That is, we consider the input size to be as small as it could possibly be.

of examples should suffice. A general rule, though, is that we must handpick examples as much as possible.

The latter question a couple

We will also define more precisely the notion of "the size of the input". On the other hand, the phrase "reducible" also as well as the sets and \( NP \).

"Polynomially
during only a polynomial number of steps.

We will also be interested, as part of comparing runtimes of algorithms, in the problem of reducing one

\[ \text{The set of problems for which there is an algorithm to "verify" the correctness of a result are the problems} \]

\[ \text{set of problems solvable by algorithms that run in polynomial time will be the set} \]

Informally, we are looking for algorithms that run in time polynomial in the size of the instance. The

study the relative running times of algorithms.

What we are going to study in this section can in some sense be described as "meta-algorithms" instead of

14 Computational Complexity and \( NP \)-Completeness
would provide a yes-no answer. Cycle, that is, a cycle that passes through each vertex exactly once. This is a problem for which an algorithm
cycles. That is, a cycle that passes through each vertex exactly once. This is a problem for which an algorithm
The first kind of problem is a decision problem. For example: Given a graph G, is there a Hamiltonian
to reduce one kind to the other. Problems that will be considered generally consist of two kinds, and one of the first things that is done is

science: Is equal to Np? We will also encounter in the course of this discussion the bi-test open problem in theoretical computer
with the last 2k + 1 summand being the cost of representing the number of edges (pairs) in the input string.
ending vertices. Any directed graph, then, can be represented in fewer than 2n^2 bits to represent the beginning and
in both directions. Each edge can be represented with 2k bits to represent the beginning and
in a vertex i less than 2k edges (can't have more than one edge from every vertex to every other vertex
In contrast, we can think about graphs and Hamiltonian in a different way. A directed graph (not a multigraph)

polynomial time, must run in time polynomial in N for any integer N to be tested for primality. N itself. Thus takes \( \leq N \) years. Thus, an algorithm for primality testing in order to be considered to run in
of the input of an integer to a primality testing algorithm, the input could reasonably be just the integer
Consider the problem of determining whether or not an integer N is prime or composite. When we think
solved in polynomial time. Only if the $dN$-complete problems in polynomial time, then all the rest of the $dN$-complete problems could be solved in polynomial time. Is this what makes the question so fundamental? If an algorithm could be found to solve any one of the problems, all problems in the class would be reducible to one another. If there is a polynomial reduction from some $dN$-complete problem $\mathcal{P}$ to $\mathcal{Q}$, then is there a polynomial reduction from $\mathcal{Q}$ to some other problem that is $dN$-complete, and as hard as any other problem that is in $dN$? What we will usually show is that for any problem $\mathcal{P}$ that is $dN$-complete, there is a polynomial reduction from $\mathcal{P}$ to some other problem that is in $dN$ and can be shown to be at least as hard as any other problem that is in $dN$. Finally, we will refer to a problem as being polynomial-many decision problems. This means we are only required to ask polynomially-many questions about the decision problems. If the decision problem is solvable in polynomial time, then so is the optimization problem. If the optimization problem is solvable in polynomial time, then so is the decision problem. If the diameter of a graph $G$ is equal to $\pi(G)$, then the diameter cannot be larger than the number of vertices. Therefore, if we assume that the diameter of a graph $G$ is equal to $\pi(G)$, then the diameter of a graph $G$ is equal to $\pi(G)$, then the diameter cannot be larger than the number of vertices. Under some circumstances, we can reduce optimization problems to decision problems. In the case of the diameter of the graph, that is, the maximum of the minimum distances between any two vertices. Given a graph $G$, what is the diameter of the graph $G$? The second kind of problem is an optimization problem.
In the case of the example below, 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.

Definition 1.4.2. 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.

Definition 1.4.1. A grammar consists of an alphabet A of symbols, including a start symbol s'.
Definition 14.4. Given two languages $L_1$ and $L_2$, the concatenation of $L_1$ and $L_2$, denoted $L_1L_2$, is the set of strings $xy$ such that $x \in L_1$ and $y \in L_2$.

We have the empty string (written $\varepsilon$) and the empty language $\emptyset$.

