11  Computational Complexity and \( NP \)-Completeness

What we are going to study in this section can in some sense be described as “meta-algorithms” instead of algorithms proper. Instead of studying the running time of algorithms to solve specific problems, we will study the relative running times of algorithms.

Informally, we are looking for algorithms that run in time polynomial in the size of the input string. The set of the problems solvable by algorithms that run in polynomial time will be the set \( P \).

The set of problems for which there is an algorithm to “verify” the correctness of a result are the problems in the set \( NP \).

We will also be interested, as part of comparing runtimes of algorithms, in the problem of reducing one problem to another problem using only a polynomial number of steps.

We will in the course of these notes make more formal definitions of “polynomial time” and “polynomially reducible,” as well as the sets \( P \) and \( NP \).

We will also define more precisely the notion of “the size of the input.” On the latter question a couple of examples should suffice. A general rule, though, is that we must handicap ourselves as much as possible. That is, we consider the input size to be as small as it could possibly be.

Consider the problem of determining whether or not an integer \( N \) is prime or composite. When we think of the input of an integer to a primality testing algorithm, the input could reasonably be just the integer \( N \) itself. This takes \( \lg N \) bits. Thus, an algorithm for primality testing, in order to be considered to run in polynomial time, must run in time polynomial in \( \lg N \) for any
integer $N$ to be tested for primality.

In contrast, we can think about graph algorithms in a different way. A directed graph (not a multigraph) on $n$ vertices has less equal $2n^2$ edges (can’t have more than one edge from every vertex to every other vertex in both directions). Each edge can be represented with $2k = 2[\lg n]$ bits to represent the beginning and ending vertices. Any directed graph, then, can be represented in fewer than $4n^2 k + 2k + 1 = O(n^2 \lg n)$ bits, with the last $2k + 1$ summand being the cost of representing the number of edges (pairs) in the input string.

We will also encounter in the course of this discussion the biggest open problem in theoretical computer science: Is $P$ equal to $NP$?

Problems that will be considered generally consist of two kinds, and one of the first things that is done is to reduce one kind to the other.

The first kind of problem is a decision problem. For example: Given a graph $G$, is there a Hamiltonian cycle, that is, a cycle that passes through each vertex exactly once? This is a problem for which an algorithm would provide a yes-no answer.

The second kind of problem is an optimization problem. For example: Given a graph $G$, what is the diameter of the graph, that is, the maximum of the minimum distances between any two vertices?

Under some circumstances, we can reduce optimization problems to decision problems. In the case of the diameter problem, we can assert that the maximum diameter cannot be larger than the number of edges, and we know that the number of edges is $O(n^2 \lg n)$ in the number $n$ of vertices. Therefore, if we ask the decision problem, “Is the diameter of $G$ equal to $m$?” for every
\(m\) from 1 to \(2n^2\), then we will have posed at most \(O(n^2)\) decision problems. If the decision problem is solvable in polynomial time, then so is the optimization problem, because we are only required to ask polynomially-many decision problems.

Finally, we will refer to a problem as being \(NP\)-complete if it is in \(NP\) and can be shown to be at least as hard as any other problem that is in \(NP\). What we will usually show is that for any problem \(M\) that is \(NP\)-complete, there is a polynomial reduction from \(M\) to some other problem \(M'\) that is \(NP\)-complete, and there is a polynomial reduction from some \(NP\)-complete problem \(M''\) to \(M\). The \(NP\)-complete problems form an equivalence class of problems that are in \(NP\) and are all polynomially reducible to one another. It is this that makes the \(P = NP\) question so fundamental: If an algorithm could be found to solve any one of the \(NP\)-complete problems in polynomial time, then all the rest of the \(NP\)-complete problems could be solved in polynomial time.

