34.4-7
Let 2-CNF-SAT be the set of satisfiable boolean formulas in CNF with exactly 2 literals per clause. Show that 2-CNF-SAT ∈ P. Make your algorithm as efficient as possible. (*Hint: Observe that \( x \lor y \) is equivalent to \( \neg x \rightarrow y \). Reduce 2-CNF-SAT to a problem on a directed graph that is efficiently solvable.)*

34.5 NP-complete problems

NP-complete problems arise in diverse domains: boolean logic, graphs, arithmetic, network design, sets and partitions, storage and retrieval, sequencing and scheduling, mathematical programming, algebra and number theory, games and puzzles, automata and language theory, program optimization, biology, chemistry, physics, and more. In this section, we shall use the reduction methodology to provide NP-completeness proofs for a variety of problems drawn from graph theory and set partitioning.

Figure 34.13 outlines the structure of the NP-completeness proofs in this section and Section 34.4. Each language in the figure is proved NP-complete by reduction from the language that points to it. At the root is CIRCUIT-SAT, which we proved NP-complete in Theorem 34.7.

34.5.1 The clique problem

A clique in an undirected graph \( G = (V, E) \) is a subset \( V' \subseteq V \) of vertices, each pair of which is connected by an edge in \( E \). In other words, a clique is a complete subgraph of \( G \). The size of a clique is the number of vertices it contains. The clique problem is the optimization problem of finding a clique of maximum size in a graph. As a decision problem, we ask simply whether a clique of a given size \( k \) exists in the graph. The formal definition is

\[
\text{CLIQUE} = \{ (G, k) : G \text{ is a graph with a clique of size } k \}
\]

A naive algorithm for determining whether a graph \( G = (V, E) \) with \( |V| \) vertices has a clique of size \( k \) is to list all \( k \)-subsets of \( V \), and check each one to see whether it forms a clique. The running time of this algorithm is \( \Omega(k^2 \binom{|V|}{k}) \), which is polynomial if \( k \) is a constant. In general, however, \( k \) could be near \( |V|/2 \), in which case the algorithm runs in superpolynomial time. As one might suspect, an efficient algorithm for the clique problem is unlikely to exist.

*Theorem 34.11*

The clique problem is NP-complete.
Proof  To show that CLIQUE $\in$ NP, for a given graph $G = (V, E)$, we use the set $V' \subseteq V$ of vertices in the clique as a certificate for $G$. Checking whether $V'$ is a clique can be accomplished in polynomial time by checking whether, for each pair $u, v \in V'$, the edge $(u, v)$ belongs to $E$.

We next prove that 3-CNF-SAT $\leq_P$ CLIQUE, which shows that the clique problem is NP-hard. That we should be able to prove this result is somewhat surprising, since on the surface logical formulas seem to have little to do with graphs.

The reduction algorithm begins with an instance of 3-CNF-SAT. Let $\phi = C_1 \land C_2 \land \cdots \land C_k$ be a boolean formula in 3-CNF with $k$ clauses. For $r = 1, 2, \ldots, k$, each clause $C_r$ has exactly three distinct literals $l_{r1}', l_{r2}',$ and $l_{r3}'$. We shall construct a graph $G$ such that $\phi$ is satisfiable if and only if $G$ has a clique of size $k$.

The graph $G = (V, E)$ is constructed as follows. For each clause $C_r = (l_{r1}' \lor l_{r2}' \lor l_{r3}')$ in $\phi$, we place a triple of vertices $v_{r1}', v_{r2}',$ and $v_{r3}'$ into $V$. We put an edge between two vertices $v_{r1}'$ and $v_{r2}'$ if both of the following hold:

- $v_{r1}'$ and $v_{r2}'$ are in different triples, that is, $r \neq s$, and
- their corresponding literals are consistent, that is, $l_{r1}'$ is not the negation of $l_{s1}'$.

This graph can easily be computed from $\phi$ in polynomial time. As an example of this construction, if we have

$$\phi = (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_3),$$

then $G$ is the graph shown in Figure 34.14.
Figure 34.14  The graph $G$ derived from the 3-CNF formula $\phi = C_1 \land C_2 \land C_3$, where $C_1 = (x_1 \lor \neg x_2 \lor \neg x_3)$, $C_2 = (\neg x_1 \lor x_2 \lor x_3)$, and $C_3 = (x_1 \lor x_2 \lor x_3)$, in reducing 3-CNF-SAT to CLIQUE. A satisfying assignment of the formula has $x_2 = 0$, $x_3 = 1$, and $x_1$ may be either 0 or 1. This assignment satisfies $C_1$ with $\neg x_2$, and it satisfies $C_2$ and $C_3$ with $x_3$, corresponding to the clique with lightly shaded vertices.