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

Languages consist of a subset of the strings of symbols in the alphabet.

Example. Let $A = \{ \varepsilon, s, q, c \}$ and let $R$ be the set of rules:

\[
\begin{align*}
q & \rightarrow q11, \quad q & \rightarrow 111q, \quad q & \rightarrow s \\
1 & \rightarrow 11q, \quad 1 & \rightarrow q10q, \quad 1 & \rightarrow s \\
0 & \rightarrow 10q, \quad 0 & \rightarrow q10q, \quad 0 & \rightarrow s \\
00 & \rightarrow q100q, \quad 00 & \rightarrow q000q, \quad 00 & \rightarrow s
\end{align*}
\]
Example Let \( S = \{ s', A, B, C \} \) with \( s \) the start state and \( C \) the termination state. Let the function \( f \) be

\[
M \quad \text{when beginning in the start state, and reading the strings in } T, \text{ will end in a termination state.}
\]

Definition 14.7. The language \( L \) accepted by a machine \( M \) consists of the strings over \( \Sigma \), such that \( f \) makes a transition to state \( f(s', a) \).

\[
\forall s \in S \quad A \times S \leftarrow \text{such that, when the machine is in state } s \text{ and is presented with symbol } a, \text{ the machine}
\]

Definition 14.6. 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 \( f \) that maps states to states.

When in doubt, we will assume that all strings are taken from

\[
\{ 0, 1 \}^* \subseteq \{ \varepsilon \} \subseteq \mathcal{L} \cup \mathcal{L} \cup \mathcal{L} \cup \mathcal{L} \cup \mathcal{L} ....
\]

Definition 14.5. The closure (Kleene closure, Kleene star) \( \mathcal{L}^* \) of a language \( \mathcal{L} \) is the set of all finite-length strings.
This machine is clearly the machine that accepts the language generated above.

and similarly for the second and third columns,

\[
V = (1^{13}V) f \quad \tilde{V} = (1^{14}V) f \quad \tilde{V} = (1^{15}V) f
\]

\[
V = (0^{13}V) f \quad \tilde{V} = (0^{14}V) f \quad \tilde{V} = (0^{15}V) f
\]

We could easily fix this by the following. Instead of the first column of transitions, we use

Note that we have abused the notation somewhat to permit us to read three symbols at once instead of one:

\[
\mathcal{C} = (111, B) f \quad \mathcal{B} = (111, V) f \quad V = (111, s) f
\]

\[
\mathcal{C} = (011, B) f \quad \mathcal{B} = (011, V) f \quad V = (011, s) f
\]

\[
\mathcal{C} = (101, B) f \quad \mathcal{B} = (101, V) f \quad V = (101, s) f
\]

\[
\mathcal{C} = (001, B) f \quad \mathcal{B} = (001, V) f \quad V = (001, s) f
\]

\[
\mathcal{C} = (110, B) f \quad \mathcal{B} = (110, V) f \quad V = (110, s) f
\]

\[
\mathcal{C} = (010, B) f \quad \mathcal{B} = (010, V) f \quad V = (010, s) f
\]

\[
\mathcal{C} = (100, B) f \quad \mathcal{B} = (100, V) f \quad V = (100, s) f
\]

\[
\mathcal{C} = (000, B) f \quad \mathcal{B} = (000, V) f \quad V = (000, s) f
\]
I is in \( P \) if and only if there exists an algorithm that decides \( I \) in polynomial time.

**Definition 14.13.** The set \( P \) of polynomial time problems is the set of languages such that a language \( L \) is decided in polynomial time by an algorithm if there is a constant \( k \) such that every string in \( L \) of \( n \) symbols is either accepted or rejected by an \( O(n^k) \) steps.

**Definition 14.12.** A language \( L \) is decided in polynomial time by an algorithm if there is a constant \( k \) such that every string in \( L \) of \( n \) symbols is accepted by an \( O(n^k) \) steps.

**Definition 14.11.** A language \( L \) is accepted by an algorithm if every string in \( L \) is accepted by a \( A \) and every string not in \( L \) is rejected by \( A \).