### 11.1 Languages, Grammars, Machines, Encodings

In theoretical computer science, there are two complementary ways of looking at problems. One is in terms of grammars and the languages they generate; the other is in terms of finite state machines and the languages they recognize. In concrete terms, the language problem is this: Given the Backus-Naur form for specifying the legal statements in a programming language, what is the set of legal programs in that language? On the flip side, given a compiler that accepts programs, what is the set of programs that it considers to be correct?
Definition 11.1. A grammar consists of an alphabet $A$ of symbols, including a start symbol $s$, and a set $R$ of transition rules of the form $x \rightarrow y$, where $x$ and $y$ are strings of symbols.

Example Let $A = \{s, a, b, c, 0, 1\}$, and let $R$ be the set of rules

\[
\begin{align*}
s & \rightarrow 000a, & a & \rightarrow 000b, & b & \rightarrow 000c \\
s & \rightarrow 001a, & a & \rightarrow 001b, & b & \rightarrow 001c \\
s & \rightarrow 010a, & a & \rightarrow 010b, & b & \rightarrow 010c \\
s & \rightarrow 011a, & a & \rightarrow 011b, & b & \rightarrow 011c \\
s & \rightarrow 100a, & a & \rightarrow 100b, & b & \rightarrow 100c \\
s & \rightarrow 101a, & a & \rightarrow 101b, & b & \rightarrow 101c \\
s & \rightarrow 110a, & a & \rightarrow 110b, & b & \rightarrow 110c \\
s & \rightarrow 111a, & a & \rightarrow 111b, & b & \rightarrow 111c 
\end{align*}
\]

Definition 11.2. The language generated by a grammar consists of all the strings in the symbols that can be generated by using some combination of the transition rules.

In the case of the example above, the language generated consists of all 10-bit strings that can be viewed as three-digit integers in octal (but written in binary) and followed by a termination character $c$.

Definition 11.3. A language $L$ over an alphabet $A$ consists of a subset of the strings of symbols in the alphabet.

Languages are sets, and as such one can form unions, intersections, and complements of languages.

We have the empty string (written $\varepsilon$) and the empty language $\emptyset$. 
Definition 11.4. Given two languages $L_1$ and $L_2$, the concatenation of $L_1$ and $L_2$ is $\{xy : x \in L_1, y \in L_2\}$.

Definition 11.5. The closure (Kleene closure, Kleene star) of a language $L$ is $L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup ...$.

When in doubt, we will assume that all strings are taken from $\mathcal{L} = \{0,1\}^*$, the set of all binary finite-length strings.

Definition 11.6. A finite state machine $M$ consists of a set $S$ of states, one of which is the start state and some subset of which are the termination states, an alphabet $A$, and a function $f : S \times A \rightarrow S$ such that, when the machine is in state $s$ and is presented with symbol $a$, the machine makes a transition to state $f(s,a)$.

Definition 11.7. The language $L$ accepted by a machine $M$ consists of the strings over $A$ such that $M$, when beginning in the start state, and reading the strings in $L$, will end in a termination state.

Example Let $S = \{s, A, B, C\}$, with $s$ the start state and $C$ the termination state. Let the function $f$ be defined by

$$f(s, 000) = A, \quad f(A, 000) = B, \quad f(B, 000) = C$$
$$f(s, 001) = A, \quad f(A, 001) = B, \quad f(B, 001) = C$$
$$f(s, 010) = A, \quad f(A, 010) = B, \quad f(B, 010) = C$$
$$f(s, 011) = A, \quad f(A, 011) = B, \quad f(B, 011) = C$$
$$f(s, 100) = A, \quad f(A, 100) = B, \quad f(B, 100) = C$$
$$f(s, 101) = A, \quad f(A, 101) = B, \quad f(B, 101) = C$$
$$f(s, 110) = A, \quad f(A, 110) = B, \quad f(B, 110) = C$$
$$f(s, 111) = A, \quad f(A, 111) = B, \quad f(B, 111) = C$$
Note that we have abused the notation somewhat to permit us to read three symbols at once instead of one. We could easily fix this by the following. Instead of the first column of transitions, we use

\[
\begin{align*}
f(s, 0) &= A_1, & f(A_1, 0) &= A_2, & f(A_3, 0) &= A \\
f(s, 1) &= A_1, & f(A_1, 1) &= A_2, & f(A_3, 1) &= A
\end{align*}
\]

and similarly for the second and third columns.

