ate 3-coloring of the interior vertices of each of the crossovers, and the desired result follows. ■

A perusal of the lists contained in the Appendix will provide many other examples of restrictions that have been analyzed for graph theoretic problems. We have attempted throughout these lists to provide as much information as possible about the complexity of various subproblems of each problem. Thus the lists can be used as one source of suggestions for restrictions that might be analyzed for a given problem. Other restrictions will be suggested by the context in which the problem arises. Instances arising in a particular application will often satisfy special constraints that could affect the complexity of the problem, even though these constraints might not be apparent at first. In the next section we discuss a special type of restriction that is often of interest for problems having numerical parameters.

4.2 Number Problems and Strong NP-Completeness

Nowhere does the need for analyzing subproblems of an NP-complete problem have more import than in the case of problems involving numbers. The reasons for this can be illustrated by considering the following "dynamic programming" approach to solving the PARTITION problem.

Let the set $A=\{a_1,a_2,\ldots,a_n\}$ and the sizes $s(a_1)$, $s(a_2),\ldots,s(a_n)$ in Z^+ constitute an arbitrary given instance of PARTITION. Define B to be equal to $\sum_{a\in A} s(a)$. If B is not evenly divisible by 2, then we know that no subset $A'\subseteq A$ can possibly satisfy

$$\sum_{a\in A'} s(a) = \sum_{a\in A-A'} s(a)$$

so we can immediately respond "no" for this instance. Otherwise, for integers $1\leqslant i\leqslant n, 0\leqslant j\leqslant B/2$, let $t(i,j)$ denote the truth value of the statement: "there is a subset of $\{a_1,a_2,\ldots,a_i\}$ for which the sum of the item sizes is exactly j." The values of all the $t(i,j)$ can be viewed as being arranged in a table, as shown in Figure 4.8.

The crux of the approach lies in the very simple procedure that can be used for filling in the table entries. It proceeds row by row, from top to bottom. For the top row, all we need do is observe that $t(1,j)=T$ if and only if either $j=0$ or $j=s(a_1)$. Each subsequent row is filled in by using the entries in the previous row. For $1<i\leqslant n$, $0\leqslant j\leqslant B/2$, the entry $t(i,j)$ in row i has the value T if and only if either $t(i-1,j)=T$ or $s(a_i)\leqslant j$ and $t(i-1,j-s(a_i))=T$. Finally, we observe that, once the entire table has been filled in, we have solved the given instance of PARTITION, because the answer is "yes" if and only if $t(n,B/2)=T$.

The reader should have no difficulty in specifying an iterative algorithm for filling in the table entries, in the manner described, in time bounded by a low order polynomial in the number of table entries (that is, polynomial

i \ j	0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	T	T	F	F	F	F	F	F	F	F	F	F	F	F
2	T	T	F	F	F	F	F	F	F	T	T	F	F	F
3	T	T	F	F	F	T	T	F	F	T	T	F	F	F
4	T	T	F	T	T	T	T	F	T	T	T	F	T	T
5	T	T	F	T	T	T	T	F	T	T	T	T	T	T

Figure 4.8 Table of $t(i,j)$ for the instance of PARTITION for which $A=\{a_1,a_2,a_3,a_4,a_5\}$, $s(a_1)=1$, $s(a_2)=9$, $s(a_3)=5$, $s(a_4)=3$, and $s(a_5)=8$. The answer for this instance is "yes," since $t(5,13)=T$, reflecting the fact that $s(a_1)+s(a_2)+s(a_4)=13=26/2$.

in nB). In fact, at first glance this might even appear to give us a polynomial time algorithm for solving PARTITION, thus proving that P=NP and obviating the need for this book. Of course this is not the case. The reason is that, by the "conciseness" requirement for reasonable encoding schemes, each integer $s(a_i)$ would be described in the input by a string of length only $O(\log s(a_i))$. Therefore the length of the entire PARTITION instance would be only $O(n\log B)$, and nB is *not* bounded by any polynomial function of this quantity. Thus, this is not a polynomial time algorithm for PARTITION.

Nevertheless, in view of this algorithm, it is clear that the NP-completeness of PARTITION (*and* its supposed intractability) depends strongly on the fact that extremely large input numbers are allowed. If any upper bound were imposed in advance on these numbers, even a bound that is polynomial in Length [I], this algorithm would be a polynomial time algorithm for the restricted problem. (In the sequel, we will be defining the term "pseudo-polynomial time algorithm" to refer to algorithms having this property.) One might expect such a bound to be satisfied in many practical applications.

For example, in scheduling problems where the numbers represent task lengths, extremely large numbers would be unlikely to occur because we actually intend to perform those tasks and we could not afford to do so if any one of them required an inordinately large amount of time. In other problems, where numbers represent empirically measured quantities, limits on the precision of measurement have the effect of limiting the range of numbers for which our algorithm must apply.

Furthermore, a pseudo-polynomial time algorithm can be useful even when there is no natural bound on the input numbers we expect. It will display "exponential behavior" only when confronted with instances containing "exponentially large" numbers, and instances of this sort might be

rare for the application we are interested in. If so, this type of algorithm might serve our purposes almost as well as a polynomial time algorithm.

Thus, the possibility of finding a pseudo-polynomial time algorithm for an NP-complete problem involving numbers can be well worth investigating. We shall see that not all such problems are like PARTITION in this regard. For some the theory of NP-completeness can be used to show that even a pseudo-polynomial time algorithm cannot exist unless P = NP. Section 4.2.1 introduces some new terminology and lays the groundwork for proving such "strong" NP-completeness results. Section 4.2.2 illustrates the proof techniques and presents our seventh "basic" NP-complete problem.

