

The NP-Completeness Column: An Ongoing Guide

DAVID S. JOHNSON

AT&T Bell Laboratories, Murray Hill, New Jersey 07974

This is the nineteenth edition of a (usually) quarterly column that covers new developments in the theory of NP-completeness. The presentation is modeled on that used by M. R. Garey and myself in our book “Computers and Intractability: A Guide to the Theory of NP-Completeness,” W. H. Freeman & Co., New York, 1979 (hereinafter referred to as “[G&J]”; previous columns will be referred to by their dates). A background equivalent to that provided by [G&J] is assumed, and, when appropriate, cross-references will be given to that book and the list of problems (NP-complete and harder) presented there. Readers who have results they would like mentioned (NP-hardness, PSPACE-hardness, polynomial-time-solvability, etc.) or open problems they would like publicized, should send them to David S. Johnson, Room 2D-150, AT&T Bell Laboratories, Murray Hill, NJ 07974 (or to dsj@btl.csnet). Please include details, or at least sketches, of any new proofs; full papers are preferred. If the results are unpublished, please state explicitly that you are willing for them to be mentioned. Comments and corrections are also welcome. For more details on the nature of the column and the form of desired submissions, see the December 1981 issue of this Journal.

THE MANY FACES OF POLYNOMIAL TIME

The concept of “polynomial time” has played a major role in the development of what we now call complexity theory. Certainly there could have been no theory of NP-completeness without it. It has long been recognized, however, that the practical significance of polynomial time does not quite come up to the level of its theoretical usefulness. This column will deal with some of the ways in which the concept falls short, as emphasized, and in one case as remedied, by major new algorithmic results.

Polynomial time certainly has its drawbacks as a precise equivalent to the more traditional notion of an “efficient” algorithm. For instance, consider the problem of testing whether a given graph has a clique of size k , for large fixed k . The fact that exhaustive search can solve this problem in time $O(n^k)$ means that the problem is in P, but this is not a particularly useful algorithmic observation for any but the smallest values of k . Such a proof of membership in P is less a positive result than what one might call a “negative-negative” result. It does not prove tractability; it merely shows that one cannot prove *intractability*, i.e.,

NP-completeness (assuming $P \neq NP$). This role of polynomial-time solvability as a negative-negative rather than as a positive result has been carried to new extremes (and to much more interesting problems) in recent work of Robertson and Seymour.

The first such result I shall discuss is especially relevant to this column, as it resolves one of the six remaining open problems from the original list of 12 in [G&J]. (The first edition of this column [Dec. 1981] covers the other six solved problems; this is the first breakthrough since then.) The newly resolved problem is SUBGRAPH HOMEOMORPHISM (FOR A FIXED GRAPH H); Robertson and Seymour show that this problem is solvable in polynomial time for all fixed graphs H . This resolution is unfortunately extremely “negative-negative”: the running times are not just exponential in $|H|$, they involve constants that are larger than a “tower of 2’s” whose *height* (number of levels of exponentiation) is worse than exponential in $|H|$. Section 1 describes the result in more detail, and some of the theory of “graph minors” that lies behind it.

Section 2 then covers an even more precipitous leap into negative-negativity, made possible by another Robertson-Seymour result. This result, the proof of a famous combinatorial conjecture due to Wagner, gives rise to the possibility of *non-constructive* proofs of polynomial-time solvability. The proof techniques involved are fairly general, and I shall list a variety of interesting problems whose membership in P has recently been established using them.

I conclude in Section 3 with a different challenge to the standard concept of polynomial time. Here the complaint is not about how horrible the polynomial is, but rather about what the polynomial measures. The nature of the challenge can be illustrated by considering the recently developed polynomial-time algorithms for linear programming. The ellipsoid method [28] and its more recent successors such as [25] have left many in the field of mathematical programming wanting more: These algorithms run in time bounded by a polynomial in the size of the input (as measured in bits). However, even if all arithmetic operations are assumed to take unit time, the algorithms’ running times are *not* bounded by a polynomial in the more traditional size parameters, i.e., the numbers n and m of variables and constraints in the instance. Thus one might, as a compromise, look for algorithms that are simultaneously polynomial in both senses. Recently, such algorithms have been found for several important special cases, and these will be discussed.

1. THE CASE OF THE HIDDEN CONSTANTS

Robertson and Seymour’s results on the SUBGRAPH HOMEOMORPHISM problem derive from an extended series of papers they are currently writing on the topic of “graph minors.” The series starts with “Graph minors I. Excluding a forest” [41] and there is currently no end in sight, although for this column we won’t have to go any higher than “Graph Minors XVI: Wagner’s conjecture”

[50]. The papers I shall discuss have not all yet appeared (and some have not yet even been written), but one can be reasonably confident about the claims made, given the solid track record of the authors. (Seymour has a Fulkerson Prize to his credit, and has previously resolved another of the twelve open problems from [G&J], by finding a polynomial-time algorithm for TOTAL UNIMODULARITY [55].) I shall be concentrating on the algorithmic consequences of the graph minor papers, but they also abound in purely graph-theoretic breakthroughs, and several of these will be mentioned in passing.

Following Robertson and Seymour, let us allow “graphs” to contain both loops and multiple edges, i.e., to be what we normally call “multigraphs.” A graph $G = (V, E)$ contains a second graph $H = (U, F)$ as a *minor* if there is a subgraph of G that can be converted to H by a sequence of “contractions.” In a contraction, two adjacent vertices and an edge between them are contracted into a single new vertex, with all other edges previously incident on either vertex now viewed as incident on the new vertex. Note that this operation can create new multiple edges and can turn multiple edges into loops. Containing H as a minor can be viewed as a generalization of the concept of containing a homeomorphic image of H . Two graphs are said to be *homeomorphic* if they can be reduced to isomorphic graphs by a series of degree-2 contractions (contractions that involve at least one vertex of degree 2). Thus, if we restrict ourselves to graphs H with no degree-2 vertices, a graph G contains a homeomorphic image of H only if it contains H as a minor. (We may restrict ourselves to such “reduced” H without loss of generality, since every graph is homeomorphic to a unique graph of this type, obtained by applying degree-2 contractions, in any order, until no more are possible.)

