Computational Unsolvability

Problems where complexity grows exponentially are now believed incapable of exact solution on even the fastest possible computers

BY LYNN ARTHUR STEEN

Problems of arrangement and scheduling are among the most common yet most vexing in all of applied mathematics. Sometimes solutions are intuitively obvious; other times they are surprisingly paradoxical. Some problems succumb to prosaic methods of attack, while other apparently similar problems are totally intractable.

The common managerial task of scheduling a given set of jobs among available staff in order to finish the work in the least possible time is a good example. Ordinarily, if the manager sees that he can't meet his deadline with existing staff, he will add an additional person. But in some cases, this additional person might increase rather than decrease the total length of time required to finish the job. The subtleties of scheduling are so deep that, although it is possible to write a computer program that will determine the most efficient schedule for a staff of two, it appears that the same problem for a staff of three requires so much computer time as to be, for all practical purposes, impossible to execute when the number of jobs is large.

Recent research has revealed a profound dichotomy in the nature of these combinatorial problems. Some are, in a specific technical sense, easy, while others are hard. The latest major problem to be successfully diagnosed in these terms is over 100 years old. It begins in a tale of four cities and ends in current research in computer networks and integrated circuit design.

Suppose an engineer wants to find the least expensive way to join cities located at the four corners of a square with a network of telephone cables. Since the cost of installing the network is roughly proportional to the total length of cable, he might expect that the best solution is to lay cable along the diagonals of the square, with a junction in the middle. But, surprisingly, this does not yield the shortest (and cheapest) network: the engineer could do about 3 percent better if he used two junctions and joined the cables at angles of 120° (see diagram). This configuration is the optimal, or best possible, solution to the engineer's problem. Because this solution was first studied by the 19th-century German geometer Jacob Steiner, the junction points where three paths meet at equal angles are today called Steiner points. (The 120° requirement for minimal path lengths is the two-dimensional analogue of the 120° angles at which soap films meet [SN: 9/20/75, p. 186].)

Modern engineers confront varieties of Steiner phenomena in problems ranging from the design of microprocessor chips to nationwide communication networks: The determination of the shortest network linking certain given vertices is one of the famous unsolved problems of combinatorial mathematics known as the Steiner minimal tree problem. (It is called a tree problem because of the resemblance between its solution network and complex branching of trees.) Although the nature of the general solution is well known, finding locations for the required Steiner points is a very difficult problem.

Until 1961 it wasn't even known if the problem could, at least in principle, be solved by a search of all possibilities. At that time Z. A. Melzak invented an algorithm (a step-by-step solution procedure) that introduced possible Steiner points in a sufficiently systematic way that it would eventually find the optimal configuration. But, despite a variety of improvements since then, Melzak's algorithm takes so much time that even on the fastest computer it is really feasible only for networks with about 15 to 20 vertices.

An experienced designer could do just about as well "eye-balling" the problem, that is, examining a scale drawing and introducing Steiner points where it looks as if they will do the most good. What makes the Steiner minimal tree problem so difficult is trying to tell a computer which of the many possible Steiner points "look good." It is precisely when the problem gets too large for a person to "eye-ball" that the computer is most needed, and that is precisely where existing programs are of no use.

The meager results of nearly two decades of work on the Steiner minimal tree problem led many researchers to speculate that the problem was in fact intractable. That it is indeed intractable has now been proved by Michael Garey, Ronald Graham and David Johnson of Bell Laboratories in Murray Hill, N.J.

To understand the nature of their result we need to examine briefly the nature of computer algorithms that are used to solve problems of scheduling or arrangement. Solving such problems involves searching through different combinations of events in space or time; thus these problems are part of what is known as combinatorial analysis. All such problems have in common a natural "tree structure" in which early tentative decisions by the solver lead to branch points in the solution process where several other options may be pursued.

