The Efficiency of Algorithms

Some mathematical problems can be solved only by methods too slow for even the fastest computers. More efficient methods have not been found, but neither has it been proved that there are no better methods

by Harry R. Lewis and Christos H. Papadimitriou

Suppose you were asked to plan an itinerary for a traveling salesman who must visit a number of cities. You are given a map on which the distances between the cities are marked and you are asked to find the shortest route that passes through all the cities and returns to the starting point. An approach to this problem that is certain to give the correct answer is to trace all the possible routes, measure their length and pick the shortest one. If the tour included more than a few cities, however, hundreds or thousands of routes would have to be checked. If there were 100 cities, then even the fastest computers would require weeks of calculation to find the shortest path.

In the search for a quicker solution you might try some less rigorous methods. One idea that seems reasonable is always to visit nearby cities before going on to those farther away. You would soon discover, however, that this procedure does not invariably yield the correct answer. Other shortcuts also fail. In fact, the best methods known for solving the problem are not much better than the obvious but laborious procedure of checking all the possible itineraries. Mathematicians now suspect that this problem and many others like it may forever remain beyond our ability to solve in any efficient way. That speculation itself, however, is unconfirmed; although no faster methods of solution have been found, neither has it been proved that faster methods do not exist.

In the problem of the traveling salesman's tour it is not the solution for a particular set of cities that is of the greatest importance but a general method for finding the solution for any cities. Such a method is called an algorithm; it is a precisely stated procedure or set of instructions that can be applied in the same way to all instances of a problem. If the problem is to be solved with the aid of a computer, an algorithm is indispensable, because only those procedures that can be stated in the explicit and unambiguous form of an algorithm can be presented to a computer. Instructions that are vague or that rely on intuition are unacceptable.

An example of an algorithm is the procedure taught in the schools for the subtraction of whole numbers. If each of the steps in this procedure is applied correctly one at a time, the algorithm will always yield the correct result. What is more, once the algorithm has been learned or stored in the memory of a computer or embodied in the circuitry of an electronic calculator, it can be applied to an infinite set of subtraction problems. With this one algorithm the difference between any two whole numbers can be determined.

In principle any problem for which an algorithm can be devised can be solved mechanically. It may therefore seem surprising that there are problems for which algorithms exist but for which we so far have no practical general solution. The algorithms for solving these problems always give a correct answer, but they often require an inordinate amount of time. The problem of the traveling salesman's tour is among these intractable tasks.

The efficiency of computer algorithms is a topic of obvious practical importance. It is also of interest in more formal areas of mathematics. There are some problems in mathematics and logic for which no algorithm can ever be written, and there are many others for which efficient, fast algorithms are already known. Between these two groups is a third class of problems that can always be solved in principle but for which there are only inefficient (and therefore largely unusable) algorithms. For some of these difficult problems mathematicians have been able to demonstrate that efficient algorithms can never be designed. For many of the most important problems, however, there is only the suspicion that good algorithms are impossible.

A given problem can have more than one algorithm for its solution. For example, children in Europe learn a procedure for subtraction slightly different from the one taught in the U.S. Both of the subtraction algorithms, however, give the same result in the same amount of time. That is not invariably the case with different algorithms for solving the same problem. One celebrated problem that can be solved by either a "fast" algorithm or a "slow" one is the problem of the Königsberg bridges.

In the 18th-century German city of Königsberg (which is now the Russian city of Kaliningrad) a park was built on the banks of the river Pregel and on two islands in the river. Within the park seven bridges connected the islands and the riverbanks. A popular puzzle of the time asked if it was possible to walk through the park by a route that crossed each of the bridges once and only once.

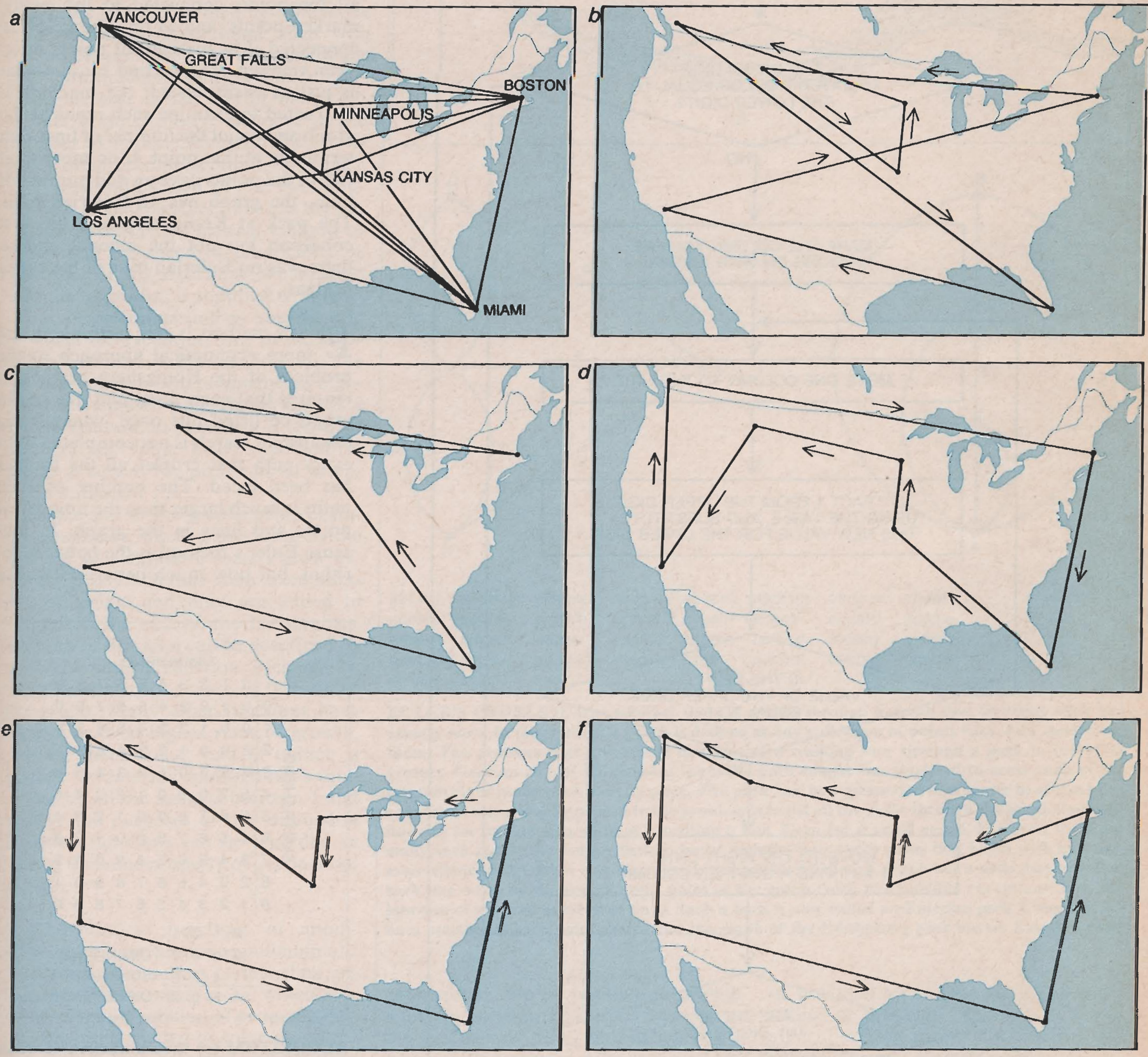
For the solution of the problem the size and shape of the islands and the length of the bridges are immaterial; the only essential information is the pattern of interconnections. This information can be presented compactly in the mathematical structure known as a graph, which is merely a set of points with lines drawn to join them. In the case of the Königsberg park each of the riverbanks and each of the islands is condensed to a single point and each of the bridges is represented by a line between two points. Thus the graph consists of four points and seven lines. If the lines are labeled, any path through the park can be specified by a simple listing of labels.

The obvious approach to the problem is to list all the paths that cross all the bridges and to eliminate from consideration those that cross any bridge more than once. This is the technique of exhaustive search, similar to the one employed in the problem of the traveling salesman. When the mathematician Leonhard Euler was presented with the problem of the Königsberg bridges, he recognized the limitations of the technique and found another method. In recognition of his contribution a path that traverses each line of a graph exactly once is now called an Eulerian path.

