# Analysis of Algorithms I: Reductions and NP-Completeness

Xi Chen

Columbia University

Recall the first NP-complete problem Circuit-SAT:

1. Input: A boolean circuit. It is a directed acyclic graph in which every vertex has in-degree $\leq 2$. There is a unique vertex that has out-degree 0 and is called the output of the circuit. The vertices with in-degree 0 are called the inputs of the circuit. Every vertex that is not an input (including the output) is labelled as one of the three Boolean gates: $\wedge$ (and) $\vee$ (or) $\neg$ (not). (See Figure 34.8 on Page 1072 for an example.)
2. Output: Yes if the circuit is satisfiable: there is an assignment of $\{0, 1\}$ to each input such that the circuit outputs 1.

**Theorem**

*Circuit-SAT is NP-complete.*

Now we use the NP-completeness of Circuit-SAT to show that 3-SAT is also NP-complete. 3-SAT: Satisfiability of a CNF Boolean formula with 3 literals in each clause (or a so-called 3-CNF formula). Here is an example of a 3-CNF:

$$\left(x_1 \vee \neg x_3 \vee \neg x_2\right) \wedge \left(x_2 \vee x_3 \vee x_4\right) \wedge \left(\neg x_1 \vee \neg x_3 \vee \neg x_4\right)$$

Formally, there are Boolean variables $x_1, \ldots, x_k$. A literal is either a variable $x_i$ itself or its negation $\neg x_i$. A 3-CNF formula is then an AND of $n$ clauses, each of which is the OR of three "distinct" literals. The problem is then, given a 3-CNF over $x_1, \ldots, x_k$, does there exist an assignment to $x_1, \ldots, x_k$ such that the formula evaluates to true: For every clause, at least one of its three literals is true. Such an assignment is called a satisfying assignment.

We prove the following theorem:

**Theorem**

*3-SAT is NP-complete.*

It is easy to show that 3-SAT is in NP: Given any assignment, checking whether it is indeed a satisfying assignment is in polynomial time. To prove that it is NP-hard, we now give a polynomial-time reduction from Circuit-SAT. (Recall from the last class why this is good enough.)

Let $Q$ denote a boolean circuit with $s$ inputs and $t$ gates, one of which is the output of $Q$. Introduce a Boolean variable for each input and each gate of the circuit, including the output. Let $z$ denote the variable corresponding to the output. Our goal is to construct a 3-CNF $\Phi$ over these $s + t$ variables so that $\Phi$ has a satisfying assignment if and only if $Q$ has a satisfying assignment.

To this end, we view $Q$ differently as $t$ equations over the $s + t$ variables: Each gate imposes a Boolean equation over some of the variables. For each of the $t$ gates of $Q$:

1. If it is a $\neg$ gate, let $a$ denote the variable of the gate and $b$ denote the (single) input variable, then it requires $a = \neg b$.
2. If it is a $\wedge$ gate, let $a$ denote the variable of the gate and $b$, $c$ denote the two input variables, then it requires $a = b \wedge c$.
3. If it is a $\vee$ gate, let $a$ denote the variable of the gate and $b$, $c$ denote the two input variables, then it requires $a = b \vee c$.

$Q$ is satisfiable if and only if there is an assignment to the $s + t$ variables such that $z = 1$ and all these $t$ equations are satisfied.

Now we continue the reduction. For each of the $t$ equations, we construct two or three clauses such that the equations is satisfied if and only if all the clauses are satisfied:

1. For an equation $a = \neg b$, we have

$$a = \neg b \;\Leftrightarrow\; \big(a \vee b\big) \wedge \big(\neg a \vee \neg b\big)$$

   Meaning that $a = \neg b$ iff both clauses above are satisfied.

1. For an equation $a = b \wedge c$, we have

$$a = b \wedge c \;\Leftrightarrow\; (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$$

2. For an equation $a = b \vee c$, we have

$$a = b \vee c \;\Leftrightarrow\; (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c)$$

Let $\Phi$ denote the conjunction of $z$ (as a clause with one single literal) and all the clauses above, two clauses for each $\neg$ gate and three clauses for each $\vee$ and $\wedge$ gate. It is easy to see that $\Phi$ can be constructed from the input circuit $Q$ in polynomial time. Also $\Phi$ has a satisfying assignment iff there is an assignment to the $s + t$ variables such that $z = 1$ and all $t$ equations are satisfied.