This machine is clearly the machine that accepts the language generated above.

We will relax a little on the formalism and refer to algorithms, not machines, and we will think of an algorithm as terminating with an output either of 1 or 0. This is easily formalized by having two termination states, one of which corresponds to the 1 and one to the 0.

**Definition 11.8.** An algorithm \( A \) **accepts** a string \( x \in \mathcal{L} \) if, given input \( x \), the algorithm reads string \( x \) and outputs 1. An algorithm \( A \) **rejects** a string \( x \in \mathcal{L} \) if, given input \( x \), the algorithm reads string \( x \) and outputs 0.

**Definition 11.9.** The **language accepted** by an algorithm \( A \) is the set of strings accepted by \( A \).

**Definition 11.10.** A language \( L \) is **decided** by an algorithm \( A \) if every string in \( L \) is accepted by \( A \) and every string not in \( L \) is rejected by \( A \).

**Definition 11.11.** A language \( L \) is **accepted in polynomial time** by an algorithm \( A \) if there is a constant \( k \) such that every string in \( L \) of \( n \) symbols is accepted by \( A \) in \( O(n^k) \) steps.
Definition 11.12. A language $L$ is decided in polynomial time by an algorithm $A$ if there is a constant $k$ such that every string in $L$ of $n$ symbols is either accepted or rejected by $A$ in $O(n^k)$ steps.

Definition 11.13. The set $P$ of polynomial time problems is the set of languages such that a language $L$ is in $P$ if and only if there exists an algorithm $A$ that decides $L$ in polynomial time.

Theorem 11.14. The set $P$ of polynomial time problems is the set of languages such that a language $L$ is in $P$ if and only if there exists an algorithm $A$ that accepts $L$ in polynomial time.

Proof. Clearly if $L$ is decided in poly time, then it is accepted in poly time, so we need only go the other direction. Assume that $L$ is accepted in poly time by an algorithm $A$. We must show that the set complement of $L$ is rejected in poly time.

If $A$ accepts $L$ in $O(n^k)$ time, then there is a constant $C$ such that any string in $L$ is accepted in leq $Cn^k$ steps. What we do is create an algorithm $A'$ that simulates algorithm $A$ for $Cn^k$ time steps on inputs of size $n$. If the simulation terminates with what would be acceptance by $A$, then $A'$ terminates with acceptance. If the simulation does not terminate with what would be acceptance by $A$, then $A'$ terminates with rejection. \qed

11.2 Verification Problems

11.2.1 Hamiltonian Cycles

Definition 11.15. Given a graph $G = (V, E)$, a Hamiltonian cycle is a simple cycle that passes through every vertex exactly once. A graph with a
Hamiltonian cycle is said to be Hamiltonian.

**Theorem 11.16.** The problem of determining whether a graph is Hamiltonian is NP-complete.

The proof of this is deferred until later.

Many problems in graph theory and combinatorics are NP-complete. To see why this might be true, we can start with the “life’s tough all over” examination of the naive approaches to solving problems in graph theory.

Consider the naive way to look at the Hamiltonian cycle problem. A graph $G$ with $n$ vertices and $cn^2$ edges (for some constant $c$) will have $n!$ permutations of the vertices. Naively, then, we would need to look at $n!$ possible cycles. But the encoding of a graph is polynomial in the number of vertices, and $n!$ is by Stirling’s formula clearly not polynomial in $n$. So an examination of all possible cycles could not be polynomial in the input size.

On the other hand, a specific cycle that would include all vertices has size $n$, and thus to verify that a specific cycle was Hamiltonian would take only polynomially many steps in the input size.

**Definition 11.17.** A **verification algorithm** is an algorithm $A$ which takes as input a problem encoding string $x$ and a certificate string $y$ and outputs 1 if $y$ is a solution to the problem and 0 otherwise. The **language** verified by a verification algorithm is

$$L = \{ x \in \mathcal{L} : \exists y \in \mathcal{L} \text{ with } A(x, y) = 1 \}$$

**Definition 11.18.** The **complexity class NP** consists of the class of languages that can be verified by a polynomial time algorithm.