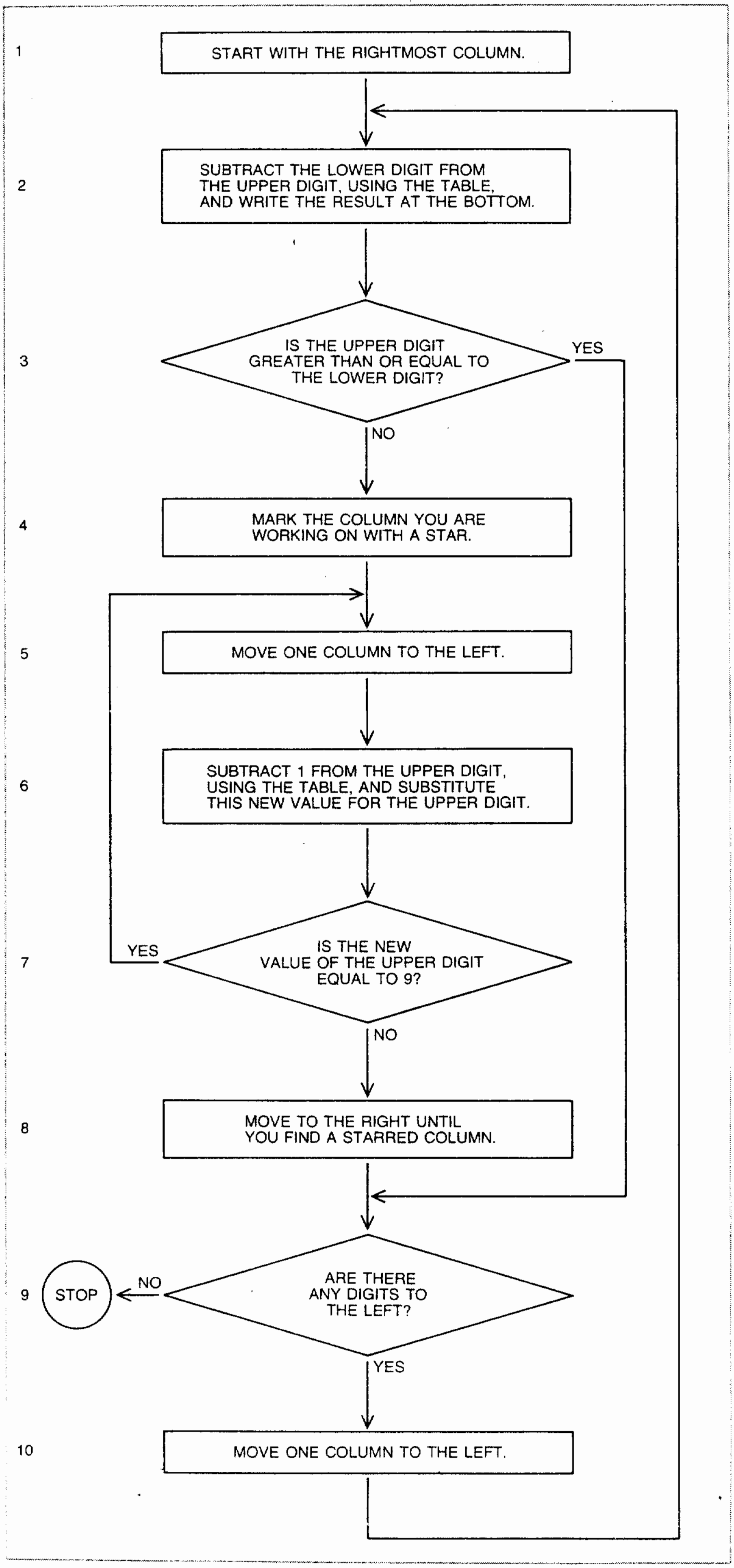
Euler wrote: "The particular problem of the seven bridges of Königsberg could be solved by carefully tabulating all possible paths, thereby ascertaining by inspection which of them, if any, met the requirement. This method of solution, however, is too tedious and too difficult because of the large number of possible combinations, and in other problems where many more bridges are involved it could not be used at all."

Euler's alternative method is much simpler. He showed that a tour of the kind sought must exist if the graph meets two conditions. First, it must be possible to go from any point in the graph to any other point by following the lines of the graph; in other words, the graph may not be disconnected. Second, every point of the graph, with two possible exceptions, must be at the junction of an even number of lines.

It is not hard to understand why a graph cannot have an Eulerian path unless it meets these conditions. All regions of the graph must be connected to one another if there is to be any path that traverses all the lines. Each point must have an even number of lines because half of them are required to reach the point and the other half to leave it. Two points with an odd number of lines can be allowed if they are chosen as the starting and finishing points of the path. Demonstrating that any graph



TRAVELING SALESMAN'S TOUR is a commonplace problem that reveals a deep deficiency in the methods available to mathematics. The problem can be solved, but only by means so arduous and time-consuming that the solution is generally impractical. The problem is, given a map showing the airline routes connecting several cities (*a*), to find the itinerary for the shortest round-trip tour of the cities. One approach to the problem that works consistently is to plot all the possible tours, measure their length and choose the shortest one. Even with only seven cities, however, there are more than 350 tours, and the number increases rapidly (as the factorial of the number of cities). Several shortcut methods might be attempted. For the salesman always to fly to the city farthest from the one where he is clearly gives a wrong answer; the resulting tour (*b*) is among the longest tours possible. Even an arbitrary route, such as one visiting the cities in alphabetical order (*c*), gives a better result. Dividing the country into segments and visiting first the East, then the Middle West and then the West gives a still shorter tour (*d*). Finally, the salesman might always fly to the nearest city, a procedure that leads to two reasonable itineraries, the difference between them depending on whether he begins at Minneapolis (*e*) or Kansas City (*f*). Even those routes are not the shortest possible. (The optimum tour is shown in the illustration on page 107.) The kind of solution sought is an algorithm, or set of instructions, that will find the shortest tour for any group of cities. No known algorithm is significantly better than exhaustive search.


1
START WITH THE RIGHTMOST COLUMN.
2
SUBTRACT THE LOWER DIGIT FROM THE UPPER DIGIT, USING THE TABLE, AND WRITE THE RESULT AT THE BOTTOM.
3
IS THE UPPER DIGIT GREATER THAN OR EQUAL TO THE LOWER DIGIT?
YES
NO
4
MARK THE COLUMN YOU ARE WORKING ON WITH A STAR.
5
MOVE ONE COLUMN TO THE LEFT.
6
SUBTRACT 1 FROM THE UPPER DIGIT, USING THE TABLE, AND SUBSTITUTE THIS NEW VALUE FOR THE UPPER DIGIT.
7
YES
IS THE NEW VALUE OF THE UPPER DIGIT EQUAL TO 9?
NO
8
MOVE TO THE RIGHT UNTIL YOU FIND A STARRED COLUMN.
9
STOP
NO
ARE THERE ANY DIGITS TO THE LEFT?
YES
10
MOVE ONE COLUMN TO THE LEFT.

that meets these conditions actually has an Eulerian path requires a somewhat more complicated argument, which we shall not present here, but Euler was able to give a rigorous proof.

It is an easy matter to express Euler's solution to this problem in an algorithm that could be executed by a computer. The first requirement, connectivity, can be established by marking some point of the graph, then similarly marking all points connected to it by lines, then all the points connected to the newly marked points, and so on. The graph is connected if at the end all points have been marked. The second requirement is just as easily tested: the machine is instructed to examine each point of the graph and count the number of lines that terminate at that point. If no more than two of the points have an odd number of lines, the graph has an Eulerian path. The park at Königsberg met the first condition but not the second, and so there was no Eulerian tour of the seven bridges.

Euler's method is unquestionably the more economical approach to the problem of the Königsberg bridges: it requires that each point and line of the graph be listed just once, whereas the exhaustive search is not completed until every path that crosses all the bridges has been listed. The number of such paths is much larger than the number of points and lines in the graph. In that sense Euler's method is the better algorithm, but how much better? How can

SUBTRAHEND	DIMINUEND									
	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	9	0	1	2	3	4	5	6	7	8
2	8	9	0	1	2	3	4	5	6	7
3	7	8	9	0	1	2	3	4	5	6
4	6	7	8	9	0	1	2	3	4	5
5	5	6	7	8	9	0	1	2	3	4
6	4	5	6	7	8	9	0	1	2	3
7	3	4	5	6	7	8	9	0	1	2
8	2	3	4	5	6	7	8	9	0	1
9	1	2	3	4	5	6	7	8	9	0

	2	*	3	*	
	~~3~~	0	~~4~~	7	DIMINUEND
−	1	4	3	8	SUBTRAHEND
	1	6	0	9	DIFFERENCE

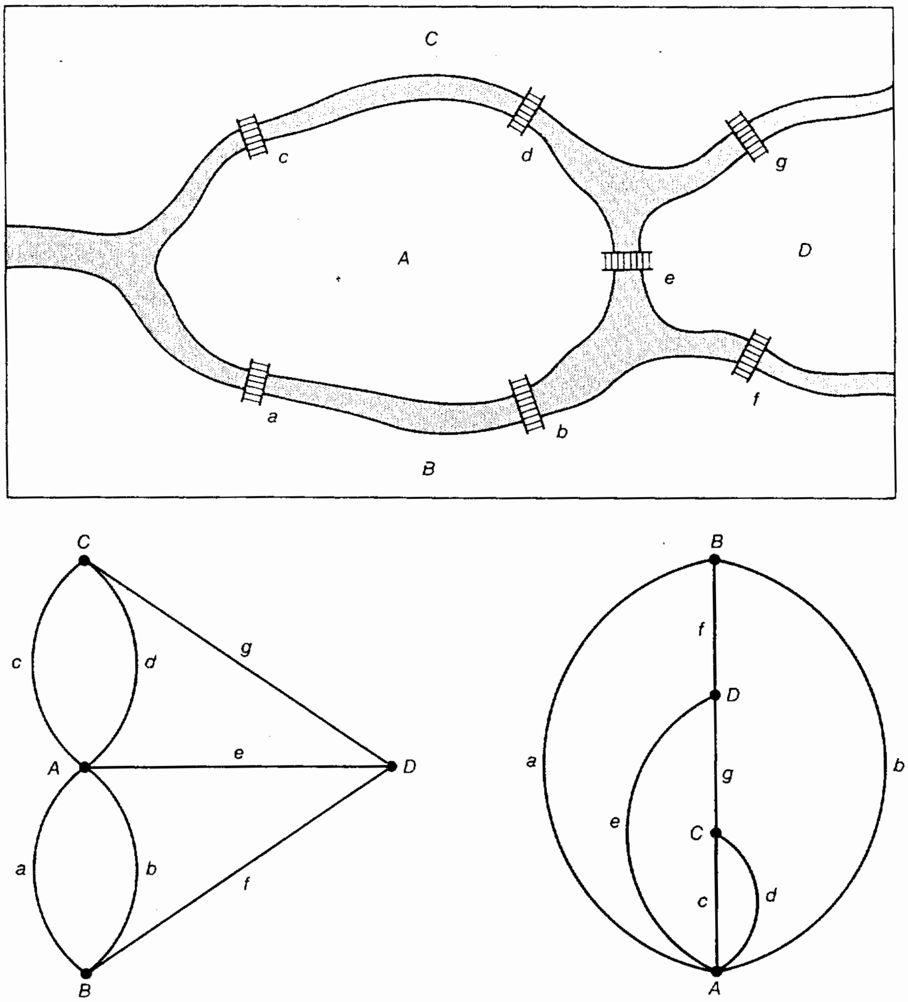
ALGORITHM for the subtraction of whole numbers defines an explicit procedure that can be followed without any need for intuition and even without an understanding of the significance each step has to the operation as a whole. The algorithm, which is assumed to incorporate the table of differences shown here, can be applied to an infinite number of subtraction problems. Other algorithms are equally effective. A method of subtraction taught in European schools, for example, differs in the treatment of borrowing where it specifies that 1 should be added to the lower digit instead of being subtracted from the upper one.

the difference be measured, and how can one tell if the difference is significant?