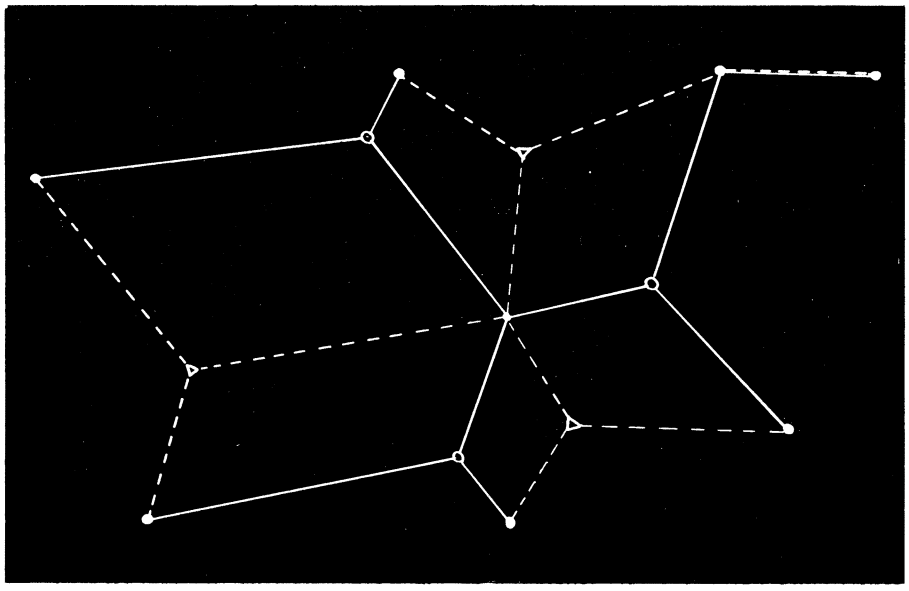
In a typical problem, there may be only a few correct routes through an enormous maze of branches. A combinatorial problem is like a huge tree with fruit at the tips of just a few twigs. How is a near-sighted bug crawling up the trunk going to select only the branches that lead to the fruit?

One way would be to have an oracle who can see the whole system at a glance. This is, in fact, how many relatively small combinatorial systems are solved—by a Gestalt-mathematician who apprehends the solution in a single act of perception. But it does not work for large systems because computers lack human insight, and humans lack the computer's memory.

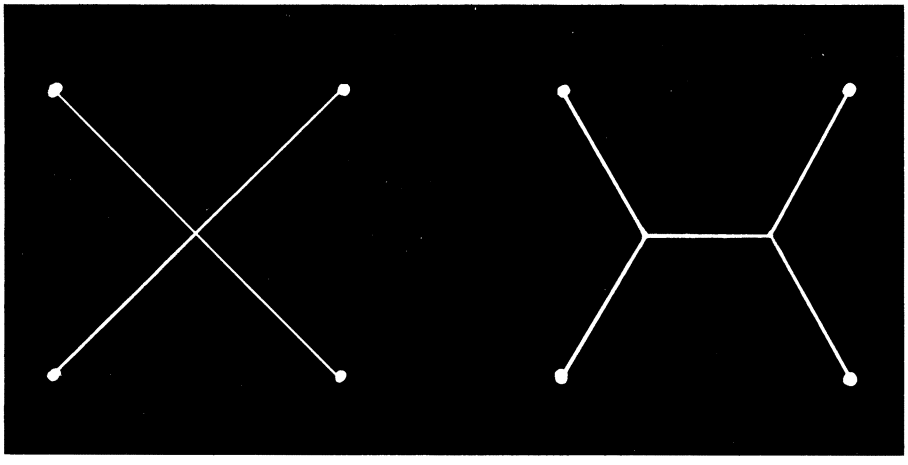
A second method would be to employ a so-called nondeterministic algorithm. "Nondeterministic" is used to describe methods that avoid the problem of determining the correct branch by following all branches simultaneously. This requires, hypothetically, a computer that replicates itself over and over again so that late in the process thousands of similar programs will be working alongside each other, simultaneously pursuing different branches of the solution tree. Whenever any one finds the fruit, the problem is solved.

Nondeterministic algorithms have a natural advantage of speed, for the time required for such an algorithm to solve a particular problem depends only on the total path length from beginning to end (called the depth of the solution tree) and not at all on the number of different branches in the tree. But they achieve this advantage by unrealistic simultaneous replication of computation power. Only in the last few years have parallel-processing computers been developed, and the number of parallel tracks is strictly limited by the nature of the hardware. So, for all practical purposes, nondeterministic algorithms are a figment of a theoretical imagination. They cannot be used for practical solution of large combinatorial problems.

The conventional way to simulate par-



Cities ● connected with a network by means of Steiner points ○ are pictured by the solid line. An alternative Steiner network linking the same cities is marked by dotted lines, using the Steiner points △. Both of these Steiner networks require less total distance than any straight-line network that did not use Steiner points, but it is not immediately clear which of these two Steiner networks uses least total distance.



Two ways to efficiently connect cities with telephone networks. For a square one mile on each side, the obvious method on the left requires $2\sqrt{2} = 2.83$ miles of cable. The more subtle means on the right, using cables that intersect with equal angles at two junctions, required only $1 + \sqrt{3} = 2.73$ miles of cable.

allel processing in a standard sequential computer is by means of a "backtrack" algorithm. Each time a branch point is reached, one branch is pursued and the others are stored in a stack of incompletely developed options. When the program reaches a dead end along the path it is pursuing, it backtracks to the most recently encountered unexplored branch stored in its options stack and pursues it. This process is repeated as often as necessary until one of the sequences of branches leads to a successful conclusion.