**Definition 14.10.** A language \( L \) is decided by an algorithm if an algorithm \( A \) is the set of strings accepted by \( A \).

**Definition 14.9.** The language accepted by an algorithm \( A \) is the set of strings the algorithm reads, one of which corresponds to the 1 and one to the 0.

**Definition 14.8.** An algorithm \( A \) accepts a string \( x \in \mathcal{X} \) if, given input \( x \) \( \in \mathcal{X} \) and outputs \( 0 \), the algorithm reads a string \( x \) if it rejects a string \( x \) \( \in \mathcal{X} \), the algorithm reads a string \( x \) if it accepts a string \( x \) \( \in \mathcal{X} \).

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.
With reflection, the simulation does not terminate with what would be acceptance by $A$, then $N$ terminates of size $n$. If the simulation terminates with what would be acceptance by $A$, then $N$ terminates with $C_n$ steps. What we do is create an algorithm $N$ that simulates algorithm $A$ for $C_n$ time steps on inputs of $A$ that is accepted in $T$ time, then there is a constant $C$ such that any string in $T$ is accepted in $T$ time if it is rejected in $T$ time.

If $N$ accepts in $T$ time, then $A$ is accepted in $T$ time.

Proof: Clearly if $N$ is rejected in $T$ time, then $A$ is rejected in $T$ time.

Theorem 1.14. The set of polynomial time problems is the set of languages such that a
On the other hand, a specific cycle that would include all vertices has size \( u \), and thus to verify that a Hamiltonian cycle was Hamiltonian would take only polynomially many steps in the input size.

For no doubly check not polynomial in \( u \). So by the definition of a graph is polynomial in the number of vertices, and \( u \) by shrinking edges (for some constraint \( c \)) will have \( u \) permutations of the vertices. Clearly, then, we would need to look at a graph \( G \) with \( u \) vertices and \( c \).

Consider the naive way to look at the Hamiltonian cycle problem. A graph \( G \) with \( u \) vertices and \( c \)

The proof of this is deferred until later.

**Theorem 14.16.** The problem of determining whether a graph is Hamiltonian is \( \mathbf{NP} \)-complete.

**Definition 14.15.** Given a graph \( G \), a Hamiltonian cycle is a simple cycle that passes through every vertex exactly once.
between the cities (edge weights), find a tour of total weight less than \( W \) that includes all the cities.

**Definition 1.4.9.** The Traveling Salesman problem is: Given a set of cities (vertices), and distances reducibility of Hamiltonian cycles to the Traveling Salesman problem.

Let's look more carefully at the question of reducibility. To get a flavor for the arguments involved, we'll prove

1.3. Reducibility

It is generally believed that \( \text{NP} \neq \text{P} \), but this is not known.

Clearly also \( \text{P} \subseteq \text{NP} \), since deciding a language is more stringent than verifying a language.

Clearly, the Hamiltonian-cyclic problem is in \( \text{NP} \).

a polynomial time algorithm.

**Definition 1.4.18.** The complexity class \( \text{NP} \) consists of the class of languages that can be verified by

\[
\{ 1 = (\exists X, Y \in \{ 0, 1 \}^* : Y \in X) : X \in \text{L} \} = \text{NP}.
\]

otherwise, The language verified by a verification algorithm is

encoding string \( x \) and a certificate string \( y \) and outputs \( 1 \) if \( y \) is a solution to the problem and 0

**Definition 1.4.17.** A verification algorithm is an algorithm \( A \) which takes as input a problem and
Definition 14.21. A language $L$ is polynomial-time reducible to a language $L'$, which we will denote $L \leq \text{poly} L'$, if there exists a polytime function $f : \Sigma^* \rightarrow \Sigma^*$ such that for all $x \in \Sigma^*$, we have $x \in L$ if and only if $f(x) \in L'$.

To prove the theorem, it remains only to prove/observe that the creation of $\mathcal{C}$ takes time that is polynomial.

Cycle. We now run the TSP algorithm $T$ on $G$. Looking for a tour of weight $n$. That tour must be a Hamiltonian cycle.

For every part of vertices $n$, in $G$ for which there is no edge, we create an edge of weight 2 in $G$.

