# 7

# NP-Completeness

## 7.1 NP

The complexity class $NP$ contains thousands of natural problems, coming from diverse areas of graph theory, combinatorics, operations research, number theory and mathematical logic. We present a few examples in this section. In order to simplify the proofs of problems belonging to $NP$, we first give a simple characterization of the class $NP$, based on the notion of the guess-and-verify algorithm.

**Theorem 7.1** *A language $A \subseteq \Sigma^*$ is in NP if and only if there exist a language $B \subseteq \Sigma^*\#\Lambda^*$ in $P$, where $\Lambda$ is an alphabet and $\# \notin \Sigma \cup \Lambda$, and a polynomial function $p$, such that*

$$x \in A \iff (\exists y \in \Lambda^*)\,[|y| \leq p(|x|), x\#y \in B]. \tag{7.1}$$

*Proof.* First, assume that $A = L(M)$ for a one-tape NTM $M = (Q, \Gamma, \Sigma, \delta, q_0)$ with time bound $p(n)$ for some polynomial $p$. Also, assume that for any configuration of $M$, there are at most $k$ choices for the next move; that is, $|\delta(q, a)| \leq k$, for any pair $(q, a) \in Q \times \Gamma$. Let $\Lambda = \{\eta_1, \eta_2, \ldots, \eta_k\}$. Recall that, in Theorem 6.23, we have constructed a DTM $M^*$ to simulate an NTM $M$, using a string $y \in \Lambda^*$, written in the second tape, to decide which choice among the $k$ possible next moves to take. We observe that if $x \in A$, then there is a string $y \in \Lambda^*$ of length $p(|x|)$ such that $M^*$ accepts $x$ with respect to the string $y$ in the second tape. Conversely, if $x \notin A$, then $M^*$ never accepts $x$. In other words, let $B$ be the set of strings $x\#y \in \Sigma^*\#\Lambda^*$, with $|y| \leq p(|x|)$,

such that $M^*$ accepts $x$ with respect to $y$. Then, set $B$ is in $P$ and satisfies (7.1).

Conversely, if sets $A$, $B$, and function $p$ satisfy (7.1), then the following NTM $M$ accepts set $A$:

(1) On input $x$, $M$ guesses a string $y \in \Lambda^*$, with $|y| \leq p(|x|)$, in a separate tape.

(2) $M$ verifies whether $x\#y \in B$ or not. It accepts $x$ if $x\#y \in B$.

It is clear that the above NTM $M$ works in time $r(p(n))$ for some polynomial $r$. Since the composition of two polynomial functions is still a polynomial function, $M$ is a polynomial-time NTM. Therefore, $A$ belongs to $NP$. □

Suppose that sets $A$, $B$ and polynomial function $p$ satisfy (7.1), and that $x$ is a string in $A$. Then, we call a string $y \in \Lambda^*$, with length $|y| \leq p(|x|)$, a *witness* or a *certificate* of $x \in A$, if $x\#y \in B$.

It is interesting to compare the above characterization of the class $NP$ with the characterization of r.e. sets given in the Projection Theorem (Theorem 5.8). We have argued that the class $P$ is a mathematical formulation of feasibly computable sets. Thus, it is the counterpart of the class of recursive sets in complexity theory. The above theorem, then, suggests that the class $NP$ is the counterpart of the class of r.e. sets in complexity theory. Based on this analogy, people have attempted to apply the ideas and techniques of computability to attack the $P$ versus $NP$ problem.

Another interesting observation about the class $NP$ is that a great number of puzzles and games are just special instances of some famous problems in $NP$, as their solutions are usually hard to find but easy to verify. We will also present some of these puzzles in this section.

Our first example is a fundamental problem in Boolean algebra, called the *satisfiability* problem. In order to define this problem, let us first review some basic concepts in Boolean algebra.

A Boolean function is a function whose variable values and function value all are in {TRUE, FALSE}. For convenience, we denote TRUE by 1 and FALSE by 0. In the following table, we show three basic boolean functions: $\wedge$ (*conjunction*), $\vee$ (*disjunction*), and $\neg$ (*negation*), where $\wedge$ and $\vee$ are Boolean functions of two variables and $\neg$ is a Boolean function of one variable. Such a table is called a *truth-table*.

| $x$ | $y$ | $x \wedge y$ | $x \vee y$ | $\neg x$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

For simplicity, we also write $xy$ for $x \wedge y$, $x + y$ for $x \vee y$ and $\overline{x}$ for $\neg x$. It is easy to see that conjunction and disjunction follow the commutative,

associative, and distributive laws. An interesting and important law about negation is De Morgan's law: $\overline{x \cdot y} = \overline{x} + \overline{y}$ and $\overline{x + y} = \overline{x} \cdot \overline{y}$.

A Boolean formula is a formula over Boolean variables, with operations $\wedge$, $\vee$ and $\neg$. It represents a Boolean function in the same way that an arithmetic formula represents a function on numbers. It is not hard to see that all Boolean functions can be represented as a Boolean formula using these three basic operations.

We say a Boolean formula $F$ over variables $x_1, x_2, \ldots, x_n$ is *satisfiable* if there is an assignment $t : \{x_1, \ldots, x_n\} \to \{0, 1\}$ that makes $F$ true when we replace each variable $x_i$ by $t(x_i)$. For instance,

$$F_1(x_1, x_2, x_3) = (x_1x_2 + x_2\overline{x}_3 + \overline{x}_1x_3)(\overline{x}_1\overline{x}_2x_3 + x_1\overline{x}_2x_3)$$

is satisfiable, because $F_1 = 1$ by the assignment $t(x_1) = t(x_2) = 0$, and $t(x_3) = 1$; and

$$F_2(x_1, x_2, x_3) = (x_1x_2 + x_2\overline{x}_3 + \overline{x}_1\overline{x}_3)(\overline{x}_1\overline{x}_2x_3 + x_1\overline{x}_2x_3)$$

is unsatisfiable, because $F_2 = 0$ for all eight different assignments to $x_1, x_2$ and $x_3$. We call an assignment $t$ that makes $F$ true a *truth assignment*.

Now, we define the satisfiability problem:

> SATISFIABILITY (SAT): Given a Boolean formula $F$, determine whether $F$ is satisfiable or not.

In the following, for each problem $\Pi$ of the form "Given input $x$, determine whether $x$ has property $\pi$," we let $\Pi$ denote the set of inputs $x$ having property $\pi$. For instance, SAT denotes the set of all satisfiable Boolean formulas.

**Example 7.2** SAT *is in NP.*

*Proof.* A nondeterministic algorithm for SAT can be described in the guess-and-verify form as follows:

(1) On an input formula $F$ over variables $x_1, \ldots, x_n$, guess an assignment $t : \{x_1, \ldots, x_n\} \to \{0, 1\}$.

(2) Check whether $t$ satisfies $F$.

The above algorithm is obviously correct. We show that it can be implemented by a two-worktape NTM $M$ in polynomial time. First, the guessing step (1) can be implemented by writing down $n$ bits $t_1, t_2, \ldots, t_n \in \{0, 1\}$ in the second work tape. Then, in the verifying step (2), $M$ copies the formula $F$ to the first work tape, with each variable $x_i$ repaced by the $i$th bit $t_i$ in the second tape. It then evaluates the formula $F'$, and accepts the input if $F' = 1$. It is easy to see that the set of legal Boolean formulas is, like the set of arithmetic formulas, context-free. Therefore, the evaluation of a formula $F'$ with no variables can be done by parsing the formula and then evaluates

the parse tree, and can be done in polynomial time (cf. Example 3.24 and Exercise 6 of Section 6.2). □

Boolean formulas are a natural mathematical tool to encode logical statements. The following examples illustrate this idea.

**Example 7.3** *Formulate the following puzzle as a Boolean formula $F$, and solve it by determining whether $F$ is satisfiable:*

> *With five games left in the regular season, the team mathematician reported to the manager of the baseball team A: "We can still make the playoffs if we can beat team B, and in case team B beats team C then team C beats team D, and in case we lose to team C then team D also beats either team B or team C." Later, among the remaining five games, each team won at least one game, but no team won all its remaining games, and there was no tie game. Did team A have any chance to make the playoffs? If so, how?*

*Solution.* The five games left are apparently (1) A vs. B, (2) A vs. C, (3) B vs. C, (4) B vs. D, and (5) C vs. D. Let us use variable $x_{uv}$ to denote the predicate "U beats V." Then, the condition reported by the team mathematician can be represented by the following Boolean formula $F$ over five Boolean variables $x_{ab}$, $x_{ac}$, $x_{bc}$, $x_{bd}$, $x_{cd}$:

$$x_{ab} \cdot (x_{bc} \rightarrow x_{cd}) \cdot (\overline{x}_{ac} \rightarrow (\overline{x}_{bd} + \overline{x}_{cd})),$$

where $x \rightarrow y$ denotes "if $x$ then $y$," which is equivalent to $\overline{x} + y$. The formula $F$ can be simplified as follows:

$$\begin{aligned} & x_{ab} \cdot (x_{bc} \rightarrow x_{cd}) \cdot (\overline{x}_{ac} \rightarrow (\overline{x}_{bd} + \overline{x}_{cd})) \\ &= x_{ab} \cdot (\overline{x}_{bc} + x_{cd}) \cdot (x_{ac} + \overline{x}_{bd} + \overline{x}_{cd}) \\ &= x_{ab}\overline{x}_{bc}x_{ac} + x_{ab}\overline{x}_{bc}\overline{x}_{bd} + x_{ab}\overline{x}_{bc}\overline{x}_{cd} \\ &\quad + x_{ab}x_{cd}x_{ac} + x_{ab}x_{cd}\overline{x}_{bd} + x_{ab}x_{cd}\overline{x}_{cd}. \end{aligned}$$

The last formula is the sum of six terms, and so it is satisfiable if and only if at least one term in it is satisfiable. Among the six terms, we can see that the first one and the fourth one are not satisfied because A would have to win both games against B and C to satisfy them. The second term is also not satisfied since that would require B to lose all its three remaining games. The sixth term is logically unsatisfiable since it would require C beating D and also C losing to D. The third term is satisfiable with A beating B, B beating D, C beating both A and B, and D beating C. The fifth term can also be satisfied, with A beating B, B beating C, C beating A and D, and D beating B.

A more formal solution is to express the condition of "each team won at least one game, but no team won all its remaining games" in the following

formula $G$:

$$(x_{ab} + x_{ac})\ (\neg(x_{ab}x_{ac}))\ (\overline{x}_{ab} + x_{bc} + x_{bd})\ (\neg(\overline{x}_{ab}x_{bc}x_{bd}))$$
$$\cdot\ (\overline{x}_{ac} + \overline{x}_{bc} + x_{cd})\ (\neg(\overline{x}_{ac}\overline{x}_{bc}x_{cd}))\ (\overline{x}_{bd} + \overline{x}_{cd})\ (\neg(\overline{x}_{bd}\overline{x}_{cd})).$$

Then, determine whether $FG$ is satisfiable. □

**Example 7.4** *Formulate the following puzzle as a Boolean formula $F$, and solve it by determining whether $F$ is satisfiable:*

> *Three men named Lewis, Miller, and Nelson fill the positions of accountant, cashier, and manager in the leading department store in the City of NP. The following information about their jobs is known:*
>
> - *If Nelson is the cashier, Miller is the manager.*
> - *If Nelson is the manager, Miller is the accountant.*
> - *If Miller is not the cashier, Lewis is the manager.*
> - *If Lewis is the accountant, Nelson is the manager.*
>
> *What is each man's job?*

*Solution.* Denote three men by $L, M, N$, and three jobs as $a, c$ and $m$. Also write $X_y$ to denote the predicate "$X$'s job is $y$." Then, the given information implies that the following formula is satisfiable:

$$\begin{aligned} F &= (\overline{N}_c + M_m)\ (\overline{N}_m + M_a)\ (M_c + L_m)\ (\overline{L}_a + N_m) \\ &= (\overline{N}_c\overline{N}_m + \overline{N}_c M_a + M_m\overline{N}_m + M_m M_a) \\ &\quad \cdot (M_c\overline{L}_a + M_c N_m + L_m\overline{L}_a + L_m N_m). \end{aligned}$$

Next, from the extra information that the three men fill the three positions, we know that $\overline{N}_c\overline{N}_m \to N_a$, $M_m \to \overline{N}_m$, $M_m \to \overline{M}_a$, and $M_a L_m \to N_c$, etc. Therefore, $F$ can be simplified as follows:

$$\begin{aligned} F = 1 &\Longrightarrow (N_a + \overline{N}_c M_a + M_m)(M_c\overline{L}_a + M_c N_m + L_m) = 1 \\ &\Longrightarrow N_a M_c\overline{L}_a + N_a M_c N_m + N_a L_m + \overline{N}_c M_a M_c\overline{L}_a + \overline{N}_c M_a M_c N_m \\ &\quad + \overline{N}_c M_a L_m + M_m M_c\overline{L}_a + M_m M_c N_m + M_m L_m = 1 \\ &\Longrightarrow N_a M_c + N_a L_m = 1 \\ &\Longrightarrow N_a M_c L_m = 1. \end{aligned}$$

That is, Lewis is the manager, Miller is the cashier, and Nelson is the accountant. □

Next, we consider problems about graphs. We have defined the notion of directed graphs in Chapter 1. We now introduce undirected graphs. An

*undirected graph* (or, simply, a *graph*) is just like a directed graph, except that the edges have no direction; that is, each edge is a two-element subset of $V$. A graph has many different representations by a string. For instance, a graph $G$ can be represented by its *adjacency matrix* $A_G$. First, we assume that the vertices in a graph are always named as $v_1, v_2, \ldots, v_n$, for some $n \geq 0$. Then, for each graph $G = (\{v_1, \ldots, v_n\}, E)$, its adjacency matrix is an $n \times n$ Boolean matrix $A_G$, with $A_G(i,j) = 1$ if and only if $\{v_i, v_j\} \in E$. Therefore, a graph of $n$ vertices can be represented by a string $x \in \{0,1\}^*$ of length $n^2$, with $\{v_i, v_j\} \in E$ if and only if the $((i-1)n + j)$th bit of $x$ is 1.[1]

Let $G = (V, E)$ be a graph, with $V = \{v_1, \ldots, v_n\}$. The notions of paths and cycles in a graph are the same as those of a digraph. A *Hamiltonian cycle* is a cycle that passes through each vertex of the graph exactly once (except for the starting and ending vertex which is passed through twice). We now consider the following problem:

> HAMILTONIAN CYCLE (HC): Given a graph $G$, determine whether $G$ has a Hamiltonian cycle.

**Example 7.5** HC *is in NP.*

*Proof.* The problem HC can be solved by the following nondeterministic algorithm:

(1) On input $G = (\{v_1, \ldots, v_n\}, E)$, guess a permutation $(i_1, i_2, \cdots, i_n)$ of $\{1, 2, \ldots, n\}$.

(2) Check whether the permutation $(i_1, i_2, \cdots, i_n)$ determines a cycle in $G$; that is, check whether (i) for every $1 \leq j \leq n-1$, $\{v_{i_j}, v_{i_{j+1}}\} \in E$, and (ii) $\{v_{i_n}, v_{i_1}\} \in E$. If the checking is successful, then halt and accept the input $G$.

We note that the guessing step (1) can be implemented by an NTM as follows: It writes down $n$ integers between 1 and $n$, each of length $\lceil \log n \rceil$, and then deterministically checks that every integer between 1 and $n$ occurs in the list. (In other words, step (1) itself has a guessing stage and a checking stage.) It is clear that the total time required is $O(n^2 \log n)$. In addition, it is clear that step (2) can be done in time $O(n^2)$. So, this is a polynomial-time nondeterministic algorithm, and it follows that HC $\in NP$. □

**Example 7.6** (KNIGHT'S TOUR) *Find a tour of the* $8 \times 8$ *chessboard by a knight which visits each square exactly once and comes back to the starting square.*

[1] Note that for an undirected graph $G$, $A_G$ is symmetric in the sense that $A_G(i,j) = A_G(j,i)$, for all $i, j \in \{1, \ldots, n\}$, and so we can actually encode $G$ by only $n(n-1)/2$ bits. Nevertheless, as we are concerned only with polynomial-time computability of the underlying problems, this saving of the input size is irrelevant.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 6 | 33 | 40 | 27 | 44 | 29 | 46 |
| 34 | 37 | 2 | 5 | 32 | 47 | 26 | 43 |
| 7 | 4 | 39 | 36 | 41 | 28 | 45 | 30 |
| 38 | 35 | 8 | 1 | 48 | 31 | 42 | 25 |
| 9 | 64 | 13 | 56 | 17 | 24 | 49 | 58 |
| 14 | 55 | 10 | 61 | 52 | 57 | 18 | 21 |
| 63 | 12 | 53 | 16 | 23 | 20 | 59 | 50 |
| 54 | 15 | 62 | 11 | 60 | 51 | 22 | 19 |

Figure 7.1: A knight's tour.

*Solution.* KNIGHT'S TOUR can be viewed as a special instance of the problem HC. Indeed, the problem KNIGHT'S TOUR on an $n \times n$ chessboard can be formulated as a subproblem of HC: For each $n \geq 1$, let $G_n = (V_n, K_n)$, where $V_n = \{v_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq n\}$, and

$$K_n = \{\, \{v_{i,j}, v_{k,l}\} \mid i,j,k,l \in \{1,\ldots,n\}, \\ [|k-i| = 1, |l-j| = 2] \text{ or } [|k-i| = 2, |l-j| = 1]\}.$$

Then, the graph $G_n$ represents the moves of a knight in an $n \times n$ chessboard; that is, $V_n$ representes the $n^2$ squares of an $n \times n$ chessboard, and two squares in the board are neighbors in $G_n$ if and only if we can reach one square from the other by a knight's move. Therefore, KNIGHT'S TOUR on an $n \times n$ chessboard is exactly the problem of finding a Hamiltonian cycle in $G_n$. There are many solutions known to this problem, if $n$ is an even integer greater than 5. (It is an interesting exercise to prove that KNIGHT'S TOUR has no solution for odd $n$'s.) We show a solution in Figure 7.1 for the case of $n = 8$. (In Figure 7.1, the numbers in the squares indicate the order of the squares that are visited by the knight.) □

For our next problem, we say that a subset $A$ of the vertex set $V$ of a graph $G$ is a *vertex cover* for $G$ if every edge $e \in E$ contains a vertex in $A$.

> VERTEX COVER (VC): Given a graph $G$ and a positive integer $k$, determine whether $G$ has a vertex cover of size at most $k$.

**Example 7.7** VC *is in NP.*

*Proof.* Consider the following nondeterministic algorithm for VC:

(1) On input $(G, k)$, guess a $k$-element subset $C$ of vertices of $G$.

(2) Check that $C$ is a vertex cover of $G$. If the checking is successful, then accept the input $(G, k)$.

It is easy to see that this algorithm can be implemented by an NTM in polynomial time. First, step (1) can be done by writing down nondeterministically a string $c \in \{0,1\}^*$ of length $n$, and then verifies that there are exactly $k$ 1's in $c$. (This string $c$ represents a subset $C$ of $\{1, 2, \ldots, n\}$ with $i \in C$ if and only if the $i$th symbol of $c$ is 1.)

For step (2), we need only go through each edge to verify that one of its vertices is in $C$. This can be done in time $O(n^2)$. So, this is a polynomial-time nondeterministic algorithm and, hence, VC is in *NP*. □

The next problem is a generalization of the problem VC.

> HITTING SET (HS): Given subsets $A_1, A_2, \ldots, A_n$ of a set $S$ and an integer $k$, determine whether there is a subset $A \subseteq S$ of size $|A| \leq k$ such that $A \cap A_i \neq \emptyset$ for all $i = 1, \ldots, n$.

**Example 7.8** HS *is in NP.*

*Proof.* The following is a simple nondeterministic algorithm for HS:

(1) Guess a subset $A \subseteq S$ of size $k$.

(2) Verify that $A \cap A_i \neq \emptyset$ for all $i = 1, \ldots, n$. □

**Example 7.9** *Formulate the following logical puzzle as an instance of the problem* HS *and solve it by finding the correct subset.*

> *Three women, Joan, Michelle, and Tracy, made the following statements about their ages.*
>
> *Joan: "I am 22 years old. I am two years younger than Michelle. I am one year older than Tracy."*
>
> *Michelle: "I am not the youngest. I am three years different from Tracy. Tracy is 25 years old."*
>
> *Tracy: "I am younger than Joan. Joan is 23 years old. Michelle is three years older than Joan."*
>
> *If we know that, among three statements given by each person, exactly two are true. What are their ages?*

*Solution.* Let $J$, $M$, and $T$ denote Joan, Michelle, and Tracy, respectively. Let $X_i$ denote the $i$th statement of $X$. First, let us list all subsets of statments which result in a contradiction:

$$\{J_1, T_2\}, \quad \{J_2, T_3\}, \quad \{J_1, J_3, M_3\},$$
$$\{J_3, M_3, T_2\}, \quad \{J_1, T_1, M_3\}, \quad \{T_1, M_3, T_2\},$$
$$\{J_3, M_2, T_3\}$$

We also know that one statement in each of the following subsets is false:

$$\{J_1, J_2, J_3\}, \quad \{M_1, M_2, M_3\}, \quad \{T_1, T_2, T_3\}.$$

Altogether, each of the above subsets contains at least one false statement. Also, we know that there are totally three false statements. Thus, the above puzzle is equivalent to finding a hitting set of this family of subsets of size exactly three.

From the first two subsets, we know that these three statements must contain $J_1$ and $T_3$ or contain $J_2$ and $T_2$. By considering these two cases, we can find that $\{J_1, M_3, T_3\}$ is the only hitting set of size three. Therefore, they must be false statements, and other statements are all true. From these statements, we get that Joan is 23 (from $T_2$), Michelle is 25 (from $J_2$), and Tracy is 22 (from $J_3$). □

For any graph $G = (V, E)$, a subset $A \subseteq V$ is an *independent set* of $G$ if for any two vertices $u, v$ in $A$, $\{u, v\} \notin E$.

> INDEPENDENT SET (IS): Given a graph $G$ and a positive integer $k$, determine whether $G$ has an independent set of size at least $k$.

**Example 7.10** IS *is in NP.*

*Proof.* The following nondeterministic algorithm for IS is similar to that of VC:

(1) Guess a $k$-element subset $A$ of the vertex set $V$ of $G$.