Clearly, the Hamiltonian-cycle problem is in $NP$. 
Clearly also $P \subseteq NP$, since deciding a language is more stringent than verifying a language.

It is generally believed that $P \neq NP$, but this is not known.

11.3 Reducibility

Let's look more carefully at the question of reducibility. To get a flavor for the arguments involved, we'll prove reducibility of Hamiltonian cycles to the travelling salesman problem.

**Definition 11.19.** The travelling salesman problem is: Given a set of cities (vertices), and distances between the cities (edge weights), find a tour of total weight less than $W$ that includes all the cities.

**Theorem 11.20.** The Hamiltonian cycle problem is polynomially reducible to the TSP. Thus, if TSP is NP-complete, then so is Hamiltonian cycle.

*Proof.* Assume that we have an algorithm $T$ that solves the TSP. Given a graph $G = (V, E)$ with $n = |V|$ for which we wish to solve the Hamiltonian cycle problem, create the following graph $G'$: The vertices $V'$ of $G'$ correspond to the vertices $V$ of $G$. For every edge $(u, v)$ in $G$ we create an edge of weight 1 in $G'$. For every pair of vertices $u, v$ in $G$ for which there is no edge $(u, v)$ we create an edge of weight 2 in $G'$.

We now run the TSP algorithm $T$ on $G'$ looking for a tour of weight $n$. That tour must be a Hamiltonian cycle.

To prove the theorem it remains only to prove/observe that the creation of $G'$ takes time that is polynomial in $n$. But this is easy. Enumerating the vertices of $G$ so as to copy the vertex set is linear in $n$. Creating an adjacency
matrix for the edges is quadratic in \( n \), with one pass to create the matrix and then one pass over the edge list of \( G \) to set the 1 and 2 values in the matrix. Creating the edge list for \( G' \) is then quadratic in \( n \); we just read the adjacency matrix and create edges of two different weights as needed.  \( \square \)

**Definition 11.21.** A language \( L_1 \) is polynomial-time reducible to a language \( L_2 \), which we will write \( L_1 \leq_P L_2 \), if there exists a poly time function \( f : \mathcal{L} \rightarrow \mathcal{L} \) such that for all \( x \in \mathcal{L} \), we have \( x \in L_1 \) if and only if \( f(x) \in L_2 \).

The function \( f \) will be called the **reduction function** and the algorithm \( F \) that computes \( f \) will be called the **reduction algorithm**.

**Theorem 11.22.** If \( L_1, L_2 \in \mathcal{L} \) are languages such that \( L_1 \leq_P L_2 \), then \( L_2 \in P \) implies that \( L_1 \in P \).

**Definition 11.23.** A language \( L \) is **NP-hard** if for every \( L' \in NP \) we have \( L' \leq_P L \).

That is, a language is NP-hard if it is no more than polynomially easier than any problem in \( NP \).

**Definition 11.24.** A language \( L \) is **NP-complete** if it is NP-hard and if \( L \in NP \).

**Theorem 11.25.** If any NP-complete problem is solvable in poly time, then \( P = NP \). If any NP-complete problem can be proved not to be solvable in poly time, then no NP-complete problem can be solved in poly time.
11.4 Some \( NP \)-Complete Problems

**Definition 11.26.** A **complete graph** is a graph \( G = (V, E) \) for which every pair of vertices is joined by an edge.

**Definition 11.27.** A **k-clique** in a graph \( G = (V, E) \) is a subgraph of \( G \) with \( k \) vertices that is a complete graph.

**Definition 11.28.** A **vertex cover** of a graph \( G \) is a subset \( S \subseteq V \) such that every edge of \( G \) is incident on some vertex of \( S \).

**Definition 11.29.** A graph \( G \) is **k-colorable** if there exists an assignment of the “colors” \( 1, 2, \ldots, k \) to the vertices of \( G \) such that no two adjacent vertices have the same color.

