

1. This is to help you recall Boolean formulae. A Boolean function f over r variables a_1, a_2, \dots, a_r is a function $f : \{0, 1\}^r \rightarrow \{0, 1\}$ which assigns 0 or 1 to each possible assignment of values to the variables. One can specify a Boolean function in several ways including a truth table. Here is a truth table for a function on 3 variables a_1, a_2, a_3 .

a_1	a_2	a_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Suppose we are given a Boolean function on r variables a_1, a_2, \dots, a_r via a truth table. We wish to express f as a CNF formula using variables a_1, a_2, \dots, a_r .

It may be easier to first think about expressing using a DNF formula (a disjunction of one more conjunctions of a set of literals). For instance the function above can be expressed as

$$(\bar{a}_1 \wedge \bar{a}_2 \wedge a_3) \vee (\bar{a}_1 \wedge a_2 \wedge \bar{a}_3) \vee (a_1 \wedge \bar{a}_2 \wedge a_3) \vee (a_1 \wedge a_2 \wedge \bar{a}_3) \vee (a_1 \wedge a_2 \wedge a_3).$$

- What is a CNF formula for the function? *Hint:* Think of the complement function and complement the DNF formula.
- Describe how one can express an arbitrary Boolean function f over r variables as a CNF formula over the variables using at most 2^r clauses.

Solution: We consider the Boolean function \bar{f} which is the complement of f . We can express \bar{f} in DNF form using at most 2^r terms. We then complement the resulting DNF formula to obtain our desired CNF formula which has at most 2^r clauses.

For the example function we obtain a DNF formula for \bar{f} as

$$(\bar{a}_1 \wedge \bar{a}_2 \wedge \bar{a}_3) \vee (\bar{a}_1 \wedge a_2 \wedge a_3) \vee (a_1 \wedge \bar{a}_2 \wedge \bar{a}_3).$$

Thus the CNF formula for f is obtained by complementing this DNF formula and we obtain:

$$(a_1 \vee a_2 \vee a_3) \wedge (a_1 \vee \bar{a}_2 \vee \bar{a}_3) \wedge (\bar{a}_1 \vee a_2 \vee a_3).$$

■

2. A *Hamiltonian cycle* in a graph G is a cycle that goes through every vertex of G exactly once. Deciding whether an arbitrary graph contains a Hamiltonian cycle is NP-hard.

A **tonian cycle** in a graph G is a cycle that goes through at least *half* of the vertices of G . Prove that deciding whether a graph contains a tonian cycle is NP-hard.

Solution (duplicate the graph): I'll describe a polynomial-time reduction from HAMILTONIANCYCLE. Let G be an arbitrary graph. Let H be a graph consisting of two disjoint copies of G , with no edges between them; call these copies G_1 and G_2 . I claim that G has a Hamiltonian cycle if and only if H has a tonian cycle.

\Rightarrow Suppose G has a Hamiltonian cycle C . Let C_1 be the corresponding cycle in G_1 . C_1 contains exactly half of the vertices of H , and thus is a tonian cycle in H .

\Leftarrow On the other hand, suppose H has a tonian cycle C . Because there are no edges between the subgraphs G_1 and G_2 , this cycle must lie entirely within one of these two subgraphs. G_1 and G_2 each contain exactly half the vertices of H , so C must also contain exactly half the vertices of H , and thus is a *Hamiltonian* cycle in either G_1 or G_2 . But G_1 and G_2 are just copies of G . We conclude that G has a Hamiltonian cycle.

Given G , we can construct H in polynomial time easily. ■

Solution (add n new vertices): I'll describe a polynomial-time reduction from HAMILTONIANCYCLE. Let G be an arbitrary graph, and suppose G has n vertices. Let H be a graph obtained by adding n new vertices to G , but no additional edges. I claim that G has a Hamiltonian cycle if and only if H has a tonian cycle.

\Rightarrow Suppose G has a Hamiltonian cycle C . Then C visits exactly half the vertices of H , and thus is a tonian cycle in H .

\Leftarrow On the other hand, suppose H has a tonian cycle C . This cycle cannot visit any of the new vertices, so it must lie entirely within the subgraph G . Since G contains exactly half the vertices of H , the cycle C must visit every vertex of G , and thus is a Hamiltonian cycle in G .