As with SUBGRAPH HOMEOMORPHISM, MINOR CONTAINMENT has important applications relating to the embeddability of graphs on surfaces. Indeed, in one case it has the *same* application. Kuratowski’s Theorem, proved in 1930 [29], says that a graph is planar if and only if it contains no subgraph homeomorphic to K_5 or $K_{3,3}$. (K_5 is the complete graph on 5 vertices and $K_{3,3}$ is the complete bipartite graph with three vertices on each side). The analogous theorem is true for minors (and is almost as old): A graph is planar if and only if it contains neither of the two graphs *as a minor* [64]. In “Graph Minors VIII” [44], Robertson and Seymour resolve a long-open problem by extending this result to higher genera. They show that for any k , there is a finite class C_k of graphs such that a graph G has genus k if and only if it contains none of the graphs in C_k as a minor. A second major result, from “Graph Minors XII” [46], is that for any graph H , those graphs G that do not contain H as a minor all have bounded genus “in essence.” (The reader is referred to that paper for the complicated qualifications hinted at by the above “in essence.”)

In another similarity with SUBGRAPH HOMEOMORPHISM (and indeed with SUBGRAPH ISOMORPHISM), MINOR CONTAINMENT is NP-complete when both G and H are given as part of the instance. (Note, however,

that although the former two problems become polynomial-time solvable when both G and H are trees [6,34,40], MINOR CONTAINMENT remains NP-complete [56,58].) Until now, the most interesting (and challenging) complexity questions about SUBGRAPH HOMEOMORPHISM and MINOR CONTAINMENT have concerned the cases where H is fixed. Let us denote by SUBGRAPH HOMEOMORPHISM(H) the problem of determining whether a given graph G contains a homeomorphic image of H , and let us denote by MINOR CONTAINMENT(H) the problem of determining whether G contains H as a minor. For certain simple kinds of H , e.g. triangles or collections of independent edges, these problems can easily be seen to be polynomial-time solvable. (One need only test for the presence of a cycle or a large enough matching.) By extensions of Kuratowski's theorem [24,65], one can also test individually for homeomorphic images of the two forbidden subgraphs for planarity, $K_{3,3}$ and K_5 . (A 4-connected non-planar graph must contain a homeomorphic image of K_5 [65]; A 3-connected non-planar graph must either be K_5 or contain a homeomorphic image of $K_{3,3}$ [24]. See [4,27] for efficient algorithms based on the latter result.) For only slightly more complicated H , however, such as K_6 , the status of the problems is not at all clear. Thus the new results of Robertson and Seymour (in "Graph Minors XIII" [47]), showing that both SUBGRAPH HOMEOMORPHISM(H) and MINOR CONTAINMENT(H) are in P for all graphs H , represent a major breakthrough.

Moreover, for good measure, Robertson and Seymour also show membership in P for an important related class of problems. These are the problems FIXED-VERTEX SUBGRAPH HOMEOMORPHISM(H): Given $G = (V, E)$ and a 1-1 map f from the vertices of H to those of G , does G contain a homeomorphic image of H in which each vertex of H is identified with its image under f ? Previously this latter problem was only known to be in P for a few non-trivial graphs H : stars, with or without multiple edges (network flow), triangles [31] (with or without multiple edges [54]), and a pair of independent edges [37,54,57]. Even the case of three independent edges has remained open until now. Moreover, this and its generalization to larger sets of independent edges are important problems in their own right, for they are closely related to certain multi-commodity flow problems. In this guise they are usually referred to as " k -PATH" problems for fixed k , and restated as follows: Given a graph G and disjoint pairs (s_i, t_i) of vertices, $1 \leq i \leq k$, does G contain k vertex-disjoint paths, one connecting each pair (s_i, t_i) ? As a consequence of the Robertson and Seymour results, these problems too are in P (for all fixed k).

Perhaps surprisingly, this last result, that the k -PATH problem is in P for all k , is enough to imply all the others. The reader may verify that each of the problems in turn (MINOR CONTAINMENT(H), SUBGRAPH HOMEOMORPHISM(H), FIXED-VERTEX SUBGRAPH HOMEOMORPHISM(H), and the k -PATH problem) can be reduced to a polynomial number of calls to instances of its successor, with the fixed graphs H' (or

numbers k') involved in these calls depending only on H . If we followed this approach, however, we might obtain some relatively high-order polynomials by the time we reached $\text{MINOR CONTAINMENT}(H)$. For instance, according to my back-of-the-envelope calculations, solving $\text{MINOR CONTAINMENT}(K_5)$ might require us to solve as many as $|V|^{10}$ instances of the 15-PATH problem, and solving $\text{MINOR CONTAINMENT}(K_{10})$ might require us to solve the 105-PATH problem $|V|^{70}$ or more times.

Robertson and Seymour take a different approach, and thus avoid such high-degree polynomials. In fact, they obtain algorithms for $\text{MINOR CONTAINMENT}(H)$ that are asymptotically faster than those they obtain for the k -PATH problems ($O(|V|^3)$ for each fixed H versus $O(|V|^2|E|)$ for each fixed k). The particular fixed graph H (or the fixed number k) only affects the constant of proportionality. Unfortunately, for any instance $G = (V, E)$ that one could fit into the known universe, one would easily prefer $|V|^{70}$ to even *constant* time, if that constant had to be one of Robertson and Seymour's.

That last statement is made without knowing precisely what the constants are. In their proofs, Robertson and Seymour do not so much describe the algorithms in question as describe procedures by which the algorithms can be constructed. The computation of the constants may be viewed as the first step of such a procedure. For the most straightforward subcase of $\text{MINOR CONTAINMENT}(H)$ (that in which H is planar), a procedure for computing the relevant constant C_H is presented explicitly in "Graph Minors V" [43]. It is unlikely that anyone will ever actually perform this computation, given the sizes of the numbers involved, but I can at least sketch the procedure:

The computation takes 9 steps, but most of the damage is concentrated in just three of them. Each of these is an application of some variant on the "tower of 2's" generator $t(k)$, defined as follows: $t(1) = 2$, $t(k) = 2^{t(k-1)}$, $k > 1$. Let n be the number of vertices in H . The first application computes a value θ_1 that is at least $t(n/2)$, i.e., a tower of 2's that is $n/2$ levels high. The second application yields θ_2 as a tower of 2's that is θ_1 tall. The third application (occurring in Robertson and Seymour's step 7) yields a tower of 2's whose height is something like triply exponential in θ_2 . The final constant C_H is then somewhat larger than this. Note that this process is primitive recursive, and so does not yield numbers as large as, say, Ackermann's function would. I think most would agree, however, that they are big enough.