For every edge of $G$, in $G$ we create an edge of weight 1 in $G$.

For every vertex $v$ in $G$, we create a graph $G_v$. The vertices of $G_v$ correspond to the vertices of $G$. The vertices of $G_v$ correspond to the vertices of $G$. Given a graph $G$, the TSP is $NP$-complete, then so is Hamiltonian cycle.
solvable in poly time.

complete problem can be proved not to be solvable in poly time, then no NP-complete problem can be

Theorem 14.25. If any NP-complete problem is solvable in poly time, then $P = NP$. If any NP-

Definition 14.24. A language $L$ is NP-complete if it is NP-hard and if $L \in NP$.

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

Definition 14.23. A language $L$ is NP-hard if for every $T \in NP$ we have $T \geq_{p} L$.

Theorem 14.22. If $T_1, T_2 \in NP$ are languages such that $T_1 \geq_{p} T_2$, then $T_2 \in P$ implies that $T_1 \in P$.

called the reduction algorithm.

The function $f$ will be called the reduction function and the algorithm that computes $f$ will be

if and only if $f(x) \in T_2$. 

Definition 14.3.2. A feedback edge set is a subset $F \subseteq E$ such that every cycle of $G$ contains an edge in $F$.

Definition 14.3.1. A feedback vertex set is a subset $S \subseteq V$ such that every cycle of $G$ contains a vertex in $S$.

Definition 14.3.0. The chromatic number of a graph $G$ is the smallest $k$ such that $G$ is $k$-colorable.

The vertices of $G$ such that no two adjacent vertices have the same color.

Definition 14.2.9. A graph $G$ is $k$-colorable if there exists an assignment of the "colors" 1, 2, ..., $k$ to incident on some vertex of $S$.

Definition 14.2.8. 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 14.2.7. A $k$-clique in a graph $G$ is a subgraph of $G$ with $k$ vertices that is a complete graph.

Definition 14.2.6. A complete graph is a graph $G$ in which every pair of vertices is joined by an edge.

Definition 14.4. Some NP-Complete Problems
Definition 14.34. 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 14.35. The following problems are all $\text{NP}$-complete.

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

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?

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

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

Definition 14.33. A set cover for a family of sets $S_1, S_2, \ldots, S_n$ is a subfamily of $k$ sets $S_1, S_2, \ldots, S_k$ such that
We will prove this and the preceding theorem in order by reducing each problem to a previous problem.

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

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

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

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

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

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

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

**Theorem 14.36.** The following problems are all NP-complete.
Now having skipped the hard part we can continue with 14.35 part (3).

Theorem 14.38. The satisfiability problem for Boolean expressions in CNF is NP-complete.

Of the previous theorem, so we will defer this one also.

Theorem 14.37. The satisfiability problem is NP-complete.

We will start with Theorem 14.35, part (1) but we won’t prove it (yet).

\[
\begin{array}{c}
\text{feedback vs set} \\
\text{feedback edge set} \\
\text{unit ham cycle} \\
\text{(excat cover)} \\
\text{coverability} \\
\text{(3-SAT)} \\
\text{CNF-SAT} \\
\text{(SAT)}
\end{array}
\]
Thus, if either $x < -\frac{1}{2}$ or $x > \frac{1}{2}$, we have $y = 1$ and $\frac{1}{2} - y > 0$, then for some $i$, $i = 1$. If $y = 1$, then either $x \geq \frac{1}{2}$ or $x \leq -\frac{1}{2}$. Then the replacement expression has value 1. If $y = 0$, then either $x > \frac{1}{2}$ or $x < -\frac{1}{2}$. Let's set $y = 1$.

This is true for the following reason. The original expression is 1 if and only if some variable $x_i = 1$. If so, expression (2) is 1 for some assignment of the variables.

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 new variables.