(2) Check that, for any two vertices $v_j, v_k$ in $A$, $\{v_j, v_k\} \notin E$. □

**Example 7.11** (EIGHT-QUEEN PROBLEM) *Place eight queens in an* $8 \times 8$ *chessboard without them attacking each another. (In the chess game, a queen attacks another piece if they are located in the same row, or in the same column, or in the same diagonal.)*

*Solution.* This is a special instance of the problem IS. Let $H_n = (V_n, Q_n)$, where $V_n = \{v_{i,j} \mid 1 \le i \le n, 1 \le j \le n\}$, and

$$Q_n = \{\, \{v_{i,j}, v_{k,l}\} \mid i, j, k, l \in \{1, \ldots, n\}, \\ [i = k] \text{ or } [j = l] \text{ or } [|k - i| = |l - j| \neq 0]\}.$$

Similar to Example 7.6, we can treat $V_n$ as the $n^2$ squares of an $n \times n$ chessboard. Then, the graph $H_n$ represents a queen's moves in an $n \times n$ chessboard. Thus, the EIGHT-QUEEN PROBLEM is exactly the problem of finding an independent set for $H_8$. In general, the problem IS on $(H_n, n)$ is called the $n$-QUEEN PROBLEM. There are many solutions known to this problem for $n \ge 4$. We show a solution in Figure 7.2 for the case of $n = 8$. □

Our next example is a combinatorial problem.

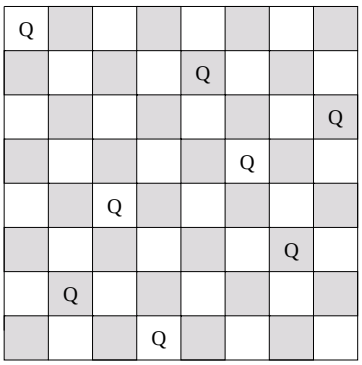


Figure 7.2: A solution to the eight-queen problem.

THREE-DIMENSIONAL MATCHING (3DM): Given three pairwisely disjoint sets $A$, $B$ and $C$, each of $n$ elements, and a set $W \subseteq A \times B \times C$, determine whether $W$ has a subset $W'$ of exactly $n$ triples such that each element in $A \cup B \cup C$ appears in the $n$ triples of $W'$ exactly once. (Such a subset $W'$ is called a *three-dimensional matching* of sets $A$, $B$ and $C$.)

**Example 7.12** 3DM *is in NP.*

*Proof.* Consider the following nondeterministic algorithm:

(1) On input $(A, B, C, W)$, guess a subset $W' \subseteq W$ of size $|A|$.

(2) Check that each element of $A \cup B \cup C$ occurs in $W'$ exactly once. If the checking is successful, then accept the input.

It is easy to see that step (1) can be done by an NTM in time $O(m)$ where $m = |W|$, and step (2) can be done by a DTM in time $O(|A|)$. □

Our next example is a famous problem in operations research. Suppose $u$ and $v$ are two $n$-dimensional integer vectors. We write $u \geq v$ to denote that the $i$th element of $u$ is greater than or equal to the $i$th element of $v$ for all $i = 1, 2, \ldots, n$.

INTEGER PROGRAMMING (IP): Given an $n \times m$ integer matrix $A$ and an $n$-dimensional integer vector $b$, determine whether there exists an $m$-dimensional integer vector $x$ such that $Ax \geq b$.

⋆ **Example 7.13** IP *is in NP.*

*Proof (Sketch).* It is natural to try the following nondeterministic algorithm for IP:

(1) Guess an $m$-dimensional integer vector $x$.

(2) Verify that $Ax \geq b$.

The correctness of the above algorithm is straightforward. However, it is not clear that it can be implemented by an NTM in polynomial time. We note that, in order to make this algorithm to work in polynomial time, we need to find a *witness* $x$ of length polynomially bounded by the total size of the input $(A, b)$. That is, we need the following lemma from linear algebra. (See Exercise 8 of this section for the proof.)

**Lemma 7.14** *For any $n \times n$ integer matrix $A$ and $n$-dimensional integer vector $b$, if there is an $m$-dimensional integer vector $y$ satisfying $Ay \geq b$, then there is an $m$-dimensional integer vector $x$ satisfying $Ax \geq b$ such that the absolute value of each integer in $x$ is bounded by $2(\alpha q)^{2q+2}$, where $\alpha$ is the maximum absolute value of elements in $A$ and $b$, and $q = \max\{m, n\}$.*

With this lemma, we can modify step (1) of the above algorithm as follows:

(1′) Guess an $m$-dimensional integer vector $x$, with the absolute value of each integer in $x$ bounded by $2(\alpha q)^{2q+2}$.

Note that the binary representation of an integer $t \leq 2(\alpha q)^{2q+2}$ is of length $\leq \log(2(\alpha q)^{2q+2}) = O(q \log q \log \alpha)$, which is polynomially bounded by the input size. So, after the modification, the algorithm runs in polynomial time. □

**Example 7.15** *Formulate the following logical puzzle as an instance of the problem IP, and solve it by solving the corresponding system of inequalities.*

> *Professor X presented a colloquium talk in the Department of Complexity. When he started to talk, he noticed that the 15 people of the audience satisfy the following conditions:*
>
> - *There were more students than professors.*
> - *There were more male professors than male students.*
> - *There were more male students than female students.*
> - *There was at least one female professor.*
>
> *Five minutes later, a person rushed into the room. However, the appearance of this person did not change the above four conditions. Is this person a man or a woman, a student or a professor?*

*Solution.* Let $s_1, s_2, p_1, p_2$ denote, respectively, the numbers of male students, female students, male professors, and female professors, both before and after the last person entering the room. From the given conditions, we have the following inequalities:

(1) $15 \leq s_1 + s_2 + p_1 + p_2 \leq 16$,

(2) $s_1 + s_2 \geq p_1 + p_2 + 1$,

(3) $p_1 \geq s_1 + 1$,

(4) $s_1 \geq s_2 + 1$,

(5) $p_2 \geq 1$.

From (1) and (2), we obtain

(6) $s_1 + s_2 \geq 8$,

(7) $p_1 + p_2 \leq 7$.

Furthermore, by (4) and (6), we have $s_1 \geq 5$. By (3), $p_1 \geq 6$. By (5) and (7), we obtain $p_1 = 6$ and $p_2 = 1$. It follows that $s_1 = 5$ and $8 \leq s_1 + s_2 \leq 9$. This implies that $3 \leq s_2 \leq 4$. We conclude that the last person coming into the room is a female student. □

All the examples above have simple guess-and-verify algorithms. In the following, we show an example in *NP*, which does not have such a simple guess-and-verify algorithm. Instead, we prove that it is in *NP* by a nondeterministic *recursive* algorithm.

PRIMALITY TESTING (PRIME): Given a positive integer $p$, written in the binary form, determine whether $p$ is a prime number.

To show that PRIME is in *NP*, we need the following result of number theory. We omit the proof of this lemma.

**Lemma 7.16** *An odd integer $p > 2$ is a prime if and only if there exists an integer $a \in \{1, 2, \ldots, p-1\}$ such that*

*(a)* $a^{p-1} \equiv 1 \pmod{p}$, *and*

*(b)* $a^{(p-1)/q} \not\equiv 1 \pmod{p}$, *for all prime factors $q$ of $p-1$.*

⋆ **Example 7.17** PRIME *is in NP.*

*Proof.* Lemma 7.16 suggests that to check whether $p$ is a prime, we can guess an integer $a \in \{1, \ldots, p-1\}$ and then check conditions (a) and (b). So, we only need to show that conditions (a) and (b) can be done in polynomial time. We note, however, that condition (b) involves the checking of an inequality over all prime factors $q$ of $p-1$, and so it seems to require a solution to the *integer factoring* problem, which asks, for a given integer $n$, to find all its prime factors. The integer factoring problem is apparently more difficult than the problem PRIME, and so this looks like a vicious cycle.

Fortunately, we note that since $p$ is an odd integer, a prime factor $q$ of integer $p-1$ must be less than $p/2$. This fact allows us to use a recursive, nondeterministic algorithm to solve the problem PRIME:

(1) For input $p$, guess an integer $a$, $1 \leq a \leq p-1$, and a sequence of distinct numbers $q_1, q_2, \ldots, q_k$, $1 \leq k \leq \log p$, with each $q_i$ between 2 and $(p-1)/2$.

(2) Recursively check that each $q_i$, $1 \leq i \leq k$, is a prime number.

(3) Check that $q_1, q_2, \ldots, q_k$ are the only prime factors of $p-1$.

(4) Check that $a^{p-1} \equiv 1 \pmod{p}$.

(5) Check that $a^{(p-1)/q_i} \not\equiv 1 \pmod{p}$, for all $1 \leq i \leq k$.

First, we claim that conditions (3), (4) and (5) can be verified by a DTM in time $O((\log p)^4)$. To see this, we note that the modulo exponentiation $x^y \bmod p$, with $x, y \leq p$, can be done by $O(\log p)$ modulo multiplications as follows: Write $y$ in the binary form as $y = y_b y_{b-1} \cdots y_0$; that is, $y = \sum_{j=0}^{b} y_j 2^j$, with each $y_j \in \{0,1\}$. Then,

$$x^y \bmod p = \prod_{0 \leq j \leq b, y_j = 1} (x^{2^j} \bmod p) \bmod p.$$

Since each $(x^{2^j} \bmod p)$ can be found by $j$ modulo multiplications, $x^y$ can be found by $2b$ modulo multiplications, where $b = \lfloor \log p \rfloor + 1$. Therefore, conditions (3), (4) and (5) can be done by $O((\log p)^2)$ modulo multiplications (or divisions). Since each modulo multiplication is over two integers of length $\log p$, it can be done in time $O((\log p)^2)$. Thus, the total checking time for conditions (3), (4) and (5) is $O((\log p)^4)$.

Next, we note that condition (2) can be done by applying the above algorithm recursively to $q_1, \ldots, q_k$. Note that each $q_i$ is bounded by $(p-1)/2$, and so the depth of the recursive calls (the maximum number of nested recursive calls of the algorithm) is bounded by $\log p$. Let $m(p)$ denote the total computation time of the above algorithm on input $p$. Then, for the nondeterministic guess of step (1) which guesses the correct prime factors $q_1, \ldots, q_k$ of $p-1$, we have the following recurrence relation on function $m$:

$$m(p) \leq c(\log p)^4 + \sum_{i=1}^{k} m(q_i),$$

for some constant $c > 0$. Note that each $q_i$, $1 \leq i \leq k$, is between 2 and $(p-1)/2$, and that $\sum_{i=1}^{k} \log q_i \leq \log p$. It follows that

$$\sum_{i=1}^{k} (\log q_i)^5 \leq (\log p)^5 - (\log p)^4$$

and, hence, $c(\log p)^5$ is an upper bound of $m(p)$ for the recurrence inequality. This means that the shortest accepting path of this nondeterministic algorithm takes time $O((\log p)^5)$ to accept the input $p$. So, the algorithm is a polynomial-time algorithm. □

The above example showed that for any prime $p$, there is a *short proof* for the fact that $p$ is a prime. This short proof involves the correct guess at each step of the recursive nondeterministic algorithm and the corresponding verifications of conditions (3), (4) and (5).

**Example 7.18** *Prove that* 683 *is a prime.*

*Proof.* We apply the recursive nondeterministic algorithm for PRIME on input $p = 683$. First, for input 683, we guess $a = 73$, $q_1 = 2$, $q_2 = 11$ and $q_3 = 31$. For this guess, we verify the following conditions:

(3) $p - 1 = 682 = 2 \cdot 11 \cdot 31$.

(4) $73^{682} \equiv 1 \pmod{683}$.

(5) $73^{341} \equiv 682 \pmod{683}$, $73^{62} \equiv 256 \pmod{683}$, and $73^{22} \equiv 3 \pmod{683}$.

Next, we need to prove that $q_1$, $q_2$ and $q_3$ are primes. For $q_1 = 2$, it is clearly a prime. For $q_2 = 11$, we may guess $(6; 2, 5)$ and verify the following conditions:

(3) $11 - 1 = 2 \cdot 5$.

(4) $6^{10} \equiv 1 \pmod{11}$.

(5) $6^5 \equiv 10 \pmod{11}$, and $6^2 \equiv 3 \pmod{11}$.

Also, we note that 2 and 5 are obviously primes. This proofs that 11 is a prime.

For $q_3 = 31$, we may guess $(3; 2, 3, 5)$ and verify that

(3) $31 - 1 = 2 \cdot 3 \cdot 5$.

(4) $3^{30} \equiv 1 \pmod{31}$.

(5) $3^{15} \equiv 30 \pmod{31}$, $3^{10} \equiv 25 \pmod{31}$, and $3^6 \equiv 16 \pmod{31}$.

Finally, we note that 2, 3 and 5 are obviously primes, and so we conclude that 31 is a prime.

This completes the proof that 683 is a prime. □

An interesting fact about PRIME is that its complement, the set of nonprime numbers is also in *NP*. In fact, it is easy to determine that an integer $n \geq 4$ is a composite number by the following straightforward guess-and-verify algorithm: Guess an integer $m$, $2 \leq m \leq n - 1$, and verify that $m$ divides $n$. We recall that it is not known whether $NP = co\text{-}NP$, nor is it known whether $NP \cap co\text{-}NP = P$. Also, most natural examples of problems in *NP* are not known to be in *co-NP* unless they are actually in *P*. Thus, PRIME is one of very few candidates of problems in $(NP \cap co\text{-}NP) - P$.

## Exercise 7.1

**1.** For any set $A$, let $A' = \{w\#0^n \mid w \text{ is a suffix of some } x \text{ in } A \text{ with } |x| = n\}$ and $A'' = \{w \mid w \text{ has a suffix } x \text{ in } A\}$. Show that if $A$ is in *NP*, then $A'$ and $A''$ are also in *NP*.

**2.** For any set $B \subseteq \{0,1\}^*\#\{0,1\}^*$, let $prefix(B) = \{x\#u \mid x, u \in \{0,1\}^*, (\exists v \in \{0,1\}^*)\, x\#uv \in B\}$.

(a) Assume that there is a polynomial function $p$ such that $x\#y \in B$ implies $|y| \leq p(|x|)$. Show that if $B \in NP$, then $prefix(B) \in NP$.

(b) Show that if $A = \{x \mid (\exists u)[|u| \leq p(|x|), x\#u \in B\}$ for some polynomial function $p$ and if $prefix(B) \in P$, then $A \in P$.

**3.** A function $f : \{0,1\}^* \to \{0,1\}^*$ is *polynomial-time computable* if there is a polynomial-time DTM that computes the function $f$ (i.e., on input $x$, it halts in $p(|x|)$ moves for some polynomial $p$, with the string $f(x)$ on the output tape). A function $f : \{0,1\}^* \to \{0,1\}^*$ is *polynomially honest* if there is a polynomial function $p$ such that $|f(x)| \leq p(|x|)$ and $|x| \leq p(|f(x)|)$ for all $x \in \{0,1\}^*$. Assume that $f$ is polynomial-time computable and is polynomially honest.

(a) Assume that $f$ is one-to-one and onto. Show that if $A \in NP$, then $f(A) \in NP$ and $f^{-1}(A) \in NP$.

(b) Show that if $P = NP$, then the following functions are polynomial-time computable:

$$Max_f(u,v) = \max\{f(x) \mid u \leq_{\text{lex}} x \leq_{\text{lex}} v\},$$
$$Min_f(u,v) = \min\{f(x) \mid u \leq_{\text{lex}} x \leq_{\text{lex}} v\},$$

where $\leq_{\text{lex}}$ is the lexicographic ordering on $\{0,1\}^*$.

**4.** Formulate the following puzzles as instances of problems in $NP$, and then solve the puzzles by finding the witnesses to the instances:

(a) Six men, A, B, C, D, E, and F, are the only members eligible for the offices of President, Vice-President, and Secretary in Club NP. The following preferences of these men are known:

- A won't be an officer unless E is President.
- B won't serve if he outranks C.
- B won't serve with F under any conditions.
- C won't serve with both E and F.
- C won't serve if F is President or B is Secretary.
- D won't serve with C or E unless he outranks them.
- E won't be Vice-President.
- E won't be Secretary if D is an officer.
- E won't serve with A unless F serves too.
- F won't serve unless either he or C is President.

How can the three offices be filled satisfying all above conditions?

(b) Police arrested four men as the suspects of a murder case. They made the following statements when questioned:

John: "Nick did it."
Nick: "Bill did it."
Dan: "I didn't do it."
Bill: "Nick lied when he said I did it."

If exactly one of them is the real murderer and exactly one of the above statements is true, who is the murderer?

(c) Brown, Jones, and Smith are employed by the village of NP as fireman, policeman, and teacher, though not necessarily respectively. Someone in the village reported that:

- Brown and the teacher are neighbors.
- Jones and the teacher are neighbors.
- Both Brown and Smith are neighbors of the fireman.
- Both the policeman and the fireman are neightbors of Jones.
- The men are all neighbors.

However, the truth of the matter is that only two of these statements are true. Can you determine the job which each man holds?

(d) Both the Smiths and the Taylors have two young sons under eleven. The names of the boys, whose ages rounded off to the nearest year are all different, are Arthur, Bert, Carl, and David. Taking the ages of the boys only to the nearest year, the following statements are true:

- Arthur is three years younger than his brother.
- Bert is the oldest.
- Carl is half as old as one of the Taylor boys.
- David is five years older than the younger Smith boy.
- The total ages of the boys in each family differ by the same amount today as they did five years ago.

How old is each boy, and what is each boy's family name?"

**5.** The *threshold function* $T_{n,k}$ is a Boolean function defined as follows:

$$T_{n,k}(x_1, \cdots, x_n) = \begin{cases} 1 & \text{if there are at least } k \text{ 1's in } x_1, \cdots, x_n, \\ 0 & \text{otherwise.} \end{cases}$$

(a) Construct Boolean formulas $F_{n,k}$ using operations $\wedge$, $\vee$, $\neg$, and variables $x_1, x_2, \ldots, x_n$ such that it computes the function $T_{n,k}$.

(b) Show that there exist Boolean formulas $G_{n,k}$ using operations $\wedge$, $\vee$, $\neg$, variables $x_1, x_2, \ldots, x_n$, and some auxiliary variables $y_1, \ldots, y_m$, such that (i) the length of $G_{n,k}$ is bounded by $n^c$ for some constant $c > 0$, and (ii) an assignment $t$ satisfies $G_{n,k}$ if and only if $t$ assigns value 1 to at least $k$ variables in $\{x_1, \ldots, x_n\}$.

**6.** Prove that the following integers are primes: 293, 587, 65537, 214177.

**7.** Show that the following problems are in *NP*:

(a) LONGEST-PATH (LP): Given a graph $G$ and an integer $k > 0$, determine whether $G$ has a simple path of at least $k$ edges. (A path is *simple* if no vertex appears twice.)

(b) Traveling Salesman Problem (TSP): Given a complete graph $G = (V, E)$ with a cost function $c : E \to \mathbf{N}$, and an integer $k$, determine whether there is a tour (i.e., a Hamiltonian cycle) of $G$ with the total edge cost bounded by $k$.

(c) Graph Isomorphism (GIso): Given two graphs $G$ and $H$, determine whether $G$ is isomorphic to $H$. (Two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$ are *isomorphic* if $|V_1| = |V_2|$, and there is a one-to-one, onto mapping $f : V_1 \to V_2$ such that $\{u, v\} \in E_1$ if and only if $\{f(u), f(v)\} \in E_2$. The function $f$ is called an isomorphism.)

(d) Bounded PCP: Given a finite set of ordered pairs $(x_1, y_1), \ldots, (x_n, y_n)$ of strings over an alphabet $\Sigma$, and an integer $K$ (in the unary form $1^K$), determine whether there is a finite sequence of integers $(i_1, i_2, \ldots, i_m)$, with each $i_j \in \{1, \ldots, n\}$ and $m \leq K$, such that

$$x_{i_1} x_{i_2} \cdots x_{i_m} = y_{i_1} y_{i_2} \cdots y_{i_m}.$$

(e) Bounded Tiling: Given a finite number of types $t_0, t_1, \ldots, t_n$ of colored tiles and an integer $K$ (in the unary form $1^K$), determine whether it is possible to cover a $K \times K$ square by colored tiles of these types, starting with a tile of type $t_0$ at the lower left corner. (See Exercise 6 of Section 5.6 for the details of the definition.)

⋆ **8.** Let $A$ be an $n \times n$ integer matrix, and $b$ an $n$-dimensional integer vector. Let $\alpha$ be the maximum absolute value of integers in $A$ and $b$, and $q = \max\{m, n\}$.

(a) Show that if $B$ is a square submatrix of $A$, then $|\det(B)| \leq (\alpha q)^q$, where $\det(B)$ denotes the determinant of matrix $B$.

(b) Show that if $\text{rank}(A) = r < m$, then there exists a nonzero vector $z$ such that $Az = 0$ and the maximum absolute value of integers in $z$ is bounded by $(\alpha q)^q$.

(c) Assume that $Ax \geq b$ has an integer solution $x$. Let $a_i$ denote the $i$th row of $A$, $b_i$ the $i$th component of $b$, and $e_j$ the $j$th $m$-dimensional unit vector (i.e., all components of $e_j$ is 0, except that the $j$th component is 1). Let $x$ be a solution for $Ax \geq b$ that maximizes the number of elements in the following set:

$$\mathcal{A}_x = \{a_i \mid b_i \leq a_i x \leq b_i + (\alpha q)^{q+1}, 1 \leq i \leq n\} \cup \{e_j \mid |x_j| \leq (\alpha q)^q, 1 \leq j \leq m\}.$$

Prove that the rank of $\mathcal{A}_x$ is $m$.

(d) Use (c) above to prove Lemma 7.14.

## 7.2 Polynomial-Time Reducibility

We have seen, in Section 5.4, that the technique of reducibility is useful in proving a problem undecidable. Namely, suppose we know that $A$ is unde-

cidable and $A \leq_m B$, then $B$ is also undecidable. In this section, we extend the notion of reducibility to a polynomial time-bounded version and apply it to show that many problems in $NP$ have the *same* computational complexity, in the sense that their deterministic time complexity is within a polynomial factor of each other.

Let $A, B \subseteq \Sigma^*$ be two languages. We say set $A$ is *polynomial-time many-one reducible* (or, simply, *polynomial-time reducible*) to $B$, and write $A \leq_m^P B$, if there exists a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that for every $x \in \Sigma^*$,

$$x \in A \iff f(x) \in B.$$

That is, $A \leq_m^P B$ if $A \leq_m B$ by a polynomial-time computable reduction function $f$. The following properties are analogous to Propositions 5.24 and 5.26:

**Proposition 7.19** *(a)* $A \leq_m^P A$.

*(b) If* $A \leq_m^P B$ *and* $B \leq_m^P C$, *then* $A \leq_m^P C$.