To explain how C_H finds its way into the algorithms, I must now take a detour back to a topic mentioned in Column 16 [Sept. 1985]. That column concerned restricted classes of graphs and the complexity results that had been proved for them. Among the classes mentioned were the *partial k -trees*, for fixed k . Partial k -trees can be inductively characterized as follows: (1) A complete graph on k vertices is a k -tree. (2) If $G = (V, E)$ is a k -tree and $V' \subseteq V$ is a set of k vertices that induces a complete subgraph in G , then the graph obtained by adding a new vertex v to V together with an edge from v to every vertex in V' is also a k -tree.

(Note that a 1-tree is simply a tree.) A graph G is a *partial k -tree* if there is some k -tree G' of which it is a subgraph, and Robertson and Seymour in this case say that G has “tree-width” bounded by k . The main result of “Graph Minors V” [43] is the following “planar obstruction theorem”:

THEOREM: *If H is a planar graph, then any graph G that does not contain H as a minor has tree-width C_H or less.*

This theorem, combined with the results and techniques of [2,3] mentioned in Column 16, is already enough to show that $\text{MINOR CONTAINMENT}(H)$ is in P for all planar graphs H , as I shall now sketch. (Membership in P was proved by slightly different means in “Graph Minors II” [42].) The needed auxiliary results are as follows. In [2], Arnborg, Corneil, and Proskurowski describe an $O(|V|^{k+1})$ algorithm for testing whether a graph G is a partial k -tree and, if so, deducing its k -tree-structure. Given a partial k -tree G along with a description of its underlying k -tree structure, Arnborg and Proskurowski show in [3] how to solve various problems in linear time (for fixed k) by generalizing standard dynamic programming algorithms that work for trees. (The hidden constant here is simply a singly exponential function of k .) It is not difficult to see that these techniques extend to $\text{MINOR CONTAINMENT}(H)$, for fixed planar H . Thus we can proceed as follows.

1. Test (in time $O(|V|^{C_H+1})$) whether G is a partial C_H -tree, generating its underlying tree structure if it is. If G is not a partial C_H -tree, then it must contain H as a minor. Otherwise,
2. Use dynamic programming and the knowledge obtained in Step 1 to determine whether G contains H as a minor (in linear time with a hidden constant that is singly exponential in C_H).

The time bound here is, of course, somewhat larger than the $O(|V|^3)$ claimed above. In “Graph Minors XIII” [47], Robertson and Seymour show how to get the C_H out of the exponent for $|V|$, and extend the algorithm so that it works for all H , not just the planar ones. In order to get C_H out of the exponent, the key idea is to settle for a weaker width test. For technical and aesthetic reasons, they also change the type of width they are measuring, replacing “tree-width” by the related concept of “branch-width,” from “Graph Minors X” [45]. (Branch-width is roughly $2/3$ times tree-width, and the “branch decompositions” upon which it is based are equally amenable to dynamic programming algorithms.) The test they settle for doesn’t compute branch-width precisely, but rather, given a proposed value w , either outputs a branch decomposition of width $3w$, or verifies that the branch-width of G is greater than w . For fixed w , the running time is $O(|V||E|)$, with the hidden constant only singly exponential in w . Together with an appropriate dynamic programming algorithm for $\text{MINOR CONTAINMENT}(H)$ for fixed H and graphs with width- $3w$ branch-decompositions, this is enough to yield the desired low-order polynomial time

bounds for MINOR CONTAINMENT(H) when H is fixed and planar. (In fact, for such H Robertson and Seymour can reduce the running time from $O(|V|^3)$ to $O(|V|^2)$.)

To extend this approach to non-planar graphs H , and also to the k -PATH problem for fixed k , requires several more ideas. I only have room to sketch them briefly. First, instead of dealing with the two types of problems directly, Robertson and Seymour solve a common generalization, the FOLIO DETERMINATION(ϵ, ξ) problem, for every fixed $\epsilon, \xi > 0$. (The definition of this problem is not exactly intuitive, requiring three auxiliary concepts and a page of text.) After showing that FOLIO DETERMINATION(ϵ, ξ) can be solved for graphs with bounded branch-width, they then examine what can be done for graphs of high branch-width. In this case, their goal is to find some vertex in G that is “irrelevant” to the problem, i.e., can be deleted without affecting the answer. Once this has been found, they have reduced the problem to one of smaller size and can proceed inductively.

The search for the irrelevant vertex is based on two observations. First, they show that if one can find in G a large clique minor (subgraph contractible to a large complete graph), one will have relatively little trouble finding an irrelevant vertex therein. Second, if one can find a large grid minor (subgraph contractible to a 2-dimensional grid) that “does not stand out among the other sections of the grid close to it in any significant way,” then any vertex in that section is irrelevant. (This is not actually *shown* in [47]; for the proof the reader will have to wait for the forthcoming “Graph Minors XIV” and “Graph Minors XV” [48,49].) Given these two observations, Robertson and Seymour complete their algorithm description by proving that at least one of the two desired structures must exist and showing how it can be found in time $O(|V||E|)$. (The hidden constants here are even worse than the C_H .)

Since only $O(|V|)$ successive irrelevant vertices need be found, we thus obtain an $O(|V|^2|E|)$ running time for solving FOLIO DETERMINATION(ϵ, ξ) when ϵ and ξ are fixed. The abovementioned running times of $O(|V|^3)$ and $O(|V|^2|E|)$ for MINOR CONTAINMENT(H) and the k -PATH problem then follow, as do times of $O(|V|^2|E|)$ for FIXED-VERTEX SUBGRAPH HOMEOMORPHISM(H) and $O(|V|^{U+2}|E|)$ for SUBGRAPH HOMEOMORPHISM(H), where $H = (U, F)$. The story is not over, however. In the next section, we shall see some surprising consequences of the fact that MINOR CONTAINMENT(H) is in P for every fixed H . Moreover, readers seeking still-open problems in this area need only turn to the directed version of SUBGRAPH HOMEOMORPHISM(H). Although the complexity of the fixed-vertex version of this problem has been determined for all fixed H [15], this has no major implications for the unfixed vertex case since the fixed-vertex version turns out to be NP-complete for all but a very narrow class of graphs. To date only a few results are known for the unfixed vertex case, including instances both of NP-completeness and of polynomial-time solvability [15].