**Definition 11.30.** The **chromatic number** of a graph \( G \) is the smallest \( k \) such that \( G \) is \( k \)-colorable.

**Definition 11.31.** A **feedback vertex set** is a subset \( S \subseteq V \) such that every cycle of \( G \) contains a vertex in \( S \).

**Definition 11.32.** A **feedback edge set** is a subset \( F \subseteq E \) such that every cycle of \( G \) contains an edge in \( F \).

**Definition 11.33.** A **set cover** for a family of sets \( S_1, S_2, \ldots, S_n \) is a subfamily of \( k \) sets \( S_{i_1}, S_{i_2}, \ldots, S_{i_k} \) such that

\[ \cup_{j=1}^{k} S_{i_j} = \bigcup_{j=1}^{n} S_j? \]

**Definition 11.34.** An **exact cover** for a family of sets \( S_1, S_2, \ldots, S_n \) is a set cover consisting of pairwise disjoint sets.
Theorem 11.35. The following problems are all \textit{NP}-complete.

1. (SAT, or satisfiability) Is a Boolean expression satisfiable?

2. (CNF SAT) Is a Boolean expression in conjunctive normal form satisfiable?

3. (3-CNF SAT) Is a Boolean expression in conjunctive normal form, in which every product term is a sum of at most three variables, satisfiable?

4. (k-colorability) Is an undirected graph colorable with \(k\) colors?

5. (exact cover) Given a family of sets \(S_1, S_2, ..., S_n\), does there exist a set cover consisting of pairwise disjoint sets?
Theorem 11.36. The following problems are all NP-complete.

1. (k-clique) Does an undirected graph have a clique of size $k$?

2. (vertex cover) Does an undirected graph have a vertex cover of size $k$?

3. (feedback vertex set) Does an undirected graph have a feedback vertex set of $k$ vertices?

4. (feedback edge set) Does an undirected graph have a feedback edge set of $k$ edges?

5. (directed Hamiltonian cycle) Does a directed graph have a Hamiltonian cycle?

6. (set cover) Given a family of sets $S_1, S_2, ..., S_n$, does there exist a set cover?

7. (Hamiltonian cycle) Does an undirected graph have a Hamiltonian cycle?

We will prove this and the preceding theorem in order by reducing each problem to a previous problem.
We will start with Theorem 10.35, part (1) but we won’t prove it (yet).

**Theorem 11.37.** The satisfiability problem is NP-complete.

Theorem 10.35, part (2) can then be proved, but we would need the machinery that is defined in the proof of the previous theorem, so we will defer this one also.

**Theorem 11.38.** The satisfiability problem for Boolean expressions in CNF is NP-complete.

Now (having skipped the hard part) we can continue with 10.35, part (3).

**Theorem 11.39.** The 3-SAT problem is NP-complete.

*Proof.* We will reduce any SAT problem in CNF to a SAT problem in which the conjoined terms have lessequal three variables. Assume that we have a
product of sums of the form

\[ x_1 + x_2 + ... + x_k \]  \hspace{1cm} (1)

for \( k \geq 4 \). We can replace any such sum by

\[
(x_1 + x_2 + y_1)(x_3 + y_1 + y_2)(x_4 + y_2 + y_3)...(x_{k-2} + y_{k-2} + y_{k-3})(x_{k-1} + x_k + y_{k-3})
\]  \hspace{1cm} (2)

using new variables \( y_1, y_2, ..., y_{k-3} \). For example, we have

\[ x_1 + x_2 + x_3 + x_4 = (x_1 + x_2 + y_1)(x_3 + x_4 + y_1). \]

**Claim:** The original expression (1) is 1 for some assignment of the variables if and only if the replacement expression (2) is 1 for some assignment of the variables.

This is true for the following reason. The original expression is 1 if and only if some variable \( x_i = 1 \). If so, then set \( y_j = 1 \) for \( j \leq i - 2 \) and set \( y_j = 0 \) for \( j > i - 2 \). Then the replacing expression has the value 1.