Given G , we can construct H in polynomial time easily. ■

3. *Big Clique* is the following decision problem: given a graph $G = (V, E)$, does G have a clique of size at least $n/2$ where $n = |V|$ is the number of nodes? Prove that *Big Clique* is NP-hard.

Solution: Recall that an instance of **CLIQUE** consists of a graph $G = (V, E)$ and integer k . (G, k) is a YES instance if G has a clique of size at least k , otherwise it is a NO instance. For simplicity we will assume n is an even number.

We describe a polynomial-time reduction from **CLIQUE** to **BIG CLIQUE**. We consider two cases depending on whether $k \leq n/2$ or not. If $k \leq n/2$ we obtain a graph $G' = (V', E')$ as follows. We add a set of X new vertices where $|X| = n - 2k$; thus $V' = V \uplus X$. We make X a clique by adding all possible edges between vertices of X . In addition we connect each vertex $v \in X$ to each vertex $u \in V$. In other words $E' = E \cup \{(u, v) \mid u \in V, v \in X\} \cup \{(a, b) \mid a, b \in X\}$. If $k > n/2$ we let $G' = (V', E')$ where $V' = V \uplus X$ and $E' = E$, where $|X| = 2k - n$. In other words we add $2k - n$ new vertices which are isolated and have no edges incident on them.

We make the following relatively easy claims that we leave as exercises.

Claim 1. Suppose $k \leq n/2$. Then for any clique S in G , $S \cup X$ is a clique in G' . For any clique $S' \in G'$ the set $S' \setminus X$ is a clique in G .

Claim 2. Suppose $k > n/2$. Then S is a clique in G' iff $S \cap X = \emptyset$ and S is a clique in G .

Now we prove the correctness of the reduction. We need to show that G has a clique of size k if and only if G' has a clique of size $n'/2$ where n' is the number of nodes in G' .

\Rightarrow Suppose G has a clique S of size k . We consider two cases. If $k > n/2$ then $n' = n + 2k - n = 2k$; note that S is a clique in G' as well and hence S is a big clique in G' since $|S| = k \geq n'/2$. If $k \leq n/2$, by the first claim, $S \cup X$ is a clique in G' of size $k + |X| = k + n - 2k = n - k$. Moreover, $n' = n + n - 2k = 2n - 2k$ and hence $S \cup X$ is a big clique in G' . Thus, in both cases G' has a big clique.

\Leftarrow Suppose G' has a clique of size at least $n'/2$ in G' . Let it be S' ; $|S'| \geq n'/2$. We consider two cases again. If $k \leq n/2$, we have $n' = 2n - 2k$ and $|S'| \geq n - k$. By the first claim, $S = S' \setminus X$ is a clique in G . $|S| \geq |S'| - |X| \geq n - k - (n - 2k) \geq k$. Hence G has a clique of size k . If $k > n/2$, by the second claim S' is a clique in G and $|S'| \geq n'/2 = (n + 2k - n)/2 = k$. Therefore, in this case as well G has a clique of size k .

■

4. Recall the following k COLOR problem: Given an undirected graph G , can its vertices be colored with k colors, so that every edge touches vertices with two different colors?
- (a) Describe a direct polynomial-time reduction from 3COLOR to 4COLOR.

Solution: Suppose we are given an arbitrary graph G . Let H be the graph obtained from G by adding a new vertex a (called an *apex*) with edges to every vertex of G . I claim that G is 3-colorable if and only if H is 4-colorable.

\Rightarrow Suppose G is 3-colorable. Fix an arbitrary 3-coloring of G , and call the colors “red”, “green”, and “blue”. Assign the new apex a the color “plaid”. Let uv be an arbitrary edge in H .

- If both u and v are vertices in G , they have different colors.
- Otherwise, one endpoint of uv is plaid and the other is not, so u and v have different colors.

We conclude that we have a valid 4-coloring of H , so H is 4-colorable.

\Leftarrow Suppose H is 4-colorable. Fix an arbitrary 4-coloring; call the apex’s color “plaid” and the other three colors “red”, “green”, and “blue”. Each edge uv in G is also an edge of H and therefore has endpoints of two different colors. Each vertex v in G is adjacent to the apex and therefore cannot be plaid. We conclude that by deleting the apex, we obtain a valid 3-coloring of G , so G is 3-colorable.