However, we are not done yet: Some clauses of $\Phi$ have only one or two literals. We replace each such clause in $\Phi$ as follows:

1. For each clause in $\Phi$ with two literals $(a \vee b)$: Introduce a new variable $p$; and replace $(a \vee b)$ by:

$$\big(a \vee b \vee p\big) \wedge \big(a \vee b \vee \neg p\big)$$

2. There is only one clause in $\Phi$ that has a single literal: $(z)$. Introduce two new variables $p$ and $q$; and replace $(z)$ by:

$$\big(z \vee p \vee q\big) \wedge \big(z \vee \neg p \vee q\big) \wedge \big(z \vee p \vee \neg q\big) \wedge \big(z \vee \neg p \vee \neg q\big)$$

Let $\Phi$ denote the resulting 3-CNF. $\Phi$ can be constructed from $Q$ in polynomial time; and $\Phi$ is satisfiable if and only if $Q$ is satisfiable.

This gives us a polynomial-time reduction from Circuit-SAT to 3-SAT and thus, 3-SAT is NP-hard and NP-complete as well. So, from now on, we can use 3-SAT as our starting point, instead of Circuit-SAT, and we will see that it makes life a lot easier.

Integer Linear Inequalities (ILL): Given a set of $\{0, 1\}$ variables as well as a set of linear inequalities over them, does there exist a $\{0, 1\}$ assignment that satisfies all the linear inequalities?

**Theorem**

*ILL is NP-complete.*

It is easy to see that ILL is in NP. To prove ILL is NP-hard, we give a simple polynomial-time reduction from 3-SAT. Given a 3-CNF $\Phi$ over $m$ variables $x_1, \ldots, x_m$ and $n$ clauses, we construct a system of linear inequalities as follows. It has the same set of variables.

For each clause $(a \vee b \vee c)$ in $\Phi$, where $a, b$ and $c$ are literals, we add a new linear inequality $\alpha + \beta + \gamma \geq 1$ as follows: If $a = x_i$ for some $i \in [m]$, then $\alpha = x_i$; if $a = \neg x_i$ for some $i \in [m]$, then set $\alpha = (1 - x_i)$, and similarly with $\beta$ and $\gamma$. It is easy to show that

$$(a \vee b \vee c) = 1 \;\Leftrightarrow\; \alpha + \beta + \gamma \geq 1$$

because $a$ is true if and only if $\alpha = 1$. For example:

$$(x_1 \vee x_2 \vee \neg x_3) = 1 \;\Leftrightarrow\; x_1 + x_2 + (1 - x_3) \geq 1$$

So $\Phi$ is satisfiable if and only if the linear system has a solution. This reduction is clearly polynomial time. So ILL is NP-complete.

Max Independent Set: Given an undirected graph $G = (V, E)$, a subset $I \subseteq V$ of vertices is an independent set if vertices in $I$ are pairwise nonadjacent. Decision problem: Given $G$ and a number $k$, decide if $G$ has an independent set of size $\geq k$.

**Theorem**

*Max Independent Set is NP-complete.*

Again, membership in NP is trivial: Given a set $I \subseteq V$ of vertices, checking whether $|I| \geq k$ is an independent set is easy. To show it is NP-hard, we give a polynomial-time reduction from 3-SAT. Let $\Phi$ denote a 3-CNF with $m$ variables and $n$ clauses $C_1, \ldots, C_n$.

We construct from $\Phi$ the following graph $G = (V, E)$:

1. $V$ has exactly $3n$ vertices $v_{i,j}$, $i \in [n]$ and $j \in [3]$. One vertex for each literal of each clause $C_1, \ldots, C_n$: $v_{i,j}$ corresponds to the $j$th literal in the $i$th clause $C_i$.
2. $E$ has the following edges. For each clause, add three edges to connect its three vertices $v_{i,1}, v_{i,2}$ and $v_{i,3}$. In addition, we connect all the complementary literals: If there are two clauses in $\Phi$, say the $i$th and the $i'$th with $i \neq i'$, such that the $j$th literal of the $i$th clause is the negation of the $j'$th literal of the $i'$th clause, then connect $v_{i,j}$ and $v_{i',j'}$.

It is clear that $G$ can be constructed in polynomial time. We show that $\Phi$ is satisfiable iff $G$ has an independent set of size $= n$.

If $\Phi$ is satisfiable, then $G$ has an independent set of size $n$: Take any satisfying assignment of $\Phi$. Let $I$ denote the following set of vertices: For each $i \in [n]$, pick ONE and ONLY ONE vertex $v_{i,j}$ that corresponds to a true literal (Claim: Such a vertex always exists, why?), and add it to $I$. From the construction, it is clear that $|I| = n$. Moreover, show that $I$ is an independent set of $G$.

