Chapter 2

NP Completeness II

By Sariel Har-Peled, December 10, 2013[1] Version: 1.04

2.1 Max-Clique

We remind the reader, that a ***clique*** is a complete graph, where every pair of vertices are connected by an edge. The MaxClique problem asks what is the largest clique appearing as a subgraph of G. See Figure 2.1.

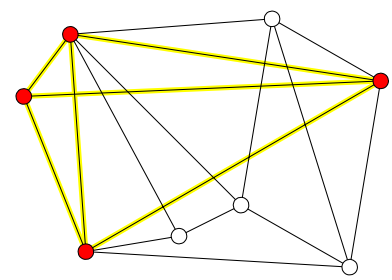


Figure 2.1: A clique of size 4 inside a graph with 8 vertices.

MaxClique

Instance: A graph G
Question: What is the largest number of nodes in G forming a complete subgraph?

Note that MaxClique is an *optimization* problem, since the output of the algorithm is a number and not just true/false.

The first natural question, is how to solve MaxClique. A naive algorithm would work by enumerating all subsets $S \subseteq V(\mathsf{G})$, checking for each such subset S if it induces a clique in G (i.e., all pairs of vertices in S are connected by an edge of G). If so, we know that G_S is a clique, where G_S denotes the ***induced subgraph*** on S defined by G; that is, the graph formed by removing all the vertices are not in S from G (in particular, only edges that have both endpoints in S appear in G_S). Finally, our algorithm would return the largest S encountered, such that G_S is a clique. The running time of this algorithm is $O(2^n n^2)$ as can be easily verified.

Suggestion 2.1.1. When solving any algorithmic problem, always try first to find a simple (or even naive) solution. You can try optimizing it later, but even a naive solution might give you useful insight into a problem structure and behavior. $\boxed{\mathcal{TIP}}$

We will prove that MaxClique is NP-Hard. Before dwelling into that, the simple algorithm we devised for MaxClique shade some light on why intuitively it should be NP-Hard: It does not seem like there is any way of avoiding the brute force enumeration of all possible subsets of the vertices of G. Thus, a problem is NP-Hard or NP-Complete, *intuitively*, if the only way we know how to solve the problem is to use naive brute force enumeration of all relevant possibilities.

How to prove that a problem X is NP-Hard? Proving that a given problem X is NP-Hard is usually done in two steps. First, we pick a known NP-Complete problem A. Next, we show how to solve any instance of A in polynomial time, assuming that we are given a polynomial time algorithm that solves X.

Proving that a problem X is NP-Complete requires the additional burden of showing that is in NP. Note, that only decision problems can be NP-Complete, but optimization problems can be NP-Hard; namely, the set of NP-Hard problems is much bigger than the set of NP-Complete problems.

Theorem 2.1.2. *MaxClique is* NP-Hard.

Proof: We show a reduction from 3SAT. So, consider an input to 3SAT, which is a formula F defined over n variables (and with m clauses).

We build a graph from the formula F by scanning it, as follows:

(i) For every literal in the formula we generate a vertex, and label the vertex with the literal it corresponds to.
Note, that every clause corresponds to the three such vertices.

(ii) We connect two vertices in the graph, if they are: (i) in different clauses, and (ii) they are *not* a negation of each other.

Let G denote the resulting graph. See Figure 2.2 for a concrete example. Note, that this reduction can be easily be done in quadratic time in the size of the given formula.

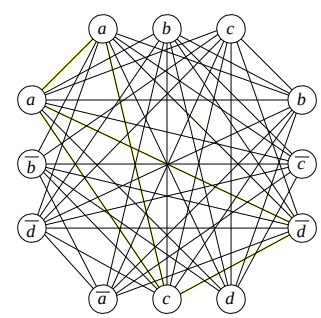


Figure 2.2: The generated graph for the formula $(a \vee b \vee c) \wedge (b \vee \overline{c} \vee \overline{d}) \wedge (\overline{a} \vee c \vee d) \wedge (a \vee \overline{b} \vee \overline{d})$.