4.2.1 Some Additional Definitions

Our new definitions will involve subproblems obtained by placing restrictions on the magnitudes of the numbers occurring in a problem instance. These restrictions will be stated in terms of two encoding-independent functions, Length: $D_\Pi \to Z^+$ and Max: $D_\Pi \to Z^+$, which we assume to be associated with any decision problem Π. Although in theory these two functions can be entirely arbitrary (just like encoding schemes), the significance of what we do with them will depend on the extent to which they reflect the following intended meanings. The function Length, as discussed in Section 2.1, is intended to map any instance I to an integer Length$[I]$ that corresponds to the number of symbols used to describe I under some reasonable encoding scheme for Π. The function Max, which has not been discussed previously, is intended to map any instance I to an integer Max$[I]$ that corresponds to the magnitude of the largest number in I.

The types of results we will be proving will be sufficiently general that each will hold for a broad class of "polynomially related" Length and Max functions. Two Length functions, say Length and Length', for a problem Π are said to be *polynomially related* if there exist polynomials p and p' such that, for all instances $I \in D_\Pi$,

$$\text{Length}[I] \leqslant p'(\text{Length}'[I])$$

and

$$\text{Length}'[I] \leqslant p(\text{Length}[I])$$

We will say that the pair of functions (Length, Max) is *polynomially related* to the pair of functions (Length', Max') if Length and Length' are polynomially related as above and there exist two-variable polynomials q and q' such that, for all $I \in D_\Pi$,

$$\text{Max}[I] \leqslant q'(\text{Max}'[I],\text{Length}'[I])$$

and

$$\text{Max}'[I] \leqslant q(\text{Max}[I],\text{Length}[I])$$

All the results we state will hold for any Length and Max functions that are polynomially related to the ones we are using.

As an example, consider the PARTITION problem, in which an instance I consists of a finite set A and a size $s(a) \in Z^+$ for each $a \in A$. Any of the following would be a suitable Length function for PARTITION:

$$\text{Length}[I] = |A| + \sum_{a \in A} \lceil \log_2 s(a) \rceil$$

$$\text{Length}[I] = |A| + \max\{\lceil \log_2 s(a) \rceil : a \in A\}$$

$$\text{Length}[I] = |A| \cdot \lceil \log_2 \sum_{a \in A} s(a) \rceil$$

Similarly, any of the following would be a suitable Max function for PARTITION:

$$\text{Max}[I] = \max\{s(a) : a \in A\}$$

$$\text{Max}[I] = \sum_{a \in A} s(a)$$

$$\text{Max}[I] = \lceil (\sum_{a \in A} s(a))/|A| \rceil$$

We leave for the reader to verify that any of the nine pairs of Length and Max functions that can be chosen using these two lists is polynomially related to any of the others.

The flexibility we are allowed in choosing Length and Max functions will enable us to avoid explicitly stating the ones we have in mind for a problem Π, since they can be inferred with sufficient accuracy from our description of a generic problem instance. An appropriate Length function is implied by what we consider to be a reasonable encoding scheme for the problem, and the latter follows from our description of the generic instance using the standard conventions set forth at the end of Section 2.1. An appropriate Max function is implied by our specifying that certain objects in the generic instance are numbers (in distinction to sets, sequences, graphs, named elements, etc.). These numbers usually will be integers, and any more complicated "number" in an instance will be viewed as being a composite of one or more separate integers, as has already been done for rational numbers. By convention, we will take Max$[I]$ to be the magnitude of the largest integer occurring in I, or 0 if no integers occur in I.

One final property will be required of the functions Max and Length. This is that, given any reasonable encoding scheme for Π, there must exist polynomial time DTMs that take as input the encoded representation of any instance $I \in D_\Pi$ and that output the values of Length$[I]$ and Max$[I]$, written in binary notation. We need this property solely because we will be

considering restrictions on instances defined in terms of Length[I] and Max[I], and we need to be able to decide whether or not a given string encodes an instance meeting these restrictions. Any natural choices for Length and Max will certainly have this property.

The definitions that follow assume that every decision problem Π has an associated Length function and an associated Max function as discussed above. Formal precision at the language level would also require that an encoding scheme be given for each problem Π. However, it is convenient to state the definitions at the problem level without this proviso, operating under our standard assumptions about the use of reasonable encoding schemes. The reader should have no difficulty in filling in the details needed to make these definitions precise at the language level, and it is more natural and informative to continue our discussions in terms of problems.

An algorithm that solves a problem Π will be called a *pseudo-polynomial time algorithm* for Π if its time complexity function is bounded above by a polynomial function of the two variables Length[I] and Max[I]. By definition, any polynomial time algorithm is also a pseudo-polynomial time algorithm, because it runs in time bounded by a polynomial in Length[I] alone. However, we have already seen an example of a pseudo-polynomial time algorithm that is *not* a polynomial time algorithm, that given for PARTITION. This shows that, even though an NP-completeness result for a problem Π rules out the possibility of solving Π with a polynomial time algorithm (unless P=NP), it does not rule out the possibility of solving Π with a pseudo-polynomial time algorithm.

To be more precise, an NP-completeness result does not *necessarily* rule out the possibility of solving Π with a pseudo-polynomial time algorithm. Many of the decision problems we have considered so far have the property that Max[I] is itself bounded by a polynomial function of Length[I], and for these problems there is no distinction between polynomial time algorithms and pseudo-polynomial time algorithms. For example, the only number that occurs in an instance of CLIQUE is the bound J, and J is constrained to be no larger than the number of vertices in the given graph. SATISFIABILITY involves no numbers at all, except for the subscripts on variables and literals, and these can be ignored because they actually are "names" rather than "numbers." (Our conventions on encoding schemes ensure that such numerical "names" will always be polynomially bounded in terms of Length[I].) The issues we are concerned with here are not relevant for problems like this, so let us give a name to the type of problem for which these issues *are* relevant. We say that a problem Π is a *number problem* if there exists no polynomial p such that $\text{Max}[I] \leq p(\text{Length}[I])$ for all $I \in D_\Pi$. The only number problem among our six basic NP-complete problems is PARTITION.