*(c) If* $B \in P$ *and* $A \leq_m^P B$, *then* $A \in P$.

*(d) If* $B \in NP$ *and* $A \leq_m^P B$, *then* $A \in NP$.

*(e) If* $B \in PSPACE$ *and* $A \leq_m^P B$ *then* $A \in PSPACE$.

*Proof.* The proofs are essentially the same as Propositions 5.24 and 5.26. The main observation is that the composition of two polynomial functions is still a polynomial and, hence, that the composition of two polynomial-time computable functions is still polynomial-time computable. □

We first present a few simple reductions among problems of similar forms to demonstrate the basic ideas of polynomial-time reductions. Recall the problem LP defined in Exercise 7(a) of Section 7.1.

**Example 7.20** HC $\leq_m^P$ LP.

*Proof.* Let $G = (V, E)$ be an input instance of HC. We need to find a graph $G' = (V', E')$ and an integer $k > 0$ such that $G$ contains a Hamiltonian cycle if and only if $G'$ has a simple path of length $k$. In addition, the graph $G'$ and integer $k$ must be computable from graph $G$ in polynomial time.

Note that a Hamiltonian cycle is a simple path except that the starting vertex and the ending vertex are the same. Thus, this construction is very simple: Assume that $V = \{v_1, \ldots, v_n\}$. Then, we let $V' = V \cup \{v_0, u_0, u_1\}$ and

$$E' = E \cup \{\{v_0, v_i\} \mid \{v_1, v_i\} \in E\} \cup \{\{u_0, v_0\}, \{u_1, v_1\}\}.$$

We also let $k = n + 2$.

Now, suppose that $G$ has a Hamiltonian cycle $(v_1, v_{i_2}, \ldots, v_{i_n}, v_1)$, then the path $(u_1, v_1, v_{i_2}, \ldots, v_{i_n}, v_0, u_0)$ in $G'$ has length $n + 2$. Conversely, we note that $G'$ contains only $n + 3$ vertices, and so any simple path of length $n + 2$ in $G'$ must pass through each vertex exactly once. Since both vertices $u_1$ and

$u_0$ are of degree one, the path must be $(u_1, v_1, v_{i_2}, \ldots, v_{i_n}, v_0, u_0)$ for some permutation $(i_2, \ldots, i_n)$ of $\{2, 3, \ldots, n\}$. Furthermore, $(v_{i_n}, v_0) \in E'$ implies $(v_{i_n}, v_1) \in E$. Therefore, $(v_1, v_{i_2}, \ldots, v_{i_n}, v_1)$ is a Hamiltonian cycle in $G$.

The above proved that $G \in$ HC if and only if $(G', k) \in$ LP. In addition, it is clear the construction can be easily done in polynomial time. Therefore, this is a polynomial-time reduction from HC to LP. □

We say two problems $A$ and $B$ are *polynomial-time equivalent* (under the reduction $\leq_m^P$), and write $A \equiv_m^P B$, if $A \leq_m^P B$ and $B \leq_m^P A$.

**Example 7.21** VC $\equiv_m^P$ IS.

*Proof.* We observe that problems VC and IS have a very simple relation: A set $A \subseteq V$ is a vertex cover of the graph $G = (V, E)$ if and only if the set $V - A$ is an independent set of $G$. From this observation, we can see that the mapping from $(G, k)$ to $(G, |V| - k)$ is a reduction for both IS $\leq_m^P$ VC and VC $\leq_m^P$ IS. □

We have defined the problem GIso in Exercise 7(c) of Section 7.1. A variation of the problem GIso is the graph isomorphism problem over directed graphs.

> Digraph Isomorphism (DGIso): Given two digraphs $G_1 = (V_1, A_1)$ and $G_2 = (V_2, A_2)$, determine whether there is a one-to-one, onto function $f : V_1 \to V_2$ such that $(u, v) \in A_1$ if and only if $(f(u), f(v)) \in A_2$.

**Example 7.22** GIso $\equiv_m^P$ DGIso.

*Proof.* We first consider the reduction from GIso to DGIso: For any two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, we construct two directed graphs $G_1' = (V_1, E_1')$ and $G_2' = (V_2, E_2')$, with

$$E_i' = \{(u, v), (v, u) \mid u, v \in V_i, \{u, v\} \in E_i\},$$

for $i = 1, 2$. Then, it is clear that $G_1$ and $G_2$ are isomorphic if and only if $G_1'$ and $G_2'$ are isomorphic.

For the reduction DGIso $\leq_m^P$ GIso, suppose that we are given two digraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. For each digraph $G_i$, $i \in \{1, 2\}$, we construct an undirected graph $G_i' = (V_i', E_i')$ by replacing each edge $(u, v) \in E_i$ by a subgraph $H_{u,v}$ of nine vertices, with two of the vertices identified with $u$ and $v$ (see Figure 7.3). It is clear that if $f$ is an isomorphism function between $G_1$ and $G_2$, then the corresponding mapping $g$ which maps each $H_{u,v}$ to $H_{f(u),f(v)}$ is an isomorphism between $G_1'$ and $G_2'$. On the other hand, suppose that $g$ is an isomorphism between $G_1'$ and $G_2'$. Then, it is easy to

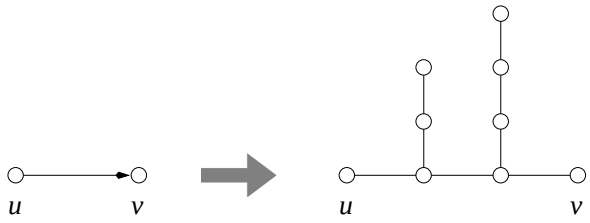


Figure 7.3: The subgraph $H_{u,v}$.

see that each subgraph $H_{u,v}$, with $u, v \in V_1$, must be mapped to a subgraph $H_{y,z}$, with $y, z \in V_2$. In addition, by the design of the subgraphs $H_{u,v}$, the corresponding mapping between $G_1$ and $G_2$ must be an isomorphism.

It is clear that the above reductions can be constructed in polynomial time. So, we conclude that GIso $\equiv_m^P$ DGIso. □

Next, we introduce a subproblem of SAT, called 3SAT, which is very useful in the study of the *NP* theory. We first introduce more notations about Boolean formulas. In a Boolean formula, we define a *literal* to be a variable $x_i$ or its negation $\overline{x}_i$. We say a Boolean formula $F$ is an *elementary sum* if it is a sum of literals, and it is a *CNF* (*conjunctive normal form*) if it is a product of elementary sums. Each elementary sum $C$ which occurs as a factor of a CNF $F$ must have the property of $[C = 0 \Rightarrow F = 0]$. Such an elementary sum is called a *clause* of the Boolean function $F$. A CNF $F$ is called a 3-*CNF* if each clause of $F$ contains exactly three literals about three distinct variables. That is, a 3-CNF formula must have the following form:

$$(z_{i_1} + z_{i_2} + z_{i_3})\,(z_{i_4} + z_{i_5} + z_{i_6})\,\cdots\,(z_{i_{3m-2}} + z_{i_{3m-1}} + z_{i_{3m}}),$$

where each $z_j$ is either a variable $x_j$ or its negation $\overline{x}_j$.

The following are two subproblems of SAT:

CNF-SAT: Given a CNF $F$, determine whether $F$ is satisfiable.

3SAT: Given a 3-CNF $F$, determine whether $F$ is satisfiable.

**Example 7.23** SAT $\leq_m^P$ CNF-SAT.

*Proof.* Define

$$c(x, y, z) = (x + y + \overline{z})(\overline{x} + z)(\overline{y} + z).$$

Then, it is easy to check that

(a) An assignment $t$ satisfies $c(x, y, z)$ if and only if $t(x) + t(y) = t(z)$.

(b) An assignment $t$ satisfies $c(\overline{x}, \overline{y}, \overline{z})$ if and only if $t(x) \cdot t(y) = t(z)$.

Now, for each Boolean formula $F$, we construct a CNF $F^*$ as follows:

(1) Initial $F^* = 1$.

(2) While $F$ contains at least one operator $+$ or $\cdot$, do the following:

(2.1) Select a subformula $x + y$ (or, $xy$) of $F$, where $x$ and $y$ are two literals. Let $G$ be the Boolean formula obtained by replacing the subformula $x + y$ (or, respectively, $xy$) in $F$ by a new variable $z$. Also, Set $G^* := F^* \cdot c(x, y, z)$ (or, respectively, $F^* \cdot c(\overline{x}, \overline{y}, \overline{z})$).

(2.2) Reset $F := G$, and $F^* := G^*$.

(3) Let $F^* := F^* \cdot F$. (Note that $F$ must be a single literal.)

It is clear that the final $F^*$ is a CNF formula. By the observations (a) and (b), it is easy to see that at the end of step (2.1), we must have the relation that $FF^*$ is satisfiable if and only if $GG^*$ is satisfiable. Note that, at the beginning, $FF^*$ is just the original input formula $F$ and, at the end, $GG^*$ is the output formula $F^*$. Thus, we have that the input $F$ is satisfiable if and only if the output $F^*$ is satisfiable. Furthermore, the while loop will be executed at most $|F|$ times, and so it is a polynomial-time algorithm. It follows that the mapping from $F$ to $F^*$ is a polynomial-time reduction from SAT to CNF-SAT. □

Note that the above reduction actually reduces a Boolean formula $F$ to a CNF $F^*$ with each clause in $F^*$ having at most three literals. So, it is easy to modify this reduction to SAT $\leq_m^P$ 3SAT (see Exercise 1 of this section).

In the above examples, the two problems involved in the reductions have similar forms. Therefore, the reductions are just simple local changes on problem instances. When two problems in question are of different forms, the reduction between them are usually more complicated. However, there are some basic ideas we can apply. First, we note that a reduction function $f$ for $A \leq_m^P B$ is an efficient way to transform a problem instance of $A$ to a problem instance of $B$. Thus, if we know how to encode the data structures of problem $A$ in terms of data structures of problem $B$, then the reduction $f$ may become simple. For instance, we have seen, through the study of logical puzzles in the last section, that Boolean formulas and linear inqualities are useful tools to encode logical and numerical relations. Using this expressive power of Boolean formulas and linear inqualities, reductions from many problems in $NP$ to SAT and IP are easy to construct.

**Example 7.24** VC $\leq_m^P$ IP.

*Proof.* Let $(G, k)$ be an instance of VC, where $G = (V, E)$ is a graph and $k$ is a positive integer. Suppose $V = \{v_1, \cdots, v_n\}$. We need to define an instance of IP that encodes the condition of a vertex cover $C$ of $G$. First, for each vertex $v_i$, $1 \leq i \leq n$, we define a variable $x_i$. Then, we define the following inequalities:

(1) $0 \leq x_i \leq 1$, for $i = 1, \ldots, n$.

(2) $x_i + x_j \geq 1$, for each $\{v_i, v_j\} \in E$.

(3) $x_1 + x_2 + \cdots + x_n \leq k$.

We note that with condition (1), a solution to the above system of inequalities encodes a subset $C$ of $V$, with $x_i = 1$ indicating that $v_i \in C$. Under this interpretation, condition (2) means that each edge in $E$ contains at least one vertex in $C$, and condition (3) means that $|C| \leq k$. Together, we see that the above system of inequalities has a solution $\{x_1, \ldots, x_n\}$ if and only if $C = \{v_i \in V \mid x_i = 1\}$ is a vertex cover of $G$ of size $\leq k$. Therefore, this is a reduction from VC to IP.

Finally, we observe that the mapping from $(G, k)$ to the above inequalities can be easily done in polynomial time. We conclude that VC $\leq_m^P$ IP. □

**Example 7.25** 3DM $\leq_m^P$ SAT.

*Proof.* Given an instance $W \subseteq A \times B \times C$ of the problem 3DM, we need to construct a Boolean formula $F$ such that $F \in$ SAT if and only if $W \in$ 3DM. First, for each $w \in W$, we define a Boolean variable $x_w$. Then, we formulate the statement that $W$ contains a three-dimensional matching $W'$ as a Boolean formula $F$ over variables $x_w$, with $x_w = 1$ indicating $w \in W'$. That is, under this interpretation, the formula $F$ asserts:

(1) Each $a \in A \cup B \cup C$ occurs in some $w \in W'$; and

(2) An element $a \in A \cup B \cup C$ cannot occur in two different $u, v \in W'$.

For each $w \in W$ and each $a \in A \cup B \cup c$, we write $a \in w$ to denote that $a$ is one of the three elements in $w$. Then, condition (1) can be formulated as

$$F_1 = \prod_{a \in A \cup B \cup C} \sum_{w \in W, a \in w} x_w,$$

and condition (2) can be formulated as

$$F_2 = \prod_{a \in A \cup B \cup C} \prod_{\substack{u, v \in W, u \neq v \\ a \in u \cap v}} (\overline{x}_u + \overline{x}_v).$$

The required formula is $F = F_1 F_2$.

For any $W' \subseteq W$, define an assignment $t_{W'}$, with $t_{W'}(x_w) = 1$ if and only if $w \in W'$. Then, under the above interpretation, we can see that $W$ contains a three-dimensional matching $W'$ if and only if $t_{W'}$ satisfies $F$. Furthermore, it is easy to see that formula $F$ can be constructed from $W$ in polynomial time. So, we conclude that 3DM $\leq_m^p$ SAT. □

Another idea for the construction of polynomial-time reductions among problems in *NP* is from Theorem 7.1. In Theorem 7.1, we showed that, for a

set $A \in NP$, each instance $x \in A$ has a witness that certifies that this instance belongs to set $A$. Although two problems in $NP$ may look very different, they must share this common property. Since a reduction function $f$ for $A \leq_m^P B$ must map an instance $x$ in $A$ with witness $w_x$ to an instance $y$ in $B$ with witness $w_y$, the idea of the construction, then, is to design the function $f$ $f$ to *preserve* the witnesses of the two problms. That is, the function $f$ not only maps $x$ to $y$, but also maps $w_x$ to $w_y$. We illustrate this idea in the following examples.

**Example 7.26** 3SAT $\leq_m^P$ VC.

*Proof.* To show that 3SAT $\leq_m^P$ VC, we need to construct, from a 3-CNF formula $F$, a graph $G_F$ and an integer $K$ such that $F$ is satisfiable if and only if $G_F$ has a vertex cover of size $K$. The witness to a Boolean formula $F$ in 3SAT is a truth assignment, and the witness to $(G_F, K)$ in VC is a vertex cover $S \subseteq V$ of size $K$. So, the main idea of the design for $G_F$ is to create a subgraph $H_1$ in $G_F$ that corresponds to the variables in $F$ such that a possible vertex cover of $H_1$ corresponds to an assignment of variables in $F$.

More precisely, suppose that $F$ has $m$ clauses $C_1, \ldots, C_m$ over $n$ variables $x_1, \ldots, x_n$. For each variable $x_i$ in $F$, we design two vertices $v_i$ and $\overline{v}_i$ in $H_1$ with an edge between them. Then, any minimum vertex cover $S_1$ of $H_1$ must contain exactly one of these two vertices for every $i$, $1 \leq i \leq n$. In other words, each minimum cover $S_1$ corresponds to an assignment $t$ on variables, with $t(x_i) = 1$ if and only if $v_i \in S_1$.

In addition to $H_1$, we also need a second part $H_2$ of $G_F$ that relates the vertices in $H_1$ in a way like the clauses in $F$ relate the variables $x_1, \ldots, x_n$. That is, we need to encode the clauses $C_1, \ldots, C_m$ in $H_2$, based on the above interpretation between vertex covers $S_1$ of $H_1$ and assignments $t$ on variables $\{x_1, \ldots, x_n\}$. To do so, we create a triangle $T_j$ for each clause $C_j$, $1 \leq j \leq m$, and connects $T_j$ with $H_1$ in such a way that if variable $x_i$ (or, its negation $\overline{x}_i$) is in $C_j$ then $T_j$ is connected to $v_i$ (or, respectively, to $\overline{v}_i$).

Together, the whole graph $G_F$ can be described as follows:

(1) $G_F$ has $2n + 3m$ vertices: $v_i$, $\overline{v}_i$, for $1 \leq i \leq n$, and $u_{j,1}, u_{j,2}, u_{j,3}$ for $1 \leq j \leq m$.

(2) For each $i$, $1 \leq i \leq n$, there is an edge between vertices $v_i$ and $\overline{v}_i$.

(3) For each $j$, $1 \leq j \leq m$, there are three edges between $u_{j,1}$, $u_{j,2}$ and $u_{j,3}$.

(4) For each pair $(i, j)$, with $1 \leq i \leq n$ and $1 \leq j \leq m$, if $x_i$ (or $\overline{x}_i$) is the $k$th literal of $C_j$, then there is an edge between $v_i$ (or, $\overline{v}_i$) and $u_{j,k}$, where $k = 1, 2$ or 3.

Figure 7.4 shows the graph $G_F$ with respect to the formula $F = (x_1 + x_2 + x_3)(\overline{x}_1 + \overline{x}_2 + x_3)$.

We observe that any vertex cover $S$ of $G_F$ must include at least one vertex in each pair $v_i$ and $\overline{v}_i$, and at least two vertices in each triangle $\{u_{j,1}, u_{j,2}, u_{j,3}\}$. So, it includes at least $n + 2m$ vertices.

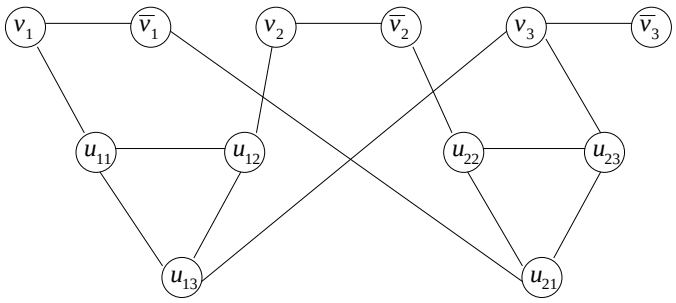


Figure 7.4: The graph $G_F$.

From this basic relation between $F$ and $G_F$, it is easy to see that $F$ is satisfiable if and only if $G_F$ has a vertex cover of size at most $n + 2m$. First, suppose that $F$ is satisfiable. Then, there exists a truth assignment $t$ for $F$. From this $t$, we can find a vertex cover $S$ for $G_F$ as follows:

(a) For each $i$, $1 \leq i \leq n$, $S$ contains $v_i$ if $t(x_i) = 1$, and $S$ contains $\overline{v}_i$ if $t(x_i) = 0$.

(b) For each $j$, $1 \leq j \leq m$, let $k$ be the least integer in $\{1, 2, 3\}$ such that the $k$th literal of $C_j$ is true under $t$. Add $u_{j,\ell}$ to $S$, for $\ell \in \{1, 2, 3\} - \{k\}$.

To check that $S$ is indeed a vertex cover, we note that the only edges which are not obviously covered by $S$ are edges between a vertex $v_i$ (or, $\overline{v}_i$) and a vertex $u_{j,k}$. Note, however, that if such an edge exists, it means $x_i$ (or, respectively, $\overline{x}_i$) is a literal of clause $C_j$. By the design (b) above, either this literal is true under $t$ and, hence, $x_i$ (or, respectively, $\overline{x}_i$) is in $S$, or it is false under $t$ and then $u_{j,k}$ must be in $S$. This shows that $S$ is a vertex cover of $G_F$.

Conversely, suppose that $G_F$ has a vertex cover $S$ of size at most $n + 2m$. From the basic observation, we know that $S$ is of size exactly $n + 2m$ and includes exactly one vertex from each pair $v_i$ and $\overline{v}_i$ and exactly two vertices from each triangle $u_{j,1}, u_{j,2}$ and $u_{j,3}$. Define an assignment $t$ by $t(x_i) = 1$ if and only if $v_i \in S$, for $i = 1, \ldots, n$. We claim that $t$ satisfies $F$. To see this, we note that for each $j$, $1 \leq j \leq m$, if $u_{j,k} \notin S$, and if the $k$th literal of $C_j$ is $x_i$ (or, $\overline{x}_i$), then there is an edge between $v_i$ (or, respectively, $\overline{v}_i$) and $u_{j,k}$, and so we must have $v_i \in S$ (or, respectively, $\overline{v}_i \in S$). By the definition of $t$, the $k$th literal thus is satisfied. Since for each $j$, there is exactly one vertex $u_{j,k} \notin S$, we know that each clause is satisfied by $t$. Thus, $t$ satisfies $F$.

Finally, we observe that the above construction from $F$ to $(G_F, n + 2m)$ can clearly be done in polynomial time. Hence, we have proved that 3SAT $\leq_m^P$ VC. □

**Example 7.27** 3SAT $\leq_m^P$ HC.

*Proof.* Let $F$ be a 3-CNF formula of $n$ variables $x_1, \ldots, x_n$ and $m$ clauses $C_1, \ldots, C_m$. We need to construct a graph $G_F$ such that $F \in$ 3SAT if and

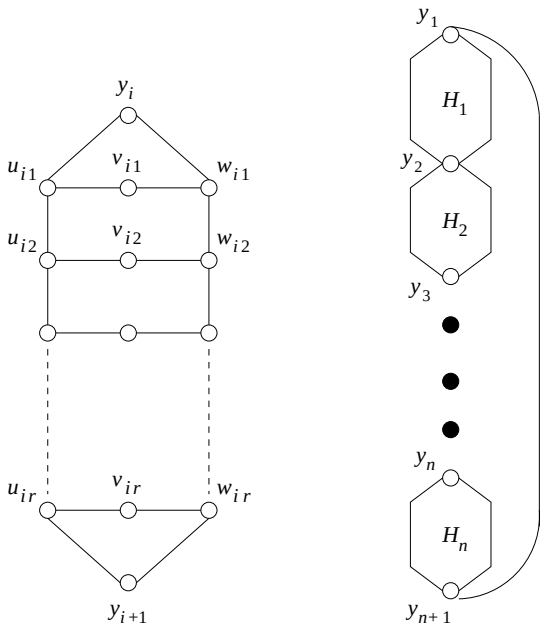


Figure 7.5: The graphs $H_i$ and $H$.

only if $G_F \in$ HC. Again, we try to design a reduction $f$ that preserves the witnesses. Here, a witness to a formula $F$ in 3SAT is a truth assignment, and a witness to a graph $G$ in HC is a Hamiltonian cycle in $G$. So, the basic idea is to divide the graph $G_F$ into $n$ subgraphs $H_i$, $1 \leq i \leq n$, each having two different Hamiltonian paths, corresponding to two assignments to variable $x_i$. Altogether, there are $2^n$ different Hamiltonian paths, corresponding to $2^n$ different assignments on variables $x_1, x_2, \ldots, x_n$.