For a graph with only four points and seven lines both techniques are fast enough to be considered practical. Suppose, however, that more islands and bridges were added to the park, or in other words that more points and lines were added to the graph. If the problem is being solved by Euler's method, each new point merely adds one item to the list of points that must be checked. If the paths are to be examined by exhaustive search, on the other hand, then with each new point and line the size of the list is multiplied by some factor. A moderate increase in the size of the graph results in an explosive increase in the number of paths. Ultimately the list of paths must become prohibitively long.

In this comparison of the two solutions to Euler's problem there is the basis for a completely general method of evaluating the speed or practicality or efficiency of any algorithm. We imagine that the algorithm is supplied with larger and larger inputs, and we note the rate at which the execution time of the algorithm increases. In this way it is possible to make unequivocal judgments of algorithms. Exhaustive search not only is a slower method; in general it is too slow to be of any value. Euler's method remains practical for problems of essentially unlimited size.

As the size of the graphs being examined increases, the lists produced by the method of exhaustive search grow exponentially. Each time some fixed number of points and lines are added to the graph the size of the list doubles. Growth of this kind can be described by a mathematical function such as 2^n, where n is some measure of the size of the graph. Many other functions have similar or even higher rates of growth. Among them are n^n and $n!$ (which is read as "n factorial" and signifies n multiplied by all the integers between 1 and n). For the purposes of this discussion all these functions can be regarded as having the same property of exponential growth.



abcdef	abcdeg	abcged	abcgf	abdcef	abdceg	abdgec	abdgf	A: abcde	= 5
acdbfe	acdbfg	acdefb	acdeg	acgebf	acged	acgfbd	acgfbe	B: abf	= 3
adceg	adgebf	adgec	adgfbc	adgfbe	aefbcd	aefbcg	aefbdc	C: cdg	= 3
aegdbf	aegdc	abegcd	abegdc	abef	adcbfe	adcbfg	adcefb	D: efg	= 3
aefbdg	aegcbf	aegcd							

EULER'S PROBLEM asks whether there is a path through a graph that traverses each line exactly once. In this context a graph is defined as any collection of points with lines connecting them. The problem was first stated in terms of a walking tour through a park in the 18th-century German city of Königsberg (*top*); the path sought was required to cross each of the seven bridges in the park exactly once. The park can be represented as a graph in at least two equivalent ways. One approach to the problem is to list all the paths through the graph that continue as far as they can without repeating a line. Even for a small graph, however, there are many such paths; the sample listing shown includes only those paths that begin with line *a*. A more efficient algorithm was discovered by Leonhard Euler. A graph has a path that traverses each line once, he showed, if every point of the graph (with two possible exceptions) is at the junction of an even number of lines. Such a path is now called an Eulerian path. Counting the lines meeting at each point shows that the graph of the Königsberg park has no Eulerian path.

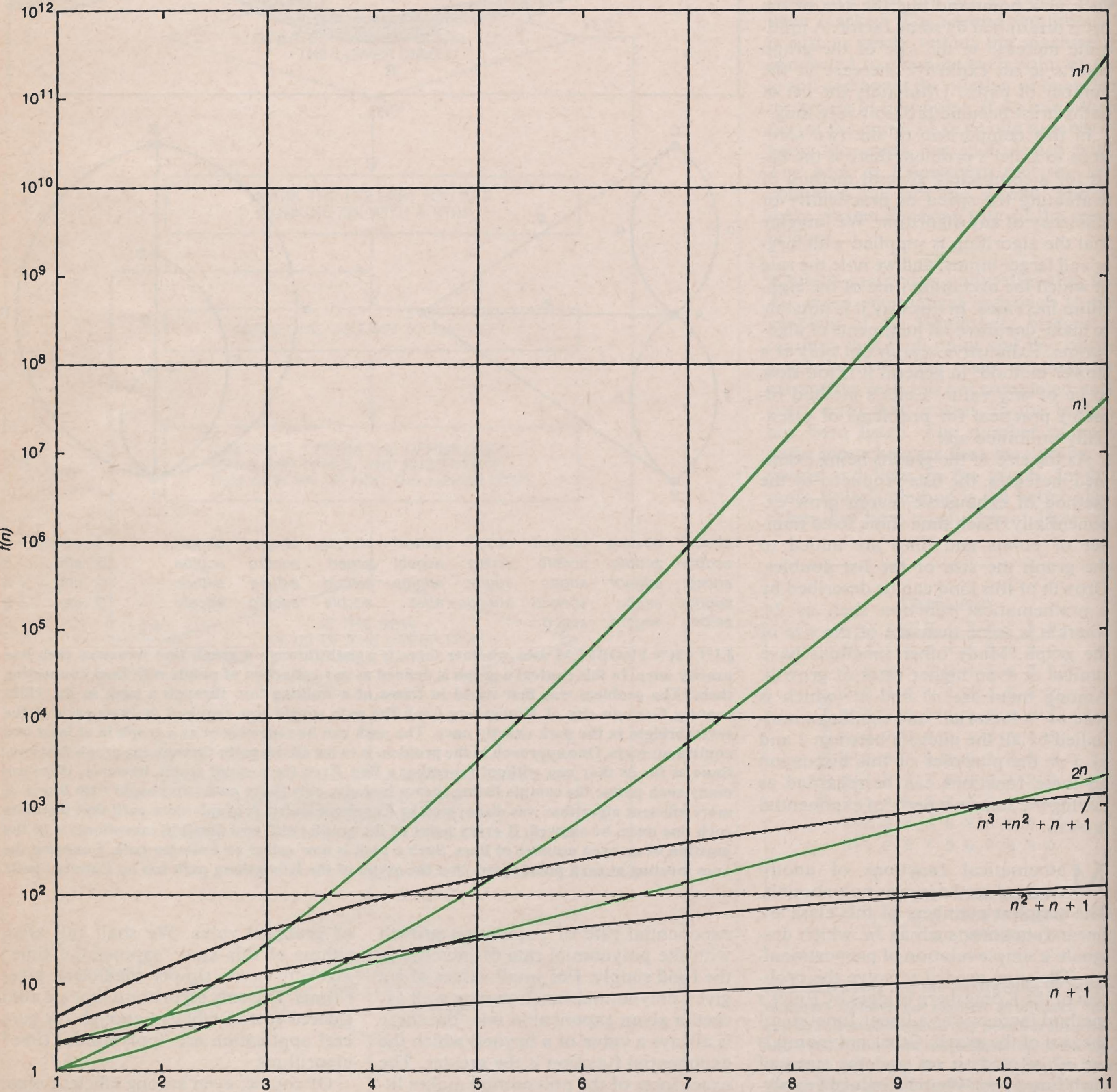
Mathematical functions of another kind are known as polynomials. The simplest members of this class are linear functions, such as $3n$, which designate a simple relation of proportionality. The time needed to solve the problem of the Königsberg bridges by Euler's method increases as a linear function of the size of the graph. Other polynomials are n^2, n^3 and so on, and the sums of such functions. What distinguishes polynomials from exponential functions is that n never appears in an exponent.

For sufficiently large values of n any exponential function will overtake and exceed any polynomial function. It was the certainty of this result that dismayed Thomas Malthus when he compared the exponential rate of population growth with the polynomial rate of increase in the food supply. For small values of n a given polynomial function may well exceed a given exponential one, but there is always a value of n beyond which the exponential function is the greater. The exact form of the polynomial makes little difference, except in changing the point at which the polynomial function is overtaken.

It is now generally agreed by computer scientists that algorithms whose execution time increases exponentially as a function of the size of the input are not of practical value. We shall call algorithms of this kind "exponential time" algorithms, or simply inefficient algorithms. The only algorithms that are considered fast or efficient enough for general application are "polynomial time" algorithms.

Of course, even among efficient algorithms some are faster than others, but for the purposes of this discussion it is important only to distinguish polynomial-time algorithms as a class from exponential-time algorithms. Moreover, this system of classification has the advantage that it makes the speed of an al-

FUNCTION	TYPE	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$	$n = 7$	$n = 8$	$n = 9$	$n = 10$	$n = 11$
$f(n) = n + 1$	POLYNOMIAL	2	3	4	5	6	7	8	9	10	11	12
$f(n) = n^2 + n + 1$	POLYNOMIAL	3	7	13	21	31	43	57	73	91	111	133
$f(n) = n^3\ \ n^2 + n + 1$	POLYNOMIAL	4	15	40	85	156	259	400	585	820	1,111	1,464
$f(n) = 2^n$	EXPONENTIAL	2	4	8	16	32	64	128	256	512	1,024	2,048
$f(n) = n!$	EXPONENTIAL	1	2	6	24	120	720	5,040	40,320	362,880	3,628,800	39,916,800
$f(n) = n^n$	EXPONENTIAL	1	4	27	256	3,125	46,656	823,543	16,777,216	387,420,489	10,000,000,000	285,311,670,611



RATE OF GROWTH in the execution time of an algorithm as the size of the problem increases determines the practicality of the algorithm. The rates of growth are described by mathematical functions that can be classified as either polynomial or exponential. The values of n in the table at the top and in the graph at the bottom represent some measure of the size of a problem; the values of the functions correspond to execution time. For any exponential function and any polynomial one there is a value of n beyond which the exponential function is always the greater. For this reason algorithms whose execution time increases as an exponential function of the problem size are considered inefficient and in general are of no practical value. Algorithms whose execution time increases as a polynomial function, in comparison, are efficient. In the problem of the Königsberg bridges the execution time for Euler's method increases as a polynomial function such as $n + 1$; the method of exhaustive tabulation requires an amount of time given by an exponential function such as 2^n.