We must show that this transformation of $\phi$ into $G$ is a reduction. First, suppose that $\phi$ has a satisfying assignment. Then each clause $C_i$ contains at least one literal $l'_r$ that is assigned 1, and each such literal corresponds to a vertex $v'_r$. Picking one such "true" literal from each clause yields a set $V'$ of $k$ vertices. We claim that $V'$ is a clique. For any two vertices $v'_r, v'_s \in V'$, where $r \neq s$, both corresponding literals $l'_r$ and $l'_s$ are mapped to 1 by the given satisfying assignment, and thus the literals cannot be complements. Thus, by the construction of $G$, the edge $(v'_r, v'_s)$ belongs to $E$.

Conversely, suppose that $G$ has a clique $V'$ of size $k$. No edges in $G$ connect vertices in the same triple, and so $V'$ contains exactly one vertex per triple. We can assign 1 to each literal $l'_r$ such that $v'_r \in V'$ without fear of assigning 1 to both a literal and its complement, since $G$ contains no edges between inconsistent literals. Each clause is satisfied, and so $\phi$ is satisfied. (Any variables that do not correspond to a vertex in the clique may be set arbitrarily.)

In the example of Figure 34.14, a satisfying assignment of $\phi$ has $x_2 = 0$ and $x_3 = 1$. A corresponding clique of size $k = 3$ consists of the vertices corresponding to $\neg x_2$ from the first clause, $x_3$ from the second clause, and $x_1$ from the third clause. Because the clique contains no vertices corresponding to either $x_1$ or $\neg x_1$, we can set $x_1$ to either 0 or 1 in this satisfying assignment.
Observe that in the proof of Theorem 34.11, we reduced an arbitrary instance of
3-CNF-SAT to an instance of CLIQUE with a particular structure. It might seem
that we have shown only that CLIQUE is NP-hard in graphs in which the vertices
are restricted to occur in triples and in which there are no edges between vertices
in the same triple. Indeed, we have shown that CLIQUE is NP-hard only in this
restricted case, but this proof suffices to show that CLIQUE is NP-hard in general
graphs. Why? If we had a polynomial-time algorithm that solved CLIQUE on
general graphs, it would also solve CLIQUE on restricted graphs.

It would not have been sufficient, however, to reduce instances of 3-CNF-SAT
with a special structure to general instances of CLIQUE. Why? It might have been
the case that the instances of 3-CNF-SAT we chose to reduce from were “easy,”
and so we would not have reduced an NP-hard problem to CLIQUE.

Observe also that the reduction used the instance of 3-CNF-SAT but not the
solution. It would have been a mistake for the polynomial-time reduction to have
been based on knowing whether the formula \( \phi \) is satisfiable, since we do not know
how to determine this information in polynomial time.

### 34.5.2 The vertex-cover problem

A vertex cover of an undirected graph \( G = (V, E) \) is a subset \( V' \subseteq V \) such that
if \((u, v) \in E\), then \(u \in V' \) or \(v \in V'\) (or both). That is, each vertex “covers” its
incident edges, and a vertex cover for \( G \) is a set of vertices that covers all the edges
in \( E \). The size of a vertex cover is the number of vertices in it. For example, the
graph in Figure 34.15(b) has a vertex cover \( \{w, z\} \) of size 2.

The vertex-cover problem is to find a vertex cover of minimum size in a given
graph. Restating this optimization problem as a decision problem, we wish to
determine whether a graph has a vertex cover of a given size \( k \). As a language, we
define

\[
\text{VERTEX-COVER} = \{(G, k) : \text{graph } G \text{ has a vertex cover of size } k\}
\]

The following theorem shows that this problem is NP-complete.

**Theorem 34.12**
The vertex-cover problem is NP-complete.

**Proof** We first show that VERTEX-COVER \( \in \) NP. Suppose we are given a graph
\( G = (V, E) \) and an integer \( k \). The certificate we choose is the vertex cover \( V' \subseteq V \)
itselt. The verification algorithm affirms that \( |V'| = k \), and then it checks, for
each edge \((u, v) \in E\), that \(u \in V' \) or \(v \in V'\). This verification can be performed
straightforwardly in polynomial time.

We prove that the vertex-cover problem is NP-hard by showing that CLIQUE \( \leq_p \)
VERTEX-COVER. This reduction is based on the notion of the “complement” of