To be more precise, our graph $G_F$ contains $n$ subgraphs $H_1, \ldots, H_n$. Each subgraph $H_i$ is a ladder of $3r + 2$ vertices, as shown in Figure 7.5(a), where $r$ is an integer parameter to be determined later. We identify the bottom vertex $y_{i+1}$ of subgraph $H_i$ with the top vertex of $H_{i+1}$, and join vertex $y_1$ with vertex $y_{n+1}$ by an edge. Then, we obtain a graph $H$ of $G_F$, as shown in Figure 7.5(b).

We note that each subgraph $H_i$ has two Hamiltonian paths from $y_i$ to $y_{i+1}$, one beginning with the edge from $y_i$ to $u_{i,1}$, and the other beginning with the edge from $y_i$ to $w_{i,1}$. They correspond to the two assignments to variable $x_i$. So, a Hamiltonian cycle from $y_1$ back to $y_1$ corresponds to an assignment to all variables.

To complete the construction of the graph $G_F$, we need to design some new vertices and edges to encode the relations among variables that are defined

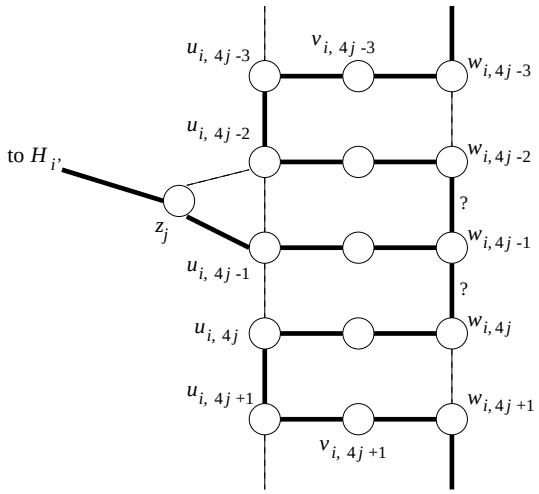


Figure 7.6: $Q$ cannot pass both $w_{i,4j-2}$ and $w_{i,4j}$.

by the clauses of $F$. First, we let $r = 4m$ so that there are four levels of the ladder corresponding to each clause $C_j$. Now, for each clause $C_j$, $1 \leq j \leq m$, we define a new vertex $z_j$. Then, we connect $z_j$ to $u_{i,4j-1}$ and $u_{i,4j-2}$ if $C_j$ contains literal $x_i$, and connect $z_j$ to $w_{i,4j-1}$ and $w_{i,4j-2}$ if $C_j$ contains literal $\overline{x}_i$. Thus, each $z_j$ has degree 6.

We note that these new vertices and edges are designed in such a way that a Hamiltonian cycle of $G_F$ must essentially follow the same path as a Hamiltonian cycle of $H$. That is, it must go from $y_1$ to $y_2$, passing through all vertices in $H_1$, then it goes from $y_2$ to $y_3$, passing through all vertices in $H_2$, etc. The new vertices $z_j$ must be visited between two vertices $u_{i,4j-1}$ and $u_{i,4j-2}$ or between two vertices $w_{i,4j-1}$ and $w_{i,4j-2}$ for some $i = 1, \ldots, n$. This fact can be verified as follows: Suppose otherwise that a Hamiltonian cycle $Q$ contains edge $\{u_{i,4j-1}, z_j\}$ and then jumps to subgraph $H_{i'}$ for some $i' \neq i$. Since each vertex $v_{i,k}$ has only degree 2, $Q$ must contain edges $\{u_{i,k}, v_{i,k}\}$ and $\{v_{i,k}, w_{i,k}\}$. This implies that $Q$ must contain both edges $\{u_{i,4j-2}, u_{i,4j-3}\}$ and $\{u_{i,4j}, u_{i,4j+1}\}$. Furthermore, $w_{i,4j-1}$ and $w_{i,4j-2}$ are not connected to $z_j$ since $C_j$ cannot contain both $x_i$ and $\overline{x}_i$. Thus, $Q$ must contain both edges $\{w_{i,4j-1}, w_{i,4j}\}$ and $\{w_{i,4j-1}, w_{i,4j-2}\}$. These observations lead to a situation as shown in Figure 7.6, which is a contradiction. (In Figure 7.6, the thick lines denote the path $Q$.)

Now, we are ready to prove that $F$ is satisfiable if and only if $G_F$ has a Hamiltonian cycle. First, suppose that $F$ is satisfiable and that $t$ is a truth assignment for $F$. Let $Q'$ be the Hamiltonian cycle of $H$, with the edge $\{y_i, u_{i,1}\}$ in $Q'$ if and only if $t(x_i) = 1$, for $1 \leq i \leq n$. We then modify $Q'$ as follows: For each vertex $z_j$, $1 \leq j \leq m$, find a literal in $C_j$ which is assigned with value 1 by $t$. If this literal is $x_i$ (or, $\overline{x}_i$), then $Q'$ must contain edge

$\{u_{i,4j-1}, u_{i,4j-2}\}$ (or, respectively, edge $\{w_{i,4j-1}, w_{i,4j-2}\}$); we replace this edge by two edges $\{u_{i,4j-1}, z_j\}$ and $\{z_j, u_{i,4j-2}\}$ (or, respectively, by edges $\{w_{i,4j-1}, z_j\}$ and $\{z_j, w_{i,4j-2}\}$). Clearly, after the modification, $Q'$ becomes a cycle $Q$ that passes through all vertices in $G_F$ exactly once, i.e. a Hamiltonian cycle of $G_F$.

Conversely, suppose that $G_F$ has a Hamiltonian cycle $Q$. From the above analysis, we know that $Q$ must pass through each vertex $z_j$ *locally* through $u_{i,4j-1}$ and $u_{i,4j-2}$ or through $w_{i,4j-1}$ and $w_{i,4j-2}$. Let $t(x_i) = 1$ if and only if edge $\{y_i, u_{i,1}\}$ is in $Q$. Then, we claim that $t$ satisfies every clause $C_j$, $1 \leq j \leq m$. To see this, we check that if $Q$ contains edges $\{u_{i,4j-1}, z_j\}$ and $\{z_j, u_{i,4j-2}\}$, then $x_i$ is a literal in $C_j$. In addition, $Q$ must contain edge $\{y_i, u_{i,1}\}$ and so $t(x_i) = 1$. Similarly, if $Q$ contains edges $\{w_{i,4j-1}, z_j\}$ and $\{z_j, w_{i,4j-2}\}$, then we must have $\overline{x}_i$ as a literal in $C_j$ and $t(x_i) = 0$. Either way, $C_j$ is satisfied by $t$.

Finally, we remark that the construction from $F$ to graph $G_F$ can be done in polynomial time and, hence, this is a polynomial-time reduction from 3SAT to HC. □

## Exercise 7.2

**1.** Prove that CNF-SAT $\leq_m^P$ 3SAT.

**2.** In this exercise, we study an alternative proof for Example 7.25. Let us replace conditions (1) and (2) by condition (3): For every $a \in A \cup B \cup C$, there is one $w \in W'$ such that (i) $a \in w$, and (ii) if $w' \in W'$, $w' \neq w$ then $a \notin w'$. Translate this condition into a Boolean formula $G$ over variables $x_w$ and show that $G$ is satisfiable if and only if $W$ has a three-dimensional matching $W'$.

**3.** Construct the following reductions:

(a) 3SAT $\leq_m^p$ 3DM.
(b) VC $\leq_m^P$ HC.
(c) HC $\leq_m^P$ 3SAT.
(d) VC $\leq_m^P$ HS.

**4.** Consider the following variations of the problem 3SAT:

> 3SAT-EXACT-ONE: Given a 3-CNF $F$, determine whether there is an assignment $t$ on variables in $F$ that assigns value TRUE to exactly one literal in each clause of $F$.
>
> 3SAT-NOT-ALL: Given a 3-CNF $F$, determine whether there is an assignment $t$ on variables in $F$ that assigns value TRUE to either one or two literals (but not all three literals) in each clause of $F$.

Show that 3SAT, 3SAT-EXACT-ONE and 3SAT-NOT-ALL are polynomial time equivalent under $\leq_m^P$.

⋆ **5.** Consider the following problems:

SUBGRAPH ISOMORPHISM (SGISO): Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, determine whether there is a one-to-one mapping $f : V_1 \to V_2$ such that, for all $u, v \in V_1$, $\{u, v\} \in E_1$ implies $\{f(u), f(v)\} \in E_2$.

GRAPH AUTOMORPHISM (GAUTO): Given a graph $G = (V, E)$, determine whether there is a one-to-one function $f : V \to V$ other than the identity function such that, for all $u, v \in V$, $\{u, v\} \in E$ if and only if $\{f(u), f(v)\} \in E$.

Prove all the polynomial-time reductions you can find among the three problems GISO, SGISO and GAUTO.

## 7.3 Cook's Theorem

In Section 5.4, we have defined a complete r.e. set as an r.e. set $A$ with the property that $B \leq_m A$ for all r.e. sets $B$. In a sense, a complete r.e. set $A$ is the *hardest* r.e. set meaning that if there were a TM solving problem $A$ then every r.e. set $B$ could be solved by a TM. We now extend this notion to the class $NP$. We say a set $A$ is *NP-complete* if

(a) $A$ is in $NP$; and

(b) $B \leq_m^P A$ for all $B \in NP$.

(A problem $A$ satisfying condition (b) alone is called *NP-hard*.) So, an $NP$-complete set $A$ is one of the hardest sets in $NP$, in the sense that if we have a deterministic polynomial-time algorithm for $A$ then every set in $NP$ has such an algorithm and so $P = NP$. In other words, an $NP$-complete set $A$ has the property that $A \in P$ if and only if $P = NP$.

**Example 7.28** *Show that if $P = NP$, then all nonempty proper subsets $A$ of $\Sigma^*$ are NP-complete.*

*Proof.* Let $a_0 \in \Sigma^* - A$ and $a_1 \in A$ be two fixed string. If $P = NP$, then every set $B$ in $NP$ can be reduced to set $A$ by the function

$$f(x) = \begin{cases} a_1, & \text{if } x \in B, \\ a_0, & \text{otherwise,} \end{cases}$$

because the question of whether $x \in B$ can be decided in polynomial time. □

**Example 7.29** *If $A$ is known to be NP-complete, is $A^2$ always NP-complete?*

*Solution.* The answer is NO if $P \neq NP$. Let $A \subseteq \{0,1\}^*$ be any $NP$-complete set. For every $x$, define $d(x)$ to be the string $x$ with each symbol doubled (e.g., $d(0100) = 00110000$). Define

$$B = \{d(x) \mid x \in A\} \cup \{y \mid |y| \text{ is odd }\} \cup \{\varepsilon\}.$$

Then, $B$ is still $NP$-complete: If $\varepsilon \in A$ then $A \leq_m^P B$ by function $d$; otherwise, if $\varepsilon \notin A$ then $A \leq_m^P B$ by function $f$, where $f(x) = d(x)$ if $x \neq \varepsilon$ and $f(\varepsilon) = d(x_0)$ for some fixed $x_0 \in \overline{A} - \{\varepsilon\}$. However, $B^2 = \{0,1\}^*$, because every string $z$ of odd length is the concatenation of $z$ and the empty string $\varepsilon$, and every nonempty string $z$ of even length is the concatenation of two strings of odd length. □

We are going to see that most problems studied in Section 7.2 are $NP$-complete. In the following, we first show that SAT is $NP$-complete by presenting a generic reduction from a problem $A$ in $NP$ to SAT.

⋆ **Theorem 7.30** (Cook's Theorem) SAT *is NP-complete.*

*Proof.* We have shown in Example 7.2 that SAT is in $NP$. To prove this theorem, we need to prove that for every set $A$ in $NP$, $A \leq_m^P$ SAT. Or, equivalently, we need to construct, from a given NTM $M$ with a polynomial time bound $p(n)$, and a given input string $x \in \Sigma^*$, a Boolean formula $F_x$ such that $M$ accepts $x$ if and only if $F_x$ is satisfiable.

Before we describe how to construct $F_x$, let us study the computation of $M$ on an input string $x$ more carefully. First, let us assume that $M$ is an NTM of the type defined in Section 4.1. That is, its initial state is $s$; its final state is $h$; and it has a single one-way infinite tape. (We call the leftmost cell of the tape the 0th cell, and the one to its right the first cell, etc.) Thus, its initial configuration on input $x$ is $(s, \mathtt{B}x\underline{\mathtt{B}})$.

We note that if $x \in L(M)$, then there is an accepting computation path of $M$ on $x$ of length at most $p(n) + 1$ (i.e., containing at most $p(n) + 1$ configurations). In addition, since the tape head of $M$ can move to the left or right at most one cell at a time, each configuration has length at most $p(n) + n + 2$.

Let $r(n) = p(n) + n + 1$. Then, $x \in L(M)$ if and only if there is a sequence of configurations $\alpha_0, \alpha_1, \ldots, \alpha_{r(n)}$ such that the following conditions hold:

(a) Each $\alpha_i$ contains exactly $r(n) + 1$ symbols. (We can pad extra blanks to the right, if necessary.)

(b) $\alpha_0$ is the initial configuration of $M$ on input $x$.

(c) $\alpha_{r(n)}$ is an accepting configuration.

(d) For any $i = 0, \ldots, r(n) - 1$, either $\alpha_i \vdash_M \alpha_{i+1}$, or $\alpha_i$ is an accepting configuration and $\alpha_i = \alpha_{i+1}$. (We use $\alpha_i \vdash \alpha_{i+1}$ to denote both conditions.)

To further simplify the above conditions, let us express each configuration using exactly $r(n)+1$ symbols by attaching the state symbol $q$ to the tape symbol $a$ in the cell the tape head is scanning. Let $\Gamma$ be the set of tape symbols used by $M$, and $Q$ the set of states of $M$ (including the halting state $h$). Then, each symbol in a configuration is a symbol in $\Gamma' = \Gamma \cup (Q\#\Gamma)$. That is, the configuration

$$(q, a_{i_1}a_{i_2}\cdots a_{i_{k-1}}\underline{a_{i_k}}a_{i_{k+1}}\cdots a_{i_m})$$

is expressed by exactly $m$ symbols:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $a_{i_1}$ | $a_{i_2}$ | $\cdots$ | $a_{i_{k-1}}$ | $q\#a_{i_k}$ | $a_{i_{k+1}}$ | $\cdots$ | $a_{i_m}$ |

.

Let $s_{i,j}$ denote the $j$th symbol of the configuration $\alpha_i$. Then, the above condition can be restated as follows: $x \in L(M)$ if and only if there exists $(r(n)+1)^2$ symbols $s_{i,j} \in \Gamma'$, $i, j \in \{0, 1, \ldots, r(n)\}$, such that

(1) For each $i$, $0 \le i \le r(n)$, exactly one symbol of $s_{i,0}, s_{i,1}, \ldots, s_{i,r(n)}$ is in $Q\#\Gamma$. (So, each string $s_{i,0}s_{i,1}\cdots s_{i,r(n)}$ is a legal configuration.)

(2) The string $s_{0,0}s_{0,1}\cdots s_{0,r(n)}$ is the initial configuration of $M$ on input $x$.

(3) $s_{r(n),j} = h\#a$ for some $j$, $0 \le j \le r(n)$, and some $a \in \Gamma$. (So, the string $s_{r(n),0}s_{r(n),1}\cdots s_{r(n),r(n)}$ is an accepting configuration.)

(4) For each $i$, $0 \le i \le r(n)-1$, either $s_{i+1,0}s_{i+1,1}\cdots s_{i+1,r(n)}$ is a successor configuration of $s_{i,0}s_{i,1}\cdots s_{i,r(n)}$, or they are an identical accepting configuration.

Let $S$ be the $(r(n)+1)\times(r(n)+1)$ matrix with $s_{i,j}$ as its element in the $i$th row and $j$th column. Then, conditions (1)—(3) are just simple conditions on this matrix, which are easy to be translated to Boolean formulas. Condition (4) is, however, too complicated and needs to be replaced by more concrete subconditions. The main idea here is that we can check whether $\alpha_i \vdash \alpha_{i+1}$ by examining every four-square,  -shaped window over the $i$th and $(i+1)$st rows of the matrix $S$. In other words, if the matrix $S$ satisfies condition (1), then condition (4) is equivalent to the following condition (4′):

(4′) For every pair $(i, j)$, with $0 \le i \le r(n)-1$ and $0 \le j \le r(n)$, if $s_{i,j} = a$, $s_{i+1,j-1} = b$, $s_{i+1,j} = c$ and $s_{i+1,j+1} = d$, then $(a, b, c, d)$ must satisfy one of the following conditions:

(4.1) $a = q\#u \in (Q-\{h\})\#\Gamma$, $b = p\#v \in Q\#\Gamma$, $c, d \in \Gamma$, and $(p, c, L) \in \delta(q, u)$.

(4.2) $a = q\#u \in (Q-\{h\})\#\Gamma$, $d = p\#v \in Q\#\Gamma$, $b, c, \in \Gamma$, and $(p, c, R) \in \delta(q, u)$.

(4.3) $a = c \in \Gamma$.

(4.4) $a \in \Gamma$ and $c = p\#a \in Q\#\Gamma$.

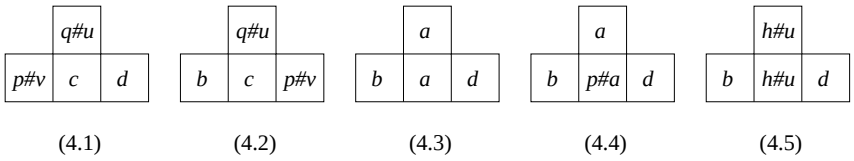


Figure 7.7: The five conditions.

(4.5) $a = c = h\#u$ for some $u \in \Gamma$.

(In the above, we assume that $s_{i,-1} = s_{i,r(n)+1} = \mathtt{B}$ for all $i$.)

The above five conditions are shown in Figure 7.7.

We now prove that (4) is equivalent to (4′), if condition (1) is satisfied. Assume that $S$ satisfies condition (1); that is, each row of $S$ is a configuration. We let $\alpha_i$ denote the $i$th row of $S$, for $i = 0, \ldots, r(n)$. We fix an $i \in \{0, \ldots, r(n)-1\}$, and call a $\square\!\!\square\!\!\square$-shaped window *the $j$th window* if the top symbol of the window is the symbol $s_{i,j}$.

First assume that $\alpha_i \vdash \alpha_{i+1}$. Then, there are three possible cases:

*Case* 1. $\alpha_i \vdash \alpha_{i+1}$ by the application of an instruction of the form $(p, c, L) \in \delta(q, u)$ of $M$. Then, $\alpha_i$ contains a unique symbol $q\#u$ in $Q\#\Gamma$; for instance, $s_{i,j} = q\#u$. Then, we know that $s_{i,k} = s_{i+1,k}$ for $k \neq j-1, j$. Therefore, the $k$th window is of the form (4.3), for $k \neq j-1, j$. Furthermore, the $j$th window must be of the form (4.1), and the $(j-1)$st window must be of the form (4.4). So, each window is of one of the five forms.

*Case* 2. $\alpha_i \vdash \alpha_{i+1}$ by the application of an instruction of the form $(p, c, R) \in \delta(q, u)$ of $M$. This case is similar to Case 1. Suppose that $s_{i,j} = q\#u$. Then, we can verify that all windows are of the form (4.3), except that the $j$th window is of the form (4.2), and the $(j+1)$st window is of the form (4.4).

*Case* 3. $\alpha_i$ is an accepting configuration and $\alpha_i = \alpha_{i+1}$. Suppose that $s_{i,j} = h\#u$ for some $u \in \Gamma$. Then, all windows are of the form (4.3), except for the $j$th window which is of the form (4.5).

Conversely, assume that every window over the $i$th and $(i+1)$st rows of $S$ is in one of the five forms. Assume that $s_{i,j} \in Q\#\Gamma$. Again, there are three cases.

*Case* 1. The $j$th window is of the form (4.1). From condition (1), we know that each row has exactly one state symbol. Therefore, the $(j-1)$st window must be of the form (4.4), and all other windows must be of the form (4.3). By condition (4,3), we know that $s_{i,k} = s_{i+1,k}$ for $k \neq j, j-1$. Also, by conditions (4.1) and (4.4), we know that the change from $(s_{i,j-1}, s_{i,j})$ to $(s_{i+1,j-1}, s_{i+1,j})$ follows an instruction $(p, c, L) \in \delta(q, u)$ in $M$. It follows that $\alpha_i \vdash \alpha_{i+1}$.

*Case* 2. The $j$th window is of the form (4.2). This is similar to Case 1.

*Case* 3. The $j$th window is of the form (4.5). Then, all other windows must be of the form (4.3), and it follows that $\alpha_i = \alpha_{i+1}$.

The above completes the proof that conditions (1)—(4) are equivalent to conditions (1)—(3) and (4′).

Now, we are ready to construct the Boolean formula $F_x$. First, we define, for each symbol $a$ in $\Gamma'$, and each pair of integers $(i, j)$ with $i, j \in \{0, \ldots, r(n)\}$, a Boolean variable $y_{i,j,a}$. In the design of the formula $F_x$, we will interprete an assignment $y_{i,j,a} = 1$ to mean that $s_{i,j} = a$. Based on this interpretation, the above four conditions on matrix $S$ can be easily translated to the following five conditions on variables $y_{i,j,a}$:

(0) For each pair $(i, j)$, with $i, j \in \{0, \ldots, r(n)\}$, there exists exactly one $a \in \Gamma'$ such that $y_{i,j,a} = 1$. (This condition fixes the relation between variables $y_{i,j,a}$ and symbols $s_{i,j}$.)

(1) For each $i$, with $0 \leq i \leq r(n)$, there exists exactly one $j$, $0 \leq j \leq r(n)$, and one $a \in Q\#\Gamma$ such that $y_{i,j,a} = 1$.

(2) $y_{0,j,a} = 1$ if and only if the the $j$th symbol of the initial configuration is $a$.

(3) $y_{r(n),j,h\#a} = 1$ for some $j \in \{0, \ldots, r(n)\}$ and some $a \in \Gamma$.

(4) For each pair $(i, j)$, with $0 \leq i \leq r(n) - 1$ and $0 \leq j \leq r(n)$,

$$y_{i,j,a}\ y_{i+1,j-1,b}\ y_{i+1,j,c}\ y_{i+1,j+1,d} = 1$$

if and only if $(a, b, c, d)$ satisfies one of the conditions (4.1)—(4.5).