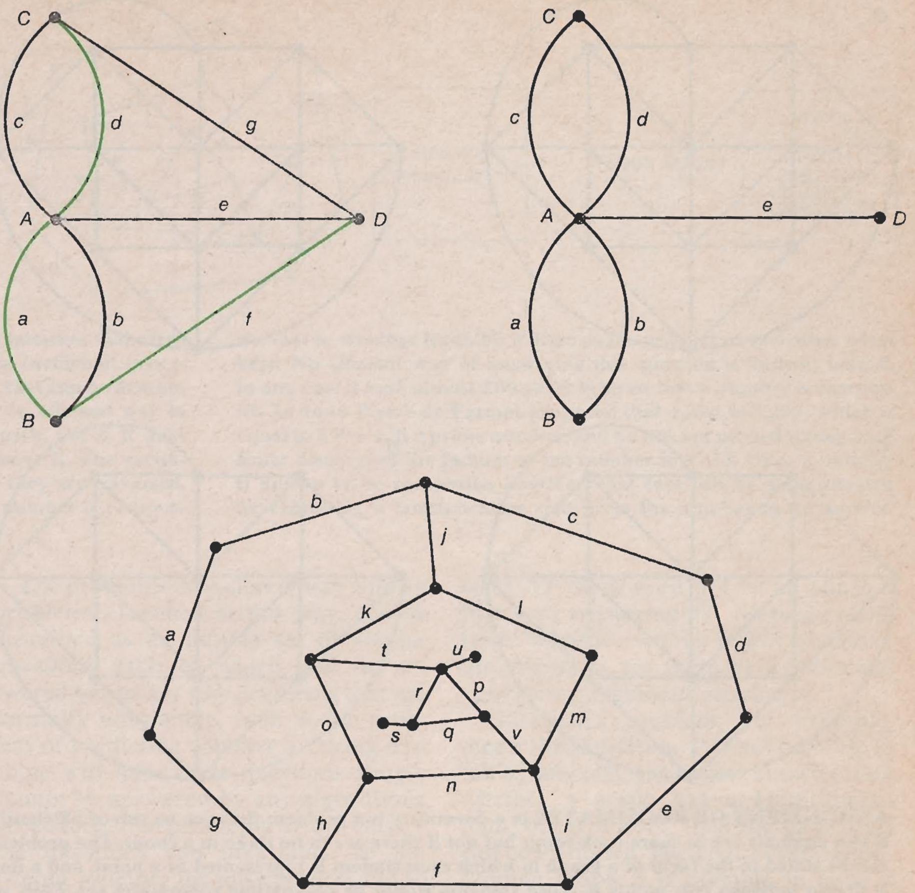
gorithm a property of the algorithm itself and independent of the machine on which it is executed. For sufficiently large problems a polynomial-time algorithm executed on even the slowest machine will find an answer sooner than an exponential-time algorithm on the fastest computer.

The mathematical properties of algorithms were studied in the 1930's by the British mathematician A. M. Turing, the inventor of the imaginary computer now called a Turing machine. The Turing machine was conceived to be an automaton equipped with an infinite supply of paper tape marked off in square regions. The machine was capable of just four actions: it could move the tape one square, it could place a mark in a square, it could erase a mark already present and at the end of a calculation it could halt. These operations were to be performed according to a sequence of instructions built into the internal mechanism. Of course, Turing never built such a machine; it was merely a conceptual device for automatically solving problems in mathematics and logic. Indeed, Turing was interested not in actually solving problems with the machine but rather in investigating what kinds of problems it could solve and what kinds it could not.

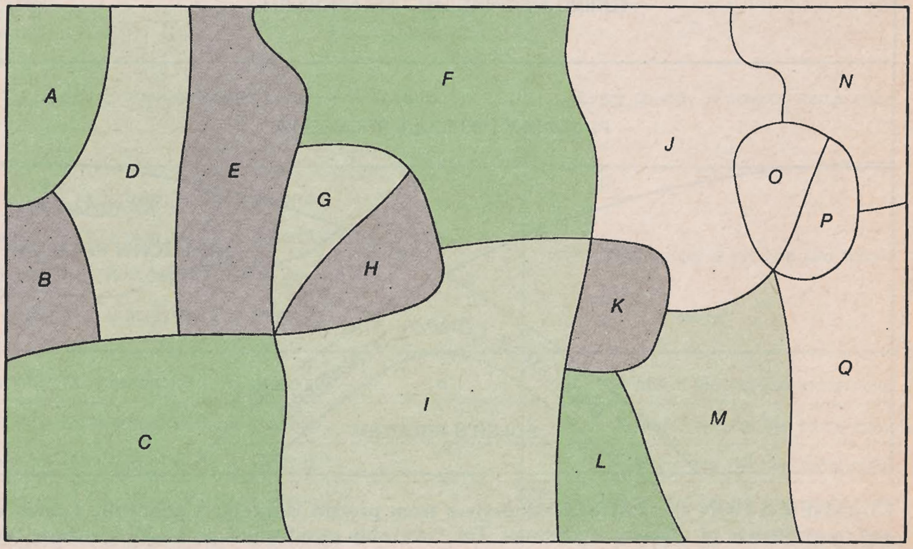
Turing discovered that even a machine as simple as this one could solve any problem for which an algorithm could be devised. The computation might be laborious and indirect, but given enough time and paper tape the machine would eventually find the solution and halt. Reduced to its essentials the Turing machine is a language for stating algorithms, as powerful in principle as the more sophisticated languages now employed for communicating with computers.

In addition to conceiving these machines Turing demonstrated their limitations. In 1936 he showed that there are problems that cannot be solved by Turing machines, and it follows that these problems cannot be solved by any automatic computer. They are problems for which algorithms cannot be written, even in principle. The example first studied by Turing is the problem of predicting whether a particular Turing machine, once it is set in motion, will ever finish its calculation and halt. Through an analysis of this problem he was able to show that there can be no general procedure for telling whether mathematical propositions are true or false. Since then a variety of other problems with the same properties have been proposed.

One result of Turing's work was to divide all imaginable problems in mathematics into two classes. Those problems for which algorithms can never be written are in a formal sense permanently unsolvable. Some instances of these problems may be solved by rare percep-



HAMILTON'S PROBLEM, formulated by the Irish mathematician William Rowan Hamilton, resembles Euler's problem superficially but asks whether there is a path through a graph that touches each point (instead of each line) exactly once. The graph derived from the park at Königsberg (*top left*) has a Hamiltonian path (*color*) although it has no Eulerian path. By removing two lines a graph is formed (*top right*) that has an Eulerian path (beginning at point *D* and proceeding, for example, along lines *e, a, b, c* and *d*) but not a Hamiltonian path. The third graph (*bottom*) has neither property. No efficient algorithm for solving Hamilton's problem is known; the available methods are fundamentally no better than exhaustive search.



THREE-COLOR-MAP PROBLEM asks whether three colors can be applied to the regions of a map so that no two regions sharing a border have the same color. The problem can be solved for any particular map by examining all the possible colorings, but such a procedure is extremely tedious; for the map shown there are 3^{16}, or about 43 million, possible colorings. No efficient algorithm for solving the problem is known, but where a three-color solution does exist it can be found in principle by guessing (a nonalgorithmic method). If there is no solution, guessing is of no value. The trial solution shown fails on reaching country *J*, which communicates with all three colors and therefore cannot be assigned any of them. The failure does not prove, however, that no three-color solution exists; indeed, a solution can be found fairly quickly by exchanging the colors of regions *K* and *L* and continuing the series of guesses in the rest of the graph.

tion or by luck, but a general method for their solution will never be found.

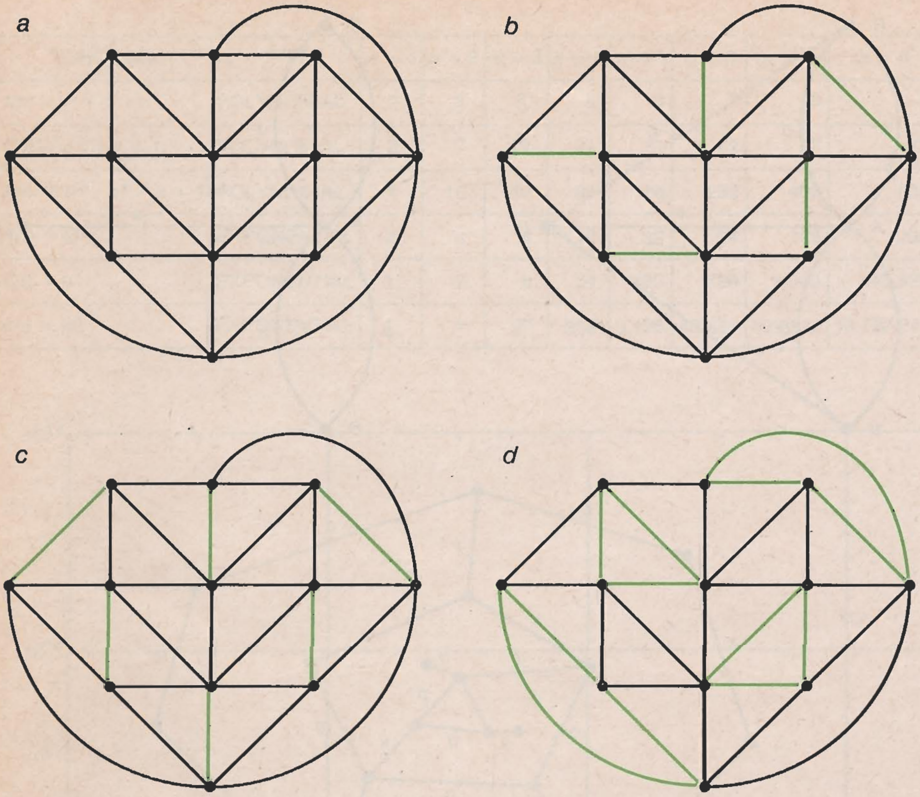
All other problems in mathematics and logic can be solved by algorithms. As we have seen, however, some algorithms are more useful than others. The class of solvable problems can therefore be divided into two subgroups: those for which there are efficient, polynomial-time algorithms and those for which there are only exponential-time algorithms. Euler's problem is a member of the class with polynomial-time solutions, since Euler's method is itself a polynomial-time algorithm. Problems that can be proved to have only exponential-time algorithms are also known, although they are rather obscure.