Conversely, assume the replacing expression has value 1. If \( y_i = 0 \), then either \( x_1 \) or \( x_2 \) is 1. If \( y_{k-3} = 1 \), then either \( x_{k-1} \) or \( x_k \) is 1. If \( y_i = 1 \) and \( y_{k-3} = 0 \), then for some \( i, 1 \leq i \leq k-4 \), we have \( y_i = 1 \) and \( y_{i+1} = 0 \). This then implies that \( x_{i-2} = 1 \). One way or another, some \( x_i \) has the value 1.

So the claim is proved. Reduction of the number of variables is possible in a polynomial number of steps, and reduction to three variables is thus possible in a poly number of transformation steps. \( \square \)
We move on to Theorem 10.35, part (4).

**Theorem 11.40.** The problem of finding a $k$-coloring of an arbitrary graph is NP-complete.

**Proof.** We will prove that 3-SAT is poly transformable into the coloring problem.

Given an expression $F$ in 3-CNF with $n$ variables and $t$ factors, we will construct in time poly in $n + t$ an undirected graph $G = (V, E)$ with $3n + t$ vertices that can be colored with $n + 1$ colors if and only if $F$ is satisfiable. Let $x_1, x_2, ..., x_n$ and $F_1, F_2, ..., F_t$ be the variables and the factors of $F$. Let $v_1, v_2, ..., v_n$ be new symbols. WLOG we can assume $n \geq 4$.

We now build a graph. The vertices “are”

1. $x_i, \bar{x}_i, v_i$ for $1 \leq i \leq n$;
2. $F_i$ for $1 \leq i \leq t$.

The edges “are”

1. all $(v_i, v_j)$ such that $i \neq j$;
2. all $(v_i, x_j)$ and $(v_i, \bar{x}_j)$ such that $i \neq j$;
3. $(x_i, \bar{x}_i)$ for $1 \leq i \leq n$;
4. $(x_i, F_j)$ if $x_i$ is not a term of factor $F_j$;
5. $(\bar{x}_i, F_j)$ if $\bar{x}_i$ is not a term of factor $F_j$.

The vertices $v_1, v_2, ..., v_n$ form a complete subgraph of $n$ vertices. Any coloring must therefore require at least $n$ colors. Each $x_i$ and $\bar{x}_i$ is connected
to each $v_i$ for $i \neq j$, so none of the $x_j$ or $\overline{x}_j$ can be the same color as any $v_i$ except possibly for $v_j$. Since each $x_i$ and $\overline{x}_i$ are adjacent, they cannot be the same color. Thus $G$ cannot be colored with as few as $n + 1$ colors unless one of $x_j$ or $\overline{x}_j$ is the same color as $v_j$ and the other is a new special color.

Consider the vertex with the special color to have been assigned the value 0. Now consider the color assigned to the $F_j$ vertices. $F_j$ is adjacent to at least $2n - 3$ of the vertices $x_j$ and $\overline{x}_j$. Since we’re assuming $n \geq 4$, for every $j$ there exists an $i$ such that $F_j$ is adjacent to both $x_i$ and $\overline{x}_i$. Since one of these two is colored with the special color, $F_j$ cannot be colored with this special color.

If $F_j$ contains some literal $y$ for which $\overline{y}$ has been assigned the special color, then $F_j$ is not adjacent to any vertex colored the same as $y$ and hence can be assigned the same color as $y$.

Otherwise, we will need a new color.

Thus, all the $F_j$ can be colored with no additional colors if and only if there is an assignment of the special color to the literals such that each factor contains some literal $y$ for which $\overline{y}$ has been assigned the special color, that is, if and only if one can assign values to the variables so that each factor contains a $y$ that is assigned the value 1. That is, we can color the graph with $n + 1$ colors if and only if $F$ is satisfiable. \hfill \Box

**Example:** Consider $F = (x_1 + x_2)(\overline{x}_1 + x_3)$. 
Now part (5) of Theorem 10.35.

**Theorem 11.41.** The colorability problem is poly transformable to the exact cover problem. Thus the exact cover problem is NP-complete.