For each condition $(k)$, $k = 0, \ldots, 4$, we define, in the following, a Boolean formula $f_k$ over variables $y_{i,j,a}$ such that $f_k$ is satisfied if and only if condition $(k)$ holds for variables $y_{i,j,a}$. (In $f_2$, we write $x_t$ to denote the $t$th symbol of input $x$.)

$$f_0 = \prod_{i=0}^{r(n)} \prod_{j=0}^{r(n)} \left[ \left( \sum_{a \in \Gamma'} y_{i,j,a} \right) \prod_{a,b \in \Gamma', a \neq b} (\overline{y}_{i,j,a} + \overline{y}_{i,j,b}) \right].$$

$$f_1 = \prod_{i=0}^{r(n)} \left[ \left( \sum_{j=0}^{r(n)} \sum_{a \in Q\#\Gamma} y_{i,j,a} \right) \prod_{0 \leq j < j' \leq r(n)} \prod_{a,b \in Q\#\Gamma} (\overline{y}_{i,j,a} + \overline{y}_{i,j',b}) \right].$$

$$f_2 = y_{0,0,\mathtt{B}}\ y_{0,1,x_1}\ y_{0,2,x_2} \cdots y_{0,n,x_n}\ y_{0,n+1,s\#\mathtt{B}}\ y_{0,n+2,\mathtt{B}} \cdots y_{0,r(n),\mathtt{B}}.$$

$$f_3 = \sum_{j=0}^{r(n)} \sum_{a \in \Gamma} y_{r(n),j,h\#a}.$$

$$f_4 = \prod_{i=0}^{r(n)-1} \prod_{j=0}^{r(n)} \sum_{(a,b,c,d) \in T} y_{i,j,a}\ y_{i+1,j-1,b}\ y_{i+1,j,c}\ y_{i+1,j+1,d},$$


Figure 7.8: These two windows cannot detect the illegal transition.

where $T = \{(a,b,c,d) \in (\Gamma')^4 \mid (a,b,c,d)$ satisfies one of the conditions (4.1)—(4.5)$\}$, and $y_{i,-1,a}$ and $y_{i,r(n)+1,a}$ denote the constant 1 if $a = \mathtt{B}$, and denote the constant 0 if $a \neq \mathtt{B}$.

Now, we let $F_x = f_0\, f_1\, f_2\, f_3\, f_4$. From the above analysis, we see that $F_x$ is satisfiable if and only if $x \in L(M)$. Thus, this is a reduction from $L(M)$ to SAT.

Finally, we remark that the construction can be done in polynomial time, as the lengths of the formulas $f_k$, for $k = 0, \ldots, 4$, are in $O(m^4 \cdot (r(n))^3)$, where $m$ is the number of symbols in $\Gamma'$, and the definition of each $f_k$ is constructive. We conclude that $L(M) \leq_m^P$ SAT. □

The critical part of the above proof is the relation between condition (4) and condition (4′). In the following examples, we study some alternatives to condition (4′).

**Example 7.31** *In the proof of Cook's Theorem, we have used conditions* (4.1) — (4.5) *on* ⊓⊓-*shaped windows over matrix* $S$ *to satisfy condition* (4). *Show that if we, instead, work with* ⊔⊓-*shaped windows over matrix* $S$, *then no conditions on these windows can satisfy condition* (4). *In other words, show that the following relation does not hold for any subset* $T \subseteq (\Gamma')^4$:

$$\left( \prod_{j=0}^{r(n)} \sum_{(a,b,c,d) \in T} y_{i,j-1,a}\, y_{i,j,b}\, y_{i,j+1,c}\, y_{i+1,j,d} \right) = 1 \iff \alpha_i \vdash \alpha_{i+1},$$

*where* $\alpha_i$ *denotes the* $i$*th configuration corresponding to the assingments to variables* $y_{i,j,a}$.

*Proof.* Note that $M$ is an NTM, and so $\delta(q,u)$ may contain more than one triple. Suppose that $\delta(q,u) = \{(p,v,L),(r,w,L)\}$, with $p \neq r$ and $v \neq w$. Then, consider the case where $s_{i,j-1} = a$, $s_{i,j} = q\#u$, $s_{i+1,j-1} = p\#a$ and $s_{i+1,j} = w$, for some $a \in \Gamma$. It is clear that $\neg(\alpha_i \vdash \alpha_{i+1})$. However, both the $(j-1)$st window and the $j$th window are clearly legal (see Figure 7.8). Thus, if the set $T$ contains all legal windows, it cannot tell that $\neg(\alpha_i \vdash \alpha_{i+1})$. □

**Example 7.32** *Show that, in the proof of Cook's Theorem, function* $f_1$ *is necessary. That is, no definition of subset* $T$ *can make the following true:* $F'_x = f_0 f_2 f_3 f_4$ *is satisfiable if and only if* $x \in L(M)$.

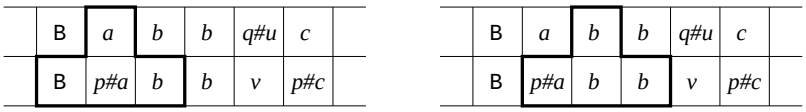


Figure 7.9: Example 7.32

*Proof.* Without condition (1), conditions (4) and (4′) are not equivalent, no matter how we define subset $T$. We note that in order to satisfy $f_4$ when $\alpha_i \vdash \alpha_{i+1}$, we must define $T$ to include conditions (4.1)—(4.5). Now, consider the case shown in Figure 7.9. It is apparent that the second row cannot be derived from the first row, since the second row has two symbols in $Q\#\Gamma$. However, all windows over this submatrix are legal according to $T$. □

⋆ **Example 7.33** *In the proof of Cook's Theorem, it is possible to remove condition* (1)*, as long as we replace condition* (4′) *by a new condition* (4″) *over six-square,* ⊞*-shaped windows. Find the subconditions over the six-square,* ⊞*-shaped windows that make* (4″) *equivalent to condition* (4)*, without the assumption of condition* (1)*.*

*Solution.* There are twelve possible legal, six-square windows with $s_{i,j-1} = a$, $s_{i,j} = b$, $s_{i,j+1} = c$, $s_{i+1,j-1} = d$, $s_{i+1,j} = e$ and $s_{i+1,j+1} = g$. We group them in three cases:

*Case* 1. There is an instruction $(p, v, L) \in \delta(q, u)$, and the window is consistent with this instruction. We show the four corresponding windows in the first row of Figure 7.10.

*Case* 2. There is an instruction $(p, v, R) \in \delta(q, u)$, and the window is consistent with this instruction. We show the four corresponding windows in the second row of Figure 7.10.

*Case* 3. All of $a, b, c$ are in $\Gamma \cup (\{h\}\#\Gamma)$, and $a = d$, $b = e$ and $c = g$. We show the four corresponding windows in the third row of Figure 7.10.

Now, define

(4″) For every pair $(i, j)$, with $0 \leq i \leq r(n)-1$ and $0 \leq j \leq r(n)$, if $s_{i,j-1} = a$, $s_{i,j} = b$, $s_{i,j+1} = c$, $s_{i+1,j-1} = d$, $s_{i+1,j} = e$ and $s_{i+1,j+1} = g$, then $(a, b, c, d, e, g)$ must be in one of the twelve forms shown in Figure 7.10.

To see that condition (4″) is equivalent to condition (4), it suffices to show that if matrix $S$ satisfies (4″) then matrix $S$ satisfies condition (1), since these twelve windows are consistent with all five subwindows of Figure 7.7. We prove this by induction. First, we know from condition (2) that the first row of $S$ contains exactly one symbol in $Q\#\Gamma$.

Next, we assume that the $i$th row of $S$ has exactly one symbol in $Q\#\Gamma$, and that it is $s_{i,j} = q\#u$. Let us call a window *the $\ell$th window* if the top middle symbol of the window is $s_{i,\ell}$. Then, the $j$th window is in the form of window

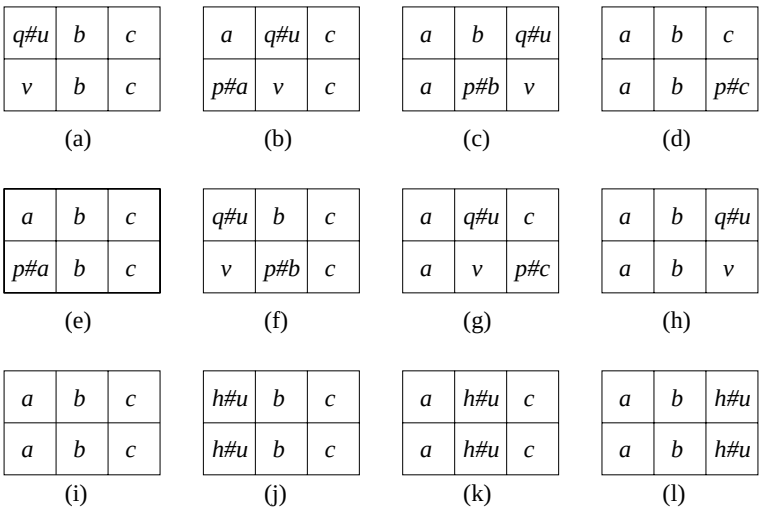


Figure 7.10: Twelve possible windows.

(b), (g) or (k) in Figure 7.10. If the $j$th window is of the form (b), then there must be an instruction $(p, v, L) \in \delta(q, u)$. It follows that the $(j-2)$nd window must be of the form (d) in Figure 7.10, the $(j-1)$st window must be of the form (c), and the $(j+1)$st window must be of the form (a), since these are the only windows which are consistent with the $j$th window. (The $(j-2)$nd window cannot be of the form (g), for otherwise there would be two symbols in $Q\#\Gamma$ in the $i$th row of $S$.)

We also claim that the $k$th window must be of the form (i) in Figure 7.10, if $k < j-2$ or $k > j+1$. To see this, we note that if $k$ satisfies the above condition, then the top three symbols of the $k$th window must be all in $\Gamma$ (by the inductive hypothesis). Therefore, the only possible window forms for it are (d), (e) and (i). However, if the $k$th window is of the form (d), then the $(k+1)$st window must be of the form (c), and so $s_{i,k+2} \in Q\#\Gamma$, which is a contradiction. Similarly, we can see that the form (e) is not possible. So, the claim is proven. It follows that the $(i+1)$st row of $S$ contains exactly one symbol in $Q\#\Gamma$.

If the $j$th window is of the form (g) or (k), then the same argument can be applied to show that the $(i+1)$st row of $S$ contains exactly one symbol in $Q\#\Gamma$. We leave the detail to the reader. This completes the induction proof. □

## Exercise 7.3

**1.** A Boolean formula which is a product of literals is called an *elementary product.* A *DNF (disjunctive normal form)* is a sum of elementary

products.

(a) Prove that a DNF Boolean formula can be switched in polynomial time to a CNF formula (with possibly more variables) that preserves satisfiability.

(b) Prove that if $P \neq NP$, then there is no polynomial-time algorithm that switches a CNF Boolean formula to a DNF formula, and preserves satisfiability.

**2.** Assume that $A$ and $B$ are two $NP$-complete sets. Show that $A \cup B$ and $A \cap B$ are not necessarily $NP$-complete, if $P \neq NP$. Is $A \cup B$ always $NP$-complete when $A \cap B = \emptyset$?

**3.** This exercise follows the notations defined in Exercise 3 of Section 7.1.

(a) Show that there exists a polynomial-time computable, polynomially honest function $f$ such that its range $f(\{0,1\}^*)$ is $NP$-complete. [Hint: For any $NP$-complete set $A$, $f$ takes an instance $x$ of $A$ and a string $y$ as the input and decides whether $y$ is a witness to $x \in A$.]

(b) Define $S_f = \{\langle u, v, y\rangle \mid y \geq_{\text{lev}} Min_f(u, v)\}$. Show that there exists a polynomial-time computable, polynomially honest function $f$ such that $S_f$ is $NP$-complete. [Hint: Design a function like part (a), except that the problem $A$ is a minimization problem.]

**4.** Assume that $P \neq NP$. Let $A$ and $B$ be two sets that are complete for *co-NP* (i.e., $\overline{A}$ and $\overline{B}$ are $NP$-complete). Is $AB$ necessarily in *co-NP*? Is $AB$ necessarily *co-NP*-complete? Is it possible that $AB$ is in $P$? Justify your answer.

⋆ **5.** Suppose that, in the proof of Cook's Theorem, we drop condition (1) and replace condition $(4')$ by $(\overline{4})$ which specifies some subconditions on five-square windows of one of the four shapes shown in Figure 7.11. Show that no such subconditions on the five-square windows can make $(\overline{4})$ equivalent to $(4')$.

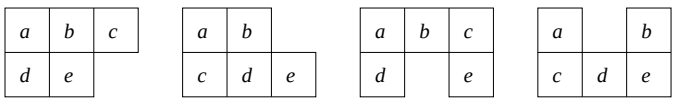


Figure 7.11: Five-square windows.

⋆ **6.** Consider the alternative proof of Cook's Theorem given in Example 7.33. Let $T \subseteq (\Gamma')^6$ be the set corresponding to the twelve windows of Figure 7.10; that is, $(a, b, c, d, e, g) \in T$ if and only if ($a$, $b$, $c$, $d$, $e$, $g$) is in one of the twelve forms in Figure 7.10, and satisfies the

corresponding conditions given in the three cases. Show that if there exists a mapping $\phi : (\Gamma')^3 \to (\Gamma')^3$ such that $(a, b, c, d, e, g) \in T$ if and only if $\phi(a, b, c) = (d, e, g)$, then $L(M)$ is in $P$.

⋆ **7.** In this exercise, we consider a different setting for the proof of Cook's Theorem. Instead of attaching the state symbol to the tape symbol currently scanned by the tape head, we may define separate Boolean variables to represent the state and the position of the tape head. That is, in addition to variables $y_{i,j,a}$ (for $a \in \Gamma$ only), we define, for each $i = 0, \ldots, r(n)$ and each $q \in Q$, a variable $Q_{i,q}$ to mean that the state symbol of $\alpha_i$ is $q$, and for each pair $(i, j)$, with $i, j \in \{0, \ldots, r(n)\}$, a variable $H_{i,j}$ to mean that, in the configuration $\alpha_i$, the tape head is scanning the $j$th symbol. To prove Cook's Theorem in this setting, then we need to define six Boolean functions $g_1, \ldots, g_6$, such that, for $k = 1, \ldots, 6$, $g_k = 1$ if and only if the condition $(k)$ below holds:

(1) For each $i = 0, \ldots, r(n)$, $\alpha_i$ is in exactly one state.

(2) For each $i = 0, \ldots, r(n)$, the head scans exactly one cell in $\alpha_i$.

(3) For each $i = 0, \ldots, r(n)$ and each $j = 0, \ldots, r(n)$, the $j$th cell of $\alpha_i$ contains exactly one symbol.

(4) $\alpha_0$ is the initial configuration of $M$ on input $x$.

(5) $\alpha_{r(n)}$ is an accepting configuration.

(6) For each $i = 0, \ldots, r(n) - 1$, $\alpha_i \vdash \alpha_{i+1}$.

Assume that $g_1, \ldots, g_5$ satisfy the condition that $g_k = 1$ if and only if condition $(k)$ holds. Also let $g_6$ be

$$\prod_{i=0}^{r(n)} \prod_{j=0}^{r(n)} \prod_{q \in Q} \prod_{u \in \Gamma} \sum_{(p,v,D) \in \delta(q,u)} \Big[ (\overline{H}_{i,j} + \overline{Q}_{i,q} + \overline{y}_{i,j,u} + H_{i+1,j+\Delta}) \cdot (\overline{H}_{i,j} + \overline{Q}_{i,q} + \overline{y}_{i,j,u} + Q_{i+1,p}) \cdot (\overline{H}_{i,j} + \overline{Q}_{i,q} + \overline{y}_{i,j,u} + y_{i+1,j,v}) \Big],$$

where $\Delta = -1$ if $D = L$, and $\Delta = 1$ if $D = R$. Find a counterexample to disprove that $G_x = \prod_{i=1}^{6} g_i = 1$ if and only if $M$ accepts $x$. Find a new function $g_6$ satisfying that $G_x = 1$ if and only if $M$ accepts $x$.

## 7.4 More NP-Complete Problems

We have seen, from Cook's Theorem, that SAT is *NP*-complete. In this section, we prove more *NP*-complete problems. First, we recall that $\leq_m^P$ is a transitive relation (Proposition 7.19(b)), and so we have:

**Proposition 7.34** *If $A$ is NP-complete, $B \in NP$ and $A \leq_m^P B$, then $B$ is also NP-complete.*

From this proposition, we can prove a problem *NP*-complete by reducing a known *NP*-complete problem to it. Therefore, most problems we studied in Section 7.2 are in fact *NP*-complete.

**Corollary 7.35** CNF-SAT, 3SAT, HC, LP, IS, VC, IP, *and* 3DM *are NP-complete.*

*Proof.* Example 7.23 established CNF-SAT as an *NP*-complete problem. Exercise 1 of Section 7.2 showed that 3SAT is *NP*-complete. The other examples of Section 7.2 showed that HC, LP, IS, VC, and IP are all *NP*-complete. Finally, Exercise 3(a) of Section 7.2 shows that 3SAT $\leq_m^P$ 3DM, and so 3DM is *NP*-complete. □

To understand how difficult an *NP*-complete problem is, we often consider its subproblems by adding restrictions to the problem. Depending on how strong the restriction is, the new subproblem may remain *NP*-complete or may become polynomial-time solvable. For instance, for the problem VC, we may restrict the input graphs to be planar graphs, and the problem is still *NP*-complete (see Exercise 2(a) of this section). On the other hand, if we require the vertex cover to be also an independent set, then the problem VC becomes in *P*.

**Example 7.36** *Does the problem* VC *remain NP-complete if each edge is to be covered by exactly one vertex?*

*Solution.* The answer is NO if and only if $P \neq NP$ (if $P = NP$ then all nontrivial sets in *P* are *NP*-complete; see Example 7.28).

Let EXACT-VC denote the set of pairs $(G = (V, E), k)$ for which there is a subset $A \subseteq V$ of size $k$ such that every edge in $E$ is covered by exactly one vertex in $A$. We note that if set $A \subseteq V$ covers every edge with exactly one vertex, then $A$ is an independent set of $G$. In addition, we know, from Example 7.21, that the complement of a vertex cover is an independent set, and so, $V - A$ is also an independent set of $G$. This means that both $A$ and $V - A$ are independent sets, and so the graph $G$ is a bipartite graph (i.e., a graph whose vertex set can be partitioned into two subsets $A$ and $B$ such that its edges are all between a vertex in $A$ and a vertex in $B$). Conversely, suppose $G = (V, E)$ is a bipartite graph with $V = A \cup B$, $A \cap B = \emptyset$ and each edge in $E$ connects a vertex in $A$ and a vertex in $B$. Then, $G$ has a vertex cover $A$ which covers exactly one vertex of each edge.

So, we have proved that $(G, k) \in$ EXACT-VC if and only if $G$ is a bipartite graph with one of its vertex subset of size less than or equal to $k$. The latter property of a graph $G$ is easily recognized in polynomial time (by, for instance, a breadth-first search over the graph). Therefore, the problem EXACT-VC is in *P*. □

In the following, we present a few more famous *NP*-complete problems. For a graph $G = (V, E)$ and a set $A \subseteq V$, the *induced subgraph* $G|_A$ of $G$

on the vertex set $A$ is the graph with the vertex set $A$ and the edge set $E_A = \{\{u, v\} \in E \mid u, v \in A\}$.

> MINMUM CONNECTIVITY GRAPH (MCG): Given $n$ subsets $X_1$, $X_2, \ldots, X_n$ of a set $X$ and a positive integer $k$, find a graph $G$ over the vertex set $X$, with at most $k$ edges, such that every induced subgraph $G|_{X_i}$ of $G$, $1 \leq i \leq n$, is connected.

**Example 7.37** *Prove that* MCG *is NP-complete.*

*Proof.* The problem MCG is in *NP*, since we can guess a graph $G$ on $X$ and check, in polynomial time, whether $G$ has at most $k$ edges and whether every $G|_{X_i}$ is connected. (To determine whether a given graph is connected, we can perform a standard depth-first search over the graph $G$, which visits each edge at most twice and so runs in linear time.)

To show that MCG is *NP*-hard, we reduce VC to it. For an instance $(G, h)$ of VC, where $G = (V, E)$, we let $X = V \cup \{x\}$, where $x \notin V$, $k = |E| + h$, and construct, for each $e \in E$, two subsets of $X$: $X_{1,e} = e$, $X_{2,e} = e \cup \{x\}$. So, together, we have $2|E|$ subsets.

We say a graph $H$ over the vertex set $X$ is *feasible* if $H|_{X_{i,e}}$ is connected, for all $e \in E$ and $i = 1, 2$. Suppose that $G$ has a vertex cover $C$ of size $\leq h$. Then, $G \cup \{\{x, v\} \mid v \in C\}$ is a feasible graph with at most $|E| + h$ edges. Indeed, $G|_{X_{1,e}}$ contains exactly one edge $e$ and so is connected; and $G|_{X_{2,e}}$ contains the edge $e$ plus an edge between $x$ and a vertex in $e \cap C$, and is also connected.

Conversely, suppose that there exists a feasible graph $G'$ with at most $|E| + h$ edges on vertex set $X$. Then, it must contain all edges in $G$ since $G'_{X_{1,e}}$ is connected for every $e \in E$. It follows that $x$ has degree at most $h$. In addition, since $G'_{X_{2,e}}$ is connected for every $e \in E$, every edge $e$ has at least one vertex connected to $x$ in $G'$. This means that $C = \{v \mid \{x, v\} \in G'\}$ is a vertex cover for $G$. Furthermore, this vertex cover has size at most $h$, because each vertex in $C$ is joined by an edge with $x$. It follows that $(G, h) \in$ VC. □

In the next problem, we say $(A, B)$ is a *partition* of a set $S$ if $A \cup B = S$ and $A \cap B = \emptyset$.

> PARTITION: Given $n$ positive integers $a_1, a_2, \cdots, a_n$, determine whether a partition $(I_1, I_2)$ of $\{1, 2, \cdots, n\}$ exists such that
>
> $$\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i.$$
>
> (Such a partition is called an *even partition* of the list $(a_1, \ldots, a_n)$.)