As an immediate consequence of this definition, we can make the following observation:

Observation 4.1 If Π is NP-complete and Π is not a number problem, then Π cannot be solved by a pseudo-polynomial time algorithm unless P=NP.

Thus, assuming that P$\neq$NP, the only NP-complete problems that are potential candidates for being solved by pseudo-polynomial time algorithms are those that are number problems.

For any decision problem Π and any polynomial p (over the integers), let Π_p denote the subproblem of Π obtained by restricting Π to only those instances I that satisfy $\text{Max}[I] \leq p(\text{Length}[I])$. Then Π_p is not a number problem. Furthermore, if Π is solvable by a pseudo-polynomial time algorithm, then Π_p must be solvable by a polynomial time algorithm. Given any input string x, all we need do is check that x encodes an instance I satisfying $\text{Max}[I] \leq p(\text{Length}[I])$ and, if so, apply the pseudo-polynomial time algorithm for Π to I. By our assumption that Max[I] and Length[I] can be computed in polynomial time, the required inequality can be checked in polynomial time. By the definition of pseudo-polynomial time algorithm, the algorithm for Π will be a polynomial time algorithm for the instances that satisfy this inequality. This motivates us to call a decision problem Π *NP-complete in the strong sense* if Π belongs to NP and there exists a polynomial p over the integers for which Π_p is NP-complete. In particular, if Π is NP-complete and Π is not a number problem, then Π is automatically NP-complete in the strong sense.

We then have the following generalization of Observation 4.1:

Observation 4.2 If Π is NP-complete in the strong sense, then Π cannot be solved by a pseudo-polynomial time algorithm unless P=NP.

This second observation provides the means for applying the theory of NP-completeness to questions about the existence of pseudo-polynomial time algorithms. We know that PARTITION cannot be NP-complete in the strong sense, because it can be solved by a pseudo-polynomial time algorithm. However, we have not yet seen any examples of number problems that *are* NP-complete in the strong sense. This situation will be rectified in the next section, where we illustrate how strong NP-completeness results can be proved.

4.2.2 Proving Strong NP-Completeness Results

The most straightforward way to prove that a number problem Π is NP-complete in the strong sense is simply to prove for some specific polynomial p that Π_p is NP-complete. For example, the TRAVELING SALESMAN problem (TS) defined in Section 2.1 is a number problem because there are no constraints on the values of either the intercity distances $d(i,j)$ or the bound B. We proved TS NP-complete by transforming HAMIL-

TONIAN CIRCUIT to it. Moreover, the instances of TS created by this transformation all have intercity distances equal to 1 or 2 and a bound B equal to the number m of cities. Thus if we take Max$[I]$ to be the larger of B and the longest intercity distance, and we take Length$[I]$ to be $m + \lceil \log_2 B \rceil + \sum_{i,j} \lceil \log_2 d(i,j) \rceil$, then all the instances created by this transformation satisfy the bound

$$\text{Max}[I] \leqslant \text{Length}[I]$$

In other words, this transformation actually shows that the *subproblem* of TS made up of all those instances satisfying the above inequality is itself NP-complete. It follows that TRAVELING SALESMAN is NP-complete in the strong sense.

In contrast, the NP-completeness proofs for KNAPSACK, MULTIPROCESSOR SCHEDULING, and SEQUENCING WITHIN INTERVALS, described in Section 3.2, all leave open the possibility that these problems can be solved by pseudo-polynomial time algorithms. It turns out that KNAPSACK *can* be solved in pseudo-polynomial time, using a dynamic programming approach similar to that we used for PARTITION, as delineated in [Dantzig, 1957]. All pseudo-polynomial time algorithms known to us are based on similar techniques, and we refer the reader to [Horowitz and Sahni, 1976], [Lawler, 1977a], [Lawler and Moore, 1969], and [Sahni, 1976] for illustrations of these techniques.

The problems MULTIPROCESSOR SCHEDULING and SEQUENCING WITHIN INTERVALS, however, do turn out to be NP-complete in the strong sense. In order to show this, it is useful to have a number problem that is NP-complete in the strong sense and that is somewhat "more numeric" than any we have seen so far. Such a problem is provided by our seventh "basic" NP-complete problem, 3-PARTITION, which is defined as follows:

3-PARTITION
INSTANCE: A finite set A of $3m$ elements, a bound $B \in Z^+$, and a "size" $s(a) \in Z^+$ for each $a \in A$, such that each $s(a)$ satisfies $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$.
QUESTION: Can A be partitioned into m disjoint sets $S_1, S_2, \ldots, S_m$ such that, for $1 \leqslant i \leqslant m$, $\sum_{a \in S_i} s(a) = B$? (Notice that the above constraints on the item sizes imply that every such S_i must contain *exactly* three elements from A.)

We prove that 3-PARTITION is NP-complete in the strong sense in two steps, first proving that the related 4-PARTITION problem is NP-complete in the strong sense. 4-PARTITION is identical to 3-PARTITION except that the set A contains $4m$ elements and each $s(a)$ must satisfy

$B/5 < s(a) < B/3$. Thus each set in the desired partition will contain exactly *four* elements.