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 new variables.

\[(2) \quad (x - y_i + v x + 1 - y_i x)(x - 2y_i + v x + 2 - y_i x) \cdots (x - y_i + v x + 2y_i + x)(1x + x + 1x)\]

For $x > 1$, we can replace any such sum by

\[(1) \quad x + \cdots + 2x + 1x\]

If $x < \frac{1}{2}$, we have a product of sums of the form $3-\frac{1}{2}y_i$. Assume that we have a product of sums of the form $3-\frac{1}{2}y_i$. Assume that we have a product of sums of the form $3-\frac{1}{2}y_i$. Assume that we have a product of sums of the form $3-\frac{1}{2}y_i$.

Proof: We will reduce any SAT problem in $\text{CNF}$ to a SAT problem in which the conflict terms have been reduced.

Theorem 14.39. The 3-SAT problem is $\text{NP}$-complete.
If \( I = 0 \), then \( 0 = 1 \). If \( I = 1 \), then \( 1 = \bar{I} \). Since the first factor is \( I \), and \( \bar{I} \) is SAT, if all middle terms we have \( x ") then start with \( x \" I \), then is SAT.

With any of these assignments, \( \bar{C} \) SAT. Conversely, \( \bar{C} \) SAT if and only if all factors equal \( I \).

If this is \( x \" x \" I \), we set \( \bar{I} = 0 \). If this is \( x \" x \" I \), we set \( \bar{I} = 1 \). For all \( x \" x \" \) or \( x \" x \" I \), then we set \( \bar{I} = 0 \). For all \( x \" x \" \) or \( x \" x \" I \), then we set \( \bar{I} = 1 \).

Now \( \bar{C} \) SAT if and only if some \( x \" \) equals \( I \).

\[
(3I + 6x + 3x)(3I + \bar{I} + 4x)(\bar{I} + I + 3x + 1x = C
\]

Example: Let \( C \) SAT. Replace \( C \) SAT. We prove \( C \) SAT. Possible in a poly number of transformation steps.

\[\square\]
2. \( \forall i \neq j \) such that \( \exists \) \( f(x_i, x_j) \) and \( \forall i \neq j \) such that \( \exists \) \( f(x_i, x_j) \) such that \( \exists \) \( f(x_i, x_j) \).

The edges are

We now build a graph. The vertices are

We assume we can assume \( n > 4 \).

Let \( x_1, x_2, \ldots, x_n \) and \( P_1, P_2, \ldots, P_n \) be the variables and the factors of \( \mathcal{F} \). Let \( v_1, v_2, \ldots, v_n \) be the new symbols.

If \( \mathcal{A}(x_1, x_2, \ldots, x_n) \) is satisfiable, the unweighted graph \( (\mathcal{A}, E') \) with \( 3n + t \) vertices that can be colored with \( n + 1 \) colors if and only if \( \mathcal{A}(x_1, x_2, \ldots, x_n) \) is satisfiable.

Given an expression \( \mathcal{A}(x_1, x_2, \ldots, x_n) \) in 3-CNF with \( n \) variables and \( t \) factors, we will construct in time poly in \( n + t \) a graph \( \mathcal{G} \) with \( 3n + t \) vertices and \( n + 1 \) colors, such that \( \mathcal{A}(x_1, x_2, \ldots, x_n) \) is satisfiable if and only if \( \mathcal{G} \) is \( (n + 1) \)-colorable.

**Theorem 14.40.** The problem of finding a \( k \)-coloring of an arbitrary graph is \( \mathsf{NP} \)-complete.

We move on to Theorem 14.35, part (4).
Otherwise, we will need a new color.

If $\mathcal{H}$ contains some literal $\phi$ with which $\phi$ has been assigned the special color, then $\phi$ is not adjacent to any vertex colored the same as $\phi$ and hence can be assigned the same color as $\phi$.

If $\mathcal{H}$ contains some literal $\phi$ with which $\phi$ cannot be colored with this special color, these two is colored with the special color, $\mathcal{H}$ is not adjacent to both $x$ and $x'$. Since one of $\mathcal{H}$ vertices, $\phi$ is adjacent to at least $2n - 3$ of the vertices $x$ and $x'$.

Consider the vertex with the special color to have been assigned the value 0. Now consider the color

the other is a new special color.

Thus cannot be colored with as few as $u + 1$ colors unless one of $x$, or $x'$ is the same color as $\phi$ and $\phi$. If $x$, or $x'$ is the same color as $\phi$ and $\phi$ is not adjacent to both $x$ and $x'$, they cannot be the same color.