If $G$ has an independent set of size $n$, then $\Phi$ is satisfiable: Let $I$ denote an independent set of size $n$. Then $I$ must consist of exactly ONE vertex from each triple $(v_{i,1}, v_{i,2}, v_{i,3})$. We construct an assignment to $x_1, \ldots, x_m$ as follows. For each $x_\ell$, $\ell \in [m]$:

1. Set $x_\ell = 1$ if $\exists$ a vertex $v_{i,j} \in I$ that corresponds to $x_\ell$;
2. Set $x_\ell = 0$ if $\exists$ a vertex $v_{i,j} \in I$ that corresponds to $\neg x_\ell$;
3. Otherwise, set $x_\ell \in \{0, 1\}$ arbitrarily.

First show that 1) and 2) above cannot both happen (because $I$ is an independent set) so there is no ambiguity. Then show that this must be a satisfying assignment of $\Phi$. It then follows that this is a polynomial-time reduction from 3-SAT to Max Independent Set and thus, the latter is also NP-complete.

Clique: A clique of an undirected graph $G = (V, E)$ is a subset of pairwise adjacent vertices. Decision problem: Given $G$ and a number $k$, decide whether there is a clique of size $\geq k$.

**Theorem**

*Clique is NP-complete.*

It is easy to see that Clique is in NP. To prove it is NP-hard, we give a simple polynomial-time reduction from Max Independent Set to Clique. To this end, let $(G, k)$, where $G = (V, E)$, denote an input of Max Independent Set. We use $G' = (V, E')$ to denote the complement of $G$: $(u, v) \in E$ if and only if $(u, v) \notin E$.

The following lemma is easy to prove:

**Lemma**

*$I \subseteq V$ is an independent set of $G$ iff $I$ is a clique of $G'$*

This immediately implies that $G$ has an independent set of size $\geq k$ if and only if $G'$ has a clique of size $\geq k$. This gives us a polynomial-time reduction from Max Independent Set to Clique, so the latter is also NP-complete.

Vertex Cover: A vertex cover of $G$ is a subset $I \subseteq V$ of vertices such that every edge $(u, v) \in E$ has at least one vertex in $I$.
Decision problem: Given $G$ and a number $k$, decide whether $G$ has a vertex cover of size $\leq k$ or not.

**Theorem**

*Vertex Cover is NP-complete.*

It is easy to see that Vertex Cover is in NP. To show it is NP-hard we give a simple polynomial-time reduction from Independent Set.

The following lemma is easy to prove:

**Lemma**

*$I \subseteq V$ is an independent set of $G$ iff $V - I$ is a vertex cover of $G$.*

This implies that $G$ has an independent set $\geq k$ if and only if $G$, the same graph, has a vertex cover of size $\leq n - k$. This gives us a reduction (Given $(G, k)$ for Max Independent Set, transform it into $(G, n - k)$ for Vertex Cover) from Max Independent Set to Vertex Cover, so the latter is also NP-complete.

Subset Sum: Given a set $S$ of integers and a target integer $t$, decide if there is a subset $T$ of $S$ that sums to exactly $t$.

**Theorem**

*Subset Sum is NP-complete.*

It is easy to see that Subset Sum is in NP. To show it is NP-hard, we give a polynomial-time reduction from Vertex Cover to Subset Sum. Given $G = (V, E)$ and $k$, we need to decide if $G$ has a vertex cover of size $= k$. (Note that $G$ has a vertex cover of size $\leq k$ iff it has a vertex cover of size $= k$, why?) We construct $t$ and $S$, a set of $|V| + |E|$ integers, as follows.

Let $V = \{1, 2, \ldots, n\}$ and $|E| = m$. $S$ has one integer $a_i$ for each vertex $i \in V$; and one integer $b_{i,j}$ for each edge $(i, j) \in E$. Each of these $n + m$ numbers have $2m + 1$ bits in binary: the leading bit + 2 bits for each edge (arrange the $m$ edges in any order; for any $\ell \in [m]$, we refer to the $(2\ell)$th and $(2\ell - 1)$th lowest bits as the edge bits that correspond to the $\ell$th edge). We set $a_i$'s and $b_{i,j}$'s:

1. The leading bit of $a_i$, $i \in [n]$, is set to be 1. For each $\ell \in [m]$, set the two edge bits that correspond to the $\ell$th edge 01 if $i$ is a vertex of the edge; and 00 otherwise.
2. For each edge $(i, j) \in E$, we set $b_{i,j}$ as follows. The leading bit of $b_{i,j}$ is set to 0. Set the two edge bits that correspond to $(i, j) \in E$ to be 01; and set all other edge bits to be 00.