Backtrack programming (and related techniques called "branch and bound" algorithms) are just systematized approaches to trial and error: They organize the trials to ensure that errors, once discovered, are never repeated. But even a systematic search of all possibilities requires an enormous amount of time. Typically, the time required for a backtrack solution grows exponentially with the size of the problem.

The facts of exponential growth overwhelm even the astonishing speed of modern computers. Typical combinatorial problems involving, say, 10 cities (or 10 tasks to be scheduled) may require about 2^{10} (roughly, 1,000) branches. Depending on the complexity of the investigation to be carried out along each branch, this can be done in about a second or so of computer time. But if the problem increases to size 50—not all uncommon—the branching increases to 2^{50}; at 1,000 branches per second, this would take over 30,000 years! Thus computer scientists view as intractable large problems whose only known solutions are achieved by backtrack programming. They are computationally unsolvable.

Much better are those algorithms that grow polynomially rather than exponentially. If the time involved is of the order of n^2 or n^3 rather than 2^n, the computation time for large problems is dramatically reduced. For example, to continue with the hypothetical computations of the previous paragraph, a computer checking out 1,000 branches a second would manage 10^2 branches in a tenth of a second, and 50^2 branches in 2.5 seconds. Even 50^3 branches would only take two minutes. So to get a rough measure of the time required to solve a problem, computer scientists look first at whether the solution algorithm grows polynomially or exponentially with the size of the problem data.

Exponential growth most often results from a solution tree that is too broad: Even though the number of steps in a correct solution may grow polynomially, if the number of unfruitful branches that must be explored is too great, the time required by the solution algorithm may grow exponentially with the size of the problem. Problems like this can be solved in polynomial time by a nondeterministic algorithm. But, as we have seen, such algorithms are idealizations, incapable of actual implementation in polynomial time.

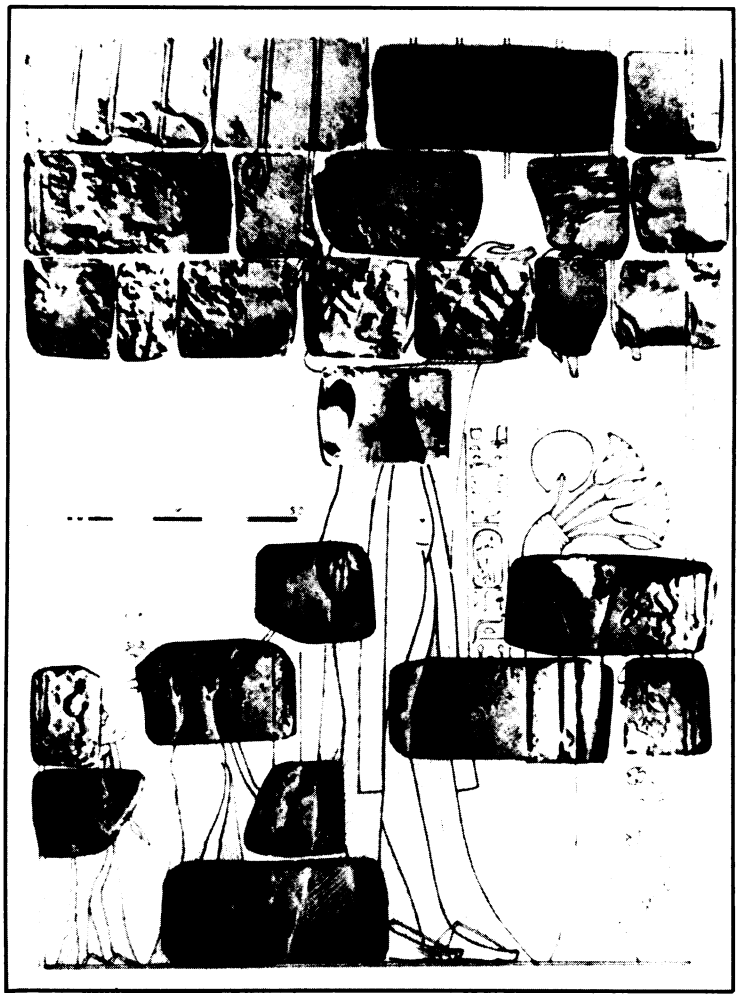
The class of problems that can, in principle, be solved by a nondeterministic algorithm of polynomial time—a class called NP, short for nondeterministic polynomial—thus includes many of the problems whose solutions actually seem to require exponential time. Whether in fact those problems that now seem to require exponential time actually cannot be done in polynomial time is not known: One of the major unsolved problems of current computer science is whether, possibly, every problem in the class NP can really be solved in polynomial time by an ordinary (deterministic) algorithm.

Massive circumstantial evidence has led virtually all informed observers to the conclusion that some problems in NP cannot be solved in polynomial time. This conclusion, if upheld, means that large problems of this type cannot be solved at all.

