**CS 3510 Algorithms** 3/28/2024

# NP Hard Problems

*Lecturer: Jan van den Brand* *Scribe(s): Ankith Thalanki*

In today's lecture we covered some additional **NP-Hard** problems. We looked at the clique problem, which we proved to be **NP-Complete** by reducing from the Independent Set problem covered before. We then looked at Vertex Cover, Set Cover, and 3-Coloring, and prove they are all **NP-Complete**. We also look at the generic n-Coloring problem where $n$ is greater than 2, and show how that is also **NP-Hard**.

## Largest Clique Set

The CliqueSet problem asks if the largest Clique in a graph $G$ is of size greater than or equal to $k$. We can prove this problem is **NP** easily by verifying every node in our declared clique is connected to every other node in the clique, proving it is indeed a clique of its declared size. We can then show that the problem is in **NP-Complete** by reducing from IndependentSet. Notice that if in $G$ we have a clique set $C_k$, where $k$ is the size of the clique, then $\forall(u,v) \notin E(u \notin C_k \vee v \notin C_k)$. This is essentially saying that if some edge $(u,v)$ is not present in the graph, then the vertices $u$ and $v$ can't both be present in $C_k$, otherwise $C_k$ wouldn't be a clique. We can derive a very similar equation for the IndependentSet problem, where we have a graph $G$ and an independent set $I_k$, where $k$ is the size of the independent set. In this problem, if an edge is present in $G$, both of the nodes cannot be in $I_k$. That can be represented by the following equation: $\forall(u,v) \in E(u \notin I_k \vee v \notin I_k)$. In order to reduce the IndependentSet problem into the CliqueSet problem, we can simply create a new graph $G'$ by swapping every edge in $G$. Then our $C_k$ in $G$ will be equivalent to $I_k$ in $G'$, allowing us to solve IndependentSet with CliqueSet, proving the problem is also in **NP-Hard**.

## Vertex Cover

The VertexCover problem asks if we can cover a graph $G$ with at most $k$ nodes, where by cover we mean that for every edge in $G$, it is adjacent to a node contained in the vertex cover set, $A$. We can represent this idea by the equation $\forall(u,v) \in E(u \in A \vee v \in A)$. This problem is **NP** as it only takes polynomial time to verify the solution by going through every edge. We can also easily prove this problem is in **NP-Complete** by proving it is **NP-Hard** by reducing from IndependentSet, via a similar strategy from the above problem. Remember for the IndependentSet problem our corresponding equation is $\forall(u,v) \in E(u \notin I_k \vee v \notin I_k)$. If we define $X_{n-k} = G - I_k$, we can redefine our problem as $\forall(u,v) \in E(u \in X_{n-k} \vee v \in X_{n-k})$, which is equivalent to our equation for VertexCover if we set $X_{n-k} = A$. Therefore we can solve IndependentSet using VertexCover. For example, if we wished to find an independent set of size $k$ in a graph $G$ where $|G| = n$, we would simply need to find a vertex cover in $G$ of maximum size $n - k$. This shows that VertexCover is **NP-Hard** and **NP-Complete**.

## Set Cover

In the SetCover problem, we define a "universe" set $\Omega$ that contains every element explored in this problem. We then consider a series of subsets $S_1, S_2, ..., S_l$ where $S_i \subseteq \Omega$. In SetCover we wish to determine if there is a collection of $k$ subsets, $C \subseteq [l]$ such that $\bigcup_{i \in C} S_i = \Omega$. We can verify this problem is in **NP**, as if we are given a set of subsets, we can determine if their union is equal to $\Omega$ in polynomial time. To prove this problem is in **NP-Complete** we need to show it is **NP-Hard**, which can be done by reducing from a similar problem, VertexCover. We can first define $\Omega$ to be $E$, or every edge in $G$. We will then define $|V|$ subsets, one subset for each node, where the subset contains all the edges adjacent to that specified node. All the subsets chosen by SetCover for this problem have corresponding vertices needed for VertexCover, and if a solution is found for SetCover, that means that there are $k$ vertices which cover every edge in $G$.