2. NON-CONSTRUCTIVE PROOFS AND OTHER MINOR RESULTS

Many complexity theorists, your columnist included, seem to have built their technical careers on the assumption that $P \neq NP$. To those who point out the implicit danger in this, we have a standard reply: A proof that $P = NP$ would not be a disaster for us; it would simply turn all our NP-completeness proofs into algorithms! Unfortunately, there is a hole in this reasoning. What if there were a proof that $P = NP$, but the proof was *non-constructive*? Until now, most of us would have thought such a possibility ludicrous. After all, how could one possibly prove that a polynomial-time algorithm exists without exhibiting it?

As a consequence of a result that will appear in “Graph Minors XVI” [50], just such a proof technique now exists. That result is the proof of an unpublished conjecture about graph minors due to K. Wagner: Suppose F is a class of graphs that is closed under minors, i.e., is such that if G is in F and H is contained as a minor in G , then H is in F . Then there exists a finite set $\{H_1, H_2, \dots, H_k\}$ of graphs such that a general graph G is in F if and only if it contains no minor isomorphic to any of the H_i , $1 \leq i \leq k$. As a consequence of the results mentioned in the previous section, Wagner’s conjecture implies that any class F of graphs closed under minors can be recognized in polynomial time, indeed in time $O(|V|^3)$ (albeit with those horrible constants of proportionality). One cannot actually exhibit the polynomial-time algorithm, however, unless one knows the graphs H_i , $1 \leq i \leq k$, and, unfortunately, Robertson and Seymour’s proof of Wagner’s conjecture is highly non-constructive. (Friedman, Robertson, and Seymour have shown that the conjecture is in fact independent of very powerful theories of arithmetic [17].)

Thus, we have a method for proving, non-constructively, that a problem is in P . Moreover, this method seems to be widely applicable (although fortunately not, so far, to any known NP-complete problems). As a result of investigations by Fellows et al. [1,12,13], membership in P has already been proved for several previously-open problems. For instance, consider the problem of determining whether a given graph can be embedded in 3-space in such a way that no two of its cycles are linked (as in links of a chain). A few instances of this problem have been solved. For instance, Conway and Gordon [7] have shown that K_6 cannot be embedded without some pair of cycles being linked. However, it is not immediately clear that the general problem is even decidable. There exists a highly exponential algorithm due to H. Schubert [53] to test whether two given cycles are linked in a given embedding, but the exponentiality of this would only be compounded if one had to consider all pairs of cycles, and the number of topologically distinct embeddings is not even finite. As observed in [13], however, the class of graphs that can be embedded with no linked cycles is clearly closed under minors, so this problem is in P . A similar result holds for the problem of determining whether a given graph can be embedded in 3-space in such a way that none of its cycles is knotted [13].

Retreating from 3-dimensional embeddings to embeddings on surfaces, consider the “disk dimension” problem: Given a graph $G = (V, E)$, together with integers g and d , can G be embedded in a surface of genus g , from which d open disks have been removed, in such a way that each vertex of G is on the boundary of one of the deleted disks? This problem is NP-complete even if g is fixed at 0, in which case the problem simply asks whether there is a planar embedding of G and a set of d faces from that embedding such that each vertex of G is on the boundary of one of the d faces [11]. When d is fixed at ∞ , the problem simply asks whether G has genus g or less, and is one of the remaining five “Open Problems” from [G&J]. Now consider what happens when both g and d are fixed. For $g = 0$ and fixed d , the problem is solvable in linear time by a result of Bienstock and Monma [5]. For $d = \infty$ and fixed g , the problem is solvable in polynomial time by a result of Filotti, Miller, and Reif [14] (with the polynomial’s degree being linear in g). We can now say, however, that the problem is solvable in time $O(|V|^3)$ for all fixed g and d , since for any such g and d , the set of graphs with embeddings of the specified form is closed under minors [13]. Not only does this resolve the previously open cases when $g > 0$ and $d < \infty$, but it also provides an asymptotic speed-up in the running times for the $d = \infty$ cases, all non-constructively.

For a less obviously topological example, consider the GATE MATRIX LAYOUT problem, also known as the problem of “multiple folding” for Programmable Logic Arrays (PLA’s). This problem asks whether a given set of nets can be rearranged (“folded”) so that it occupies just k tracks in a logic array. More formally, given an $n \times m$ Boolean matrix M and an integer k , can the columns of M be permuted so that, if in each row we change to 1 every 0 lying between the leftmost and rightmost 1’s, then no column contains more than k 1’s? This problem was proved NP-complete in [26,33], and the case $k = 2$ was shown to be in P in [8]. The existence of $O(n^2)$ algorithms for all fixed $k > 2$ is shown non-constructively by Fellows and Langston in [12], by reducing GATE MATRIX LAYOUT to a graph problem (about edge-permutations) whose yes-instances are closed under minors for each fixed k .

As a final and instructive example from the many in [1,12,13], consider the problem of determining whether a graph G has a vertex cover of size k or less, where k is fixed. This problem can clearly be solved in time $O(|V|^k |E|)$ and so is in P. Since the set of graphs with such vertex covers is closed under minors, however, we can use our non-constructive techniques to get an asymptotically much faster $O(|V|^3)$ algorithm [13]. Moreover, we can be a bit more sophisticated than this, in light of the following key observations. The first is that no graph with a vertex cover of size k or less can contain the $(2k+1)$ -vertex cycle C_{2k+1} as a minor. The second key observation is that C_{2k+1} is planar, so that the planar obstruction theorem applies, and the set of graphs with vertex covers of size k or less has bounded tree-width. The final observation is that vertex cover can be solved in linear time for graphs of bounded tree-width (provided the

underlying tree model is known) [3]. Thus we can use the techniques of the previous section to derive an improved, $O(|V|^2)$ algorithm, and do so *constructively*.