Although these two groups of problems are quite distinct, it is not always a straightforward task to assign a problem to one group or the other. Indeed, a very interesting class of problems seems to fall somewhere between them. For these problems no efficient algorithms are known and the best available solutions require exponentially increasing time, yet no one has been able to prove that the problems do not have polynomial-time solutions.

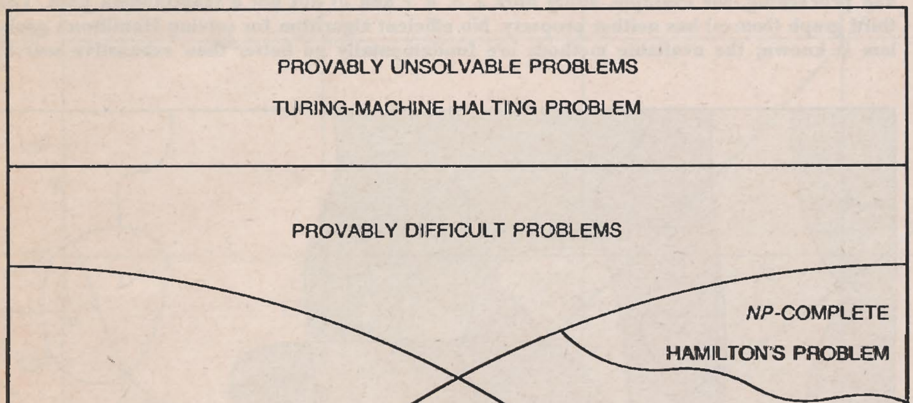
One such problem was considered in the 19th century by the Irish mathematician William Rowan Hamilton. Superficially Hamilton's problem is much like Euler's. The problem is to decide whether a given graph has a path that takes in each point exactly once (whereas Euler looked for a path that traversed each line once). Actually the tasks are quite different, and Euler's method cannot be applied to Hamilton's problem. The graph derived from the plan of the park at Königsberg has a Hamiltonian path, although, as we have seen, it has no Eulerian path. On the other hand, removing two lines results in a graph that has an Eulerian path but not a Hamiltonian one. Many other graphs have neither kind of path.

Hamilton's problem can be solved by exhaustive search; indeed, the procedure is not substantially different from that employed in listing all the possible paths that might have the Eulerian property. For Hamilton's problem, however, no efficient algorithm comparable to Euler's method has been found. The problem has been pondered by many of the best mathematicians of the past century, but the most efficient methods available today are fundamentally no better than exhaustive tabulation. On the other hand, all attempts to prove that there is no better method have also failed, and it must be considered a possibility that an efficient algorithm will be discovered tomorrow.

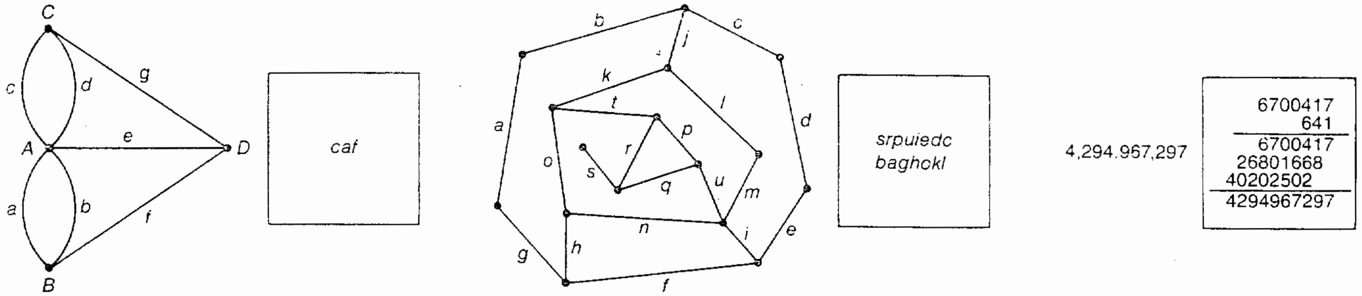
Problems that are known to have polynomial-time solutions, such as Euler's problem, are said to be members of the class *P* (for polynomial). Hamilton's problem is a member of another class,



ASSIGNMENT OF ROOMMATES in a dormitory is a problem that can be solved efficiently if two students are to share each room but not if there are to be three in a room. The problem can be stated in the form of a graph in which each student is represented as a point, and a line is drawn between two points if those students would be compatible roommates (*a*). Pairs of students can be assigned to rooms by a technique called augmentation: an incomplete assignment is made (*b*), then a complete one is found (if it exists) by a sequence of rearrangements (*c*). There is no equivalent efficient procedure for finding groups of three compatible students. Once an acceptable room assignment has been found, however, it can readily be exhibited (*d*).


PROVABLY UNSOLVABLE PROBLEMS
TURING-MACHINE HALTING PROBLEM
PROVABLY DIFFICULT PROBLEMS
NP-COMPLETE
HAMILTON'S PROBLEM
co-NP
NP
PROBLEM OF COMPOSITE NUMBERS
P
EULER'S PROBLEM

CLASSIFICATION OF PROBLEMS derives from present conjectures about the existence and nonexistence of efficient algorithms. The "provably unsolvable" problems are those for which there are no algorithms of any kind; the "provably difficult" ones have only exponential-time algorithms. Problems with known polynomial-time algorithms are assigned to the class *P* (for polynomial). The status of the remaining problems is less certain: only exponential-time algorithms are known for them, but it has not been proved that efficient algorithms do not exist. These problems are assigned to the classes *NP* and co-*NP*. The letters *NP* stand for "nondeterministic polynomial" and signify that the problems can be solved quickly by guessing. Co-*NP* includes the yes-or-no problems whose complementary no-or-yes versions are in *NP*. *NP*-complete is a subset of the class *NP* made up of problems with a special property: if any one of them could be solved by an efficient algorithm, then all other problems in the class *NP* could also be solved efficiently. The discovery of such an algorithm would constitute a proof that *P* and *NP* are identical, but most mathematicians believe such an algorithm does not exist.

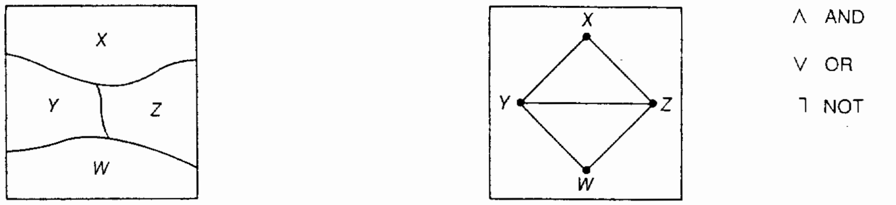


PROBLEMS IN THE CLASS *NP* ask a yes-or-no question that often can be answered only through a time-consuming, inefficient procedure, but when the answer is known to be yes, that fact can be demonstrated by a short proof. For the present there is no efficient way to determine whether a graph has a Hamiltonian path, but if it does have one, a brief "certificate" can be issued to prove it. The certificate simply lists the lines of the path in the order they are traversed. Another problem in the class *NP* asks whether a number is composite, that is, whether it can be written as the product of two other numbers. No efficient way of answering this question is known; indeed, in one case it took almost 100 years to show that a number is composite. In 1640 Pierre de Fermat proposed that 4,294,967,297, which is equal to $2^{32} + 1$, is a prime number, and he was not proved wrong until Euler discovered the factors of the number in 1732. Once a number is known to be composite, however, that fact can be demonstrated by exhibiting a multiplication that gives the number as an answer.

designated by the letters *NP*, signifying "nondeterministic polynomial." The class *NP* encompasses all the problems in *P*, or in other words *P* is a subset of *NP*. In addition *NP* includes problems whose status is less certain. They are all solvable problems in principle; they have algorithms, but for now only exponential-time algorithms are known. They may also have polynomial-time algorithms (in which case *NP* and *P* are identical), or they may prove to be permanently intractable, with only inefficient solutions.

The problems considered here, and all problems classified in this way, can be described as an infinite set of similar questions each of which can be answered yes or no. For problems that are formally unsolvable, such as the problem of predicting whether a Turing machine will halt, these questions simply cannot be answered by any algorithmic procedure. For problems of the class *P* the questions can invariably be answered, whether the answer turns out to be yes or no, by an efficient procedure. In order for a problem to qualify for the class *NP* there need not be an efficient means of answering the yes-or-no questions. What is required is that whenever the answer is yes there be a short and convincing argument proving it.

