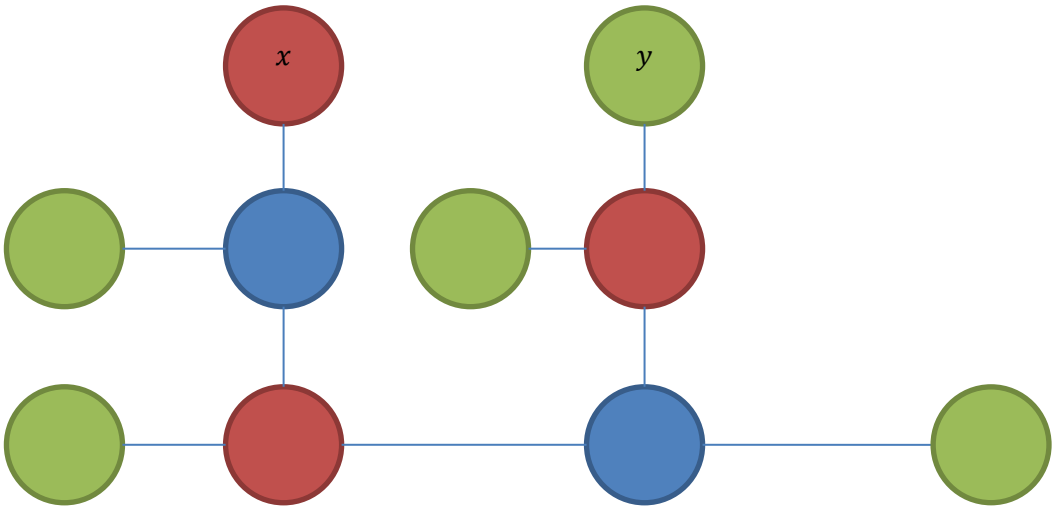
Build the gadget example:

Let $C = (x \vee y \vee \bar{z} \vee u \vee \bar{v} \vee w)$ – **an EVEN sized clause**. We create the following structure:

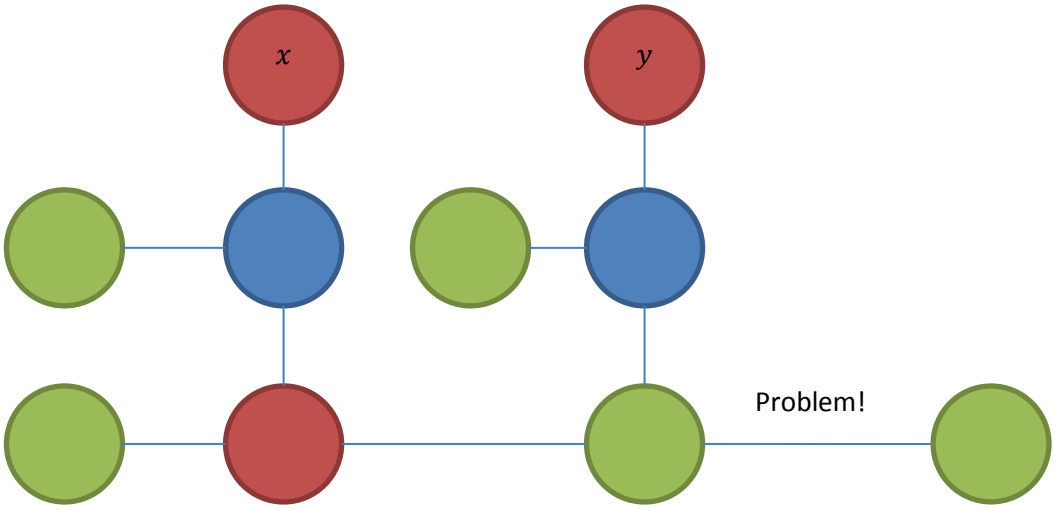


We have 2 new nodes per each variable-node $(x, y, \dots)$, and in addition we have the green node from before, here represented as different nodes (but they are all the same one).

Assume we had a satisfying assignment, where only $\bar{z} = TRUE$:



Assume we don't have a satisfying assignment:



When dealing with **ODD-sized clauses**, the right-most node simply needs to be set to **red** instead of **green**. The interchanging colors (red/green) in the lower row of nodes will get stuck for a non-satisfying assignment.