Finally, we set the target integer $t$:

$$t = k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$$

To see that this gives us a polynomial-time reduction from Vertex Cover to Subset Sum, we show that $S$ has a subset that sums to $t$ iff $G$ has a vertex cover of size $k$. To this end, we first notice that for any $T \subseteq S$, when we add up the numbers in $T$, there is no carry from the two edge bits that correspond to the $\ell$th edge to the next bit, because 1) no carry from the lower two bits, by induction; 2) every integer in $S$ has either 00 or 01 in these two bits; and 3) there are exactly three integers in $S$ that has 01 in these two bits.

If $G$ has a vertex cover $C$ with $k$ vertices, then there is a subset $T$ of $S$ that sums to $t$. Let $T$ denote the following integers in $S$:

$$T = \{a_i : i \in C\} \cup \{b_{i,j} : i \notin C \text{ or } j \notin C\}$$

First, the leading bit of $a_i$ is 1 and the leading bit of $b_{i,j}$ is 0. As $|C| = k$, the leading bits of $a_i$, $i \in C$, cancel with the first term $k \cdot 4^m$ in $t$. We only need to show that for every edge $(i, j) \in E$, the sum of numbers in $T$ has 10 in the two edge bits that corresponds to $(i, j) \in E$ (why?).

If a subset $T$ of $S$ sums to $t$, then $G$ has a vertex cover $C$ of size $= k$. Take $C = \{i : a_i \in T\}$ and we show that $C$ is a vertex cover of $G$ and $|C| = k$. Because there is no carry from any two edge bits to the next bit (the leading bit in particular), $T$ must have exactly $k$ $a_i$'s due to the term $k \cdot 4^m$ in $t$. So $|C| = k$. Again, because there is no carry from any two edge bits to the next, for each edge $(i, j) \in E$, $T$ must have exactly two numbers with 01 at the two edge bits that correspond to $(i, j)$. Thus, for each edge $(i, j) \in E$, $T$ contains either $a_i$ or $a_j$. It follows that $C$ is a vertex cover of $G$. This gives us a polynomial-time reduction from Vertex Cover to Subset Sum and thus, the latter is also NP-complete.

Directed Hamiltonian Cycle: A simple cycle that visits every vertex in a directed graph $G = (V, E)$. Decision problem: Given a directed graph $G = (V, E)$, decides if $G$ has a Hamiltonian cycle.

**Theorem**

*Directed Hamiltonian Cycle is NP-complete.*

It is clear that Directed Hamiltonian Cycle is in NP. To show it is NP-hard, we give a polynomial-time reduction from 3-SAT. Note that this proof is different from the one given in the textbook, a reduction from Vertex Cover. I found it somewhat simpler. Let $\Phi$ be a 3-CNF with $m$ variables and $n$ clauses $C_1, \ldots, C_n$.

We start with the following gadget: A doubly linked list between $u$ and $v$ of length $3n+2$ is a sequence of $3n+3$ vertices

$$\langle w_0, w_1, \cdots, w_{3n+2} \rangle$$

where $w_0 = u$ and $w_{3n+2} = v$, such that there is an edge from $w_i$ to $w_{i+1}$ and an edge from $w_{i+1}$ to $w_i$, for each $i \in [0 : 3n+1]$.

We use $G$ to denote the following directed graph: $G$ has the following $2m+2$ special vertices: $s, u_i, v_i, t$, where $i \in [m]$:

1. Add $(s, u_1)$ and $(s, v_1)$ to $E$;
2. For each $i \in [m]$, add new vertices to create a doubly linked list between $u_i$ and $v_i$ of length $3n+2$. We call the new vertices $w_{i,1}, \ldots, w_{i,3n+1}$.
3. For each $i \in [m-1]$, add $(u_i, u_{i+1}), (u_i, v_{i+1}), (v_i, u_{i+1})$ and $(v_i, v_{i+1})$ to $E$; Finally, add $(u_m, t)$, $(v_m, t)$ and $(t, s)$ to $E$.

We are not done yet (we haven't encoded the clauses in $G$ yet) but it is a good time to look at $G$ and see what is going on. The key question is then what are the Hamiltonian cycles of $G$. It is easy to see that $G$ right now has $2^m$ Hamiltonian cycles. Starting with $s$, for each $i \in [m]$, the cycle can visit all vertices in the doubly linked list between $u_i$ and $v_i$ either from $u_i$ to $v_i$ or from $v_i$ to $u_i$. After visiting all the $m$ doubly linked lists, the cycle visits $t$ and finally goes back to $s$, forming a Hamiltonian cycle.

