34.2 Polynomial-time verification

We now look at algorithms that "verify" membership in languages. For example, suppose that for a given instance \( \langle G, u, v, k \rangle \) of the decision problem PATH, we are also given a path \( p \) from \( u \) to \( v \). We can easily check whether the length of \( p \) is at most \( k \), and if so, we can view \( p \) as a "certificate" that the instance indeed belongs to PATH. For the decision problem PATH, this certificate doesn't seem to buy us much. After all, PATH belongs to P--in fact, PATH can be solved in linear time--and so verifying membership from a given certificate takes as long as solving the problem from scratch. We shall now examine a problem for which we know of no polynomial-time decision algorithm yet, given a certificate, verification is easy.

Hamiltonian cycles

The problem of finding a hamiltonian cycle in an undirected graph has been studied for over a hundred years. Formally, a hamiltonian cycle of an undirected graph \( G = (V, E) \) is a simple cycle that contains each vertex in \( V \). A graph that contains a hamiltonian cycle is said to be hamiltonian; otherwise, it is nonhamiltonian. Bondy and Murty [45] cite a letter by W. R. Hamilton describing a mathematical game on the dodecahedron (Figure 34.2 (a)) in which one player sticks five pins in any five consecutive vertices and the other player must complete the path to form a cycle containing all the vertices. The dodecahedron is hamiltonian, and Figure 34.2(a) shows one hamiltonian cycle. Not all graphs are hamiltonian, however. For example, Figure 34.2(b) shows a bipartite graph with an odd number of vertices. Exercise 34.2-2 asks you to show that all such graphs are nonhamiltonian.

We can define the hamiltonian-cycle problem, "Does a graph \( G \) have a hamiltonian cycle?" as a formal language:

\[
\text{HAM-CYCLE} = \{ \langle G \rangle : G \text{ is a hamiltonian graph} \}.
\]

How might an algorithm decide the language HAM-CYCLE? Given a problem instance \( \langle G \rangle \), one possible decision algorithm lists all permutations of the vertices of \( G \) and then checks each permutation to see if it is a hamiltonian path. What is the running time of this algorithm? If we use the "reasonable"
Encoding of a graph as its adjacency matrix, the number \( m \) of vertices in the
graph is \( \Omega(\sqrt{n}) \), where \( n = |\langle G \rangle| \) is the length of the encoding of \( G \). There
are \( m! \) possible permutations of the vertices, and therefore the running time
is \( \Omega(m!) = \Omega(\sqrt{n}!) = \Omega(2^{\sqrt{n}}) \), which is not \( O(n^k) \) for any constant \( k \). Thus, this naive
algorithm does not run in polynomial time. In fact, the hamiltonian-cycle
problem is NP-complete, as we shall prove in Section 34.5.

Verification algorithms

Consider a slightly easier problem. Suppose that a friend tells you that a
given graph \( G \) is hamiltonian, and then offers to prove it by giving you the
vertices in order along the hamiltonian cycle. It would certainly be easy
enough to verify the proof: simply verify that the provided cycle is hamiltonian
by checking whether it is a permutation of the vertices of \( V \) and whether each
of the consecutive edges along the cycle actually exists in the graph. This
verification algorithm can certainly be implemented to run in \( O(n^2) \) time,
where \( n \) is the length of the encoding of \( G \). Thus, a proof that a hamiltonian
cycle exists in a graph can be verified in polynomial time.

We define a verification algorithm as being a two-argument algorithm \( A \),
where one argument is an ordinary input string \( x \) and the other is a binary
string \( y \) called a certificate. A two-argument algorithm \( A \) verifies an input
string \( x \) if there exists a certificate \( y \) such that \( A(x, y) = 1 \). The language
verified by a verification algorithm \( A \) is

\[
L = \{ x \in \{0, 1\}^* : \text{there exists } y \in \{0, 1\}^* \text{ such that } A(x, y) = 1 \}.
\]

Intuitively, an algorithm \( A \) verifies a language \( L \) if for any string \( x \in L \), there is
a certificate \( y \) that \( A \) can use to prove that \( x \in L \). Moreover, for any string \( x \notin L \),
there must be no certificate proving that \( x \in L \). For example, in the
hamiltonian-cycle problem, the certificate is the list of vertices in the
hamiltonian cycle. If a graph is hamiltonian, the hamiltonian cycle itself offers
enough information to verify this fact. Conversely, if a graph is not
hamiltonian, there is no list of vertices that can fool the verification algorithm
into believing that the graph is hamiltonian, since the verification algorithm
carefully checks the proposed "cycle" to be sure.

The complexity class NP

The complexity class \( \text{NP} \) is the class of languages that can be verified by a
polynomial-time algorithm. More precisely, a language \( L \) belongs to \( \text{NP} \) if
and only if there exist a two-input polynomial-time algorithm \( A \) and constant \( c \)
such that

\[
L = \{ x \in \{0, 1\}^* : \text{there exists } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1 \}.
\]

We say that algorithm \( A \) verifies language \( L \) in polynomial time.