Thus $\mathcal{H}$ cannot be colored with $u$ colors if $x$, or $x'$ is connected to each $\phi$ for $\phi \neq x$ or $x'$.

The vertices $x_1, x_2, \ldots, x_n$ form a complete subgraph of $n$ vertices. Any coloring must therefore require at

3. $u \geq n$ for $\phi$. 
4. $\phi$ is not a term of a factor.

5. $\phi$ is not a term of a factor.

$x, y \in \mathcal{H}$
This gives the following table:

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( A ) or ( B )</th>
<th>( C )</th>
<th>( A ) and ( B )</th>
<th>( C )</th>
<th>( \neg A ) \lor ( \neg B ) \lor ( \neg C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Example: Consider \( \neg x \lor \neg y \lor \neg z \) is satisfiable.

\( \neg x \lor \neg y \lor \neg z \) is satisfiable if and only if there is an assignment of the special color to the literals such that each factor contains only literals for which \( y \) has been assigned.

Thus, all the \( \neg y \) can be colored with no additional colors if and only if there is in an assignment of the special color, that is, if and only if one can assign values to the variables so that each factor contains the special color; and if \( y \) is assigned the value 1, then \( \neg y \) is assigned the value 0.
\[ S \text{ is a family of sets, and } n \text{ is selected unless it is distinct from all other selected sets.} \]

\[ x \notin S \implies (x, a) \neq 0 \quad \text{And if } x \text{ and } y \text{ are not adjacent, then } S \text{ is an edge, then } C \text{ is not} \]

\[ \not\exists \quad (m, a) = \left( \bigcup_{x \in S} a, a \right) \]

\[ \text{If } C \text{ forms an exact cover, then the collection of sets } S \text{ for each } a \text{ and the singleton sets } a \text{ such that } S \text{ is a } \}

\[ (a, a) \neq \emptyset \text{ for any } a \text{ is } \mathcal{P} \quad \text{We relate } k \text{-colorings to exact covers as follows. Let } C \text{ have a } k \text{-coloring with a colored } C_a \]

\[ \{[i, e]\} = \mathcal{P}_E \quad \text{For each } e \in E \text{ and for each } i \text{ and } \}

\[ \{x, a\} = e \in \cup \{x \in [i, e] \cap \{a\} = S \}

\[ \text{For each } e \in E \text{ and for each } i \text{ and } \}

\[ \{y \geq i \geq 1 \text{ and } \}

\[ \text{chosen from} \]

\[ \text{Proof. Let } C \text{ be an undirected graph. Let } k \text{ be an integer. We construct sets whose elements are} \]

\[ \text{exact cover problem is } \mathcal{N}P \text{-complete.} \]

Theorem 14.41. The colorability problem is polynomial-time solvable if the exact cover problem problem. Thus the

Now part (g) of Theorem 14.35.
Example. Consider a set of colored objects. Each set contains exactly one object from each color. Let us assume that set $S$ contains objects of colors $c_1, c_2, c_3, \ldots, c_n$. Then, $S$ is a valid cover if for each color $c_i$, there is exactly one object of color $c_i$ in $S$. Otherwise, there exists a color $c_j$ such that there is no object of color $c_j$ in $S$. We claim that if $S$ contains an object of each color, then $S$ is a valid cover. Conversely, suppose an exact cover exists. Then, for each color $c_i$, there is exactly one object of color $c_i$ in $S$. Therefore, $S$ is a valid cover.
\{b \leq y \leq 1 : \emptyset, y\} \}

We claim that the set of vertices has at least one literal assigned the value 1. Let such a literal be \(x^y\).

Assume that \(J\) is satisfiable. Then we can assign values to the literals so that \(J = I\). Thus each factor \(J\) exists.

We now claim that \(J\) has a clique of size \(b\) if and only if \(J\) is satisfiable.

There are no more vertices in \(J\) than factors in \(J\). The number of edges is \(\geq |J|\).

To the variables in the factors so that both literals have the value 1 (so both factors can be 1).