**Example 7.38** PARTITION *is NP-complete.*

*Proof.* The problem PARTITION belongs to NP since we can guess a partition of the input instance and verify in polynomial time whether the partition is even. To show the NP-hardness of PARTITION, we reduce 3DM to it. For each instance $W \subseteq X \times Y \times Z$ of 3DM, we construct an instance of PARTITION as follows:

For each element $x$ in $X \cup Y \cup Z$, denote by $\#(x)$ the number of occurrences of $x$ in $W$. Note that if there is an element $x$ in $X \cup Y \cup Z$ such that $\#(x) = 0$, then obviously $W$ does not contain a 3-dimensional matiching, and so the reduction is trivial. Therefore, we may assume, in general, $\#(x) \geq 1$ for all $x \in X \cup Y \cup Y$. Let $X = \{x_1, x_2, \cdots, x_n\}$, $Y = \{y_1, y_2, \cdots, y_n\}$, and $Z = \{z_1, z_2, \cdots, z_n\}$. For each vector $(x_i, y_j, z_k) \in W$, construct an integer $a_{ijk}$ as follows:

$$a_{ijk} = m^i + m^{n+j} + m^{2n+k},$$

where $m = |W| + 1$. Define

$$s = \sum_{(x_i, y_j, z_k) \in W} a_{ijk} \qquad \text{and} \qquad t = \sum_{i=1}^{3n} m^i.$$

Note that, by the assumption that each $x \in X \cup Y \cup Z$ has $\#(x) \geq 1$, we have $s \geq t$. The instance of PARTITION corresponding to $W$ consists of integers $b_1 = s + 1$, $b_2 = 2(s - t) + 1$, and $a_{ijk}$ for $(x_i, y_j, z_k) \in W$. We note that the sum $S$ of all these numbers is equal to

$$(s + 1) + 2(s - t) + 1 + s = 4s - 2t + 2.$$

In other words, these numbers have an even partition if and only if there is a subset of these numbers which sums to $2s - t + 1$.

Now, we observe that if $W$ has a three-dimensional matching $W'$, then

$$\sum_{(x_i, y_j, z_k) \in W'} a_{ijk} = t,$$

since each $x_i$ occurs in $W'$ exactly once, and so it contributes exactly $m^i$ to the above sum, and similarly each $y_j$ contributes $m^{n+j}$ and each $z_k$ contributes $m^{2n+k}$ to the above sum. (An easy way to visualize the above claim it to view each number in its base-$m$ representation. In this representation, $t = 1^{3n}$.) It follows that

$$b_2 + \sum_{(x_i, y_j, z_k) \in W'} a_{ijk} = 2s - t + 1 = \frac{S}{2},$$

and so this defines an even partition.

Conversely, assume that the instance of PARTITION corresponding to $W$ has an even partition. Then, $b_1$ and $b_2$ must belong to the different sides of the partition, since $b_1 + b_2 = s + 1 + 2(s - t) + 1 > S/2$. Let

$$W' = \{(x_i, y_j, z_k) \in W \mid a_{ijk} \text{ belongs to the side of } b_2\}.$$

That is,

$$b_1 + \sum_{(x_i,y_j,z_k)\in W-W'} a_{ijk} = b_2 + \sum_{(x_i,y_j,z_k)\in W'} a_{ijk} = \frac{S}{2}.$$

Therefore, we get

$$\sum_{(x_i,y_j,z_k)\in W'} a_{ijk} = \frac{S}{2} - (2(s-t)+1) = t.$$

Note that, for all $i$, $1 \le i \le n$, $\#(x_i) \le m-1$, and so all occurrences of $x_i$ contribute at most $(m-1)m^i < m^{i+1}$ to the sum. Similarly, all occurrences of $y_j$ (and $z_k$) contribute at most $(m-1)m^{n+j} < m^{n+j+1}$ (and, respectively, $(m-1)m^{2n+k} < m^{2n+k+1}$) to the sum. Therefore, to get the sum equal to $t$, there must be exactly one occurrence of each $x_i$, $y_j$ and $z_k$ in $W'$. This implies that $W'$ is a three-dimensional matching.

Finally, we note that $S = 4s - 2t + 2$ is of length $O(n \log m)$, and so the total size of the constructed instance of PARTITION is $O(mn \log m)$. It follows that it can be constructed in time polynomial in $|W|$. □

Because of its simple form, it is easy to reduce PARTITION to other problems to establish new NP-completeness results. The following are two packing problems.

> KNAPSACK: Given $2n+2$ nonnegative integers $c_1, c_2, \ldots, c_n$, $p_1, p_2, \ldots, p_n$, $s$, and $k$, determine whether there exists $x_1$, $x_2$, $\ldots, x_n \in \{0,1\}$ satisfying

$$\sum_{i=1}^{n} c_i x_i \le s, \qquad \text{and} \qquad \sum_{i=1}^{n} p_i x_i \ge k. \tag{7.2}$$

(Intuitively, the problem KNAPSACK asks, for given costs $c_1, \ldots, c_n$ and profits $p_1, \ldots, p_n$ of $n$ items, and a given bound $s$, to select a subset of items to maximize the profits, subject to the condition that the total cost does not exceed $s$.)

**Example 7.39** KNAPSACK *is NP-complete.*

*Proof.* To show that KNAPSACK is in NP, we guess an assignment of values 0 and 1 to variables $x_1, \ldots, x_n$, and check that the assignment satisfies the constraint (7.2). It is clear that the checking of the constraint (7.2) can be done in deterministic polynomial time.

Next, we show that KNAPSACK is *NP*-hard by reducing PARTITION to it. For each instance $(a_1, a_2, \ldots, a_n)$ of PARTITION, we construct an instance $(c_1, \ldots, c_n, p_1, \ldots, p_n, s, k)$ of KNAPSACK as follows:

(1) $c_i = p_i = a_i$, for $1 \le i \le n$,

$$(2)\ \ s = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} a_i \right\rfloor,$$

$$(3)\ \ k = \left\lceil \frac{1}{2} \sum_{i=1}^{n} a_i \right\rceil.$$

Clearly, $(a_1, a_2, \cdots, a_n)$ has an even partition $(I_1, I_2)$ if and only if the corresponding assignment $(x_i = 1 \Leftrightarrow i \in I_1)$ satisfies the constrait (7.2). □

> BIN PACKING (BP): Given $n + 2$ positive integers $a_1, a_2, \cdots, a_n$, $c$, and $k$, determine whether the list $(a_1, a_2, \cdots, a_n)$ can be partitioned into $k$ sublists such that the sum of $a_i$'s in each sublist is at most $c$.

(Intuitively, the problem BP asks, for $n$ given items of size $a_1, a_2, \ldots, a_n$ and $k$ given bins, each of size $c$, whether the $n$ items can be packed in these $k$ bins.)

**Example 7.40** BP *is NP-complete.*

*Proof.* To show that BP is in *NP*, we can guess a partition of the input instance into $k$ sublists and check that the total size of items in each sublist does not exceed $c$. This can apparently be done in polynomial time.

To show that BP is *NP*-hard, we reduce PARTITION to it. For each instance $(a_1, a_2, \cdots, a_n)$ of PARTITION, we construct an instance $(a_1, a_2, \cdots, a_n, c, k)$ of BP with

$$c = \left\lfloor \frac{1}{2} \cdot \sum_{i=1}^{n} a_i \right\rfloor$$

and $k = 2$. It is easy to see that $(a_1, a_2, \cdots, a_n)$ has an even partition if and only if the corresponding instance belongs to the set BP. □

⋆ **Example 7.41** *Show that the following problem is NP-complete: Given positive integers $a_1, \ldots, a_n$, determine whether*

$$\int_0^{2\pi} \left( \prod_{i=1}^{n} \cos(a_i t) \right) dt \neq 0.$$

*Proof.* We first show by induction on $n$ that

$$\prod_{i=1}^{n} \cos(a_i t) \ = \ \frac{1}{2^n} \sum_{(I_1, I_2) \in P_n} \cos\Big( \sum_{i \in I_1} a_i t - \sum_{i \in I_2} a_i t \Big), \tag{7.3}$$

where $P_n$ is the family of all possible partitions $(I_1, I_2)$ of $\{1, 2, \cdots, n\}$, with $(I_1, I_2)$ and $(I_2, I_1)$ considered as two different partitions. For $n = 1$, there are

two possible partitions $(\{1\},\emptyset)$ and $(\emptyset,\{1\})$. In this case, the equality holds trivially. Now, consider $n \geq 2$. We recall the identity

$$\cos\alpha\cos\beta = \frac{1}{2}\big(\cos(\alpha+\beta)+\cos(\alpha-\beta)\big).$$

It follows that

$$\begin{aligned}
\prod_{i=1}^{n}\cos(a_i t) &= \frac{1}{2^{n-1}}\left[\sum_{(I_1,I_2)\in P_{n-1}}\cos\Big(\sum_{i\in I_1}a_i t-\sum_{i\in I_2}a_i t\Big)\right]\cos(a_n t)\\
&= \frac{1}{2^n}\sum_{(I_1,I_2)\in P_{n-1}}\left[\cos\Big(\sum_{i\in I_1\cup\{n\}}a_i t-\sum_{i\in I_2}a_i t\Big)\right.\\
&\qquad\left.+\cos\Big(\sum_{i\in I_1}a_i t-\sum_{i\in I_2\cup\{n\}}a_i t\Big)\right]\\
&= \frac{1}{2^n}\sum_{(I_1,I_2)\in P_n}\cos\Big(\sum_{i\in I_1}a_i t-\sum_{i\in I_2}a_i t\Big).
\end{aligned}$$

Next, note that

$$\int_0^{2\pi}\cos(at)\,dt = \begin{cases} 0 & \text{if } a\neq 0\\ 2\pi & \text{if } a=0.\end{cases}$$

Thus, by (7.3), if $(a_1,a_2,\ldots,a_n)\in$ PARTITION, then there exists at least one partition $(I_1,I_2)\in P_n$ such that $\sum_{i\in I_1}a_i-\sum_{i\in I_2}a_i=0$, and so

$$\int_0^{2\pi}\left(\prod_{i=1}^{n}cos(a_i t)\right)dt \geq 2\pi.$$

Otherwise, if $(a_1,a_2,\ldots,a_n)\notin$ PARTITION, then all partitions $(I_1,I_2)\in P_n$ have $\sum_{i\in I_1}a_i-\sum_{i\in I_2}a_i\neq 0$, and so

$$\int_0^{2\pi}\left(\prod_{i=1}^{n}cos(a_i t)\right)dt = 0.$$

In other words, this problem is equivalent to PARTITION. □

We next study an interesting geometric problem. A tree in the Euclidean plane is a connected graph with no cycles, whose vertices are points in the two-dimensional plane and edges are the line segments joining two vertices. The length of an edge connecting two points $x_1=(a_1,b_1)$ and $x_2=(a_2,b_2)$ is $d(x_1,x_2)=((a_1-a_2)^2+(b_1-b_2)^2)^{1/2}$. Let $x_1,x_2,\ldots,x_n$ be $n$ points in the Euclidean plane. A *Steiner tree* over $x_1,\ldots,x_n$ is a tree $T$ in the plane whose vertex set includes $\{x_1,\ldots,x_n\}$ as a subset. Vertices $x_1,x_2,\ldots,x_n$ are called the *terminal points* of tree $T$, and the other vertices in $T$ are called the

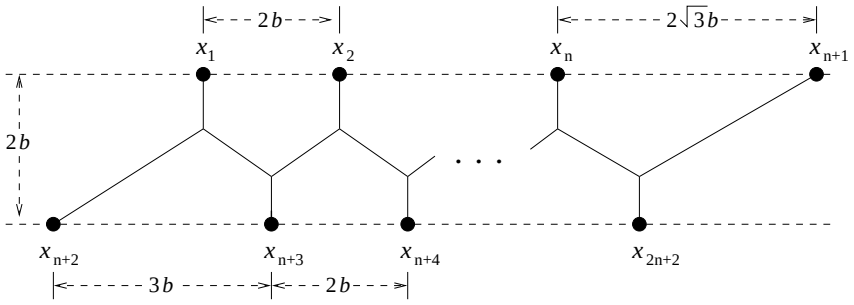


Figure 7.12: A basic Steiner minimum tree.

*Steiner points* of $T$. A *Steiner minimum tree* over $x_1, \ldots, x_n$ is the Steiner tree over $x_1, \ldots, x_n$ with the minimum total edge length. A decision version of the Steiner minimum tree problem can be formulated as follows:

> STEINER MINIMUM TREE (SMT): Given a set of $n$ integer points in the Euclidean plane and a positive integer $k$, determine whether there exists a Steiner tree over terminal points $x_1, \ldots, x_n$ whose total edge length does not exceed $k$.

**Example 7.42** SMT *is NP-complete.*

*Proof (Sketch).* We are going to present a reduction from PARTITION to SMT which depends on certain geometric properties of Steiner trees. Since the proofs of these geometric properties are tedious but not directly related to the idea of the reduction, we will omit these proofs here.

We first consider a special Steiner minimum tree as Shown in Figure 7.12. This is a Steiner minimum tree over $2n+2$ terminal points $x_1, x_2, \ldots, x_{2n+2}$, which lie in two parallel lines of distance $2b$ for some positive integer $b$. We call these two parallel lines the base lines. More precisely, the coordinates of these points are

(1) $x_i = (2ib, 2b)$, for $1 \leq i \leq n$,

(2) $x_{n+1} = ((n + \sqrt{3})2b, 2b)$,

(3) $x_{n+2} = (0, 0)$, and

(4) $x_i = ((2(i - n - 2) + 1)b, 0)$, for $n + 3 \leq i \leq 2n + 2$.

By the minimality of the total length, it can be proved that every angle in this tree is of 120 degrees. Thus, this Steiner minimum tree consists of three families of parallel edges, with the edges passing through $x_1, x_2, \ldots, x_n$ and $x_{n+3}, x_{n+4}, \ldots, x_{2n+2}$ perpendicular to the two base lines.

This Steiner minimum tree has a key property: If we fix $x_{n+1}$ and $x_{n+2}$ and move other points along the two base lines, and also keep the directions

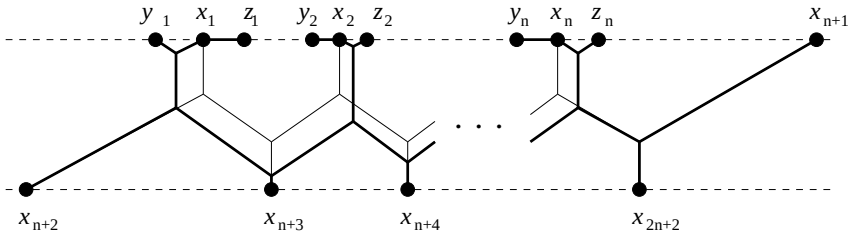


Figure 7.13: The new Steiner minimum tree.

of all edges unchanged, then the total length of the tree does not change. This length is not hard to calculate:

$$\ell = (2 + \sqrt{3})nb + 4b.$$

Furthermore, it can be proved that this is actually the minimum total length of any Steiner tree over $x_1, \ldots, x_n$, even if we allow the edge directions to change.

Now, we use this basic Steiner minimum tree structure to construct a reduction from PARTITION to SMT. Let $a_1, \ldots, a_n$ be $n$ given integers. We choose an integer $b \gg \sum_{i=1}^{n} a_i$ (i.e., $b$ is much greater than the sum of $a_i$'s), and create $2n + 2$ points $x_1, x_2, \ldots, x_{2n+2}$ as above. Next, for each $x_i$, with $1 \leq i \leq n$, on the upper base line, we add two points $y_i$ and $z_i$ on its left and its right, respectively, such that $d(y_i, x_i) = d(x_i, z_i) = a_i$. Over these $4n + 2$ points, we can define a class of Steiner trees, each corresponding to a partition $(I_1, I_2)$ of $\{1, 2, \cdots, n\}$: For each partition $(I_1, I_2)$, let $T(I_1, I_2)$ denote the tree that includes (i) the edges $\{x_i, y_i\}$ for $i \in I_1$, (ii) the edges $\{x_i, z_i\}$ for $i \in I_2$, and (iii) a Steiner minimum tree over the terminal points

$$\{x_i \mid 1 \leq i \leq 2n + 2\} \cup \{y_i \mid i \in I_2\} \cup \{z_i \mid i \in I_1\}.$$

(See Fig. 7.13.)

With $b \gg \sum_{i=1}^{n} a_i$, it can be proved that the shortest Steiner tree in the above class of trees $T(I_1, I_2)$ is actually the Steiner minimum tree over the $4n + 2$ points. In addition, this class of trees have the following properties:

(a) If there is an even partition $(I_1, I_2)$ for $(a_1, \ldots, a_n)$, then $T(I_1, I_2)$ is the shortest one with length $\ell^* = \ell + (1 + \sqrt{3}/2) \sum_{i=1}^{n} a_i$. Actually, in this case, $T(I_1, I_2)$ has edges perpendicular to the two base lines.

(b) If $(I_1, I_2)$ is not an even partition of $(a_1, \ldots, a_n)$, then $T(I_1, I_2)$ has length longer than $\ell^*$.

That is, there exists an even partition of $(a_1, \ldots, a_n)$ if and only if there exists a Steiner tree with length at most $\ell^*$ that interconnects these $4n + 2$ points. However, we note that the mapping from the list $(a_1, \ldots, a_n)$ to $(x_1, \ldots, x_{2n+2}, y_1, \ldots, y_n, z_1, \ldots, z_n; \ell^*)$ is not a reduction from PARTITION to

SMT, because $\ell^*$ and the first coodinate of the point $x_{n+1}$ are not integers. To get a correct reduction, we need to first replace each occurrence of $\sqrt{3}$ in these two numbers by a rational approximation $r$, with $\sqrt{3} < r < \sqrt{3}+\epsilon$. Then, we multiply every coordinate of the $4n+2$ points and the length $\ell^*$ by their common denominator to produce an integer instance of SMT. A careful analysis shows that, for a small enough error $\epsilon$, the new integer instance of SMT has a solution if and only if the instance $(x_1, \ldots, x_{2n+2}, y_1, \ldots, y_n, z_1, \ldots, z_n; \ell^*)$ has a solution. This completes the reduction from PARTITION to SMT.

Finally, we note that we still need to show that SMT is in *NP*. To do so, for every given instance of SMT, we guess an additional set of Steiner points, and verify that the total length of the tree interconnecting these points is bounded by the given length bound. Note that the additional Steiner points and the length of the tree may involve irrational numbers, and so approximations are required. It can be checked that the approximations can be calculated within polynomial time so that the result of calculation remains correct. We omit the detailed analysis. □

## Exercise 7.4

**1.** For each of the following problems, determine whether it is *NP*-complete or is in *P*. If it is *NP*-complete, find a reduction from a known *NP*-complete problem to it. If it is in *P*, find a polynomial-time algorithm for it.

(a) Given a graph $G$, two vertices $s$ and $t$ in $G$ and a positive integer $k$, determine whether there is a path between $s$ and $t$ of length less than or equal to $k$.

(b) Given a graph $G$, two vertices $s$ and $t$ in $G$ and a positive integer $k$, determine whether there is a path between $s$ and $t$ of length greater than or equal to $k$.

⋆ (c) Given a directed graph $G$, determine whether $G$ has a cycle of an odd length.

⋆ (d) Given a directed graph $G$, determine whether $G$ has a cycle of an even length.

⋆ (e) Given a graph $G$, determine whether $G$ has a cycle of an odd length.

(f) Given an edge-weighted complete graph $G$, three subsets $X_1$, $X_2$, $X_3$ of vertices and a positive integer $k$, determine whether there is a subgraph $H$ of $G$ of weight $\leq k$ which contains a spanning tree for each of $X_1, X_2$ and $X_3$. (A *spanning tree* for a subset $X \subseteq V$ is a connected subgraph $H$ of $G$ on the vertex set $X$ that has no cycle.)

(g) Given a graph $G = (V, E)$ and a positive integer $k$, determine whether there is a subset $T \subseteq E$ of at most $k$ edges such that

every vertex in $V$ is incident with at least one edge in $T$.

⋆ (h) REGULAR EXPRESSION NONEQUIVALENCE: Given two regular expressions $r_1$ and $r_2$ without the Kleene-star operation, determine whether $L(r_1) \neq L(r_2)$.

(i) BOUNDED PCP (see Exercise 7(d) of Section 7.1).

(j) BOUNDED TILING (see Exercise 7(e) of Section 7.1).

**2.** Prove that the following variations of the problem VC are *NP*-complete:

(a) PLANAR VC: The problem VC restricted to planar graphs. (A *planar graph* is a graph that can be drawn on the two-dimensional plane such that no two edges cross each other at a non-vertex point.)

(b) CUBIC VC: The problem VC restricted to cubic graphs. (A *cubic graph* is a graph such that every vertex has degree three.)

⋆ (c) PLANAR CONNECTED-VC-4: Given a planar graph $G = (V, E)$ with each vertex in $V$ having degree at most 4 and an integer $k > 0$, determine whether there is a vertex cover $C$ of $G$ of size $k$ such that the induced subgraph $G|_C$ on vertex set $C$ is connected.

⋆ (d) Given a graph $G$, determine whether $G$ has a vertex cover $C$ satisfying the following conditions:

(i) The subgraph $G|_C$ induced by $C$ has no isolated point.

(ii) Every vertex in $C$ is adjacent to a vertex not in $C$.

**3.** The *polar representation* of a CNF $F$ is a bipartite graph $G_F = (V_1, V_2, E)$, where $V_1$ is the set of all variables and $V_2$ is the set of all clauses in $F$, and there is an edge in $E$ between $x_i \in V_1$ and $c_j \in V_2$ if and only if the variable $x_i$ appears in the clause $c_j$ (in the form of $x_i$ or $\overline{x}_i$). Prove the following statements:

(a) $(x+y+\overline{z})(\overline{x}+z+w)(\overline{x}+z+\overline{w})(\overline{y}+z+u)(\overline{y}+z+\overline{u})$ has a planar polar representation and it is satisfiable if and only if $x + y = z$.

(b) There exists a 3-CNF formula $F$ with a planar polar representation and three variables $x$, $y$, and $z$, such that $F$ is satisfiable if and only if $xy = z$.

(c) There exists a 3-CNF formula $F$ with a planar polar representation and three variables $x$, $y$, and $z$, such that $F$ is satisfiable if and only if $x \oplus y = z$. [Hint: $x \oplus y = x(\overline{x} + \overline{y}) + (\overline{x} + \overline{y})y$.]

(d) The following problem, called PLANAR POLAR 3SAT, is *NP*-complete: Given a 3-CNF formula $F$ with a planar polar representation, determine whether $F$ is satisfiable or not. [Hint: Applying the fact that $x \oplus (x \oplus y) = y$ and $(x \oplus y) \oplus y = x$ to the construction.]