We claim that F is satisfiable iff there exists a clique of size m in G.

$\Longrightarrow$ Let $x_1, \ldots, x_n$ be the variables appearing in F, and let $v_1, \ldots, v_n \in \{0, 1\}$ be the satisfying assignment for F. Namely, the formula F holds if we set $x_i = v_i$, for $i = 1, \ldots, n$.
For every clause C in F there must be at least one literal that evaluates to TRUE. Pick a vertex that corresponds to such TRUE value from each clause. Let W be the resulting set of vertices. Clearly, W forms a clique in G. The set W is of size m, since there are m clauses and each one contribute one vertex to the clique.

$\Longleftarrow$ Let U be the set of m vertices which form a clique in G.
We need to translate the clique G_U into a satisfying assignment of F.

(i) set $x_i \leftarrow$ TRUE if there is a vertex in U labeled with x_i.

(ii) set $x_i \leftarrow$ FALSE if there is a vertex in U labeled with $\overline{x_i}$.

This is a valid assignment as can be easily verified. Indeed, assume for the sake of contradiction, that there is a variable x_i such that there are two vertices u, v in U labeled with x_i and $\overline{x_i}$; namely, we are trying to assign to contradictory values of x_i. But then, u and v, by construction will not be connected in G, and as such G_S is not a clique. A contradiction.
Furthermore, this is a satisfying assignment as there is at least one vertex of U in each clause. Implying, that there is a literal evaluating to TRUE in each clause. Namely, F evaluates to TRUE.

Thus, given a polytime (i.e., polynomial time) algorithm for Max-Clique, we can solve 3SAT in polytime. We conclude that MaxClique in NP-Hard. ■

MaxClique is an optimization problem, but it can be easily restated as a decision problem.

Clique

Instance: A graph G, integer k **Question:** Is there a clique in G of size k?

Theorem 2.1.3. *Clique is* NP-Complete.

Proof: It is NP-Hard by the reduction of Theorem 2.1.2. Thus, we only need to show that it is in NP. This is quite easy. Indeed, given a graph G having n vertices, a parameter k, and a set W of k vertices, verifying that every pair of vertices in W form an edge in G takes $O(u + k^2)$, where u is the size of the input (i.e., number of edges + number of vertices). Namely, verifying a positive answer to an instance of Clique can be done in polynomial time.

Thus, Clique is NP-Complete. ■

2.2 Independent Set

Definition 2.2.1. A set S of nodes in a graph $\mathsf{G} = (V, E)$ is an ***independent set***, if no pair of vertices in S are connected by an edge.

Independent Set

Instance: A graph G, integer k **Question:** Is there an independent set in G of size k?

Theorem 2.2.2. *Independent Set is* NP-Complete.

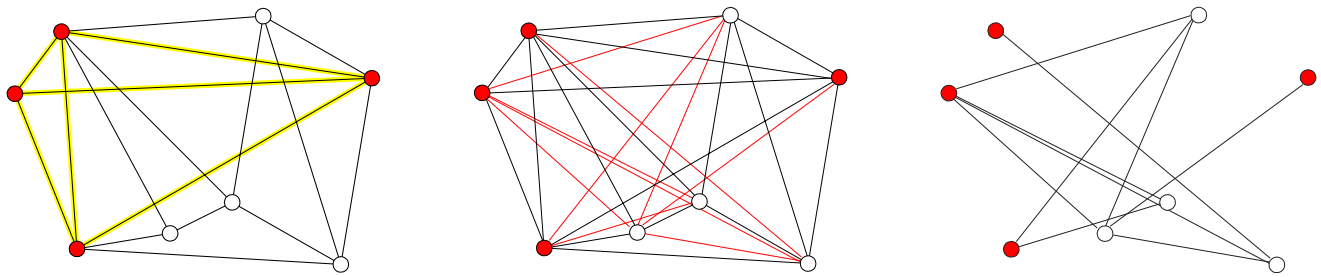
Proof: This readily follows by a reduction from Clique. Given G and k, compute the complement graph $\overline{G}$ where we connected two vertices u, v in $\overline{G}$ iff they are independent (i.e., not connected) in G. See Figure 2.3. Clearly, a clique in G corresponds to an independent set in $\overline{G}$, and vice versa. Thus, Independent Set is NP-Hard, and since it is in NP, it is NPC. ■