Theorem 4.3 4-PARTITION is NP-complete in the strong sense.
Proof: It is easy to see that 4-PARTITION belongs to NP, since all we need do is verify in polynomial time that a given partition of A has all the stated properties. We shall transform 3-DIMENSIONAL MATCHING to a restricted version of 4-PARTITION in which all the element sizes are bounded by a polynomial function of the total number of elements, and hence by a polynomial function of Length$[I]$. In particular, taking Max$[I]$ to be $\max\{s(a): a \in A\}$ we shall show that 4-PARTITION is NP-complete even when restricted to instances I with $\text{Max}[I] \leqslant 2^{16} \cdot |A|^4$.

Let $W = \{w_1, w_2, \ldots, w_q\}$, $X = \{x_1, x_2, \ldots, x_q\}$, $Y = \{y_1, y_2, \ldots, y_q\}$, and $M \subseteq W \times X \times Y$ denote an arbitrary instance of 3DM. We may assume without loss of generality that $|M| \geqslant q$. Our corresponding instance of 4-PARTITION has $|A| = 4|M|$ elements, one for each occurrence of a member of $W \cup X \cup Y$ in a triple in M and one for each triple in M.

The elements corresponding to a particular $z \in W \cup X \cup Y$ will be denoted by $z[1], z[2], \ldots, z[N(z)]$, where $N(z)$ denotes the number of triples from M in which z occurs. We shall regard $z[1]$ as being the "actual" element corresponding to z, and $z[2]$ through $z[N(z)]$ as being the "dummy" elements corresponding to z. The sizes of these elements depend on which one of W, X, or Y contains z and on the index of z within that set. These are defined as follows, where r is chosen equal to $32q$:

$$\begin{aligned}
s(w_i[1]) &= 10r^4 + ir + 1 && 1 \leqslant i \leqslant q \\
s(w_i[l]) &= 11r^4 + ir + 1 && 1 \leqslant i \leqslant q,\ 2 \leqslant l \leqslant N(w_i) \\
s(x_j[1]) &= 10r^4 + jr^2 + 2 && 1 \leqslant j \leqslant q \\
s(x_j[l]) &= 11r^4 + jr^2 + 2 && 1 \leqslant j \leqslant q,\ 2 \leqslant l \leqslant N(x_j) \\
s(y_k[1]) &= 10r^4 + kr^3 + 4 && 1 \leqslant k \leqslant q \\
s(y_k[l]) &= 8r^4 + kr^3 + 4 && 1 \leqslant k \leqslant q,\ 2 \leqslant l \leqslant N(y_k)
\end{aligned}$$

The single element corresponding to a particular triple $m_l = (w_i, x_j, y_k) \in M$ is denoted by u_l, and its size depends on the indices of its members as follows:

$$s(u_l) = 10r^4 - kr^3 - jr^2 - ir + 8$$

Notice that, if we add to $s(u_l)$ the sizes of three elements that correspond to w_i, x_j, and y_k, respectively, then the total will be equal to $40r^4 + 15$ whenever all three are "actual" elements or whenever all three are "dummy" elements. We choose this number to be our bound B, that is,

$$B = 40r^4+15\cdot|A|^3+15$$

The reader should have no difficulty verifying that this is a polynomial transformation, that the size of each element is strictly between $B/3$ and $B/5$, and that the sum of all the element sizes is $|M|\cdot B$, as required. Furthermore, we observe that the size of each element is bounded above by $12r^4 \leqslant 12\cdot 8^4\cdot|A|^4 < 2^{16}\cdot|A|^4$. Thus all that remains to be done to prove that 4-PARTITION is NP-complete in the strong sense is to show that the desired 4-partition exists if and only if M contains a matching.

First, suppose that $M' \subseteq M$ is a matching. The corresponding 4-partition is made up of $|M|$ 4-sets, each containing a u_l, a $w_i[\cdot]$, an $x_j[\cdot]$, and a $y_k[\cdot]$, where $(w_i, x_j, y_k) = m_l \in M$. If $m_l \in M'$, we group u_l with $w_i[1]$, $x_j[1]$, and $y_k[1]$. If $m_l \in M-M'$, we group u_l with "dummy" elements corresponding to w_i, x_j, and y_k. It is not hard to see that there are enough dummy elements so that this can be done, and by our previous comments the sizes of the four elements in each set will sum exactly to B. Thus we have our required 4-partition.

Now suppose we are given a 4-partition of the required form. Consider any 4-set in this 4-partition. By successively considering the sum of the element sizes modulo r, r^2, r^3, r^4, and r^5, we shall show that this 4-set must contain one element corresponding to each of the three members of that triple, all three being "actual" elements or all three being "dummy" elements. First, since $r > 4\cdot 8 = 32$, we know that the sum modulo R of the sizes of the four elements, which must equal $B \pmod r = 15$, is the same as the sum of the item sizes when each is taken modulo r beforehand. The only way these can sum to 15 is for the 4-set to contain one element that corresponds to a member of W, one that corresponds to a member of X, one that corresponds to a member of Y, and one that corresponds to a triple from M. Let w_i, x_j, and y_k denote the corresponding members of W, X, and Y, and let $m_l = (w_{i'}, x_{j'}, y_{k'})$ denote the corresponding triple from M. Then the sum of the element sizes modulo r^2 must equal $((i-i')r+15) \pmod{r^2}$ and, since $(i-i')r+15 < r^2$, we must have

$$B \pmod{r^2} = 15 = (i-i')r+15$$

It follows from this that $i = i'$. Similarly, since $(j-j')r^2+15 < r^3$, we must have

$$B \pmod{r^3} = 15 = (j-j')r^2+15$$

and hence $j = j'$, and , since $(k-k')r^3+15 < r^4$, we must have

$$B \pmod{r^4} = 15 = (k-k')r^3+15$$

and hence $k = k'$. Thus w_i, x_j, and y_k are indeed the three members of the triple m_l, and we know that the coefficient of r^4 in the sum of the element sizes is simply the sum of the individual coefficients for r^4. Our choice of these coefficients in the construction then guarantees that the only way for

them to sum to 40 is for all three elements to be "actual" elements or for all three to be "dummy" elements.