One can also view each of these $2^m$ Hamiltonian cycles of $G$ as a $\{0,1\}$ assignment to the $m$ Boolean variables $x_1, \ldots, x_m$ of $\Phi$: If the cycle traverses the $i$th linked list from $u_i$ to $v_i$, then $x_i = 1$; otherwise the cycle traverses the $i$th linked list from $v_i$ to $u_i$, then $x_i = 0$. In this way we can decode a Boolean assignment from each Hamiltonian cycle of $G$. What we will do next is to add more structures to the graph $G$ to encode the $n$ clauses of $\Phi$, so that every Hamiltonian cycle of $G$ that remains (if any) corresponds to a satisfying assignment of $\Phi$, instead of an arbitrary assignment. This implies that $G$ has a Hamiltonian cycle iff $\Phi$ is satisfiable.

To this end, we add a new vertex $h_j$ for each clause $C_j$, $j \in [n]$:

1. For each of the three literals in the $j$th clause, if it is $x_i$ for some $i \in [m]$, then we add the following two edges to $E$:

$$\left(w_{i,3j-1}, h_j\right) \quad \text{and} \quad \left(h_j, w_{i,3j}\right)$$

   So that if we traverse the $i$th linked list from $u_i$ to $v_i$, we can visit $h_j$ on the way if we have not visited it yet.

2. If it is $\neg x_i$ for some $i \in [m]$, add the following edges to $E$:

$$\left(w_{i,3j}, h_j\right) \quad \text{and} \quad \left(h_j, w_{i,3j-1}\right)$$

   So that if we traverse the $i$th linked list from $v_i$ to $u_i$, we can visit $h_j$ on the way if we have not visited it yet.

If $\Phi$ is satisfiable then $G$ has a Hamiltonian cycle. To see this, given a satisfying assignment of $\Phi$, we construct a Hamiltonian cycle starting from $s$ as follows: For each $i \in [m]$, the cycle traverses the $i$th linked list from $u_i$ to $v_i$ if $x_i = 1$; and from $v_i$ to $u_i$ if $x_i = 0$. Along the path from $u_i$ to $v_i$ (or from $v_i$ to $u_i$), visit as many clause vertices as possible: visit $h_j$ if it is not visited yet and $x_i$ is a literal in the $j$th clause $C_j$ (or $\neg x_i$ is a literal in the $j$th clause $C_j$). Since it is a satisfying assignment, every clause vertex is visited once and the cycle goes to $t$ and finally back to $s$.

It can also be shown (a more delicate argument, however) that if $G$ has a Hamiltonian cycle then $\Phi$ is satisfiable. Basically any Hamiltonian cycle must visit the $i$th linked list either from $u_i$ to $v_i$ or from $v_i$ to $u_i$. In particular, it cannot visit a clause vertex when it traverses the $i$th linked list and then jumps to another doubly linked list. It has to visit all the linked lists one by one (why?). Whether the cycle traverses the $i$th list from $u_i$ to $v_i$ or from $v_i$ to $u_i$ gives us an assignment to the $m$ Boolean variables. We only need to show that this is a satisfying assignment of $\Phi$. This gives us a polynomial-time reduction from 3-SAT to Directed Hamiltonian Cycle and thus, the latter is also NP-complete.

Hamiltonian Cycle: A simple cycle that visits every vertex in an undirected graph $G = (V, E)$. Decision problem: Given an undirected $G = (V, E)$, decide if $G$ has a Hamiltonian cycle.

**Theorem**

*Hamiltonian Cycle is NP-complete.*

It is clear that Hamiltonian Cycle is in NP. To show it is NP-hard we give a polynomial-time reduction from Directed Hamiltonian Cycle. We only sketch the proof here. Let $G$ denote a directed graph for which we need to decide if it has a Hamiltonian cycle.

We construct an undirected $G' = (V', E')$ as follows. Set $V'$:

> Replace each vertex $v$ in $G$ by three vertices $v_1, v_2$ and $v_3$.

Then we construct the edge set $E'$ of $G'$. Starting with $E' = \emptyset$:

> For each vertex $v$ in $G$, add two undirected edges $(v_1, v_2)$ and $(v_2, v_3)$ to $E'$. For each directed edge $(u, v)$ in $G$, add an undirected edge $(u_3, v_1)$ to $E'$.

Show that $G$ has a directed Hamiltonian cycle if and only if $G'$ has a Hamiltonian cycle. This gives a polynomial-time reduction from Directed Hamiltonian Cycle to Hamiltonian Cycle and thus, the latter is NP-complete as well.

Finally, check the textbook (or try to figure out by yourself) for a very simple polynomial-time reduction from Hamiltonian Cycle to the Traveling Salesman Problem and thus, we have:

**Theorem**

*TSP is NP-complete.*