The above example is instructive because it shows that one needn't necessarily know the full obstruction set $\{H_1, \dots, H_k\}$ in order to turn a non-constructive algorithm into a constructive one. It is even more instructive because of what happened when the authors of [13] informed Manuel Blum of their result. Blum congratulated them on their ingenuity, but wanted to think some more about the problem. Shortly thereafter, he and student Steve Rudich discovered that if all one wants is an $O(|V|^2)$ algorithm, there is a much more straightforward approach, one that replaces Robertson and Seymour's horrible constants by a simple 3^k and is in fact $O(|E|)$ [30]. (I leave this as an exercise to the reader, but, as a hint, the first step involves finding a maximal matching.)

Indeed, it is not clear that *any* of the results of this section are inherently non-constructive, or even that the Robertson-Seymour constants are unavoidable. Nevertheless, the non-constructive approach does appear to be a useful method for gaining quick insights into what problems *are* polynomial-time solvable, and it will be interesting to observe its impact on the field as it becomes more widely known.

3. STRONGLY POLYNOMIAL TIME

Several key choices were made when our current definition of “polynomial time” was formalized. First, it was decided to restrict attention to computational models that reflect the digital nature of current-day computers, in which memory consists of bits, and in which the time taken by a step is polynomially related to the number of digits manipulated by that step. Second, and as a consequence of the first choice, it was required that instances be describable by finite bit strings. Finally, it was determined that the *size* of an instance should reflect the length of that string (to within a fixed polynomial). These choices led to a concept that was robust both with respect to machine model and to input representation. They also, however, excluded certain popular ways of looking at computation.

One of these is based on what might be called the “real number (unit cost) RAM.” This is a random access machine in which each register holds a real number and the basic operations are memory transfers, comparisons, and (real) addition, subtraction, multiplication, and division, with each operation assumed to take unit time, irrespective of the contents of the registers involved. An input is a sequence I of real numbers that are loaded into the first $|I|$ registers, and the input size is simply taken to be $n = |I|$. Some of these real numbers may actually be integers (for instance those specifying number of variables and constraints in a given linear programming instance), and algorithms can use this information to construct register addresses, but we assume that the machine halts in an error state if ever we try to load a number into a register with a non-integral address.

Although this is in many senses an unrealistic model, it still has its uses. Among other things, it allows one to abstract away such issues as numerical stability and round-off error. It is the model, for instance, in which much of the work on the complexity of matrix multiplication has taken place. It also reflects the way in which many people continue to think about computation, and perhaps accounts for the dissatisfaction felt by some members of the mathematical programming community with the concept of “polynomial-time.” This is because key algorithms that run in polynomial time in our sense do not take a polynomially-bounded number of steps on a real number RAM.

For instance, consider algorithms based on “scaling” techniques. Scaling was introduced by Edmonds and Karp [10] as a technique for solving the “minimum cost flow” problem. In this problem, we are given a directed graph $G = (V, A)$, a capacity $c(a)$ and a cost $p(a)$ for each arc, and two specified vertices s and t . The goal is to find, among all maximum flows from s to t , one of minimum cost. (Recall that a “flow” is an assignment of a flow $f(a)$ to each arc a such that no arc’s flow exceeds its capacity and such that, for every vertex v in $V - \{s, t\}$, the sum of the flows going into v equals the sum of the flows going out. The *size* of the flow is the sum of the flows going into t , and its cost is the sum of $f(a)p(a)$ over all arcs a .)

Suppose the capacities are integral, C is the largest capacity, and $p = \lceil \log C \rceil$. Edmonds and Karp proposed solving the minimum cost flow problem in an iterative fashion, with the i th iteration yielding the solution for the “scaled-down” version of the problem in which each capacity $c(a)$ is replaced by, say, $\lfloor c(a)/2^{p-i} \rfloor$. Each iteration thus in a sense adds one bit of accuracy to the solution. In the first iteration the capacities are all 0 or 1, which makes the problem relatively straightforward. Thereafter the solution from the previous iteration serves as a sufficiently good starting point to allow the attainment of the next bit of accuracy to take the same time as it took to get the first. The resultant running time thus is proportional to $\log C$, even in the unit-cost model. (Assuming G is a dense graph, the precise Edmonds-Karp running time bound was $O(n^4 \log C)$.)

Such an algorithm meets the requirements of “polynomial time,” since the input length must reflect the binary representations of the input numbers, and so must exceed $\log C$. However, in the real number RAM model, $\log C$ does *not* contribute to the input size. Thus, even if one is willing to restrict attention to instances in which the capacities are integral, the real number RAM running time is not only not polynomial, it is not bounded at all in terms of the input size. A similar objection can be made about the recently developed “polynomial time” algorithms for linear programming, such as the ellipsoid method [28]. Although these do not use scaling techniques per se, they require a worst-case number of iterations that is a multiple of the size of the largest input number, and hence are not polynomial in the real number RAM model. (As an extreme example, Traub and Wozniakowski in [63] present a collection of 2×2 linear programming instances for which the running time of the ellipsoid method is

unbounded.)

To what extent should we take these objections seriously? One school of thought would say “not at all.” Numbers large enough for $\log C$ to make a difference don’t occur in real problems. Even if they did, an algorithm whose real number RAM time $T(n)$ did not depend on $\log C$, would end up with running time $T(n)\log C$ on a real computer, just because of the time needed to perform extended precision arithmetic. Thus the dependence of the number of iterations on $\log C$ merely increases the *exponent* of an already-present $\log C$ factor in the overall running time, and cannot be viewed as a fundamental distinction. What some may really be objecting to is not the concept of “polynomial time,” but simply the fact that the Edmonds-Karp algorithm (and the ellipsoid method) are *slow*, a fact that can more readily be attributed to the exponent on the n in their running time bounds than to the presence of the $\log C$. Reducing that exponent is much more important than getting rid of the $\log C$. (Indeed, as our sophistication at using scaling increases, scaling algorithms are beginning to take over on many problems that were originally solved without it [19,20].)

There are, however, some reasons for taking the objection seriously. Indeed Edmonds and Karp did so in [10], where after presenting their algorithm they stated as a “challenging open problem” the question of whether there was an algorithm for minimum cost flow that ran in polynomial time on a real number RAM. Such a question is at least interesting from a theoretical viewpoint, as an answer could help identify the precise source of a problem’s complexity. There is also a practical consideration, ignored in the previous paragraph. This is the existence of floating point numbers. Although the assumption above was that the input numbers would be written in binary notation, they could be, and in practice often are, input in floating point representation. In this case it might well be that $\log C$ is *exponential* in the size of the input, and so an algorithm whose running time depended on $\log C$ might not even run in “polynomial time” in the standard sense. (This may be another red herring, however. Given the bounded-precision nature of computation with floating point numbers in real computers, a bounded number of scaling iterations would normally suffice to obtain as many significant digits as the arithmetic allows.)