The total collection of $3q$ "actual" elements, one for each member of $W \cup X \cup Y$, must therefore be contained in q of our given 4-sets, each of these 4-sets consisting of one element corresponding to a triple from M and the three "actual" elements corresponding to the members of that triple. Those q triples from M provide the desired matching. ■

Theorem 4.4 3-PARTITION is NP-complete in the strong sense.

Proof: It is easy to see that 3-PARTITION belongs to NP. We shall transform the subproblem of 4-PARTITION in which all instances satisfy $\max\{s(a): a \in A\} \leqslant 2^{16}\cdot|A|^4$ to 3-PARTITION, maintaining the property that all element sizes are bounded by a polynomial function of the total number of elements.

Let $A = \{a_1, a_2, \ldots, a_{4n}\}$, bound B, and item sizes $s(a)$ satisfying $B/5 < s(a) < B/3$ and $s(a) \leqslant 2^{16}\cdot|A|^4$ be a specification of any such instance of 4-PARTITION. Our corresponding instance of 3-PARTITION will have $24n^2-3n$ elements, one for each element from A, two for each *pair* of elements from A, and $8n^2-3n$ "filler" elements.

Corresponding to each element $a_i \in A$ is a "regular" element w_i, with size defined by

$$s'(w_i) = 4\cdot(5B+s(a_i))+1$$

where we use $s'(\cdot)$ to denote the size function in our 3-PARTITION instance. Corresponding to each *pair* of elements $a_i, a_j \in A$ we have two "pairing" elements, $u[i,j]$ and $\bar{u}[i,j]$, with sizes defined by

$$s'(u[i,j]) = 4\cdot(6B-s(a_i)-s(a_j))+2$$

$$s'(\bar{u}[i,j]) = 4\cdot(5B+s(a_i)+s(a_j))+2$$

Finally, for $1 \leqslant k \leqslant 8n^2-3n$, we have a "filler" element u_k^* with size $s'(u_k^*) = 20B$. The bound B' for our 3-PARTITION instance is $64B+4$.

Once again the reader should encounter no difficulty in verifying that this is a polynomial transformation, that the size of every element is strictly between $B'/4 = 16B+1$ and $B'/2 = 32B+2$, and that the sum of all the element sizes is equal to $(8n^2-n)B'$. Furthermore, since the elements in A are constrained to have sizes no larger than $2^{16}\cdot|A|^4$, the sizes in the 3-PARTITION instance will also satisfy a polynomial bound in terms of $|A|$, hence in terms of the number of elements in the constructed instance I', hence in Length $[I]$. Thus, to complete our demonstration that 3-PARTITION is NP-complete in the strong sense, we need only show that a 3-partition exists for the constructed instance if and only if a 4-partition exists for the original instance.

First, suppose that we have a 4-partition for the original instance. The corresponding 3-partition is constructed as follows: Arbitrarily divide each 4-set $\{a_i,a_j,a_k,a_l\}$ into two 2-sets, say $\{a_i,a_j\}$ and $\{a_k,a_l\}$. Our 3-partition will then contain the two 3-sets $\{w_i,w_j,u[i,j]\}$ and $\{w_k,w_l,\bar{u}[i,j]\}$. (Notice that we could just as well have used $\bar{u}[k,l]$ instead of $u[i,j]$ and $u[k,l]$ instead of $\bar{u}[i,j]$.) The sizes of the elements in each of these 3-sets sums to B' since $s(a_i)+s(a_j)+s(a_k)+s(a_l)=B$. Doing this for each of the n given 4-sets, we obtain $2n$ 3-sets that contain all of the "regular" elements and n matched pairs of "pairing" elements. This leaves $8n^2-3n$ matched pairs of "pairing" elements and $8n^2-3n$ "filler" elements. Since the sum of the sizes of two matched "pairing" elements is $44B+4=B'-20B$, each such matched pair can be grouped with a remaining one of the "filler" elements to complete the desired 3-partition.

Now suppose that we are given a 3-partition for the constructed instance. By considering the element sizes modulo 4 we see that no 3-set can contain an odd number of "regular" elements, no 3-set can contain three "pairing" elements, and no 3-set can contain two "regular" elements and a "filler" element. It follows that the given 3-partition is made up of $2n$ 3-sets that each contain two "regular" elements and one "pairing" element, along with $8n^2-3n$ 3-sets that each contain two "pairing" elements and one "filler" element. Consider any one of the latter type of 3-sets, and let $u[i,j]$ (or $\bar{u}[k,l]$) be one of the two "pairing" elements in that set. If the other pairing element in this 3-set is not $\bar{u}[i,j]$ (or $u[k,l]$), then it must have the same size as that matching element and so can be interchanged with it to obtain an equivalent 3-partition. This operation can be repeated until we obtain a 3-partition in which *every* "filler" element occurs together with a matched pair $u[i,j]$, $\bar{u}[i,j]$. Thus, any "pairing" element that occurs with two "regular" elements in this 3-partition is such that its "match" also occurs in such a 3-set. This divides the 3-sets containing "regular" elements into n pairs of 3-sets. Since the two "pairing" elements in each such pair of 3-sets are matched, their sizes sum to $44B+4$, and hence the sizes of the four "regular" elements must sum to $84B+4$. This implies that the corresponding four elements from A form a 4-set of elements whose sizes sum to B. Therefore these n pairs of 3-sets provide the required 4-partition. ■

Notice that this last transformation, if viewed as a transformation from the general 4-PARTITION problem to 3-PARTITION, would not be enough to prove strong NP-completeness for 3-PARTITION. We needed to restrict our attention to an NP-complete subproblem of 4-PARTITION in which $\max\{s(a)\}$ was polynomially bounded. However, it is easy to see that the particular polynomial bound that we chose was not essential. Indeed, it would be convenient if we could operate with transformations like this without needing to go into the details of the subproblems and the particular polynomials involved. This can be done using the following definition and lemma.