**Proof.** Let $G = (V, E)$ be an undirected graph. Let $k$ be an integer. We construct sets whose elements are chosen from

$$S = V \cup \{[e, i] : e \in E \text{ and } 1 \leq i \leq k\}.$$

For each $v \in V$ and for each $i$, $1 \leq i \leq k$, define

$$S_{vi} = \{v\} \cup \{[e, i] : \exists x \in V \ni e = (x, v) \text{ or } e = (v, x)\}.$$

For each $e \in E$ and for each $i$, $1 \leq i \leq k$, define $T_{ei} = \{[e, i]\}$.

We relate $k$-colorings to exact covers as follows. Let $G$ have a $k$-coloring with $v$ colored $c_v$, $1 \leq c_v \leq k$. Then the collection of sets $S_{xv}$ for each $v$ and the singleton sets $T_{ei}$ such that $[e, i] \notin S_{xv}$ for any $v$ forms an exact cover.

If $e = (v, w)$ is an edge, then $c_v \neq c_w$, so $S_{xv} \cap S_{xcw} = \emptyset$. And if $x$ and $y$ are not adjacent, then $S_{xcx}$ and $S_{ycy}$ are disjoint, and no $T_{ei}$ is selected unless it is disjoint from all other selected sets.

Conversely, suppose an exact cover $S$ exists. Then for each $v$, there is a unique $c_v$ such that $S_{xv} \in S$. We claim that a $k$-coloring comes from using $c_v$ to color each vertex $v$. Suppose not. Then there is some edge $e = (v, w)$ such that $c_v = c_w = c$. Then $S_{xv}$ and $S_{xcw}$ each contain $[c, c]$, so $S$ is in fact not an exact cover. \qed
**Theorem 11.42.** CNF-SAT is poly transformable into the clique problem. Therefore the clique problem is $NP$-complete.

*Proof.* Let $F = F_1 F_2 \ldots F_q$ be an expression in CNF, with

$$F_i = x_{i1} + x_{i2} + \ldots + x_{ik_i}$$

We will construct a graph whose vertices are pairs of integers $[i, j]$, for $1 \leq i \leq q$ and $1 \leq j \leq k_i$. The first component of the pair “is” a factor and the second component “is” a literal in that factor.

The graph has an edge $([i, j], [k, l])$ such that $i \neq k$ and $x_{ij} \neq \bar{x}_{kl}$.

Intuitively, two vertices are adjacent in $G$ if they correspond to different factors and one can assign values to the variables in the factors so that both literals have the value 1 (so both factors can have the value 1).

There are no more vertices in $G$ than factors in $F$. The number of edges is no larger than the square of $|F|$.

We now claim that $G$ has a clique of size $q$ if and only if $F$ is satisfiable.

Assume that $F$ is satisfiable. Then we can assign values to the literals so that $F = 1$. Thus each factor $F_i$ has at least one literal assigned the value 1. Let such a literal be $x_{im_i}$.

We claim that the set of vertices

$$\{[i, j] : 1 \leq i \leq q\}$$

form a clique of size $q$. Otherwise, we have $i$ and $j$, with $i \neq j$, with no edge between vertices $[i, m_i]$ and $[j, m_j]$. Thus $x_{im_i} = \bar{x}_{jm_j}$. This is impossible, since $x_{im_i} = x_{jm_j} 1$ by the choice of the literals.
Conversely, assume that $G$ has a clique of size $q$. Each vertex in the clique must have a distinct first component in the pair defining the vertex, because two vertices with the same first component are never connected.

Since there are $q$ vertices, there is a 1-1 correspondence between vertices and the factors of $F$.

Let the vertices be $[i, m_i]$. Let $S_1 = \{y : x_{im_i} = y\}$ for $1 \leq i \leq q$ and $y$ a variable. Let $S_2 = \{y : x_{im_i} = \bar{y}\}$ for $1 \leq i \leq q$ and $y$ a variable. Then $S_1$ and $S_2$ are the sets of complemented and uncomplemented variables. So $S_1 \cap S_2 = \emptyset$ By setting the values of the variables in $S_1$ to 1 and the values of the variables in $S_2$ to 0, the value of every factor $F_i$ is set to 1.