At any rate, researchers have recently begun to take the issue seriously. Recognizing that the real number RAM has certain unrealistic properties, however, they have looked for algorithms that are more than just polynomial-time under that model. Instead, the algorithms must be what has now come to be called “strongly polynomial.” (This term seems preferable to the alternative and more judgmental “genuinely polynomial,” proposed by Megiddo in [35].) The basic idea is that a strongly polynomial algorithm should run in polynomial time both in the standard model and on a real number RAM, although researchers differ as to the precise details of the definition.

According to Tardos [61,62], a strongly polynomial algorithm must be designed to run on a real number RAM under arbitrary real input, must take

polynomial time in that model and, if the input numbers are all rational, the size of all computed numbers must be bounded by a polynomial in the number of registers used by the input and the maximum size of an input number. (The size of a rational number is the sum of the lengths of its numerator and denominator, under the assumption that RAM operations do not automatically remove common divisors.) Grötschel, Lovász, and Schrijver [23] retain the bound on number size, but require only that the algorithm work for rational input, with the numerator and denominator of each rational entered into separate registers. (In their unit-cost RAM model, registers must contain integers and the machine halts if a division results in a non-integral value.) In both definitions, the bound on number sizes insures that if the algorithm were converted to run on a standard model of computation, it would run in ordinary polynomial time.

Note that the bound on number sizes may be important for more reasons than the simple desire to account for the true cost of multiple precision arithmetic. A real number RAM can, in a polynomial number of steps, construct numbers whose binary representations have undergone an exponential blow-up, and then continue to operate on them at unit cost per operation. One might well worry that this fact could be exploited in non-standard ways to gain extra computing power. For instance, it only takes a little more power than we have already provided for a real number RAM to be able to solve SATISFIABILITY in polynomial time. The addition of a floor or ceiling operator will suffice [52] and it is not clear whether even that is necessary.

This raises an important side issue, as almost all the algorithms I shall be discussing use at least one of these operators, and, as Stockmeyer has shown [59,60], neither $\text{floor}(x)$ nor $\text{ceiling}(x)$ can be simulated in fewer than $\log x / \log \log x$ steps using just real addition, subtraction, multiplication, and division. Thus the mere *use* of one of these operators may be enough to prevent strong polynomiality, unless the size of the number to which it is applied is bounded by a polynomial in the number n of input registers. In the results I shall cite, care has normally been taken to insure that this last property holds (or else, as in [38], the model has explicitly been expanded to allow the needed operator at unit cost.) I shall have a bit more to say about this issue at the end of this section.

For many problems, the existence of strongly polynomial algorithms is nothing new. The numbers occurring in some problems are by definition polynomially bounded, so that *any* polynomial-time algorithm for them is strongly polynomial. For other problems, the original solvers may have found it unnecessary to resort to a sophisticated technique like scaling. Moreover, there is by now a long history of researchers who approached strong polynomiality from the other side, showing that algorithms known to be polynomial on real number RAM's could be made polynomial in the complexity theory sense. (One of the first examples of this is in a 1967 paper by Edmonds [9]. In this paper, he showed how to modify the standard algorithm for Gaussian elimination, which can suffer from exponential blow-ups in the size of intermediate results if care is not

taken, so that such blow-ups are avoided. A variety of other results along these lines are presented in [23].) Nevertheless, there remain many polynomial-time solvable problems like minimum cost flow for which, at least until recently, no strongly polynomial-time algorithms were known. For some, such as the problem of finding the greatest common divisor of two integers, strongly polynomial time seems unobtainable. For others, however, the picture is much brighter, in large part due to recent work of Eva Tardos [16,21,61,62], who has developed two separate algorithmic approaches for freeing running time from dependence on the size of the input numbers.

The first technique is based on a “Diophantine approximation” algorithm derived from Lovasz’s basis reduction algorithm [32]. Stated in terms of the minimum cost flow problem, this technique uses Diophantine approximation to round the costs to values that are polynomially bounded in the number of vertices, in such a way that the resulting instance has the same optimal solutions (although perhaps not the same optimal value) as the original. One then can use a variant on the Edmonds-Karp algorithm that scales the costs rather than the capacities (for instance, see [51]), and the $\log C$ factor (or in this case $\log P$, where P is the maximum arc cost) will be replaced by $\log n$.

This technique turns out to have a variety of applications. (One example is the problem of finding a maximum weight clique in a perfect graph [16].) The technique has a serious drawback, however, The Diophantine approximation algorithm, as used here, only satisfies an $O(n^8 \log n)$ running time bound. Thus the approach cannot be considered practical. (It also apparently does not apply to irrational input [16], and so technically does not quite fulfill Tardos’s definition of strongly polynomial given above.)

For the case of minimum cost flow, Tardos quickly replaced Diophantine approximation by more efficient techniques [61]. As with ordinary scaling, a series of rounded subproblems is solved. This time, however, each iteration reduces the number of constraints in the problem (rather than increasing the number of bits of accuracy in the solution), so the number of iterations will be polynomially bounded in n . For instance, in the minimum cost flow problem, each iteration might allow us to set one more capacity constraint to ∞ . (Once they are all infinite, the problem reduces to one of finding a shortest path.) In the short time since the algorithm of [61] appeared, there has been a rapid series of improvements on it [18,38,21]. (Interestingly enough, it is shown in [38] that a minor modification to the original Edmonds-Karp algorithm renders *it* strongly polynomial.) The current champion, due to Galil and Tardos [21] has a claimed running time bound (for dense graphs) of $O(n^4 \log n)$. This is comparable to the $O(n^4 \log C)$ bound for the original Edmonds-Karp algorithm, although not to the $O(n^3(\log n + \log P))$ bound for the current champion among scaling algorithms, due to Goldberg and Tarjan [22].