Let Π and Π' denote arbitrary decision problems with instance sets D_Π and $D_{\Pi'}$, "yes" sets Y_Π and $Y_{\Pi'}$, and specified functions Max, Length, Max', and Length', respectively. A *pseudo-polynomial transformation* from Π to Π' is a function $f: D_\Pi \to D_{\Pi'}$ such that

(a) for all $I \in D_\Pi$, $I \in Y_\Pi$ if and only if $f(I) \in Y_{\Pi'}$,

(b) f can be computed in time polynomial in the two variables $\text{Max}[I]$ and $\text{Length}[I]$,

(c) there exists a polynomial q_1 such that, for all $I \in D_\Pi$,

$$q_1(\text{Length}'[f(I)]) \geqslant \text{Length}[I]$$

(d) there exists a two-variable polynomial q_2 such that, for all $I \in D_\Pi$,

$$\text{Max}'[f(I)] \leqslant q_2(\text{Max}[I],\text{Length}[I])$$

Lemma 4.1 If Π is NP-complete in the strong sense, $\Pi' \in \text{NP}$, and there exists a pseudo-polynomial transformation from Π to Π', then Π' is NP-complete in the strong sense.

Proof: Let f be such a pseudo-polynomial transformation, with functions q_1 and q_2 as specified in the definition. We can assume without loss of generality that q_1 and q_2 have only positive integer coefficients, since they can be so modified without decreasing their values. Because Π is NP-complete in the strong sense, there is some polynomial p such that Π_p is NP-complete. Furthermore, we can choose such a p that has only positive integer coefficients, because if p_0 is any polynomial over the integers satisfying $p_0(x) \geqslant p(x)$ for all x, then Π_{p_0} will contain all the instances of Π_p and hence must be NP-complete if Π_p is. Let $\hat{p}$ be the polynomial defined by

$$\hat{p}(x) = q_2(p(q_1(x)),q_1(x))$$

We claim that the function f, when restricted to instances of Π_p, becomes a polynomial transformation from Π_p to $\Pi'_{\hat{p}}$, thus proving that $\Pi'_{\hat{p}}$ is NP-complete. First let us see that every instance I of Π_p is mapped by f to an instance of $\Pi'_{\hat{p}}$. Using the definition of Π_p and the inequalities satisfied by q_1 and q_2, we have, for each instance I of Π_p,

$$\begin{aligned}\text{Max}'[f(I)] &\leqslant q_2(\text{Max}[I],\text{Length}[I]) \\ &\leqslant q_2(p(\text{Length}[I]),\text{Length}[I]) \\ &\leqslant q_2(p(q_1(\text{Length}'[f(I)])),q_1(\text{Length}'[f(I)])) \\ &= \hat{p}(\text{Length}'[f(I)])\end{aligned}$$

Thus $f(I)$ is an instance of $\Pi'_{\hat{p}}$. Conditions (a) and (b) of the definition of pseudo-polynomial transformation, along with the fact that every instance I of Π_p satisfies $\text{Max}[I] \leqslant p(\text{Length}[I])$, then imply immediately that f meets the remaining requirements to be a polynomial transformation.

Hence $\Pi'_{\hat{p}}$ is NP-complete, and it follows that Π' is NP-complete in the strong sense. ■

This lemma frees us from having to deal with particular subproblems Π_p when proving strong NP-completeness results, a great convenience since we are rarely interested in identifying the specific polynomial involved. However, the complicated definition of pseudo-polynomial transformation might appear to be a rather formidable obstacle to using this approach. In fact, it is not as complicated as it seems. Condition (a) is identical to one of the two requirements that must be met by an ordinary polynomial transformation, and condition (b) is almost identical to the other but allows us a bit more freedom in the complexity of our transformation. Condition (c) will be met by all but the most unusual transformations, since it requires only that the transformation not cause a substantial *decrease* in input length. The heart of the definition lies in condition (d), and it serves the purpose of ensuring that the magnitude of the largest number in the constructed instance does not blow up exponentially in terms of the Max and Length of the given instance.

As a first example, the construction we used to prove Theorem 4.4 can be viewed as a pseudo-polynomial transformation from the general 4-PARTITION problem to 3-PARTITION. The 3-PARTITION problem itself earns its title as our seventh "basic NP-complete problem" because of the ease with which pseudo-polynomial transformations can be constructed from it. For instance, we can use such a transformation to show that the SEQUENCING WITHIN INTERVALS problem, proved NP-complete in Section 3.2.2, is actually NP-complete in the strong sense.

Theorem 4.5 SEQUENCING WITHIN INTERVALS is NP-complete in the strong sense.

Proof: Recall that in this problem we are given a set T of tasks, each task $t \in T$ having a length $l(t) \in Z^+$ and a time interval $[r(t), d(t)]$ within which it is to be executed, and we are asked whether the tasks can be sequenced to obey these constraints, with at most one task ever being executed at a time. In Section 3.2.2 we proved it to be NP-complete, and hence we already know that it belongs to NP. We shall give a pseudo-polynomial transformation from 3-PARTITION to SEQUENCING WITHIN INTERVALS.