2.3 Vertex Cover

Definition 2.3.1. For a graph G, a set of vertices $S \subseteq V(\mathsf{G})$ is a ***vertex cover*** if it touches every *edge* of G. Namely, for every edge $uv \in E(\mathsf{G})$ at least one of the endpoints is in S.

Vertex Cover

Instance: A graph G, integer k **Question:** Is there a vertex cover in G of size k?



(a) (b) (c)

Figure 2.3: (a) A clique in a graph G, (b) the complement graph is formed by all the edges not appearing in G, and (c) the complement graph and the independent set corresponding to the clique in G.

Lemma 2.3.2. *A set S is a vertex cover in G iff $V \setminus S$ is an independent set in G.*

Proof: If S is a vertex cover, then consider two vertices $u, v \in V \setminus S$. If $uv \in E(\mathsf{G})$ then the edge uv is not covered by S. A contradiction. Thus $V \setminus S$ is an independent set in G.

Similarly, if $V \setminus S$ is an independent set in G, then for any edge $uv \in E(\mathsf{G})$ it must be that either u or v are not in $V \setminus S$. Namely, S covers all the edges of G. ■

Theorem 2.3.3. *Vertex Cover is* NP-Complete.

Proof: Vertex Cover is in NP as can be easily verified. To show that it NP-Hard we will do a reduction from Independent Set. So, we are given an instance of Independent Set which is a graph G and parameter k, and we want to know whether there is an independent set in G of size k. By Lemma 2.3.2, G has an independent set of k iff it has a vertex cover of size $n - k$. Thus, feeding G and $n - k$ into (the supposedly given) black box that can solves vertex cover in polynomial time, we can decide if G has an independent set of size k in polynomial time. Thus Vertex Cover is NP-Complete. ■

2.4 Graph Coloring

Definition 2.4.1. A ***coloring***, by c colors, of a graph $\mathsf{G} = (V, E)$ is a mapping $C : V(\mathsf{G}) \to \{1, 2, \ldots, c\}$ such that every vertex is assigned a color (i.e., an integer), such that no two vertices that share an edge are assigned the same color.

Usually, we would like to color a graph with a minimum number of colors. Deciding if a graph can be colored with two colors is equivalent to deciding if a graph is bipartite and can be done in linear time using DFS or BFS[2].

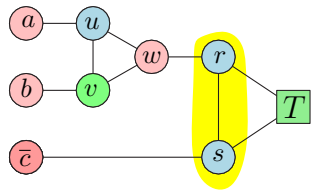
Coloring is useful for resource allocation (used in compilers for example) and scheduling type problems.

Surprisingly, moving from two colors to three colors make the problem much harder.

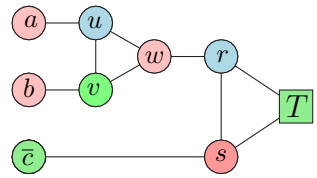
3Colorable

Instance: A graph G.
Question: Is there a coloring of G using three colors?

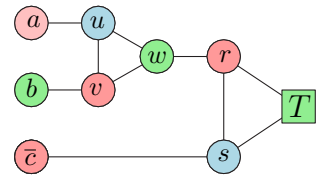
[2] If you do not know the algorithm for this, please read about it to fill this monstrous gap in your knowledge.