Partial results have also been obtained on the major remaining open question in this field, whether or not there exists a strongly polynomial-time algorithm for

the general linear programming problem. This problem can be stated as follows: Given an $m \times n$ matrix A and vectors b and c of appropriate lengths, find a vector x that minimizes the dot product cx , subject to $Ax = b$ and $x \geq 0$. Among the non-strongly polynomial-time algorithms for this problem, the new polynomial-time algorithm due to Karmarkar [25] seems to be fast enough (when implemented properly) to defuse practical objections to the lack of “strength” in its time bound. Nevertheless, the possible existence of a strongly polynomial algorithm remains intriguing from a theoretical point of view.

Several special cases have been resolved. If the number of variables takes on a fixed value k , the problem can be solved in strongly polynomial time, as in this case there are only $O(m^k)$ basic solutions, and each of these can be tested using Gaussian elimination. (Faster, linear-time algorithms for fixed k are presented by Megiddo in [36].) In [35], Megiddo gives a strongly polynomial-time algorithm for testing feasibility in the case where no row of A contains more than two non-zero entries. In [62], Tardos gives a strongly polynomial-time algorithm for the case where all entries in A are polynomially bounded in m and n but entries in both b and c can be arbitrary. (This generalizes min cost flow and includes a variety of other combinatorially-oriented linear programming problems. A dual version of the algorithm is presented in [39].)

Can the general problem be solved in strongly polynomial time? Many think not. Traub and Wozniakowski have conjectured that the problem cannot even be solved in polynomial time on an unrestricted real number RAM [63], in which *no* bound is imposed on the sizes of computed numbers. The question remains open even if we add the floor operator to the model and thus can solve SATISFIABILITY in polynomial time with it. This is because no known polynomial transformation from LINEAR PROGRAMMING to SATISFIABILITY is strongly polynomial. In the strange world of such augmented real number RAMs, it may be that there are NP-complete problems that are *not* polynomial-time equivalent. I leave this possibility, and the above open problems, for the interested reader to ponder.

REFERENCES

1. J. ABELLO, D. J. BROWN, M. R. FELLOWS, AND M. A. LANGSTON, Some applications of nonconstructive proofs for polynomial-time decidability, in preparation (1987).
2. S. ARNBORG, D. G. CORNEIL, AND A. PROSKUROWSKI, “Complexity of finding embeddings in a k -tree,” Report No. TRITA-NA-8407, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden, 1984.
3. S. ARNBORG AND A. PROSKUROWSKI, “Linear time algorithms for NP-hard problems on graphs embedded in k -trees,” Report No. TRITA-NA-8404, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden, 1984.
4. T. ASANO, An approach to the subgraph homeomorphism problem, *Theor. Comput. Sci.* **38** (1985), 249-267.

5. D. BIENSTOCK AND C. L. MONMA, On the complexity of covering vertices by faces in a planar graph, manuscript (1986).
6. M. J. CHUNG, $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees, *J. Algorithms* **8** (1987), 106-112.
7. J. CONWAY AND C. GORDON, Knots and links in spacial graphs, *J. Graph Theory* **7** (1983), 445-453.
8. N. DEO, M. KRISHNAMOORTHY, AND M. LANGSTON, Exact and approximate solutions to the gate matrix layout problem, *IEEE Trans. Computer-Aided Design* **6** (1987), 79-84.
9. J. EDMONDS, Systems of distinct representatives in linear algebra, *J. Res. Nat. Bur. Standards* **B71** (1967), 241-245.
10. J. EDMONDS AND R. M. KARP, Theoretical improvements in algorithmic efficiency for network flow problems, *J. Assoc. Comput. Mach.* **19** (1972), 248-264.
11. M. R. FELLOWS, F. HICKLING, AND M. SYSLO, A topological parameterization and hard graph problems, manuscript (December 1985).
12. M. R. FELLOWS AND M. A. LANGSTON, Nonconstructive advances in polynomial-time complexity, *Inform. Process. Lett.*, to appear.
13. M. R. FELLOWS AND M. A. LANGSTON, Nonconstructive tools in proving polynomial-time decidability, manuscript (December 1986).
14. I. S. FILOTTI, G. L. MILLER, AND J. REIF, On determining the genus of a graph in $O(V^{O(g)})$ steps, in "Proceedings 11th Ann. ACM Symp. on Theory of Computing," pp. 27-37, Association for Computing Machinery, New York, 1979.
15. S. FORTUNE, J. HOPCROFT, AND J. WYLLIE, The directed subgraph homeomorphism problem, *Theor. Comput. Sci.* **10** (1980), 111-121.
16. A. FRANK AND E. TARDOS, An application of simultaneous approximation in combinatorial optimization, in "Proceedings 26th Ann. Symp. on Foundations of Computer Science," pp. 459-463, IEEE Computer Society, Los Angeles, 1985.
17. H. FRIEDMAN, N. ROBERTSON, AND P. D. SEYMOUR, The metamathematics of the graph minor theorem, in "Applications of Logic to Combinatorics," AMS Contemporary Mathematics Series, American Math. Soc., Providence, R.I., (to appear).
18. S. FUJISHIGE, A capacity-rounding algorithm for the minimum-cost circulation problem: A dual framework of the Tardos algorithm, *Math. Programming* **35** (1986), 298-308.
19. H. N. GABOW, Scaling algorithms for network problems, in "Proceedings 24th Ann. Symp. on Foundations of Computer Science," pp. 248-257, IEEE Computer Society, Los Angeles, 1983.
20. H. N. GABOW, J. L. BENTLEY, AND R. E. TARJAN, Scaling and related techniques for geometric problems, in "Proceedings 16th Ann. ACM Symp. on Theory of Computing," pp. 135-143, Association for Computing Machinery, New York, 1984.
21. Z. GALIL AND E. TARDOS, An $O(n^2(m + n \log n) \log n)$ min-cost flow algorithm, in "Proceedings 27th Ann. Symp. on Foundations of Computer Science," pp. 1-9, IEEE Computer Society, Los Angeles, 1986.
22. A. V. GOLDBERG AND R. E. TARJAN, Solving minimum-cost flow problems by successive approximation, in "Proceedings 19th Ann. ACM Symp. on Theory of Computing," Association for Computing Machinery, New York, 1987.
23. M. GRÖTSCHHEL, L. LOVÁSZ, AND A. SCHRIJVER, "Geometric Algorithms and Combinatorial Optimization," Springer, Berlin, in press.
24. D. W. HALL, A note on primitive skew curves, *Bull. Amer. Math. Soc.* **49** (1943), 935-937.
25. N. KARMARKAR, A new polynomial-time algorithm for linear programming, *Combinatorica* **4** (1984), 373-395.
26. T. KASHIWABARA AND T. FUJISAWA, An NP-complete problem on interval graphs, in "IEEE Symp. on Circuits and Systems," pp. 82-83, IEEE, 1979.
27. P. A. KASCHUBE, "Matroids, Recursion, and the Subgraph Homeomorphism Problem," PhD Dissertation, Department of Mathematics, University of California, San Diego, Calif., 1984.