The first step in this chain of evidence was taken in 1971 by Stephen Cook of the University of Toronto who showed that each problem in the class NP can be transformed into a certain problem in mathematical logic, called the Satisfiability Problem, in such a way that any algorithm that would solve the Satisfiability Problem could be adapted to solve the other problem as well. (The Satisfiability Problem is the question of whether a Boolean logical expression can be satisfied [i.e., made true] by appropriate choice of the propositions from which the expression is built.) Cook's result shows that, in some sense, no problem in the class NP is any harder than the Satisfiability Problem.

Shortly after Cook announced his result, Richard Karp of the University of California at Berkeley showed that many

Continued on page 301



Using talatat to reconstruct a scene: Queen Nefertiti offering to the sun's disc.

finding of the ruined foundation gives the dimensions of the walls and will enable a reconstruction of the temple—visually and on paper only. A reconstruction in stone will not be attempted—there are probably not enough talatat anyway, Rainey says.

The discovery came just like in the movies. The Egyptian workers on the project were digging away near the famous temple of Karnak, in a place where the French archaeologist Henri Chevrier had found toppled statues of Ikhnaton in the 1920s. Suddenly Asmahan Shoukri, an Egyptian member of Redford's group, shouted that the workmen had found "laid stones." Redford ran down, and sure enough there it was: a bit of a wall. The workers also brought up 100 fragments of decorated relief, one of which identifies the building as the temple Gem-Pa-Aton, one of eight that Ikhnaton built to the glory of Aton around the ancient Egyptian capital of Thebes. Redford believes that the temple was built around a courtyard that was 200 to 300 yards long and that it was surrounded by a collonade of rectangular pillars bearing statues of the king. He intends to excavate further, following the wall around. The project is expected to take 10 years.

The talatat as pieced together show figures in procession doing religious things. The 100 most recently unearthed show a procession bearing the king to the temple to be received by bowing courtiers and priests. The direction in which the figures face can be used to determine which wall the stones belonged to. Since the ancient Egyptians, including Ikhnaton, built symmetrically, what is learned about any one wall will help determine the configuration of its opposite. And so eventually Egyptologists hope to know exactly what kind of building this early monotheist constructed for the public worship of his deity.

Because of his innovations and radicalism, Ikhnaton fascinates the modern imagination even more than he dismayed his contemporaries. Much has been written and speculated about him though little is really known. It may, in fact, be that his immediate successors did not quite succeed in obliterating his memory for later generations of his countrymen. It is several centuries from Ikhnaton's day to the time when Moses led the children of Israel through the Red Sea, yet some scholars would like to see an influence of his on the dawning Hebrew consciousness of monotheism and on through modern religion. Some purport to find resemblances between some of Ikhnaton's hymns to the sun and the psalms of the Hebrews. □

. . . Computation

other problems in the class NP share the distinction of the Satisfiability Problem. He called these problems NP-complete; they are sufficiently detailed to serve as prototypes for all other NP problems. Each NP problem can be transformed into any NP-complete problem and solved by appropriate adaptation of the solution algorithm for the NP-complete problem.

NP complete problems form a subclass of the class NP containing those of maximum difficulty. Karp (and others after him) showed that many famous problems of finite mathematics are in this class. These include the famous "traveling salesman problem" (find the shortest route that visits each city on a list exactly once), "0-1 integer programming" (linear programming in which variable values are limited to yes or no options) and "graph coloring" (assign a limited number of colors to regions in such a way that no regions with a common frontier receive the same color). The recent result of Garey, Graham and Johnson shows that the Steiner minimal tree problem is also of this type: It is NP-complete.

Most of the problems now known to be NP-complete have an extensive history of unsuccessful search for a polynomial-time algorithm. Recognition that they belong to the class of NP-complete problems shows that they are essentially equivalent problems. Thus the accumulated evidence of unsuccessful search for efficient algorithms for each of the several dozen NP-complete problems concatenates into an impressive record of failure.

Nearly half a century ago the mathematical logician Kurt Gödel astonished the mathematical and philosophical world by showing that in any sufficiently complex mathematical system there will always be intrinsically undecidable propositions—statements that can, by their very nature, never be proved or disproved. The status of NP-complete problems—if present beliefs are proved true—is somewhat analogous: They are problems that are sufficiently complex that, by their very nature, they cannot be solved in any practical amount of time. Gödel's work established the existence of problems that are theoretically unsolvable; NP-completeness points to the existence of problems that are computationally unsolvable.

Gödel's work on undecidable propositions led logicians away from a fruitless task (the complete formalization of all mathematics) and into more promising terrain. Similarly, the discovery of NP-completeness is right now turning applied combinatorial mathematics from the search for exact algorithms to the search for sufficiently good approximate ones. With this new focus comes a whole host of new and interesting questions concerning the establishment of standards by which an algorithm can be judged when we know that it is in the nature of things that it cannot be perfect. □