## 3-Coloring

The 3Coloring problem asks if a graph $G$ can be colored by 3 colors, such that each color is an independent set. In other words, it asks if we can make three independent sets whose union contains the entirety of $G$. The 2 coloring version of this problem amounts to just determining if there is an odd cycle or not in the graph, and is in polynomial time and thus in **P**. In order to prove this problem is in **NP**, given any coloring, we can loop through all the nodes and ensure that none of their neighbours share the same color. As this can be done in polynomial time, 3Coloring is in **NP**. In order to prove this problem is in **NP-Complete**, we need to show it is **NP-Hard**, and we can do so by reducing from 3-SAT problem. First, we can define variables and their true and false values via the following gadget:

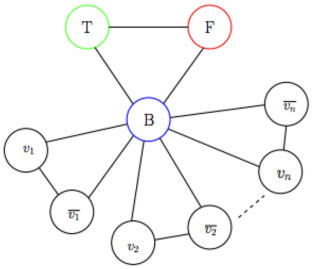


Figure 1: Example of the gadget used to enforce true and false values for the 3-SAT to 3Coloring reduction.

This gadget restricts the values of variables to two colors, which are defined by the colors the True and False nodes are colored in with. Since they art all connected to the Blue node, they can only be True and False.

Now we need to implement the gadgets that represent each clause. We can make a sort of triple or gate using the structure below: If you play around with the values of $a$, $b$, $c$, you see that the output mimics $a \vee b \vee c$, which replicates a clause in SAT-3. Each $a$, $b$, etc. represents either a true or false node created in the previous gadget. In order to enforce that

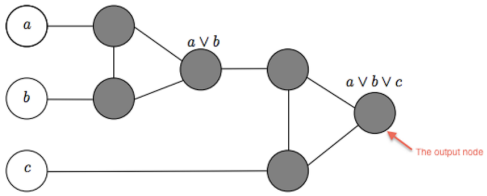


Figure 2: Enter Caption

this clause is true, we can simply connect the output to the Blue and False vertices, such that it must be true to be color able. Adding other clauses is simply a matter of repeating the above steps. Now if our instance of 3-SAT is not colorable, that means that it is impossible to solve for true, indicating it is false. If it is solvable, then our graph will be colorable, and vice-versa. Therefore we have reduced 3-SAT to 3COLORING and proved that 3COLORING is **NP-Hard**. As we proved it was **NP** earlier, we have proved it was **NP-Complete**.

## N-Coloring

After proving 3COLORING is **NP-Complete**, its relatively trivial to prove that any NCOLORING is **NP-Complete** via induction. First of all, the algorithm that verifies any coloring is valid, by looping through all nodes and ensuring it doesn't share a color with its neighbor, is still polynomial for the $n$ case, therefore showing NCOLORING is **NP**. To prove the problem is **NP-Hard** via induction, we can set our base case to be 3COLORING (which we already proved to be **NP-Hard** in the previous section). Our I.H (Induction Hypothesis) would be to assume that (N-1)COLORING, and then our I.S (Induction Step) would be to show that NCOLORING is **NP-Hard** by reducing to (N-1)COLORING. In order to solve (N-1)COLORING via NCOLORING, we can simply take the $G$, the graph associated with the (N-1)COLORING problem, and make a new graph $G'$ with an extra node connected to all other nodes. Therefore if $G$ is $n$-colorable, then $G'$ must be $n+1$-colorable due to the extra node. If $G$ is not $n$-colorable, but some $x > n$, then $G'$ will be $x+1$ colorable and not $n+1$ colorable. Therefore we have reduced (N-1)COLORING to NCOLORING and shown that NCOLORING is **NP-Hard** via Induction. Combined with what we showed earlier, this means that NCOLORING is **NP-Complete** for all $n \geq 3$.