Clique Problem Is NP-Complete

In this lecture, we prove NP-completeness of yet another problem – the clique problem.

1 What is Clique Problem

What is a clique: reminder. In a graph, a *subgraph* is what we get if we take some of the vertices of the original graph and all the edges connecting these vertices in the original graph.

A *clique* is a subgraph in which every two vertices are connected to each other. For example, two connected vertices form a clique of size 2, a triangle is a clique of size 3, etc.

Comment. The name comes from real cliques, like cliques in high school – when everyone within each clique plays with each other (but others are often excluded).

What is the clique problem. Now, we can formulate the problem:

- *given:* a graph and a natural number k;
- *find:* a subgraph of the given graph which is a clique of size k (or generate a message that there is no such subgraph).

How we will prove that the clique problem is NP-complete. We will prove this result by showing how to reduce 3-SAT to the clique problem. To describe this reduction, as usual, we will need three algorithms:

- the algorithm $U_1(x)$ that transforms a 3-CNF formula x into a graph

$$x' = U_1(x);$$

- the algorithm $U_2(y')$ that transforms the clique y' in a graph $x' = U_1(x)$ into the satisfying vector $y = U_2(y')$ for the original 3-CNF formula x; and
- the algorithm $U_3(y)$ that transforms a satisfying vector y for the 3-CNF formula x into a clique $y' = U_3(y)$ of given size in the graph $x' = U_1(x)$.

2 Algorithm U_1 That Transforms a 3-CNF Formula x into a Graph $x' = U_1(x)$

What is given. Let us assume that we have a 3-CNF formula $C_1 \,\&\, \ldots \,\&\, C_k$, where:

- C_i are clauses of length 2 or 3, i.e., formulas of the type $a \vee b$ or $a \vee b \vee c$, and
- a, b, and c are literals – i.e., Boolean variables or their negations.

We will reduce satisfiability for this formula to finding a clique of size k in the corresponding graph.

How to construct a graph. For each clause, we add as many vertices as there are literals in this clause:

- if the clause C_i is of the form $a \vee b$, we add two vertices a_i and b_i;
- if the clause C_i is of the form $a \vee b \vee c$, we add three vertices a_i, b_i, and c_i.

Vertices corresponding to the same clause are *not* connected. Now, we connect every two vertices corresponding to different clauses, with one exception – we *do not* connect vertices corresponding to opposite literals (i.e., to a variable v and its negation $\neg v$).

Example. Let us illustrate this algorithm on the example of the following 3-CNF formula

$$(p \vee \neg q) \,\&\, (\neg p \vee q \vee s) \,\&\, (\neg q \vee r \vee \neg s).$$

This formula consists of three clauses: $C_1 = p \vee \neg q$, $C_2 = \neg p \vee q \vee s$, and $C_3 = \neg q \vee r \vee \neg s$. Thus, we add three groups of vertices:

- vertices p_1 and $\neg q_1$ corresponding to the first clause;
- vertices $\neg p_2$, q_2, and s_2 corresponding to the second clause; and
- vertices $\neg q_3$, r_3, and $\neg s_3$ corresponding to the third clause.

The following pairs of vertices are connected by edges:

- the vertex p_1 is connected to q_2, s_2, $\neg q_3$, r_3, and $\neg s_3$ (but *not* to $\neg p_2$);
- the vertex $\neg q_1$ is connected to $\neg p_2$, s_2, $\neg q_3$, r_3, and $\neg s_3$ (but *not* to q_2);
- the vertex $\neg p_2$ is connected to $\neg q_3$, r_3, and $\neg s_3$ (it is also connected to some vertices corresponding to the first clause, but these connections we have already described);
- the vertex q_2 is connected to r_3 and $\neg s_3$ (but *not* to $\neg q_3$); and
- the vertex s_2 is connected to $\neg q_3$ and r_3 (but *not* to $\neg s_3$).

3 Algorithm U_2: from a Clique y' to a Satisfying Vector $y = U_2(y')$

Algorithm. For each Boolean variable v:

- if the clique contains a vertex v_i corresponding to the literal v, we take v to be true;
- if the clique contains a vertex $\neg v_i$ corresponding to the literal $\neg v$, we take v to be false.

If for some variable v, neither v nor $\neg v$ is contained in the clique, we can assign any truth value to v, e.g., the value "true".

Example. In the above example, vertices $\neg q_1$, $\neg p_2$, and r_3 form a clique. Then, according to the above algorithm U_2, we take $a =$"false", $b =$"false", and $c =$"true". For d, we can take any value, e.g., the value "true".

Comment. It is easy to see that for these values, the original formula is satisfied.

This procedure is consistent. Vertices v_i and $\neg v_j$ are never connected to each other, so they cannot belong to the same clique. So, a clique either contains a vertex corresponding to v or a vertex corresponding to $\neg v$, or neither.

Why $y = U_2(y')$ is a satisfying vector for the original formula. We have a clique of k vertices, exactly as many vertices as there are clauses in the original formula.

Vertices corresponding to the same clause *are not* connected, but all the vertices in a clique *are* connected (by definition of a clique). So, a clique can contain no more than one vertex from each clause. Since the clique has exactly as many vertices as the formula has clauses, it cannot skip any clause. So, the clique must contain exactly one vertex from each clause.

In our arrangement of Boolean values, if the clique contains a vertex p_i, then the corresponding literal a is true. Thus, the whole clause $C_i = a \vee \ldots$ is true. All the clauses are true, so the whole formula is true.

4 Algorithm U_3: from a Satisfying Vector y to a Clique $y' = U_3(y)$

Algorithm. Since the vector y of Boolean values makes the formula $C_1 \,\&\, \ldots \,\&\, C_k$ true, each clause C_i of the type $a \vee b$ or $a \vee b \vee c$ is true for the values y. This means, in turn, that at least one of the literals forming the clause is true. For each clause C_i, we pick one of the literals a which are true. The corresponding vertices p_i form the clique y'.

Example. For the above formula, one way to make it true is to have $p =$"false", $q =$"false", $r =$"true", and $s =$"true". For these Boolean values, all three clauses are true:

- the clause C_1 is true because $\neg q$ is true;
- the clause C_2 is true because $\neg p$ and s are true; and
- the clause C_3 is true because $\neg q$ and r are true.

If we pick $\neg q$ for C_1, $\neg p$ for C_2, and r for C_3, then the corresponding vertices $\neg q_1$, $\neg p_2$, and r_3 indeed form a clique.

Why they form a clique. They all belong to different clauses and they are not opposite to each other – so they are all connected, according to the algorithm U_1.

Try it yourself. Try the algorithm U_1 on the example of the formula

$$(p \vee q) \,\&\, (p \vee \neg q) \,\&\, (\neg p \vee q).$$

Homework. Now, you are hopefully ready to solve the corresponding homework problem.