28. L. G. KHACHIYAN, A polynomial algorithm in linear programming, *Dokl. Akad. Nauk. SSSR* **244** (1979), 1093-1096 (in Russian). English translation in *Soviet Math. Dokl.* **20** (1979), 191-194.
29. C. KURATOWSKI, Sur le problème des courbes gauches en topologie, *Fund. Math.* **15** (1930), 271-283.
30. M. A. LANGSTON, private communication (1987).
31. A. S. LAPAUGH AND R. L. RIVEST, The subgraph homeomorphism problem, *J. Comput. System Sci.* **20** (1980), 133-149.
32. A. K. LENSTRA, H. W. LENSTRA, JR, AND L. LOVÁSZ, Factoring polynomials with rational coefficients, *Math. Ann.* **261** (1982), 515-534.
33. M. LUBY, U. VAZIRANI, V. VAZIRANI, AND A. SANGIOVANNI-VINCENTELLI, Some theoretical results on the optimal PLA folding problem, in "Proc. Int. Conf. on Circuits and Computers," New York, 1982.
34. D. W. MATULA, Subtree isomorphism in $O(n^{5/2})$, *Ann. Discrete Math.* **2** (1978), 91-106.
35. N. MEGIDDO, Towards a genuinely polynomial algorithm for linear programming, *SIAM J. Comput.* **12** (1983), 347-353.
36. N. MEGIDDO, Linear programming in linear time when the dimension is fixed, *J. Assoc. Comput. Mach.* **31** (1984), 114-127.
37. T. OHTSUKI, The two disjoint path problem and wire routing design, "Graph Theory and Algorithms," pp. 257-267., Proc. of 17th Symp. of Research Institute of Electrical Communication, Tohoku University, Sendai, Japan, 1980.
38. J. B. ORLIN, "Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem," Report No. Technical Report No. 1615-84, MIT Sloan School of Management, Cambridge, Mass., December 1984.
39. J. B. ORLIN, A dual version of Tardos's algorithm for linear programming, *Operations Res. Lett.* **5** (1986), 221-226.
40. S. W. REYNER, An analysis of a good algorithm for the subtree problem, *SIAM J. Comput.* **6** (1977), 730-732.
41. N. ROBERTSON AND P. D. SEYMOUR, Graph minors I. Excluding a forest, *J. Combinatorial Theory Ser. B* **35** (1983), 39-61.
42. N. ROBERTSON AND P. D. SEYMOUR, Graph minors II. Algorithmic aspects of tree-width, *J. Algorithms* **7** (1986), 309-322.
43. N. ROBERTSON AND P. D. SEYMOUR, Graph minors V. Excluding a planar graph, *J. Combinatorial Theory Ser. B* **41** (1986), 92-114.
44. N. ROBERTSON AND P. D. SEYMOUR, Graph minors VIII. A Kuratowski theorem for general surfaces, manuscript (1983).
45. N. ROBERTSON AND P. D. SEYMOUR, Graph minors X. Obstructions to tree-decompositions, manuscript (1984).
46. N. ROBERTSON AND P. D. SEYMOUR, Graph minors XII. Excluding a non-planar graph, manuscript (June 1986).
47. N. ROBERTSON AND P. D. SEYMOUR, Graph minors XIII. The disjoint paths problem, manuscript (September 1986).
48. N. ROBERTSON AND P. D. SEYMOUR, Graph minors XIV. Taming a vortex, in preparation (1987).
49. N. ROBERTSON AND P. D. SEYMOUR, Graph minors XV. Surface hypergraphs, in preparation (1987).
50. N. ROBERTSON AND P. D. SEYMOUR, Graph minors XVI. Wagner's conjecture, in preparation (1987).
51. H. RÖCK, Scaling techniques for minimum cost network flows, in "Discrete Structures and Algorithms," (V. Page, Ed.), Carl Hansen, Munich, 1980.
52. A. SCHÖNHAGE, On the power of random access machines, in "Automata, Languages, and Programming," pp. 520-529, Lecture Notes in Computer Science, Vol. 71, Springer, Berlin, 1979.

53. H. SCHUBERT, Bestimmung der Primfaktorzerlegung von Verkettungen, *Math. Z.* **76** (1961), 116-148.
54. P. D. SEYMOUR, Disjoint paths in graphs, *Discrete Math.* **29** (1980), 293-309.
55. P. D. SEYMOUR, Decomposition of regular matroids, *J. Combinatorial Theory Ser. B* **28** (1980), 305-359.
56. P. D. SEYMOUR, private communication (1987).
57. Y. SHILOACH, A polynomial solution to the undirected two paths problem, *J. Assoc. Comput. Mach.* **27** (1980), 445-456.
58. P. W. SHOR, private communication (1987).
59. L. J. STOCKMEYER, "Arithmetic versus Boolean operations in idealized register machines," Report No. RC5954, IBM Research, Yorktown Heights, N.Y., 1976.
60. L. J. STOCKMEYER, private communication (1987).
61. E. TARDOS, A strongly polynomial minimum cost circulation problem, *Combinatorica* **5** (1985), 247-255.
62. E. TARDOS, A strongly polynomial algorithm to solve combinatorial linear programs, *Operations Res.* **34** (1986), 250-256.
63. J. F. TRAUB AND H. WOZNIAKOWSKI, Complexity of linear programming, *Operations Res. Lett.* **1** (1982), 59-62.
64. K. WAGNER, Über eine Eigenschaft der ebenen Komplexe, *Math. Ann.* **114** (1937), 570-590.
65. K. WAGNER, Bemerkung zu Hadwigers Vermutung, *Math. Ann.* **141** (1960), 433-451.