From our earlier discussion on the hamiltonian-cycle problem, it follows that
\( \text{HAM-CYCLE} \in \text{NP} \). (It is always nice to know that an important set is
nonempty.) Moreover, if \( L \in \text{P} \), then \( L \in \text{NP} \), since if there is a polynomial-
time algorithm to decide \( L \), the algorithm can be easily converted to a two-
argument verification algorithm that simply ignores any certificate and
accepts exactly those input strings it determines to be in \( L \). Thus, \( \text{P} \subseteq \text{NP} \).

It is unknown whether \( \text{P} = \text{NP} \), but most researchers believe that \( \text{P} \) and \( \text{NP} \)
are not the same class. Intuitively, the class P consists of problems that can be solved quickly. The class NP consists of problems for which a solution can be verified quickly. You may have learned from experience that it is often more difficult to solve a problem from scratch than to verify a clearly presented solution, especially when working under time constraints. Theoretical computer scientists generally believe that this analogy extends to the classes P and NP, and thus that NP includes languages that are not in P.

There is more compelling evidence that $P \neq NP$—the existence of languages that are "NP-complete." We shall study this class in Section 34.3.

Many other fundamental questions beyond the $P \neq NP$ question remain unresolved. Despite much work by many researchers, no one even knows if the class NP is closed under complement. That is, does $L \in NP$ imply $\overline{L} \in NP$?

We can define the **complexity class co-NP** as the set of languages $L$ such that $\overline{L} \in NP$. The question of whether NP is closed under complement can be rephrased as whether NP = co-NP. Since P is closed under complement (Exercise 34.1-6), it follows that $P \subseteq NP \cup co-NP$. Once again, however, it is not known whether $P = NP \cup co-NP$ or whether there is some language in $NP \cup co-NP$ -P. Figure 34.3 shows the four possible scenarios.

![Figure 34.3: Four possibilities for relationships among complexity classes. In each diagram, one region enclosing another indicates a proper-subset relation. (a) $P = NP = co-NP$. Most researchers regard this possibility as the most unlikely. (b) If NP is closed under complement, then NP = co-NP, but it need not be the case that $P = NP$. (c) $P = NP \cup co-NP$, but NP is not closed under complement. (d) $NP \neq co-NP$ and $P \neq NP \cup co-NP$. Most researchers regard this possibility as the most likely.](image)

Thus, our understanding of the precise relationship between P and NP is woefully incomplete. Nevertheless, by exploring the theory of NP-completeness, we shall find that our disadvantage in proving problems to be intractable is, from a practical point of view, not nearly so great as we might suppose.

**Exercises 34.2-1**

Consider the language $GRAPH-ISOMORPHISM = \{ \langle G_1, G_2 \rangle : G_1$ and $G_2$ are isomorphic graphs\}. Prove that $GRAPH-ISOMORPHISM \in NP$ by describing a polynomial-time algorithm to verify the language.

**Exercises 34.2-2**
Prove that if \( G \) is an undirected bipartite graph with an odd number of vertices, then \( G \) is nonhamiltonian.

**Exercises 34.2-3**

Show that if HAM-CYCLE \( \in \text{P} \), then the problem of listing the vertices of a hamiltonian cycle, in order, is polynomial-time solvable.

**Exercises 34.2-4**

Prove that the class NP of languages is closed under union, intersection, concatenation, and Kleene star. Discuss the closure of NP under complement.

**Exercises 34.2-5**

Show that any language in NP can be decided by an algorithm running in time \( 2^{O(n)} \) for some constant \( k \).

**Exercises 34.2-6**

A **hamiltonian path** in a graph is a simple path that visits every vertex exactly once. Show that the language HAM-PATH = \( \{ \langle G, u, v \rangle : \text{there is a hamiltonian path from } u \text{ to } v \text{ in graph } G \} \) belongs to NP.

**Exercises 34.2-7**

Show that the hamiltonian-path problem can be solved in polynomial time on directed acyclic graphs. Give an efficient algorithm for the problem.

**Exercises 34.2-8**

Let \( \phi \) be a boolean formula constructed from the boolean input variables \( x_1, x_2, ..., x_k \), negations (\( \neg \)), AND's (\( \land \)), OR's (\( \lor \)), and parentheses. The formula \( \phi \) is a **tautology** if it evaluates to 1 for every assignment of 1 and 0 to the input variables. Define TAUTOLOGY as the language of boolean formulas that are tautologies. Show that TAUTOLOGY \( \in \text{co-NP} \).

**Exercises 34.2-9**

Prove that \( \text{P} \subseteq \text{co-NP} \).

**Exercises 34.2-10**

Prove that if \( \text{NP} \neq \text{co-NP} \), then \( \text{P} \neq \text{NP} \).

**Exercises 34.2-11**
Let $G$ be a connected, undirected graph with at least 3 vertices, and let $G^3$ be the graph obtained by connecting all pairs of vertices that are connected by a path in $G$ of length at most 3. Prove that $G^3$ is hamiltonian. (Hint: Construct a spanning tree for $G$, and use an inductive argument.)

The name "NP" stands for "nondeterministic polynomial time." The class NP was originally studied in the context of nondeterminism, but this book uses the somewhat simpler yet equivalent notion of verification. Hopcroft and Ullman [156] give a good presentation of NP-completeness in terms of nondeterministic models of computation.