Initially, two vertices are adjacent in \(J\) if they correspond to different factors and one can assign values

\[x^y \neq f^y \text{ and } y \neq i \text{ such that } f \neq i \text{ and } f^y \neq i^y\] The graph has an edge \(e \in E(J)\) for \(f \neq i\). The component of the pair "a factor and the second component of a literal in that factor. The

We will construct a graph whose vertices are pairs of integers \([f, y]\) for \(f \neq i\), \(b \geq y \geq 1\) and \(b \geq i \geq 1\).

\[y^x + \cdots + \bar{a}^x + \bar{b}^x = I\]

**Proof:** Let \(f^y \in F\) be an expression in CNF, with \(b\) \(F\) is \(NP\)-complete.

**Theorem 14.42:** CNF-SAT is poly-time transformable into the clique problem. Therefore the clique problem...
And clearly the transformation is polytime.

So \( F \) is satisfiable.

in \( S^2 \) to \( 0 \), the value of every variable \( F \) is set to 1.

variables. So \( S^1 \cup S^2 \).

Then \( S^1 \) and \( S^2 \) are the sets of complemented and uncomplemented

for \( I \) and \( b \geq i \geq 1 \). Let \( \{ h = \mu_{m,x} : h \} = S^2 \). Let \( \{ h = \mu_{m,x} : h \} = S^1 \).

Since there are \( b \) vertices, there is a 1-I correspondence between vertices and the factors of \( F \).

connected.

Conversely, assume that \( F \) has a clique of size \( b \). 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

formed a clique of size \( b \). Otherwise, we have \( i \) and \( j \) with \( i \neq j \) with no edge between vertices

form a clique of size \( b \). Otherwise, we have \( i \) and \( j \) with no edge between vertices

this is impossible, since there must have a distinct first

Thus, it is impossible, since there must have a distinct first
There are 3-cliques $11, 31$ and $12, 22, 32$.

\[
\{ (\delta f, \xi f), (\xi f, \eta f), (\eta f, \gamma f) \} = \mathcal{A}
\]

\[
\{ [\xi, \eta] = \delta f, [\zeta, \eta] = \xi f, [\zeta, \xi] = \eta f, [\xi, \xi] = \eta f \} = \Lambda
\]

\[
(\eta f + \xi f)(\delta f + \xi f)(\gamma f + \eta f) = \mathcal{A}
\]

**Example:** Let \( \mathcal{C} \) have a clique of size 0 if and only if \( \mathcal{A} \) is SAT.

That is, we want an edge if the vertices are from different facets and one can make an edge by defining \( \mathcal{H} = \mathcal{H} = \mathcal{H} \).

\[
\{ \eta f \neq \xi f, f \neq r : (r, f) \in [r, q] \} = \mathcal{A}
\]

\[
\{ t q > f > g > b > g > f : [f, q] \} = \mathcal{A}
\]

**Example:** We have \( \mathcal{H} = \mathcal{H} = \mathcal{H} \).
not 32. Similarly, if 12, then we must have 31 and 41, but 31 $\leftarrow$ 42 and not 41.

block. But we can't do 11 and 12 and two others. If 11, then we must have 22 and 32, but 22 $\leftarrow$ 31 and

We will need one from each block, that is for each first coordinate, to get a clique, since have a 00 inside each

\[
\begin{array}{cccc|cccc|c}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 42 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 41 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 32 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 31 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 22 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 21 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 12 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & \text{Not SAT} \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
(\varphi_i + \varphi_i)(\varphi_i + \varphi_i)(\varphi_i + \varphi_i)(\varphi_i + \varphi_i) = \mathcal{A}
\]
\[
I = \bar{\alpha} = \bar{\beta} = \bar{\gamma}
\]

\[
\{\alpha, \beta\} = \bar{\gamma}
\]

\[
\{\bar{\gamma}\} = \bar{\gamma}
\]