Let $A = \{a_1, a_2, \ldots, a_{3m}\}$, $B \in Z^+$, and $s(a_1), s(a_2), \ldots, s(a_{3m})$ constitute an arbitrary instance of 3-PARTITION. The corresponding instance of SEQUENCING WITHIN INTERVALS is given by

$$T = A \cup \{t_i : 1 \leqslant i < m\}$$

$$l(t) = \begin{cases} 1 & \text{if } t = t_i, 1 \leqslant i < m \\ s(a_j) & \text{if } t = a_j \in A \end{cases}$$

$$r(t) = \begin{cases} iB + i - 1 & \text{if } t = t_i, 1 \leqslant i < m \\ 0 & \text{if } t = a_j \in A \end{cases}$$

$$d(t) = \begin{cases} iB + i & \text{if } t = t_i, 1 \leqslant i < m \\ mB + m - 1 & \text{if } t = a_j \in A \end{cases}$$

This transformation clearly can be performed in time polynomial in the input length alone, and the length of the constructed instance is polynomially related to the length of the given instance, so conditions (b) and (c) of the definition of pseudo-polynomial transformation are met. Furthermore, the largest number in the constructed instance is $mB + m - 1$, so condition (d) is met. All that remains to be shown is that condition (a) is met, just as in our usual NP-completeness proofs.

Any sequence that satisfies the specified constraints must execute each task t_i, $1 \leqslant i < m$, from time $iB + i - 1$ to time $iB + i$, as shown in Figure 4.9. This leaves m separate blocks of time, each of length exactly B, and since this is just enough time in total to accommodate all the tasks $t \in A$, each block must be completely filled. These blocks therefore play the same role as the sets $S_1, S_2, \ldots, S_m$ in the desired partition of A. It follows that the desired sequence exists if and only if the desired partition exists for the given 3-PARTITION instance.

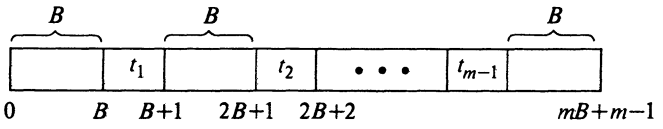

Figure 4.9 The form required of a sequence meeting the constraints of an instance of SEQUENCING WITHIN INTERVALS obtained by transforming an instance of 3-PARTITION in the proof of Theorem 4.5.

Thus condition (a) is met, and we indeed have given a pseudo-polynomial transformation from 3-PARTITION to SEQUENCING WITHIN INTERVALS. By Lemma 4.1, this proves that the latter problem is NP-complete in the strong sense. ■

We suggest as an exercise that the reader try to construct a similar transformation from 3-PARTITION to the MULTIPROCESSOR SCHEDULING problem defined in Section 3.2.1. Our lists of NP-complete problems contain a number of other problems that are proved NP-complete in the strong sense with comparable ease, merely by slightly modifying earlier proofs that used PARTITION to use 3-PARTITION instead. The straightforward nature of these modifications is indicative of the usefulness of 3-PARTITION.

We conclude this section with an example of how a pseudo-polynomial transformation from 3-PARTITION can be useful for proving an ordinary NP-completeness result for a problem that is *not* a number problem. In fact, this problem will involve no numbers at all!

Recall the SUBGRAPH ISOMORPHISM problem defined in Section 2.1: Given two graphs G and H, is H isomorphic to a subgraph of G? We proved this problem NP-complete in Section 3.2.1 simply by noting that it contains CLIQUE as a special case. However, there is one important subproblem of SUBGRAPH ISOMORPHISM that is known to belong to P. This is the problem SUBTREE ISOMORPHISM in which both G and H are required to be *trees* (a tree is a connected graph that contains no cycles). A polynomial time algorithm for this subproblem has been obtained by Edmonds and Matula [1976] (see also [Reyner, 1977]).

Our philosophy of trying to narrow in on the "boundary" between easy and hard subproblems of an NP-complete problem then suggests the following question: What if only *one* of G and H is required to be a tree? In one case the answer is immediate. The version in which only H is required to be a tree contains HAMILTONIAN PATH as a subproblem and hence is NP-complete. The case in which only G is required to be a tree is more interesting. We know that H cannot be a subgraph of such a G unless it is acyclic (contains no cycles), but this does not imply that H must be a tree, since it might be disconnected. In general, an acyclic graph is called a *forest*, with only connected forests being trees (see Figure 4.10).

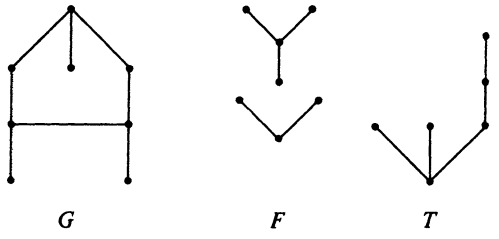

Figure 4.10 Examples of a graph G, forest F, and tree T. G is a graph but not a forest, and F is a forest but not a tree. F is *not* a subforest of T, but each tree in F is a subtree of T.

Let us give the name SUBFOREST ISOMORPHISM to the subproblem of SUBGRAPH ISOMORPHISM in which G is required to be a tree and H is required to be a forest. Despite the similarity of this problem to the polynomially solvable SUBTREE ISOMORPHISM problem, we have the following theorem:

Theorem 4.6 SUBFOREST ISOMORPHISM is NP-complete.

Proof: Membership in NP follows from that for SUBGRAPH ISOMORPHISM. We shall give a pseudo-polynomial transformation from 3-PARTITION to SUBFOREST ISOMORPHISM, and the result will follow by Lemma 4.1.

Let $A = \{a_1, a_2, \ldots, a_m\}$, $B \in Z^+$, and $s(a_1), s(a_2), \ldots, s(a_m)$ in Z^+ constitute an arbitrary instance of 3-PARTITION. The corresponding instance of SUBFOREST ISOMORPHISM is illustrated in Figure 4.11.