**4.** The *nonpolar representation* of a CNF $F$ is a graph $G_F = (V, E)$ with the vertex set $V$ consisting of all literals and all clauses in $F$, and the edge set $E$ consisting of all pairs $\{x, \overline{x}\}$ over all variables $x$ in $F$ plus

all literal-clause pairs $\{z, c\}$ such that $z$ occurs in $c$. Prove the following statements:

(a) If the nonpolar representation of a CNF formula $F$ is planar, then its polar representation must also be planar. However, the converse is not necessarily true.

(b) The following problem, called PLANAR NONPOLAR 3SAT, is *NP*-complete: Given a 3-CNF formula $F$ with a planar nonpolar representation, determine whether $F$ is satisfiable or not.

**5.** A $(k, \ell)$-*CNF* $F$ is a CNF such that each clause contains exactly $k$ literals and each variable occurs in at most $\ell$ clauses. The problem $(k, \ell)$-SAT is the problem 3SAT restricted to $(k, \ell)$-CNF's. Prove the following results:

(a) For any $k > 0$, every $(k, k)$-CNF $F$ is satisfiable.

(b) For any $\ell > 0$, $(2, \ell)$-SAT is polynomial-time solvable. (In fact, the problem 2SAT, which is the problem 3SAT restricted to CNF's with at most 2 literals in each clause, is polynomial-time solvable.)

$\star$ (c) $(3, 4)$-SAT is *NP*-complete.

## 7.5 NP-Complete Optimization Problems

In the last two sections, we have formally defined *NP*-complete problems as a class of *decision problems*. In the applications, we often deal with *search problems*, i.e., problems whose answers are not simple *yes* or *no*. For instance, consider the vertex cover problem. The problem VC defined in Section 7.1 is a decision problem: For a given graph $G$ and a given integer $k > 0$, determine whether there is a vertex cover of $G$ of size at most $k$. A more natural formulation of this problem is the following search problem:

MIN-VC: For a given graph $G$, find a minimum vertex cover of $G$.

In fact, people often refer to MIN-VC as the vertex cover problem, and often say this search problem is *NP*-complete. So, what is the exact relation between these two problems, and is there a formal way to define *NP*-complete search problems? We try to clarify it here.

First, we can easily formulate a search problem as a function from strings to strings. For instance, MIN-VC is just the *multi-valued* function mapping graph $G$ to one of its vertex cover of the minimum size. Then, with an appropriate coding of graphs and integers by binary strings, MIN-VC can be viewed as a multi-valued function from finite strings to finite strings. For a single-valued function $f$, we have defined its deterministic time complexity. In particular, $f$ is said to be polynomial-time computable if there is a polynomial-time DTM $M$ such that on each input $x$, $M$ halts with the output $f(x)$. This definition can be easily extended to multi-valued functions. Namely, a multi-valued function $f$ is *polynomial-time computable* if there is a polynomial-time

DTM $M$ such that on each input $x$, $M$ halts with an output $y$ which is one of the values of $f(x)$.

To formally define the notion of an *NP*-complete multi-valued function, we need to define (a) how a nondeterministic TM computes a function, and (b) what it means by a polynomial-time reduction between functions. Since the formal definition is tedious, we try to avoid it here. Instead, we content ourselves with an informal description and a practical example.

First, we define a new notion of reducibility between two functions which is more general than the many-one reduction between two sets (a decision problem can be considered as a function whose outputs are either 0 or 1). We say a function $f$ is *polynomial-time Turing reducible* to $g$, denoted by $f \leq_T^P g$, if there is a polynomial-time DTM $M$ such that for any $x$, $M$ computes $f(x)$ with the *help* of the function $g$ as an *oracle*. By using $g$ as an oracle, we mean that $M$ has a subprocedure $M_g$ which, when asked for the value $g(z)$, can provide the answer $w = g(z)$ to $M$ *free*, meaning that the computation from $z$ to $g(z)$ does not count toward the time complexity of $M$ (though the time to prepare the query string $z$ is still counted toward its time complexity). Based on this new notion of reducibility, we say that a search problem $f$ is *NP-complete* if the following conditions hold:

(a) A decision version $S_f$ of $f$ is an *NP*-complete set.

(b) This decision version $S_f$ and the search version $f$ are *polynomial-time Turing-reducible* to each other.

**Example 7.43** *Show that* MIN-VC *is an NP-complete search problem.*

*Proof.* The problem VC is the natural decision version for MIN-VC. It is easy to see how to use MIN-VC as an oracle to solve problem VC in polynomial time: For any graph $G$ and integer $k$, we first ask MIN-VC to find a minimum vertex cover $C$ of $G$, and let $k^*$ be the size of $C$. Then, we answer YES to the instance $(G, k)$ if and only if $k^* \leq k$.

Next, we show how to reduce MIN-VC to VC. For any graph $G = (V, E)$, we show a polynomial-time algorithm that finds a minimum vertex cover $C$ of $G$, with the help of the oracle VC:

(1) We ask $|V| + 1$ questions "Is $(G, j) \in$ VC?" for $j = 0, 1, \ldots, |V|$, to the oracle VC, and let $k^* = \min\{j \mid (G, j) \in \text{VC}\}$.

(2) Set $U := V$ and $K := k^*$.

(3) Repeat the following until $K = 0$:

  (3.1) For each $v \in U$, consider the induced subgraph $G|_{U-\{v\}}$ on the vertex set $U - \{v\}$, and ask the oracle whether $(G|_{U-\{v\}}, K-1) \in$ VC.

  (3.2) Select one $v \in U$ with $(G|_{U-\{v\}}, K-1) \in$ VC. Reset $U := U - \{v\}$ and $K := K - 1$.

(4) Output the set $C := V - U$.

To see that this algorithm works correctly, we need to show the following two properties of the algorithm:

(a) At the beginning of each iteration of step (3.1), the induced subgraph $G|_U$ has a minimum vertex cover of size $K$.

(b) In each iteration of step (3), there always exists a vertex $v \in U$ satisfying $(G|_{U-\{v\}}, K-1) \in \mathrm{VC}$.

These properties can be proved easily by induction. It is clear that at the first iteration of step (3.1), $U = V$ and $K = k^*$, and so $G|_U$ is just $G$ itself and its minimum vertex cover is of size $K$. So, property (a) is satisfied at the first iteration.

Now, assume that property (a) is satisfied at the $j$th iteration, for some $1 \leq j \leq k^*$. From property (a), we know that there is a set $V^* \subseteq V$ of size $K = k^* - j + 1$ which is a minimum vertex cover of $G|_U$. Then, for any $v \in V^*$, $V^* - \{v\}$ is a vertex cover of $G|_{U-\{v\}}$. Therefore, property (b) is also satisfied at the $j$th iteration of step (3). Furthermore, we note that, in the $j$th iteration, if we select a particular $v$ satisfying $(G|_{U-\{v\}}, K-1) \in \mathrm{VC}$ and reset $U$ to be $U - \{v\}$ and $K$ to be $K - 1$, then property (a) is satisfied at the $(j+1)$st iteration. This completes the induction proof of properties (a) and (b).

From property (a), it follows that the output $C$ is a vertex cover of $G$, since the induced subgraph $G|_U$, with respect to the final $U$, has a vertex cover of size 0 (i.e., the final $U$ is an independent set of $G$).

Finally, we note that step (3) runs for $k^* \leq |V|$ iterations, and each iteration asks $|V|$ queries to the oracle and so this oracle algorithm works within polynomial time. (Note that the answers to the queries cost us nothing when we consider the Turing reduction.) □

**Proposition 7.44** *If $f \leq_T^P g$ and $g$ is polynomial-time solvable, then $f$ is also polynomial-time solvable.*

*Proof.* If $f \leq_T^P g$, then there is a polynomial-time DTM $M_f$ that computes $f$ with the help of the oracle $g$. Since $M_f$ on any input $x$ of length $n$ runs in time $p(n)$ for some polynomial $p$, both the number of queries that can be asked by $M_f$ and the length of each query $y$ are bounded by $p(n)$ ($M_f$ must take at least $|y|$ moves to write down the string $y$ in order to ask for $g(y)$). Now, suppose that $M_g$ is a DTM computing $g$ in time $q(n)$ for some polynomial $q$. Then, we can compute $f(x)$ without using the oracle as follows:

> On input $x$, we run DTM $M_f$ on $x$ and, when a query for $g(y)$ is made by $M_f$, we simulate $M_g$ on input $y$ to get the answer.

On an input of length $n$, $M_f$ makes at most $p(n)$ queries, each of length at most $p(n)$. So, the total simulation time on $M_g$ taken by the above algorithm to answer these queries is at most $p(n) \cdot q(p(n))$. Thus, the total running time $t(n)$ of the above deterministic algorithm is bounded by a function polynomial in $n$: $t(n) \leq p(n)(q(p(n)) + 1)$. □

From the above proposition, we see that if one of the problems VC or MIN-VC is solvable in polynomial time, then the other problem is also solvable in polynomial time. Therefore, the *NP*-completeness of VC implies that MIN-VC is solvable in polynomial time if and only if all *NP*-complete problems are solvable in polynomial time. This result thus justifies the use of the term *NP*-completeness on search problems such as MIN-VC.

**Approximation to Optimization Problems.** Many *NP*-complete search problems belong to the class of *optimization* problems which asks for the maximum or the minimum solution to a given instance. These includes a number of problems we studied in the last few sections: VC, IS, KNAPSACK, SMT, and BIN PACKING. Note that, in practice, one does not really require the optimum solutions to these problems. A *nearly* optimum solution is sufficient for most applications. Thus, a reasonable approach to attack these problems is to consider their approximation versions. For instance, instead of finding the minimum-size vertex cover of a given graph, we may ask whether it is possible to find a vertex cover $C \subseteq V$ of a graph $G(V, E)$ such that $|C|$ is at most 20% larger than the size of the minimum vertex cover of $G$. In the following, we introduce a general framework to study the computational complexity of this type of approximation problems.

We formally define an *optimization problem* $\Pi$ to consist of three components: $\Pi = (\Sigma, S, v)$, where $\Sigma$ is an alphabet, $S \subseteq \Sigma^* \times \Sigma^*$ is a relation between strings, and $v$ is a function from $S$ to $\mathbf{N}$.[2] Intuitively, each string $x \in \Sigma^*$ represents a problem *instance* of $\Pi$. For each instance $x$, a string $y \in \Sigma^*$ is a *solution* to $x$ if $(x, y) \in S$. For each solution $y$ to instance $x$, there is an associated value $v(x, y) \in \mathbf{N}$. Thus, an optimization problem $\Pi = (\Sigma, S, v)$ is to find, for each $x \in \Sigma^*$, a solution $y$ to $x$ with

$$v(x, y) = \text{opt}\,\{v(x, z) \mid (x, z) \in S\},$$

where opt = max if $\Pi$ is a maximization problem, and opt = min if $\Pi$ is a minimization problem. For any instance $x \in \Sigma^*$ to $\Pi$, we also define $v^*(x)$ to denote the value of the optimum solution to $x$; that is, $v^*(x) = \text{opt}\,\{v(x, z) \mid (x, z) \in S\}$.

As a simple example, the minimization problem MIN-VC can be defined in this framework as follows: MIN-VC $= (\{0, 1\}^*, S_{\text{VC}}, v_{\text{VC}})$, where $S_{\text{VC}}$ consists of all pairs $(x, y) \in (\{0, 1\}^*)^2$ such that (i) $x$ encodes a graph $G = (V, E)$ (under a fixed coding scheme), (ii) $y$ encodes a subset of $V$, and (iii) $y$ is a vertex cover of $x$, and $v_{\text{VC}}(x, y) = |y|$.

An optimization problem $\Pi = (\Sigma, S, v)$ is *polynomially bounded* if it has the following properties:

[2] In general, the value of $v(x, y)$ may be a rational number (or a real number) instead of an integer. In most cases, though, it can be converted to an integer-valued function without changing the characteristic of the problem.

(i) There exists a polynomial function $p$ such that for any $(x, y) \in S$, $|y| \leq p(|x|)$.

(ii) The set $S$ is polynomial-time computable.

(iii) The function $v$ is polynomial-time computable.

It is not hard to verify that the optimization problems we studied in the last few sections are all polynomially bounded.

**Proposition 7.45** *If $P = NP$, then every polynomially bounded optimizaton problem $\Pi = (\Sigma, S, v)$ is solvable in polynomial time.*

*Proof.* First, assume that $\Pi$ is a maximization problem. It has the following decision problem version:

$$\Pi_D = \{(x, q) \in \Sigma^* \times \mathbf{N} \mid (\exists y \in \Sigma^*)\,[(x, y) \in S, v(x, y) \geq q\}.$$

This decision problem is apparently in $NP$. So, by the assumption that $P = NP$, we can solve this decision problem in deterministic polynomial time.

For any instance $x \in \Sigma^*$, we can first use $\Pi_D$ as an oracle to find $v^*(x)$ by a binary search. Note that $v$ is polynomial-time computable, and so $0 \leq v(x, y) \leq 2^{p(|x|)}$ for some polynomial $p$. Therefore, the binary search must find $v^*(x)$ in $p(|x|)$ steps. By Proposition 7.44, there is a deterministic polynomial-time algorithm for function $v^*(x)$.

Next, we consider the decision problem

$$\Pi'_D = \{(x, w) \in (\Sigma^*)^2 \mid (\exists z \in \Sigma^*)\,[(x, wz) \in S, v(x, wz) \geq v^*(x)\}.$$

Again, this problem is obviously in $NP$ and, by the assumption, is in $P$. We can use $\Pi'_D$ as an oracle to find a solution $y$ such that $v(x, y) \geq v^*(x)$:

(1) Let $w = \varepsilon$.

(2) Ask the oracle whether $(x, w0) \in \Pi'_D$. If the answer is YES, then we reset $w := w0$, else we reset $w := w1$.

(3) If $|w| = p(|x|)$, then halt and output $w$; otherwise, go back to step (2).

It is easy to see that the above algorithm finds a solution $y$ to $x$ with value $v(x, y) = v^*(x)$. (The above method of searching for the maximum solution $y$ is a variation of binary search, and is called *prefix search*.) In addition, it works within polynomial time and uses an oracle in $P$. (Note that if $(x, y) \in S$, then $|y| \leq p(|x|)$ for some polynomial function $p$.) It follows from Proposition 7.44 that $\Pi$ is polynomial-time solvable.

The proof for the case of minimization problems $\Pi$ is almost identical. □

Now, we consider the approximation to optimization problems. Let $r$ be a real number satisfying $r > 1$. For a maximization problem $\Pi = (\Sigma, S, v)$, its approximation version, with the approximation ratio $r$, is defined as follows:

> $r$-APPROX-$\Pi$: For a given input $x \in \Sigma^*$, find a solution $y$ such that $(x, y) \in S$, and $v(x, y) \geq v^*(x)/r$.

Similarly, for a minimization problem $\Pi$, its approximation version, with the approximation ratio $r$, is as follows:

> $r$-APPROX-$\Pi$: For a given input $x \in \Sigma^*$, find a solution $y$ such that $(x, y) \in S$, and $v(y) \leq r \cdot v^*(x)$.

The above problems may be further generalized to include the problems whose approximation ratio $r$ is actually a function $r(n)$ depending on the input size $n$, instead of a constant real number. For instance, for a minimization problem $\Pi$, $\sqrt{n}$-APPROX-$\Pi$ denotes the problem of finding a solution $y$ to the input $x$ such that $v(y) \leq \sqrt{n} \cdot v^*(x)$.

The approximation version of an *NP*-complete optimization problem is not necessarily *NP*-complete. In general, we may classify the approximation problems into three different classes according to their computational complexity:

(1) The first class consists of approximation problems that are *NP*-complete for all approximation ratios $r > 1$. Among these problems, we may further divide them into subclasses according to finer approximation ratios $r(n)$. For instance, a problem $r$-APPROX-$\Pi_1$ may be *NP*-complete for all constants $r > 1$, but is polynomial-time solvable for $r(n) = \log n$; while another problem $r$-APPROX-$\Pi_2$ may be *NP*-complete for $r(n) = c \cdot \log n$ for all $c > 0$. Then, we consider problem $\Pi_2$ as harder to approximate than problem $\Pi_1$.

(2) The second class consists of approximation problems that are *NP*-complete for some small approximation ratios, and are polynomial-time solvable for larger approximation ratios. That is, an approximation problem $\Pi$ is in this class if there exists a constant $r_0 > 1$ such that $r$-APPROX-$\Pi$ is *NP*-complete for $r < r_0$ and is polynomial-time solvable for $r > r_0$.

(3) The third class consists of all approximation problems that are polynomial-time solvable for all ratios $r > 1$. That is, a problem $r$-APPROX-$\Pi$ belongs to the third class if, for each $k \geq 1$, there is a polynomial-time algorithm $A_k$ for the problem $(1 + 1/k)$-APPROX-$\Pi$. We call such a sequence $A_k$ of algorithms a *polynomial-time approximation scheme (PTAS)*. Note that the run-time of each algorithm $A_k$ of a PTAS is polynomial in the input size, but not necessarily polynomial in the parameter $k$. If, in addition, there is a single algorithm $A$ that finds approximation solutions within the approximation ratio $r = 1 + 1/k$ in time polynomial in both the input length and $k$, then is is called a *fully polynomial-time approximation scheme* (FPTAS). A FPTAS is the best we can expect for an *NP*-complete optimization problem, short of proving $P = NP$.

To prove approximation problems belonging to the first or the second class, we need to construct *approximation-preserving reductions* between approximation problems. These reductions are usually modifications of the reductions between decision versions of the optimization problems and are often more

complicated than the original reductions. We only present a few of them to demonstrate the general techniques.

We first consider a famous problem TSP (see Exercise 7(b) of Section 7.1).

**Example 7.46** TSP *is NP-complete.*

*Proof.* There is a simple reduction from HC to TSP: For any graph $G = (V, E)$, we map it to a complete graph $G' = (V, E')$ with the costs $c(e) = 1$ if $e \in E$, and $c(e) = 2$ if $e \notin E$, and let $K = |V|$. Then, it is clear that $G$ has a Hamiltonian cycle if and only if $G'$ has a tour of total cost $K$.

The Turing reduction of the optimization version to the decision version is easy. We leave it to the reader as an exercise. □

The optimization version of TSP is to find, for a given graph $G$ with the cost function $c$ on its edges, find a minimum-cost tour of $G$. For each graph $G$ with a cost function $c$ on its edges, let $K^*(G, c)$ denote the minimum cost of a tour of $G$. The approximation version of TSP is as follows.

> $r$-APPROX-TSP: Given a complete graph $G = (V, E)$ with a cost function $c : E \to \mathbf{N}$, find a tour of $G$ with the total cost less than or equal to $r \cdot K^*(G, c)$.

**Example 7.47** *The problem* $r$-APPROX-TSP *is NP-complete for all* $r > 1$.

*Proof.* Let $r$ be a fixed constant $r > 1$. We modify the reduction from HC to TSP as follows: For any graph $G = (V, E)$, we define a complete graph $G' = (V, E')$ with the costs $c(\{u, v\}) = 1$ if $\{u, v\} \in E$, and $c(\{u, v\}) = r|V|$ if $\{u, v\} \notin E$.

Assume that $|V| = n$. If $G$ has a Hamiltonian cycle, then that tour in $G'$ is of cost $n$; otherwise, a minimum-cost tour in $G'$ is of cost $\geq (n - 1) + rn$. Thus, it is easy to see how to solve the question of whether $G \in HC$ using $r$-APPROX-TSP as an oracle:

> For a given graph $G$, construct graph $G'$ and the cost function $c$ as described above. Ask the oracle to find a tour $T$ of $G'$ with the cost at most $r \cdot K^*(G', c)$. Accept graph $G$ (i.e., decide that $G \in$ HC) if and only if the cost of $T$ is $\leq rn$.

Notice that if $G$ has a Hamiltonian cycle, then $K^*(G', c) = n$, and so the cost of the approximation solution $T$ is $\leq rn$. On the other hand, if $G$ does not have a Hamiltonian cycle, then the cost of $T$ is $\geq K^*(G', c) \geq rn + (n - 1)$. Thus, the above is a reduction for HC $\leq_T^P$ $r$-APPROX-TSP. □

Another famous *NP*-compelte optimization problem belonging to the first class is the problem MAX-CLIQUE.

> MAX-CLIQUE: Given a graph $G = (V, E)$, find a complete subgraph of the maximum size.

It is known that the problem $r(n)$-Approx-Clique is *NP*-complete for all functions $r(n) = n^c$, with $0 < c < 1$. In other words, if $P \neq NP$, then there is no polynomial-time algorithm that can find a clique of a given graph, guaranteed to have size $\geq \omega(G)/n^c$ for any $0 < c < 1$, where $\omega(G)$ is the size of the maximum clique of a graph $G$. This is the strongest nonapproximability result among all *NP*-complete optimization problems. The proof of this result requires a novel notion of *probabilistically checkable proofs* and algebraic coding theory, and is beyond our scope.

Now we turn our attention to the approximation problems in the second class. First, we formulate the problem 3Sat into an optimization problem.

> Max-3Sat: Given a 3-CNF formula $F = C_1C_2\cdots C_m$, find a Boolean assignment $t$ on its variables such that $t$ satisfies the maximum number of clauses $C_i$ in $F$.

It is clear that Max-3Sat is *NP*-complete. Indeed, the approximation to Max-3Sat is also *NP*-complete. The proof of the this result is, again, based on the study of probabilistically checkable proofs, and we omit it here.

**Theorem 7.48** *The problem* $r$-Approx-3Sat *is NP-complete for some* $r > 1$.

Note that this approximation problem is obviously solvable in polynomial time for $r = 2$ (see Exercise 5(b) of Section 7.4). Thus, $r$-Approx-3Sat belongs to the second class.

In the last few sections, we have seen a number of polynomial-time reductions from 3Sat to the devision versions of other *NP*-complete optimization problems. Some of these reductions can be modifed to preserve approximability properties.

**Example 7.49** *The problem* $r$-Approx-VC *is NP-complete for some* $r > 1$.

*Proof.* We modify the polynomial-time reduction from 3Sat to VC constructed in Example 7.26 as follows: Assume that $F = C_1C_2\cdots C_m$ is a 3-CNF formula over $n$ variables $x_1, x_2, \ldots, x_n$. We let $l_{jk}$, $k = 1, 2, 3$, to denote the three literals in each clause $C_j$. From this formula $F$, we define a graph $H_F = (V, E)$ as follows:

(1) $V$ has $3m$ vertices $u_{jk}$, for $1 \leq k \leq 3$ and $\leq j \leq m$.