So $F$ is satisfiable.

And clearly the transformation is poly time.

\[ \square \]

**Theorem 11.43.** The clique problem is poly transformable into the vertex cover problem. Therefore the vertex cover problem is $NP$-complete.

**Proof.** Consider the complement $\bar{G} = (V, \bar{E})$ of a graph $G$, with edges exactly where $G$ does not have edges.

**Claim:** A set $S \subseteq V$ is a clique in $G$ if and only if $V \setminus S$ is a vertex cover of $\bar{G}$.

If $S$ is a clique, then no edge in $\bar{G}$ connects two vertices in $S$. Thus every edge in $\bar{G}$ is incident upon at least one vertex in $V \setminus S$, and $V \setminus S$ is a vertex cover.

Similarly, if $V \setminus S$ is a vertex cover, then every edge of $\bar{G}$ is incident upon at least one vertex of $V \setminus S$. So no edge of $\bar{G}$ connects two vertices in $V$. So every pair of vertices of $S$ is connected in $G$, and that means $S$ is a clique.
To decide the clique problem of size $k$, construct $\tilde{G}$ and decide whether it has a vertex of size $|V| - k$.

And the transformation is poly time. \hfill \qed

**Theorem 11.44.** The vertex cover problem is poly transformable into the feedback vertex set problem. Therefore, the feedback vertex set problem is NP-complete.

**Proof.** Let $G = (V, E)$ be an undirected graph. Let $D$ be the directed graph obtained by replacing each edge of $G$ by two directed edges. Thus $D = (V, E')$ where $E' = \{(v, w), (w, v) : (v, w) \in E\}$. Since every edge in $E$ has been replaced by a cycle in $D$, a set $S$ is a feedback vertex set for $D$ if and only if $S$ is a vertex cover for $G$.

And the transformation is clearly poly time. \hfill \qed

**Theorem 11.45.** The vertex cover problem is poly transformable into the feedback edge set problem. Therefore, the feedback edge set problem is NP-complete.

**Proof.**

**Theorem 11.46.** The vertex cover problem is poly transformable into the Hamiltonian cycle problem for directed graphs. Therefore, the Hamiltonian cycle problem for directed graphs is NP-complete.

**Proof.**

**Theorem 11.47.** The Hamiltonian cycle problem for directed graphs is poly transformable into the Hamiltonian cycle problem for undirected graphs. Therefore, the Hamiltonian cycle problem for undirected graphs is NP-complete.
Proof. Let $G = (V, E)$ be a directed graph. Let $U = (V \times \{0, 1, 2\}, E')$ be an undirected graph where $E'$ consists of edges

1. $([v, 0], [v, 1])$ for $v \in V$
2. $([v, 1], [v, 2])$ for $v \in V$
3. $([v, 2], [w, 0])$ if and only if $(v, w)$ is a directed edge in $E$.

We have expanded each vertex in $V$ into a path of three vertices. Since a Hamiltonian cycle in $U$ must include all the vertices, it must take the vertices in $0, 1, 2$ order in the second component (or in order $2, 1, 0$). WLOG we assume the former. So the edges of type $3$ defined above form a directed Hamiltonian cycle in $G$. Conversely, a Hamiltonian cycle in $G$ has edges $(v, w)$ in $G$ that correspond to the path $[v, 0], [v, 1], [v, 2], [w, 0]$ in $U$. 

Theorem 11.48. The vertex cover problem is poly transformable into the set cover problem. Therefore, the set cover problem is $NP$-complete.

Proof. Let $G = (V, E)$ be an undirected graph. For $1 \leq i \leq |V|$, let $S_i$ be the set of edges incident on vertex $v_i$. Clearly, $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ is a set cover for the $S_i$ if and only if $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ is a vertex cover for $G$. 

\qed