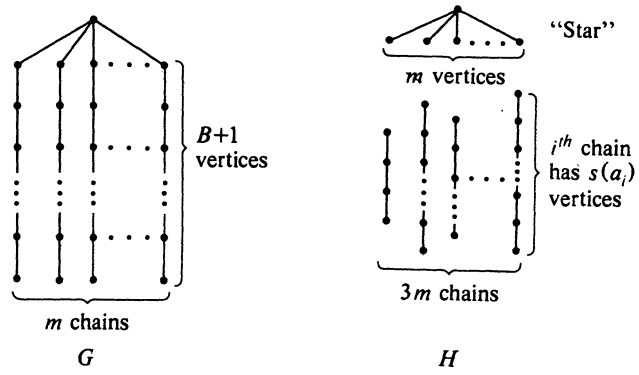

Figure 4.11 The tree G and forest H corresponding to an instance of 3-PARTITION in the proof of Theorem 4.6.

The tree G consists of m chains of $B+1$ vertices each, all attached at one end to an additional common vertex. The forest H consists of $3m+1$ trees, including one "star" on $m+1$ vertices and $3m$ chains, each corresponding to a particular element $a \in A$ and having $s(a)$ vertices.

Any isomorphism from H to a subgraph of G must map the center of the star to the single high-degree vertex of G. The m neighbors of the center of the star in H then must be mapped to the m neighbors of that vertex in G. This leaves m chains, each of B vertices, in G to which the remaining $3m$ chains in H must be mapped by the isomorphism. The mapping of these chains from H to the remainder of G corresponds to a partition of the elements of A into m sets and, by our construction, can be completed if and only if the elements in each set have sizes summing exactly to B. Thus the required isomorphism from H to a subgraph of G will exist if and only if the required 3-partition of A exists.

This confirms condition (a) of a pseudo-polynomial transformation. It is easy to see that this transformation can be performed in time polynomial in m and B, so condition (b) is satisfied. The total number of vertices in G

and H is $2(mB+1)$, so condition (c) is satisfied. Finally, there are no numbers in the constructed instance, so condition (d) holds. Thus by Lemma 4.1, SUBFOREST ISOMORPHISM is NP-complete in the strong sense, which implies that it is NP-complete in the ordinary sense as well. ■

4.3 Time Complexity as a Function of Natural Parameters

So far in this chapter we have motivated the study of subproblems mainly on the basis of the fact that in practice it is often the subproblem, rather than the general problem, that we are called upon to solve. Having mapped the boundary between the NP-complete subproblems and the polynomial time solvable subproblems, one is better prepared to focus the search for algorithms in potentially profitable directions when such a subproblem arises.

Results concerning subproblems also can be used to help guide the search for algorithms that solve the general problem. If the general problem is NP-complete, we know that an exponential time algorithm will be required (unless P=NP), but there are a variety of ways in which the time complexity of an algorithm can be "exponential," some of which might be preferable to others. This is especially evident when, as is customary in practice, we consider time complexity expressed in terms of natural problem parameters instead of the artificially constructed "input length."

For example, consider the MULTIPROCESSOR SCHEDULING problem of Section 3.2.1. Here a collection of natural parameters might consist of the number n of tasks, the number m of processors, and the length L of the longest task. The ordinary NP-completeness result for this problem proved in Section 3.2.1 implies that, unless P=NP, MULTIPROCESSOR SCHEDULING cannot be solved in time polynomial in the three parameters n, m, and $\log L$. However, one can still ask whether it is possible to have an algorithm with time complexity polynomial in m^n and $\log L$, or polynomial in n^m and $\log L$, or polynomial in n, m, and L, or even polynomial in $(nL)^m$.

Our complexity results for subproblems shed some light on these questions. The original NP-completeness result for MULTIPROCESSOR SCHEDULING actually shows that the subproblem in which m is restricted to the value 2 is NP-complete, thus ruling out an algorithm polynomial in n^m and $\log L$ (unless P=NP), since such an algorithm would be a polynomial time algorithm for this subproblem. Our subproblem results do not rule out an algorithm polynomial in m^n and $\log L$, and indeed exhaustive search algorithms having such a time complexity can be designed. Analogously, the strong NP-completeness result for MULTIPROCESSOR SCHEDULING claimed in Section 4.2.2 rules out an algorithm polynomial in n, m, and L (unless P=NP). It leaves open the possibility of an algorithm polynomial in $(nL)^m$ (which would give a pseudo-polynomial time algorithm for each *fixed* value of m), and again such an algorithm can be shown to exist.

Thus by considering the subproblems obtained by placing restrictions on one or more of the natural problem parameters, we obtain useful information about what types of algorithms are possible for the general problem. Care must be taken to ensure that the parameters we choose are sufficiently representative of instance size that Length $[I]$ can be expressed as a polynomial function of them (so that the class of polynomial time algorithms for the problem is identical to the class of algorithms polynomial in the selected parameters), but otherwise we may choose whatever parameters seem most natural and relevant. A general NP-completeness result then will imply that the problem cannot be solved in time polynomial in all the chosen parameters, and information obtained by restricting these parameters can be meaningful with regard to other types of general algorithms.

Although questions concerning strong NP-completeness and pseudo-polynomial time algorithms are especially relevant here, analyses of this type also can be applied fruitfully to problems that are not number problems, since all problems have natural numerical parameters like sizes of sets, values of bounds, etc. Thus, for instance, the NP-completeness of 3-SATISFIABILITY rules out the possibility (unless P=NP) of an algorithm for SATISFIABILITY that runs in time polynomial in $(mn)^M$, where m is the number of clauses, n is the number of literals, and M is the maximum number of literals per clause, whereas for the CLIQUE problem an n^D algorithm is possible, where n is the number of vertices and D is the maximum vertex degree. Thus the theory of NP-completeness can be used to guide our search not only for polynomial time algorithms, but for exponential time algorithms as well.