We can easily transform G into H in polynomial time by brute force. ■

(b) Prove that k COLOR problem is NP-hard for any $k \geq 3$.

Solution (direct): The lecture notes include a proof that 3COLOR is NP-hard. For any integer $k > 3$, I'll describe a direct polynomial-time reduction from 3COLOR to k COLOR.

Let G be an arbitrary graph. Let H be the graph obtain from G by adding $k - 3$ new vertices a_1, a_2, \dots, a_{k-3} , each with edges to every other vertex in H (including the other a_i 's). I claim that G is 3-colorable if and only if H is k -colorable.

\Rightarrow Suppose G is 3-colorable. Fix an arbitrary 3-coloring of G . Color the new vertices a_1, a_2, \dots, a_{k-3} with $k - 3$ new distinct colors. Every edge in H is either an edge in G or uses at least one new vertex a_i ; in either case, the endpoints of the edge have different colors. We conclude that H is k -colorable.

\Leftarrow Suppose H is k -colorable. Each vertex a_i is adjacent to every other vertex in H , and therefore is the only vertex of its color. Thus, the vertices of G use only three distinct colors. Every edge of G is also an edge of H , so its endpoints have different colors. We conclude that the induced coloring of G is a proper 3-coloring, so G is 3-colorable.

Given G , we can construct H in polynomial time by brute force. ■

Solution (induction): Let k be an arbitrary integer with $k \geq 3$. Assume that j COLOR is NP-hard for any integer $3 \leq j < k$. There are two cases to consider.

- If $k = 3$, then k COLOR is NP-hard by the reduction from 3SAT in the lecture notes.
- Suppose $k > 3$. The reduction in part (a) directly generalizes to a polynomial-time reduction from $(k-1)$ COLOR to k COLOR: To decide whether an arbitrary graph G is $(k-1)$ -colorable, add an apex and ask whether the resulting graph is k -colorable. The induction hypothesis implies that $(k-1)$ COLOR is NP-hard, so the reduction implies that k COLOR is NP-hard.

In both cases, we conclude that k COLOR is NP-hard. ■

To think about later:

6. Let G be an undirected graph with weighted edges. A Hamiltonian cycle in G is **heavy** if the total weight of edges in the cycle is at least half of the total weight of all edges in G . Prove that deciding whether a graph contains a heavy Hamiltonian cycle is NP-hard.

Solution (two new vertices): I'll describe a polynomial-time reduction from the Hamiltonian *path* problem. Let G be an arbitrary undirected graph (without edge weights). Let H be the edge-weighted graph obtained from G as follows:

- Add two new vertices s and t .
- Add edges from s and t to all the other vertices (including each other).
- Assign weight 1 to the edge st and weight 0 to every other edge.

The total weight of all edges in H is 1. Thus, a Hamiltonian cycle in H is heavy if and only if it contains the edge st . I claim that H contains a heavy Hamiltonian cycle if and only if G contains a Hamiltonian path.

\Rightarrow First, suppose G has a Hamiltonian path from vertex u to vertex v . By adding the edges vs , st , and tu to this path, we obtain a Hamiltonian cycle in H . Moreover, this Hamiltonian cycle is heavy, because it contains the edge st .

\Leftarrow On the other hand, suppose H has a heavy Hamiltonian cycle. This cycle must contain the edge st , and therefore must visit all the other vertices in H contiguously. Thus, deleting vertices s and t and their incident edges from the cycle leaves a Hamiltonian path in G .

Given G , we can easily construct H in polynomial time by brute force. ■

Solution (smartass): I'll describe a polynomial-time reduction from the standard Hamiltonian cycle problem. Let G be an arbitrary graph (without edge weights). Let H be the edge-weighted graph obtained from G by assigning each edge weight 0. I claim that H contains a heavy Hamiltonian cycle if and only if G contains a Hamiltonian path.

\Rightarrow Suppose G has a Hamiltonian cycle C . The total weight of C is at least half the total weight of all edges in H , because $0 \geq 0/2$. So C is a heavy Hamiltonian cycle in H .

\Leftarrow Suppose H has a heavy Hamiltonian cycle C . By definition, C is also a Hamiltonian cycle in G .

Given G , we can easily construct H in polynomial time by brute force. ■