(2) For each $j$, $1 \leq j \leq m$, $u_{j1}, u_{j2}, u_{j3}$ form a triangle.

(3) There is an edge $\{u_{jk}, u_{j'k'}\}$, if $j \neq j'$ and $l_{jk} = \neg\, l_{j'k'}$.

(Recall that in the graph $G_F$ of Example 7.26, we used two vertices $v_i$ and $\overline{v}_i$ to encode the literal $x_i$ and $\overline{x}_i$, and connected each $u_{jk}$ with $v_i$ (or $\overline{v}_i$) if $l_{jk} = x_i$ (or, respectively $\overline{x}_i$). Here, we do not use vertices $v_i$ and $\overline{v}_i$, and connect $u_{jk}$ and $u_{j'k'}$ directly if they encode $x_i$ and $\overline{x}_i$, respectively.)

We first observe two basic relations between the formula $F$ and the graph $H_F$.

*Claim* 1. For any assignment $t$ on $\{x_1, x_2, \ldots, x_n\}$, there is a vertex cover $C$ of $H_F$ with $|C| = 3m - s(t)$, where $s(t)$ is the number of clauses in $F$ that are satisfied by $t$.

*Proof.* We define set $C$ as follows:

(1) In each clause $C_j$ that is satisfied by $t$, pick one literal $l_{jk}$ with $t(l_{jk}) = 1$, and put two vertices $u_{jk'}$, with $k' \in \{1,2,3\} - \{k\}$, in $C$.

(2) If $C_j$ is not satisfied by $t$, then put all three vertices $u_{jk}$, $k \in \{1,2,3\}$, in $C$.

It is clear that $|C| = 3m - s(t)$, and that the edges of the triangles over $\{u_{j1}, u_{j2}, u_{j3}\}$, for all $1 \leq j \leq m$, are covered by vertices in $C$. We note that if $\{u_{jk}, u_{j'k'}\} \in E$ and $j \neq j'$, then $l_{jk} = \neg l_{j'k'}$. Therefore, one of these literals is false under the assignment $t$, and hence the corresponding vertex $u_{jk}$ or $u_{j'k'}$ must be in $C$. Therefore, $C$ is a vertex cover of $H_F$.

*Claim* 2. For any vertex cover $C$ of $H_F$, there is an assignment $t$ of variables $\{x_1, x_2, \ldots, x_n\}$ such that $s(t) \geq 3m - |C|$.

*Proof.* We define the assignment $t$ as follows: $t(x_i) = 1$ if there is a literal $l_{jk} = x_i$ such that $u_{jk} \notin C$; and $t(x_i) = 0$ otherwise. We check that if $|C \cap \{u_{j1}, u_{j2}, u_{j3}\}| < 3$, then $t$ satisfies the clause $C_j$. To see this, assume that $u_{jk} \notin C$ for some $k = 1,2,3$. We check that if $l_{jk} = x_i$ for some $i = 1, \ldots, n$, then we have $t(x_i) = 1$ and, hence, $t$ satisfies $C_j$. On the other hand, if $l_{jk} = \overline{x}_i$ for some $i = 1, \ldots, n$, then it is not possible to have $x_i = l_{j'k'}$ and $u_{j'k'} \notin C$, for that would leave the edge $\{u_{jk}, u_{j'k'}\}$ not covered by $C$. Therefore, we must have $t(x_i) = 0$ and, hence, $t$ satisfies clause $C_j$.

Let $sat^*(F)$ denote the maximum $s(t)$ under any assignment $t$, and $vc^*(H_F)$ the size of the minimum vertex cover of $H_F$. The above Claims 1 and 2 imply that

$$sat^*(F) = 3m - vc^*(H_F).$$

Now, suppose that there is a polynomial-time algorithm that can find a vertex cover $C$ of $H_F$ of size $\leq r \cdot vc^*(H_F)$, with $1 < r < 7/6$. We claim that we can apply this algorithm to find an assignment $t$ of $F$ that satisfies at least $sat^*(F)/s$ clauses, where $s = 1/(7 - 6r)$. To see this, we first define $t$ as in Claim 2 above. Then, we have

$$\begin{aligned} sat^*(F) - s(t) &\leq (3m - vc^*(H_F)) - (3m - |C|) \\ &= |C| - vc^*(H_F) \\ &\leq (r-1) \cdot vc^*(H_F) \leq 3(r-1)m. \end{aligned}$$

Next, by a simple greedy algorithm, we can find an assignment that satisfies at least $m/2$ clauses of $F$; and, hence, we have $sat^*(F) \geq m/2$. Together, we get

$$s(t) \geq sat^*(F) - 3(r-1)m \geq (7 - 6r)sat^*(F) = \frac{sat^*(F)}{s}.$$

The above shows that $s$-Approx-3Sat $\leq_T^P$ $r$-Approx-VC. Now, from Example 7.48, there exists some $s > 1$ such that $s$-Approx-3Sat is $NP$-complete.

Thus, for the corresponding $r = (7s-1)/(6s)$, the problem $r$-APPROX-VC is also $NP$-complete. □

The following subproblem of MAX-3SAT is useful in proving other nonapproximability results.

> MAX-3SAT-$b$: Given a 3-CNF formula $F = C_1C_2\cdots C_m$, with each variable $x$ in $F$ occurring in at most $b$ clauses (in the form of $x$ or $\overline{x}$), find a Boolean assignment $t$ on its variables such that $t$ satisfies the maximum number of clauses $C_i$ in $F$.

**Theorem 7.50** *The problem $r$-APPROX-3SAT-3 is NP-complete for some $r > 1$.*

The proof of the above result is an approximation-preserving reduction from $r$-APPROX-3SAT to $r$-APPROX-3SAT-3. We omit the proof (cf. Exercise 5(c) of Section 7.4).

We let VC-$d$ denote the problem VC restricted to graphs of degree $d$.

**Example 7.51** *For each $d \geq 3$, the problem $r$-APPROX-VC-$d$ is NP-complete for some $r > 1$.*

*Proof.* In the proof of Example 7.49, we observe that if each variable $x_i$ occurs in at most three clauses, then each vertex $u_{jk}$ in $H_F$ has degree at most four, since there are at most two vertices $u_{j'k'}$ whose corresponding literals $l_{j'k'}$ are equal to the negation of $l_{jk}$. Therefore, the reduction in Example 7.49 is actually a reduction from $s$-APPROX-3SAT-3 to $r$-APPROX-VC-4. In the following, we reduce the problem $r$-APPROX-VC-4 to $t$-APPROX-VC-3.

Given a graph $G = (V, E)$ in which every vertex has degree at most four, we construct a graph $G'$ of degree at most three as follows: For each vertex $x$ of degree $d$ in $G$ ($1 \leq d \leq 4$), construct a path $p_x$ of $2d-1$ vertices to replace it as shown in Figure 7.14. Among the $2d-1$ vertices, let $\overline{c}_x$ be the set of $d$ vertices which connect to the old neighbors of $x$ (denoted by the black dots in Figure 7.14), and $c_x$ the set of the other $d-1$ vertices (denoted by the gray dots in Figure 7.14).

Note that this path $p_x$ has a unique minimum vertex cover $c_x$ of size $d-1$. However, this vertex cover only covers edges in path $p_x$ but not the original edges incident on $x$. The set $\overline{c}_x$ is also a vertex cover of $p_x$. It has size $d$, but it covers all edges in $p_x$ plus all edges that are originally incident on $x$.

Assume that $|V| = n$ and $|E| = m$. We observe that if $C$ is a vertex cover of graph $G$, then

$$C' = \Big( \bigcup_{x \in C} \overline{c}_x \Big) \cup \Big( \bigcup_{x \notin C} c_x \Big)$$

is a vertex cover of $G'$. In addition, $|C'| = |C| + 2m - n$ (Note that the set $\bigcup_{x \in V} \overline{c}_x$ has size $2m$.)

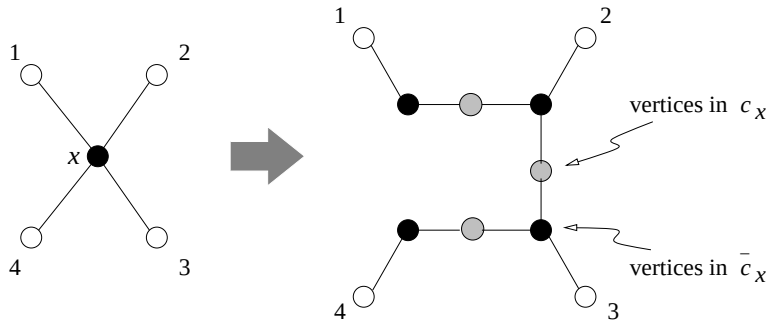


Figure 7.14: Path $p_x$.

Conversely, for each vertex cover $C'$ of $G'$, the set

$$C = \{x : \overline{c}_x \cap C' \neq \emptyset\}$$

is a vertex cover of $G$. Furthermore, $|C| \leq |C'| - 2m + n$.

Together, we have

$$vc^*(G') = vc^*(G) + 2m - n,$$

where $vc^*(G)$ is the size of the minimum vertex cover for $G$.

Now, assume that there is a polynomial-time algorithm $A$ for the problem $t$-Approx-VC-3 for some $t > 1$. Then, this algorithm on graph $G'$ produces a vertex cover $C'$ of size $\leq t \cdot vc^*(G')$. Correspondingly, we have a vertex cover $C$ of $G$ of size

$$\begin{aligned} |C| &\leq |C'| - 2m + n \leq t \cdot vc^*(G') - 2m + n \\ &= t(vc^*(G) + 2m - n) - 2m + n \\ &= t \cdot vc^*(G) + (t-1)(2m - n). \end{aligned}$$

Note that $m \leq 4 \cdot vc^*(G)$. It follows that

$$|C| \leq t \cdot vc^*(G) + 8(t-1)vc^*(G) = (9t-8)vc^*(G).$$

That is, we have an algorithm for the problem $(9t-8)$-Approx-VC-4.

Thus, the above is a $\leq_T^P$-reduction from $r$-Approx-VC-4 to $t$-Approx-VC-3. Since $r$-Approx-VC-4 is *NP*-complete for some $r > 1$, we conclude that $t$-Approx-VC-3 is *NP*-complete for the corresponding $t = (r+8)/9 > 1$. □

Recall the notion of Steiner tress defined in Section 7.4. We consider a variation of the problem SMT.

> Bottleneck Steiner Tree (BST): Given a set $P = \{p_1, p_2, \ldots, p_n\}$ of $n$ points in the Euclidean plane and a positive integer $k$, find a Steiner tree over the terminal points $p_1, p_2, \ldots, p_n$,

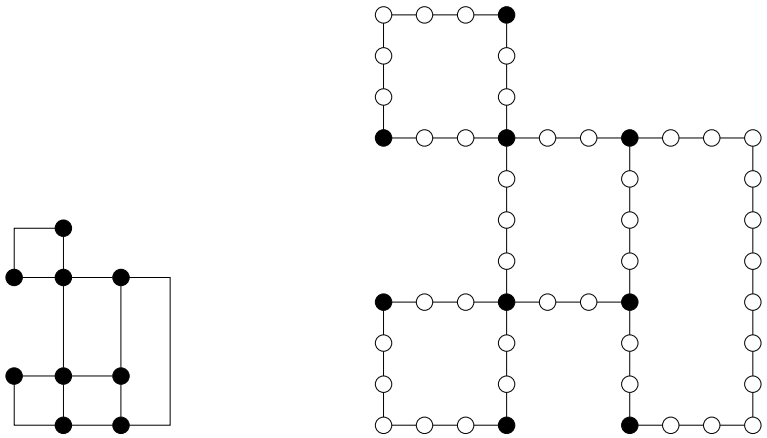


Figure 7.15: (a) A planar graph $G$. (b) The set $P_G$.

with at most $k$ Steiner points, such that the length of the longest edges in the tree is minimized.

⋆ **Example 7.52** *For $1 < r < \sqrt{2}$, $r$-*Approx-BST *is NP-complete.*

*Proof.* Let $r$ be a fixed real number, with $1 < r < \sqrt{2}$. Suppose that there exists a polynomial-time algorithm $A$ for the problem $r$-Approx-BST. We show how to apply algorithm $A$ to solve the *NP*-complete problem Planar Connected-VC-4 (see Exercise 2(c) of Section 7.4).

Let planar graph $G = (V, E)$ and integer $k$ be an given instance of the problem Planar Connected-VC-4. Then, we can embed graph $G$ into the Euclidean plane so that all edges consist of horizontal and vertical segments of length at least $2k + 2$, so that every two edges meet at an angle of 90 degrees or 180 degrees. On any edge $\{u, v\}$ of length $\ell$, we select $\ell' = \lceil \ell \rceil - 1$ interior points $x_1, \ldots, x_{\ell'}$ between $u$ and $v$ such that $|x_1 - u| = |x_{\ell'} - v| = 1$, and $|x_i - x_{i+1}| \leq 1$, for $1 \leq i \leq \ell' - 1$. Note that $\ell > 2k + 2$, and so $\ell' \geq 2k + 2$. We let $P_G$ be the set of these interior points over all edges in $G$. (See Figure 7.15, in which the circles ○ denote the points in $P_G$, and the dark circles ● denote vertices in $G$.)

We claim that $G$ has a connected vertex cover of size $k$ if and only if the algorithm $A$ on input $P_G$ produces a Steiner tree $T$ with at most $k$ Steiner points such that the maximum edge length in $T$ is less than $\sqrt{2}$. From this claim, the problem Planar Connected-VC-4 can be solved as follows: On input $(G, k)$, apply algorithm $A$ to input $P_G$ and accept if the maximum edge length in the resulting tree is less than $\sqrt{2}$. In other words, Planar Connected-VC-4 is $\leq_T^P$-reducible to $r$-Approx-BST.

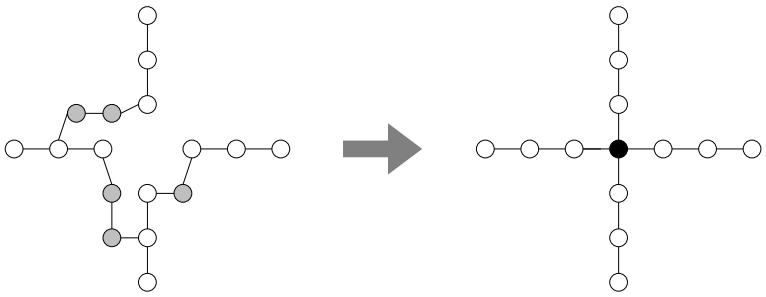

Figure 7.16: A single Steiner point suffices.

It remains to prove the claim. First, assume that $G$ has a connected vertex cover $C$ of size $k$. Then, by choosing the $k$ points in $C$ as the Steiner points, we can construct a Steiner tree $T_1$ on $P_G$ with $k$ Steiner points such that the maximum edge length in $T_1$ is at most 1. This means that the approximation algorithm $A$ on input $P_G$ produces a Steiner tree $T$ with $k$ Steiner points such that the maximum edge length in $T$ is at most $r < \sqrt{2}$.

Conversely, assume that the approximation algorithm $A$ on input $P_G$ produces a Steiner tree $T$ with $k$ Steiner points such that the maximum edge length in $T$ is less than $\sqrt{2}$. We first note that the set $P_G$ of terminal points has the following properties:

(a) Any two terminal points at two different edges of $G$ have a distance at least $\sqrt{2}$.

(b) Any two terminal points at two nonadjacent edges of $G$ have a distance at least $2k + 2$.

From (b), we know that we cannot connect terminal points at two nonadjacent edges by Steiner points only. Therefore, every Steiner point connects only terminal points at two adjacent edges. Furthermore, property (a) implies that two terminal points at two adjacent edges must be connected by at least one Steiner point in $T$. Now, for any two adjacent edges that are connected by some Steiner points, we can move one of these Steiner points to the location of the original vertex point in $G$ that is the end point of these two edges. We connect this new Steiner point to all its original adjacent terminal points, and remove other Steiner points which connect the terminal points in these adjacent edges. (See Figure 7.16, in which all gray Steiner points are replaced by a single black Steiner point.) By this process, we create a Steiner tree $T'$ with at most $k$ Steiner points such that the maximum edge length in $T'$ is at most 1. In addition, all Steiner points in $T'$ are the original vertices of $G$. They form a connected vertex cover $C$ of size at most $k$ for $G$. This completes the proof of our claim. □

Finally, we remark that a number of *NP*-complete optimization problems, including a large class of packing problems and geometric problems, are known

to belong to the third class. We list some of them below without proofs, since the proof techniques involved are mainly those for the design of algorithms and are not directly related to the techniques we are studying in this book.

> KNAPSACK (optimization version): Given $2n+1$ nonnegative integers $c_1, c_2, \ldots, c_n, p_1, p_2, \ldots, p_n$, and $s$, find $x_1, x_2, \ldots, x_n \in \{0,1\}$ that maximize the value $\sum_{i=1}^{n} p_i x_i$, subject to the condition $\sum_{i=1}^{n} c_i x_i \leq s$.

**Example 7.53** *The optimization version of* KNAPSACK *has a fully polynomial-time approximation scheme (FPTAS).*

**Example 7.54** *The following optimization problems have polynomial-time approximation schemes (PTAS's):*

(a) BIN PACKING (optimization version): Given $n+1$ positive integers $a_1, a_2, \cdots, a_n$, and $c$, find a partition of the list $(a_1, a_2, \cdots, a_n)$ into the minimum number of sublists such that the sum of $a_i$'s in each sublist is at most $c$.

(b) STEINER MINIMUM TREE (SMT): Given $n$ points $x_1, x_2, \ldots, x_n$ in the Euclidean plane, find the Steiner minimum tree over these points.

(c) RECTILINEAR SMT: The problem SMT with respect to the rectilinear distance instead of Euclidean distance. (In the two-dimensional plane, the rectilinear distance between two points $x = (a_1, b_1)$ and $y = (a_2, b_2)$ is $d_{\text{rec}}(x, y) = |a_1 - a_2| + |b_1 - b_2|$.)

(d) EUCLIDEAN TSP: Given $n$ points $x_1, x_2, \ldots, x_n$ in the Euclidean plane, find a tour of these points of the minimum total edge length.

(e) PLANAR VC: The problem MIN-VC restricted to planar graphs.

(f) RECTANGULAR PARTITION: Given a rectilinear polygon that contains some rectilinear holes, partition it into hole-free rectangles with the minimum total edge length. (A *rectilinear polygon* is one whose edges are all horizontal or vertical.)

(g) CONVEX PARTITION: Given a convex polygon that contains some convex holes, partition it into hole-free convex polygons with the minimum total edge length. (A polygon is *convex* if the line segments connecting any two interior points of the polygon lie completely inside the polygon.)

(h) $k$-MINIMUM SPANNING TREE: Given $n$ points $x_1, x_2, \ldots, x_n$ in the Euclidean plane, find $k$ points from them to minimize the length of the minimum spanning tree on these $k$ points.

## Exercise 7.5

**1.** For each of the following search problems, formulate an appropriate decision version of it and show that the decision version and the search version are equivalent under polynomial-time Turing reduciblity.

(a) MAX-3SAT.

(b) The optimization version of KNAPSACK.

(c) LP (optimization version): Given a graph $G$, find a longest simple path in $G$.

(d) Given a positive integer $n$, find its largest prime factor.

**2.** For each of the following optimization problems, show that it is *NP*-complete:

(a) Given a directed graph, find a minimum subset of edges such that every directed cycle contains at least one edge in the subset.

(b) Given a directed graph, find a minimum subset of vertices such that every directed cycle contains at least one vertex in the subset.

(c) Given integers $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n, s$ and $t$, find $x_1, x_2, \ldots, x_n \in \{0, 1\}$ that maximize the value $x_1 + x_2 + \cdots + x_n$ subject to the following constraints:

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq s,$$
$$b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \leq t.$$

⋆ (d) Given a graph, find a wheel of the maximum size. (A *wheel* of size $k$ is a subgraph of $k+1$ vertices, in which $k$ vertices form a simple cycle and the other vertex is connected to all these $k$ vertices.)

**3.** Show that for each of the following optimization problems $\Pi$, there exists an approximation ratio $r > 1$ such that $r$-APPROX-$\Pi$ is *NP*-complete.

(a) NETWORK SMT: Given a graph $G = (V, E)$, an edge-weight function $w : E \to \mathbf{N}$ and a subset $P \subseteq V$, find a connected subgraph with the minimum total edge weight that interconnects vertices in $P$.

(b) CONNECTED-VC: Given a graph $G$, find a minimum vertex cover $C$ such that the subgraph $G|_C$ induced by $C$ is connected.

(c) TSP WITH TRIANGLE INEQUALITY: Given a complete graph $G$ and an distance function $d : E \to \mathbf{N}$ that satisfies the triangle inequality, find a Hamiltonian cycle with the minimum total distance. (A distance function $d : E \to \mathbf{N}$ satisfies the *triangle inequality* if $d(\{a, b\}) + d(\{b, c\}) \geq d(\{a, c\})$ for any three vertices $a, b, c$.)

⋆ (d) TSP WITH (1,2)-DISTANCE: Given a complete graph $G$ and an edge-weight function $d : E \to \{1, 2\}$, find a Hamiltonian cycle with the minimum total distance.

⋆ **4.** For a graph $G$, its *edge-square graph* $G^2$ is a graph obtained from $G$ by replacing each edge $\{u, v\}$ by a copy of $G$, called $G_{u,v}$, and connecting both $u$ and $v$ to every vertex in $G_{u,v}$.

(a) Show that $r$-Approx-LP is $NP$-complete for some $r > 1$.

(b) Show that if the longest simple path in $G$ has length $\ell$, then the longest simple path in $G^2$ has length at least $\ell^2$. Moreover, given a path of length $m$ in $G^2$, a path of length $\sqrt{m} - 1$ in $G$ can be found in polynomial-time.

(c) Show that $r$-Approx-LP is $NP$-complete for all $r > 1$.

⋆ **5.** Show that for each of the following optimization problems $\Pi$, there exists a constant $\varepsilon > 0$ such that $n^\varepsilon$-Approx-$\Pi$ is $NP$-complete.

(a) Vertex Coloring: Given a graph $G = (V, E)$, find a coloring of $V$ (i.e., a function $c : V \to \{1, 2, \ldots, m\}$) with the minimum number $m$ of colors such that no two adjacent vertices have the same color.

(b) Edge Coloring: Given a graph $G = (V, E)$, find a coloring of $E$ (i.e., a function $c : E \to \{1, 2, \ldots, m\}$) with the minimum number $m$ of colors such that no two adjacent edges (edges with a common vertex) have the same color.