Hamilton's problem, for example, meets this condition. It is not possible to tell by any efficient means known today whether a graph has a Hamiltonian path, but if it does, then the path itself can be exhibited. Hence for every Hamiltonian graph it is possible to issue a "certificate" that proves its membership in this special class of graphs. Such a



[(X is red) ∨ (X is blue) ∨ (X is green)] ∧ [(Y is red) ∨ (Y is blue) ∨ (Y is green)] ∧ [(W is red) ∨ (W is blue) ∨	Every country is colored some color.
(W is green)] ∧ [(Z is red) ∨ (Z is blue) ∨ (Z is green)] ∧	
[(¬X is red) ∨ (¬X is blue)] ∧ [(¬X is red) ∨ (¬X is green)] ∧ [(¬X is blue) ∨ (¬X is green)] ∧	No country is colored two colors.
[(¬Y is red) ∨ (¬Y is blue)] ∧ [(¬Y is red) ∨ (¬Y is green)] ∧ [(¬Y is blue) ∨ (¬Y is green)] ∧	
[(¬W is red) ∨ (¬W is blue)] ∧ [(¬W is red) ∨ (¬W is green)] ∧ [(¬W is blue) ∨ (¬W is green)] ∧	
[(¬Z is red) ∨ (¬Z is blue)] ∧ [(¬Z is red) ∨ (¬Z is green)] ∧ [(¬Z is blue) ∨ (¬Z is green)] ∧	
[(¬X is red) ∨ (¬Y is red)] ∧ [(¬X is blue) ∨ (¬Y is blue)] ∧ [(¬X is green) ∨ (¬Y is green)] ∧	X and Y are not the same color.
[(¬X is red) ∨ (¬Z is red)] ∧ [(¬X is blue) ∨ (¬Z is blue)] ∧ [(¬X is green) ∨ (¬Z is green)] ∧	X and Z are not the same color.
[(¬Y is red) ∨ (¬Z is red)] ∧ [(¬Y is blue) ∨ (¬Z is blue)] ∧ [(¬Y is green) ∨ (¬Z is green)] ∧	Y and Z are not the same color.
[(¬Y is red) ∨ (¬W is red)] ∧ [(¬Y is blue) ∨ (¬W is blue)] ∧ [(¬Y is green) ∨ (¬W is green)] ∧	Y and W are not the same color.
[(¬W is red) ∨ (¬Z is red)] ∧ [(¬W is blue) ∨ (¬Z is blue)] ∧ [(¬W is green) ∨ (¬Z is green)]	W and Z are not the same color.

PROPOSITIONAL CALCULUS serves as a universal language for problems in the class *NP*. The problem considered here is that of applying three colors to a map or its equivalent graph. A sentence in the propositional calculus is made up of statements, such as "*X* is red," joined by the logical connectives "and," "or" and "not." If two statements are joined by a logical "and," the sentence is true only if both statements are true; a logical "or" requires that at least one of the statements be true, and "not" signifies that a statement is false. The sentence representing the map-coloring problem has three parts: the first two establish that every country has exactly one color and the third part lists the countries that cannot have the same color. The map-coloring problem is thereby reduced to the problem of determining whether this sentence can be satisfied, that is, whether it is possible to assume some statements to be true and others false in such a way that there are no contradictions. Since all problems in *NP* could be expressed as sentences in the propositional calculus, an efficient general method for solving the satisfiability problem could be applied to all those problems. No such method is known. The sentence given here can be satisfied by assuming, for example, that only the statements "*X* is red," "*Y* is blue," "*Z* is green" and "*W* is red" are true.

certificate would name the lines in the graph in the order the path traverses them. Finding the path might require weeks of tabulation, but once it has been found it can easily be exhibited. Another problem that belongs to the class *NP* is the question of whether a whole number is composite, that is, whether it can be written as the product of two other numbers. Again there is no known efficient procedure for answering the question, but if the number is indeed composite, there is a succinct proof of that fact, namely a correctly worked-out multiplication with the number on the bottom line.

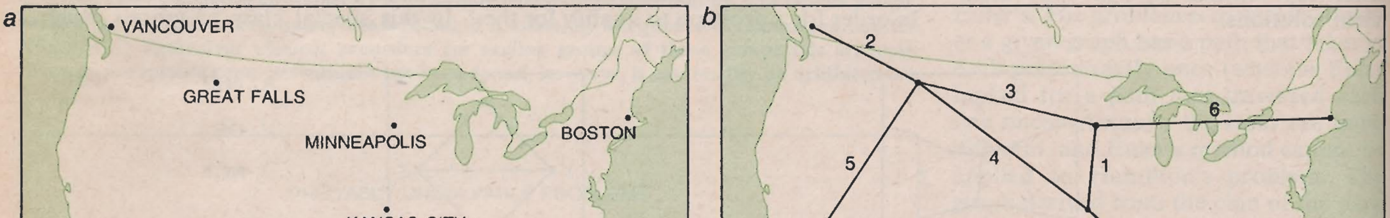
Care must be taken when asking the yes-or-no question of a problem in the class *NP*, since the complementary no-or-yes problem might not be in the same class. For example, the complement of Hamilton's problem, in which one is asked to show that a graph does not have a path passing once through each point, may well not be in the class *NP*. For now the only way to demonstrate the absence of such a path is to list all possible paths, and such a proof is too lengthy to qualify as a certificate of membership in *NP*. On the other hand, the complement of the composite-number problem, which asks if a number is prime, turns out to be in the class *NP*. The reason, which is far from obvious, is that relatively short proofs demonstrating that a number has no factors other than 1 and itself were discovered in 1975 by Vaughan Pratt of the Massachusetts Institute of Technology. Still, it is not known whether the composite-number problem and its complement are in the class *P*.

It is easy to show that every problem in the class *P* is also in the class *NP*. If a problem is in *P*, then by definition there is an efficient algorithm for it. To produce a short and convincing proof that the answer to some instance of the problem is yes, all we need to do is follow the algorithm; a record of its operation constitutes the required certificate.

Another way of defining *NP* is as the class of yes-or-no problems that can be solved by guessing certificates. If one is given an instance of a problem in *NP* for which the answer happens to be yes, then with luck one may discover the required certificate fairly quickly by making a sequence of guesses; if the answer is no, guessing cannot possibly yield an answer any faster than an exhaustive search could. For example, in solving Hamilton's problem one might find a correct path (if there is one) on the first try by tracing small portions of the path and guessing at each stage how to proceed. Such a procedure, it should be emphasized, is not an algorithm. It could be made into an algorithm only by crossing off each trial path as it is tested and checking all possible paths, but that is equivalent to the method of exhaustive search.

A mathematical procedure defined in terms of lucky guesses may seem bizarre, but it is a quite legitimate approach to defining the problems in the class *NP*. In principle the procedure could even be mechanized by building a device called a nondeterministic Turing machine. This device can do all that an ordinary Turing machine can do; in addition, at some points in its operation it may have more than one choice of what to do next. Such a machine would be considered to answer yes to a question if there were some sequence of choices that could lead it to a yes conclusion. *NP*, the class of nondeterministic polynomial-time problems, consists of precisely those problems whose yes instances can be identified by machines making comparatively short guessing computations.

The inclusion of guessing in the definition of these problems suggests strongly



SPANNING-TREE PROBLEM **calls for the shortest network of lines connecting a set of points, or equivalently the shortest railroad system connecting a set of cities (*a*). If the lines are allowed to meet only at cities, the problem can be solved by an efficient procedure called the greedy algorithm. First the closest pair of cities are joined, then the next-closest and so on (*b*). Lines joining cities that are already connected indirectly (such as line *4* here) are omitted. The result is the optimum spanning tree (*c*). An even shorter network is possible if lines are allowed to meet at isolated junction points. At each such point three lines should meet at angles of 120 degrees, but there is no efficient algorithm for determining where junction points should be introduced. The optimum network (*d*) was found by exhaustive search.**

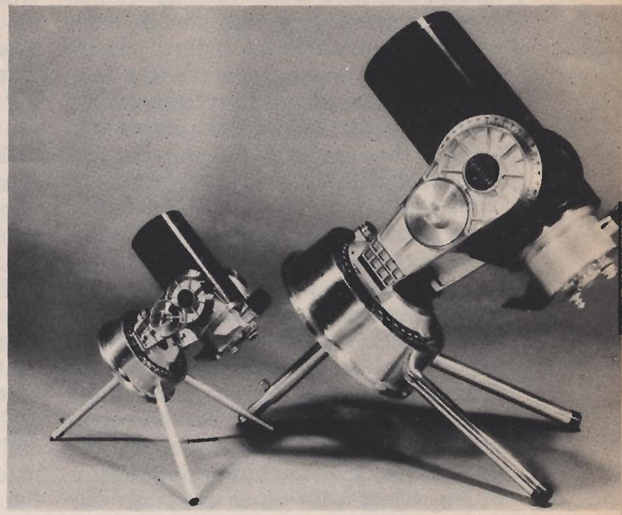
to many mathematicians that P and NP are not the same set and hence that efficient algorithms can never be found for the intractable problems in the class NP. If every problem in NP were actually in P, then all the guesswork and luck could be replaced by some systematic procedure without great sacrifice in time. It is hard to believe the ability to guess and be lucky could win so little.

The class NP includes a variety of commonly encountered problems that seem to defy efficient solution. We have already mentioned Hamilton's problem and the problem of composite numbers. Another example is known as the matching problem. It can be considered in terms of the task faced by the colleges every September, when a new class of freshmen must be assigned to shared dormitory rooms.

For the sake of simplicity let us assume that all the information gathered about the students' smoking habits, bedtime hours, taste in music and so forth results in a single yes-or-no decision as to the compatibility of each possible pair of students. The entire class can then be represented as a graph in which the points correspond to students and a line is drawn connecting every two students who can be placed in the same room. If each room holds just two students, the assignment can be made efficiently by a clever polynomial-time algorithm discovered by Jack Edmonds of the University of Waterloo. If each room is to be shared by three students, however, there is no known efficient algorithm. The problem is in the class NP, since all yes instances have succinct certificates: an acceptable room assignment, once it is discovered, can easily be exhibited. Of course, a solution could be found by exhaustive search, albeit inefficiently. With luck a suitable assignment, if there is one, can be guessed quickly.