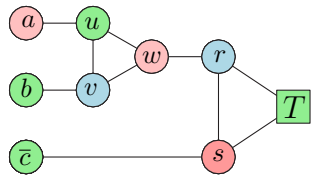
(1) $a \Leftarrow F$, $b \Leftarrow F$, $\overline{c} \Leftarrow F$



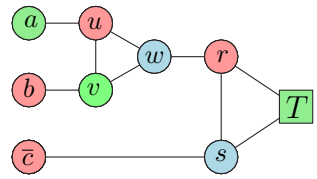
(2) $a \Leftarrow F$, $b \Leftarrow F$, $\overline{c} \Leftarrow T$



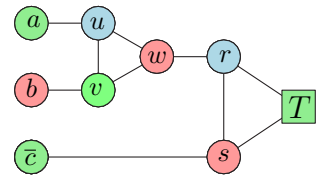
(3) $a \Leftarrow F$, $b \Leftarrow T$, $\overline{c} \Leftarrow F$



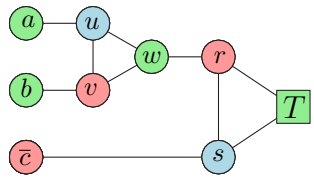
(4) $a \Leftarrow F$, $b \Leftarrow T$, $\overline{c} \Leftarrow T$



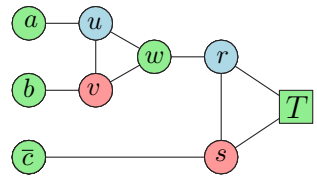
(5) $a \Leftarrow T$, $b \Leftarrow F$, $\overline{c} \Leftarrow F$



(6) $a \Leftarrow T$, $b \Leftarrow F$, $\overline{c} \Leftarrow T$



(7) $a \Leftarrow F$, $b \Leftarrow T$, $\overline{c} \Leftarrow T$



(8) $a \Leftarrow T$, $b \Leftarrow F$, $\overline{c} \Leftarrow F$

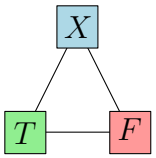
Figure 2.4: The clause $a \vee b \vee \overline{c}$ and all the three possible colorings to its literals. If all three literals are colored by the color of the special node F, then there is no valid coloring of this component, see case (1).

Theorem 2.4.2. *3Colorable is* NP-Complete.

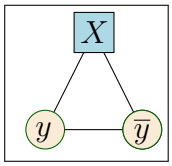
Proof: Clearly, 3Colorable is in NP.

We prove that it is NP-Complete by a reduction from 3SAT. Let $\mathcal{F}$ be the given 3SAT instance. The basic idea of the proof is to use gadgets to transform the formula into a graph. Intuitively, a ***gadget*** is a small component that corresponds to some feature of the input.

The first gadget will be the ***color generating gadget***, which is formed by three special vertices connected to each other, where the vertices are denoted by X, F and T, respectively. We will consider the color used to color T to correspond to the TRUE value, and the color of the F to correspond to the FALSE value.



For every variable y appearing in $\mathcal{F}$, we will generate a ***variable gadget***, which is (again) a triangle including two new vertices, denoted by x and $\overline{y}$, and the third vertex is the auxiliary vertex X from the color generating gadget. Note, that in a valid 3-coloring of the resulting graph either y would be colored by T (i.e., it would be assigned the same color as the color as the vertex T) and $\overline{y}$ would be colored by F, or the other way around. Thus, a valid coloring could be interpreted as assigning TRUE or FALSE value to each variable y, by just inspecting the color used for coloring the vertex y.