There is a clique \(\{11, 21, 31, 43\}\) and the line.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th></th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 1 0 1 0 1 1 0 0 0 1 0 0 1 1
0 0 0 1 1 1 1 0 0 0 1
1 0 0 0 1 1 1 0 1 0 1
0 1 1 1 0 0 0 1 0 1 0
1 1 0 0 0 1 0 1 1 1 0
0 0 1 1 1 1 1 0 0 0 0
1 0 1 1 1 1 1 1 1 1 0
0 0 0 1 1 1 1 0 0 0 0
1 0 1 1 1 1 1 1 1 1 0
1 0 1 1 1 1 1 1 1 1 0
1 0 1 1 1 1 1 1 1 1 0
(\bar{\beta} + \alpha \beta + \beta \bar{\beta}) (\bar{\beta} + \alpha \beta + \beta \bar{\beta}) (\alpha \beta + \alpha \beta + \beta \bar{\beta}) (\alpha \beta + \alpha \beta + \beta \bar{\beta})

A more complicated example:
Since every \( E \in G(n, \alpha) : (a, a, (a, a)) = E \), where \( (E,\Lambda) = \Lambda \) is the graph obtained by replacing each edge

**Proof.** Let \( G \) be an undirected graph. Let \( \Lambda \) be the directed graph obtained by replacing each edge

Therefore, the feedback vertex set problem is \( \mathbb{NP} \)-complete.

**Theorem 14.43.** The clique cover problem is poly transformable into the feedback vertex set problem.

\[ \square \]

And the transformation is poly time.

To decide the clique problem of size \( k \), construct \( G \) and decide whether \( G \) has a vertex of size \( k \).

This means \( \Lambda \) is a clique.

So no edge of \( G \) connects two vertices in \( \Lambda \). So every pair of vertices of \( \Lambda \) is connected in \( G \), and that \( S \setminus \Lambda \) is a vertex cover, then every edge of \( S \setminus \Lambda \) is incident upon at least one vertex of \( S \).

Similarly, \( S \setminus \Lambda \) is a vertex cover, then every edge of \( S \setminus \Lambda \) is incident upon at least one vertex of \( S \).

**Claim:** A set \( \Lambda \subseteq S \) is a clique if \( \Lambda \subseteq S \) and only if \( \Lambda \subseteq S \) is a vertex cover of \( G \).

**Proof:** Consider the complement of a graph \( G \) with edges exactly where \( G \) does not have edges.

Therefore, the clique cover problem is \( \mathbb{NP} \)-complete.
consists of edges.

\[ \mathcal{F} \] be an undirected graph where \[ \mathcal{F} = \{ 0,1,2 \} \times \{ \Lambda \} = \mathcal{G} \cup \mathcal{G}^{\prime} \] is a directed graph. Let \[ \mathcal{G} = \{ 0,1 \} \times \{ \Lambda \} \] be a directed graph.

**Proof.** Let \( G = (\mathcal{F}, \mathcal{E}) \) be a directed graph. Let \( \mathcal{E} \) be an undirected graph. Therefore, the Hamiltonian cycle problem for undirected graphs is \( NP \)-complete.

**Theorem 1.4.7.** The Hamiltonian cycle problem for directed graphs is polytranslatable into the Hamiltonian cycle problem.

\[ \square \]

**Proof.**

Therefore, the Hamiltonian cycle problem for directed graphs is \( NP \)-complete.

**Theorem 1.4.6.** The vertex cover problem is polytranslatable into the Hamiltonian cycle problem.

\[ \square \]

**Proof.**

Therefore, the feedback edge set problem is \( NP \)-complete.

**Theorem 1.4.5.** The vertex cover problem is polytranslatable into the feedback edge set problem.

\[ \square \]

**Proof.**

And the transformation is clearly poly time.

Vertex cover for \( G \) 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 edge in \( G \).
\[ \square \text{ Clearly, } S \subseteq \mathcal{E} \Rightarrow \{ y_i \} y_i \text{ is a vertex cover for } G. \]

\[ \square \text{ Therefore, the set cover problem is } \mathcal{NP}\text{-complete.} \]

Theorem 4.48. The vertex cover problem is poly-time transformable into the set cover problem. Therefore:

\[ \square \]

Let \( \mathcal{A} \subseteq \mathbb{Z} \cup \mathbb{Z} \) in \([0, m] \cup [2, a] \cup [1, a] \cup [0, a] \)

\[ \square \]