Map coloring is a problem in the class NP that concerns mathematicians more than it does cartographers. The question is whether the countries on a given map can be colored with a given number of colors so that no two countries that share a border have the same color. It is easy to find out if a map can be colored with two colors: it can be if there are no places on the map where an odd number of countries meet at one point. It is even easier to tell if a map can be colored with four colors; indeed, there is no need even to look at the map, since Kenneth Appel and Wolfgang Haken of the University of Illinois proved in 1975 that four colors suffice for any map. Surprisingly, however, no efficient algorithm is known for determining whether three colors are enough for a given map. The problem is in the class NP, since a correctly colored map can serve to certify a yes answer.

Map coloring can be regarded as a



special case of another problem called graph coloring. Any map can be converted into a graph by reducing each country to a point and drawing a line between two points if the corresponding countries share a border. Coloring the graph is then equivalent to coloring the map, subject to the rule that two points connected by a line cannot have the same color. Graph coloring, however, is a more general problem, with applications outside graph theory. For example, a graph can represent the scheduling of work in a factory. Each point of the graph stands for some job to be done, and two points are connected by a line if the jobs cannot be done concurrently, perhaps because they require the same piece of machinery. A coloring of the graph with three colors would then supply a schedule dividing the work of the factory into three shifts. Like map coloring, the graph-coloring problem is in the class *NP*.

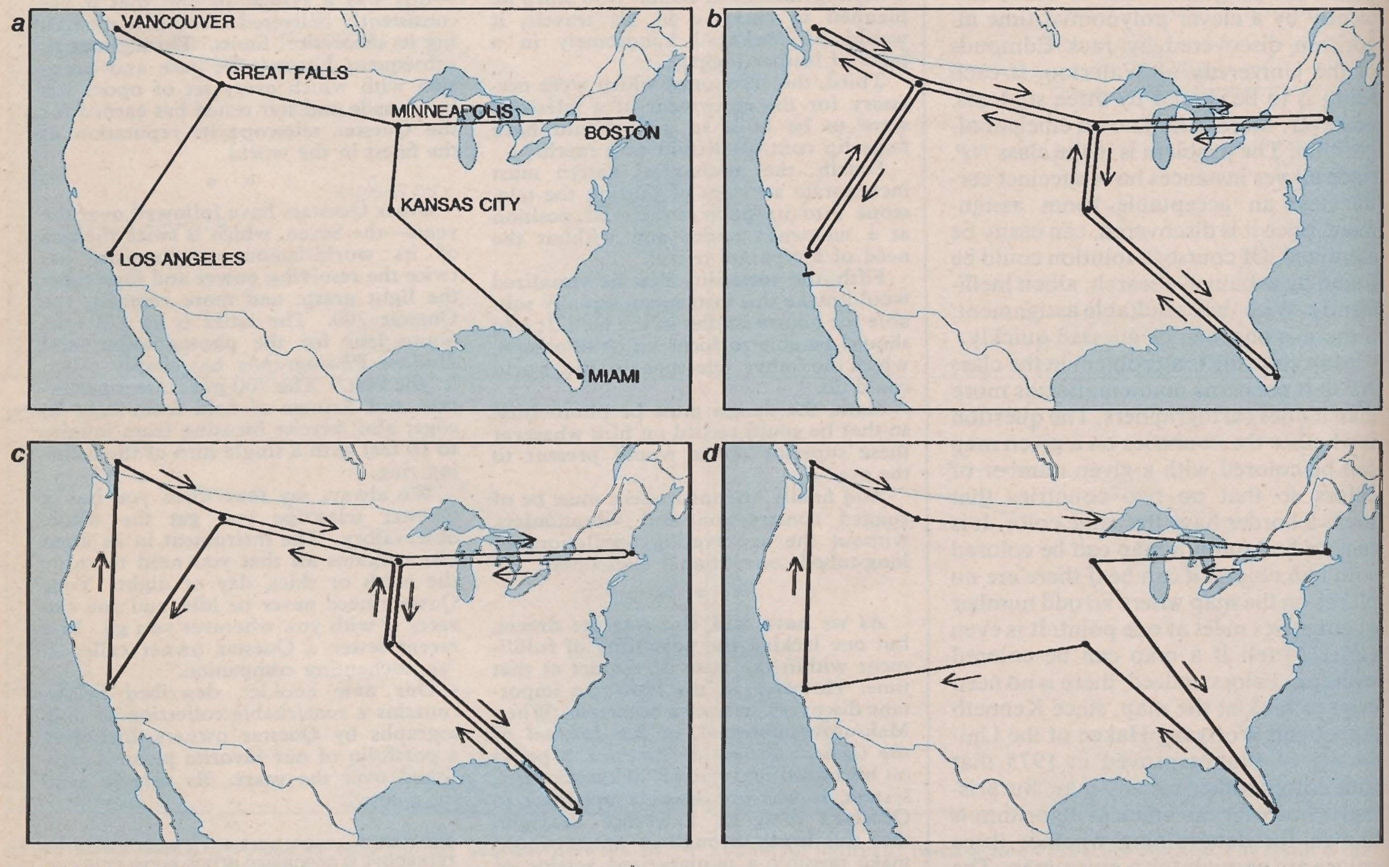
It often happens that if one problem can be solved efficiently, so can many others. For example, if an efficient algorithm could be found for the problem of graph coloring, it could be applied with only minor modifications to the problems of map coloring and factory scheduling. Map coloring and factory scheduling are therefore said to be efficiently reducible to graph coloring. In the past several years it has become apparent that some of the problems in the class *NP* have a remarkable property: all the problems in *NP* are efficiently reducible to them. These elite problems within the class *NP* are called *NP*-complete. If any one of them has an efficient algorithm, then every problem in *NP* can be solved efficiently.

The first proof that a problem is *NP*-complete was presented in 1971 by Stephen A. Cook of the University of Toronto. His reasoning follows a path essentially parallel to the path of Turing's earlier work on mathematical machines and their relation to problems of formal logic. Cook stated his proof in terms of the propositional calculus, the formal language in which separate logical statements, which individually may be either true or false, are joined together by the lexical elements "and," "or" and "not." In general a sentence in the propositional calculus can be shown to be either true or false depending on which of its component statements are assumed to be true or false. Certain sentences, however, cannot be true under any interpretation because they are self-contradictory. Sentences that cannot be made true are said to be unsatisfiable.

Cook employed the propositional calculus to describe the operation of the nondeterministic Turing machines, the mechanized guessing devices essential to the definition of the class *NP*. He showed that the calculations of any such machine can be described succinctly by sentences of the propositional calculus. When the machine is given a yes instance of a problem in *NP*, its operation is described by a satisfiable sentence, whereas the operation of a machine given a no instance is described by a sentence that cannot be satisfied.

It follows from Cook's proof that if one could efficiently determine whether a sentence in the propositional calculus can be satisfied, one could also determine efficiently in advance whether the problem presented to a nondeterministic Turing machine will be answered yes or no. Since the problems in the class *NP* are by definition all those that can be solved by nondeterministic Turing machines, one would then have an efficient method for solving all those problems. The catch, of course, is that there is no known efficient method of determining whether a sentence in the propositional calculus can be satisfied.

Cook's argument states in essence that the propositional calculus is a universal language for describing problems in the



SHORTEST-TOUR PROBLEM cannot be solved exactly by any known efficient algorithm. Mathematicians have therefore devised solutions that are good even if they are not optimal. One efficient procedure draws a tour guaranteed to be no more than twice the shortest length. The method begins with the optimum spanning tree (*a*), which can be found efficiently with the greedy algorithm. The spanning tree can be converted into a tour simply by traversing each line once in each direction (*b*). This tour is clearly just twice as long as the spanning tree itself; the spanning tree in turn must be shorter than any tour of the cities, since a tour could be made into a tree (albeit a tree with no branches) by omitting one segment. The tour produced by this method can generally be improved by taking shortcuts (*c*, *d*).

class *NP*. Every instance of such a problem corresponds to a sentence in that language, and if the sentence is satisfiable, the instance has a yes answer. Many other problems have since been shown to be *NP*-complete because the satisfiability problem can efficiently be reduced to them. Hamilton's problem, the problem of matching groups of three roommates and the problem of coloring graphs with three colors are all *NP*-complete. The first to point out the broad applicability of this theory was Richard M. Karp of the University of California at Berkeley. Similar investigations were independently conducted by the Russian mathematician P. A. Levin. Since *NP*-complete problems capture the difficulty of all other problems in *NP*, it is widely thought today that all *NP*-complete problems are computationally intractable. A proof that a problem is *NP*-complete is usually considered a strong argument for abandoning further efforts to devise an efficient algorithm for its solution.