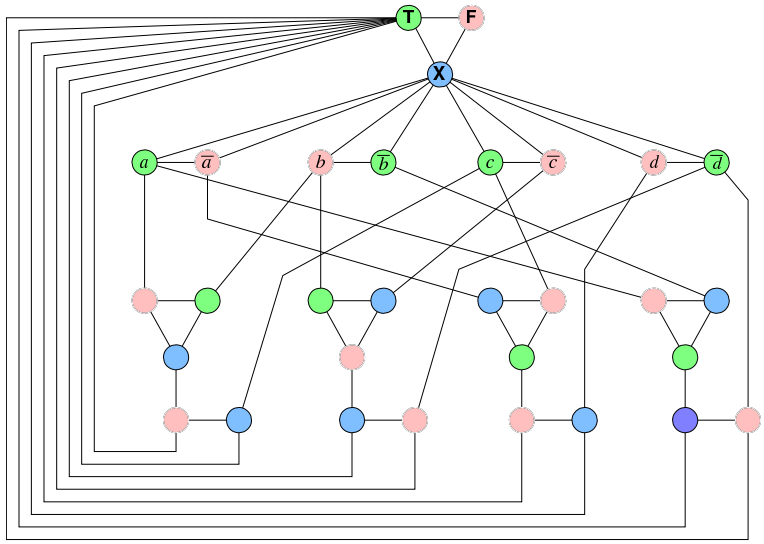
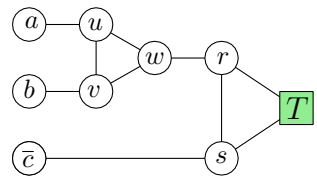
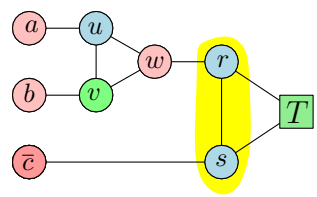


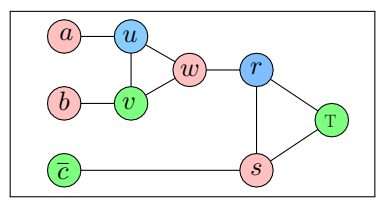
Figure 2.5: The formula $(a \vee b \vee c) \wedge (b \vee \overline{c} \vee \overline{d}) \wedge (\overline{a} \vee c \vee d) \wedge (a \vee \overline{b} \vee \overline{d})$ reduces to the depicted graph.



Finally, for every clause we introduce a ***clause gadget***. See the figure on the right – for how the gadget looks like for the clause $a \vee b \vee \overline{c}$. Note, that the vertices marked by a, b and $\overline{c}$ are the corresponding vertices from the corresponding variable gadget. We introduce five new variables for every such gadget. The claim is that this gadget can be colored by three colors if and only if the clause is satisfied. This can be done by brute force checking all 8 possibilities, and we demonstrate it only for two cases. The reader should verify that it works also for the other cases.



Indeed, if all three vertices (i.e., three variables in a clause) on the left side of a variable clause are assigned the F color (in a valid coloring of the resulting graph), then the vertices u and v must be either be assigned X and T or T and X, respectively, in any valid 3-coloring of this gadget (see figure on the left). As such, the vertex w must be assigned the color F. But then, the vertex r must be assigned the X color. But then, the vertex s has three neighbors with all three different colors, and there is no valid coloring for s.



As another example, consider the case when one of the variables on the left is assigned the T color. Then the clause gadget can be colored in a valid way, as demonstrated on the figure on the right.

This concludes the reduction. Clearly, the generated graph can be computed in polynomial time. By the above argumentation, if there is a valid 3-coloring of the resulting graph G, then there is a satisfying assignment for $\mathcal{F}$. Similarly, if there is a satisfying assignment for $\mathcal{F}$ then the G be colored in a valid way using three colors. For how the resulting graph looks like, see Figure 2.5.

This implies that 3Colorable is NP-Complete. ■

Here is an interesting related problem. You are given a graph G as input, and you know that it is 3-colorable. In polynomial time, what is the minimum number of colors you can use to color this graph legally? Currently, the best polynomial time algorithm for coloring such graphs, uses $O\left(n^{3/14}\right)$ colors.