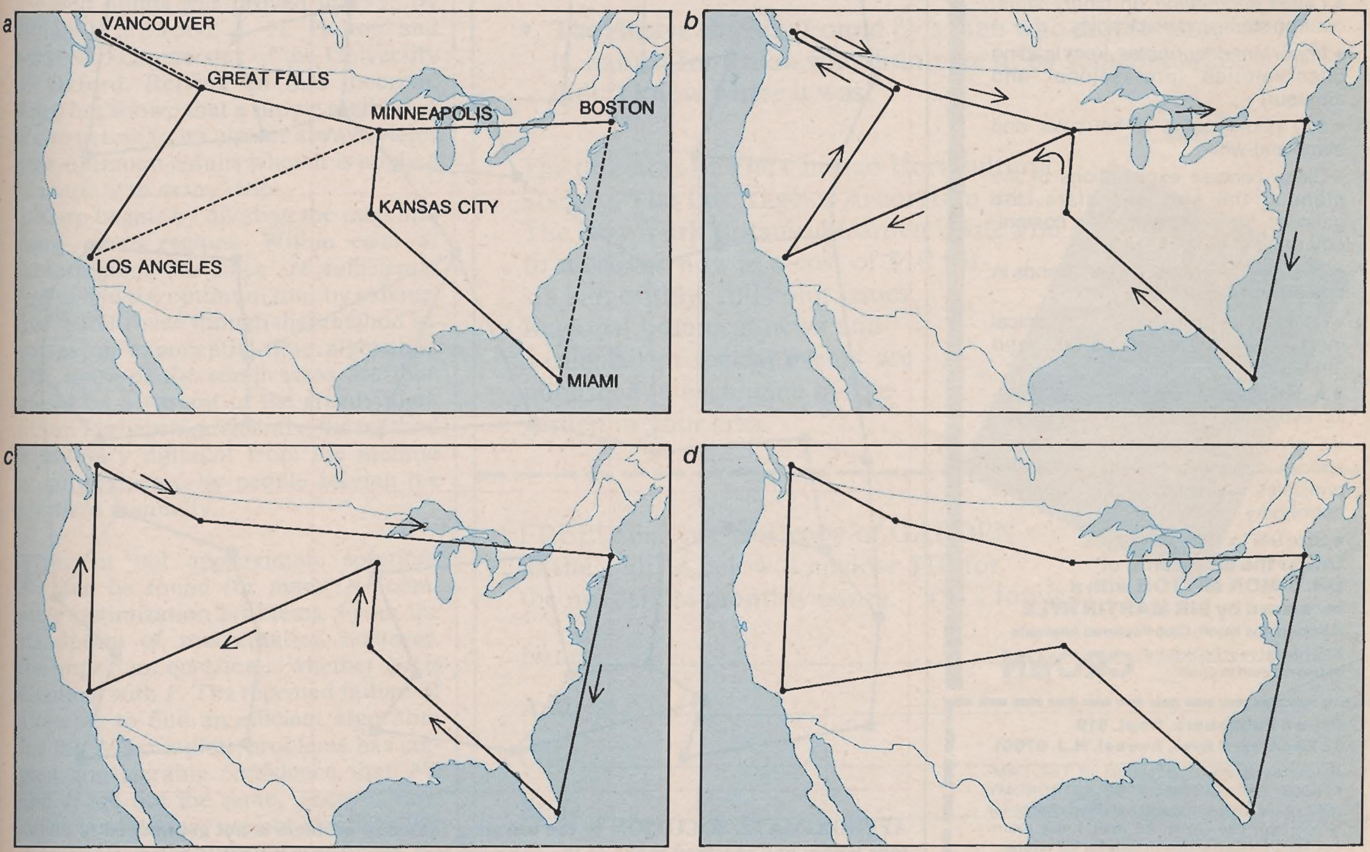
Even the assumption that all *NP*-complete problems are intractable would not settle all questions about the class *NP*. In addition to the mystery of the *NP*-complete problems there is an even more obscure area: problems in *NP* for which no efficient algorithms are known but which have not been proved to be *NP*-complete either. The problem of composite numbers is one of these.

Not all problems that can be solved by a computer are of the yes-or-no type. Another common type is the optimization problem. For example, suppose one is given the positions of some cities on a map and asked to find the shortest possible railroad network connecting them. In one version of this problem one is allowed to lay down a straight section of track between any two cities, but one is not allowed to install isolated junction points; tracks can be joined only at cities. One property of the solution to this problem is immediately apparent: the optimum network can never include a closed circuit, because if it did, the network could be made shorter simply by omitting one link in the circuit. Thus the best network always branches like a tree, and the problem itself is called the spanning-tree problem.

The spanning-tree problem can be solved correctly and quite efficiently by a method called the greedy algorithm, devised by Joseph B. Kruskal of Bell Laboratories. The procedure is simply to connect the closest pair of cities, then the next-closest and so on without adding any superfluous lines (lines joining cities that are already linked indirectly). It is far from obvious that this method always yields the shortest network, but it does, and it has the pleasant property of requiring no foresight and no reconsideration of earlier decisions.

The greedy algorithm can be relied on to find the shortest network between cities under the rules specified, but in general that network will not be the shortest possible one. Further savings can be achieved by establishing junction points between cities. The properties of networks with such junctions were studied by the Swiss mathematician Jakob Steiner. It can be shown that any shortest network must be arranged so that each junction point is made up of three lines that meet at angles of 120 degrees. This rule provides some guidance in evaluating networks, but there are many possible networks with Steiner junction points. No algorithm has been discovered that finds the best network quickly.

The problem of the traveling salesman's tour is closely related. Again one is given a set of cities, but now one is asked to find the shortest round-trip tour. As a first guess the greedy algorithm suggests that perhaps the salesman should always go to the nearest city he has not yet visited, but this procedure



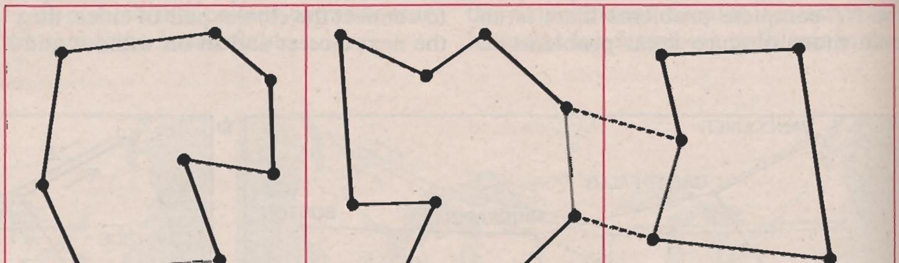
IMPROVED ALGORITHM for the traveling salesman problem yields a tour that is certain to be no more than 50 percent longer than the optimum. The procedure was devised by Nicos Christofides of the Imperial College of Science and Technology. The first step is again to generate the shortest spanning tree. All the cities that are linked to an odd number of cities are then singled out; in this example all the cities except Kansas City have an odd number of connections. These cities are next linked in pairs by a procedure similar to the one employed in matching pairs of students to yield a tour (*a*) that can be improved by making shortcuts (*b, c*). The result is only a little longer than the optimum tour (*d*), found by exhaustive search. Note that in the shortest tour the line between the two closest cities is omitted.

does not work. Indeed, the problem is notorious for having resisted all attempts to find an efficient solution.

Optimization problems are not fundamentally different from those that ask yes-or-no questions; in fact, every optimization problem can be rewritten in a yes-or-no form. In the traveling salesman problem, for example, we might be given a length along with a set of cities and asked to state whether a tour can be constructed that does not exceed the specified length. This yes-or-no problem cannot be harder than the associated optimization problem, because if the optimum tour could be found by some efficient method, it would be a trivial task to determine whether it exceeds a given number. Hence if the yes-or-no version is computationally intractable, one cannot hope to solve the optimization problem itself efficiently. For this reason certain optimization problems, such as that of the traveling salesman's tour and that of placing Steiner junction points, are said to be *NP*-complete.

Optimization problems are encountered often in engineering, economics, operations research and other fields. The discovery that at least some of these problems are *NP*-complete is therefore of considerable practical interest. Since the *NP*-complete problems probably have no efficient algorithms, there would seem to be little point in expending further effort in seeking optimum solutions. An alternative that has recently been adopted is to seek approximate solutions that are good even if they are not precisely optimal.

One technique that has been applied to the traveling salesman problem offers a solution that may not be optimum but is guaranteed to be no worse than twice the optimum path. The procedure starts with the shortest spanning tree, which can be generated efficiently by the greedy algorithm. This network can be converted into a tour of the cities by traversing each line twice and returning to the origin. It is known that the optimum spanning tree must be shorter than any possible tour of the cities, since a tour can be converted into a spanning tree (albeit one without any branches) by simply omitting one segment. Thus twice the length of the optimum spanning tree cannot be longer than twice the optimum tour. The meth-



APPROXIMATE SOLUTION to the traveling salesman problem is not guaranteed to find a tour within any specified range of the optimum tour, but when it is applied to many instances of the problem, it is not far wrong very often. The algorithm, which was devised by Richard M. Karp of the University of California at Berkeley, divides a map into many small regions, each one containing only a few cities. Within each region the optimum tour is found by exhaustive search, a procedure that is practical since the number of points is small. Each of the small tours is then regarded as a single entity to be linked to the others, a task that can be performed efficiently by an algorithm similar to the greedy algorithm for finding an optimum spanning tree.

od is a polynomial-time algorithm. Recently Nicos Christofides of the Imperial College of Science and Technology in London has found a way to improve the algorithm so that it yields a tour guaranteed to be no more than half again as long as the optimum.

A more profound compromise gives up not only the requirement that a solution be optimal but also the insistence that a less than optimum solution be guaranteed to fall within a specified range. Instead the assurance is given that the solution will not often deviate far from the optimum. An underlying assumption of such techniques is that the maps encountered in practice are not concocted to confound basically plausible techniques; such maps are encountered frequently only when they are constructed by computer scientists to reveal the flaws in methods proposed by their colleagues. Indeed, if the salesman's-tour algorithms discussed above are applied to "natural" maps, they deliver far more than they promise. The resulting tours are not 100 percent or 50 percent longer than the optimum but closer to 5 percent.

A reasonable assumption about the properties of many maps is that cities are randomly placed. A theorem describing the statistical properties of optimum tours through such randomly distributed points was proved in 1958 by Jillian Beardwood, J. H. Halton and John M. Hammersley of the University of Oxford. Relying on that theorem, Karp has shown that a simple method of constructing tours almost always yields near-optimum results when it is applied to maps with many cities.

Karp begins by dividing the map into many small regions. Within each of these regions the cities are sufficiently few to find the optimum tour by exhaustive search, even though that method involves an exponential-time algorithm. The tours of the small areas are then linked by a variant of the greedy algorithm. Perhaps significantly, the method is not very different from the method usually adopted by people solving the problem manually.

Efficient but approximate solutions can be found for many *NP*-complete optimization problems. From the standpoint of mathematics, however, the important question is whether *NP* is identical with *P*. The repeated failure of attempts to find an efficient algorithm for the *NP*-complete problems has created considerable confidence that *NP* and *P* are not the same. There is now suspicion that they are not identical, but the proof of their distinctness may be beyond present mathematical capabilities. The question may join that select group of mathematical enigmas that remain unresolved for decades, and the solution may have to await the development of new methods in mathematics.