Algorithmic Implications of the Graph Minor Theorem[*]

Daniel Bienstock
Dept. of Civil Engineering
Columbia University
New York, NY 10027

Michael A. Langston
Dept. of Computer Science
University of Tennessee
Knoxville, TN 37996

A chapter prepared for the
Handbook of Operations Research and Management Science:
Volume on Networks and Distribution

[*]Page proofs and related correspondence should be sent to M. A. Langston.

Algorithmic Implications of the Graph Minor Theorem

Daniel Bienstock and Michael A. Langston

1 Introduction

In the course of roughly the last ten years, Neil Robertson and Paul Seymour have led the way in developing a vast body of work in graph theory. One of their most celebrated results is a proof of an old and intractable conjecture in graph theory, previously known as Wagner's Conjecture, and now known as the Graph Minor Theorem. The purpose of this chapter is to describe some of the algorithmic ramifications of this powerful theorem and its consequences.

Significantly, many of the tools used in the proof of the Graph Minor Theorem can be applied to a very broad class of algorithmic problems. For example, Robertson and Seymour have obtained a relatively simple polynomial-time algorithm for the disjoint paths problem (described in detail later), a task that had eluded researchers for many years. Other applications include combinatorial problems from several domains, including network routing, utilization and design. Indeed, it is a critical measure of the value of the Graph Minor Theorem that so many applications are already known for it. Only the tip of the iceberg seems to have surfaced thus far. Many more important applications are being reported even as we write this.

The entire graph minors project is immense, containing almost 20 papers whose total length may exceed 600 pages. Thus we focus here primarily on some of the main algorithmic ideas, although a brief sketch of related issues is necessary. We assume the reader is familiar with basic concepts in graph theory [BM]. Except where noted otherwise, all graphs we consider are finite, simple and undirected.

2 A Brief Outline of the Graph Minors Project

Three of the key notions employed are *minors*, *obstructions* and *well-quasi-orders*, and we examine them in that order.

Minors. Given graphs H and G, we say that H is a minor of G (or that G contains H as a minor) if a graph isomorphic to H can be obtained by removing from G some vertices and edges and then contracting some edges in the resulting subgraph. Thus every graph is a minor of itself, and the single vertex graph is a minor of every nonempty graph. For a slightly less trivial example, see Figure 1, which illustrates that the wheel with four spokes (W_4) is a minor of the binary three-cube (Q_3).

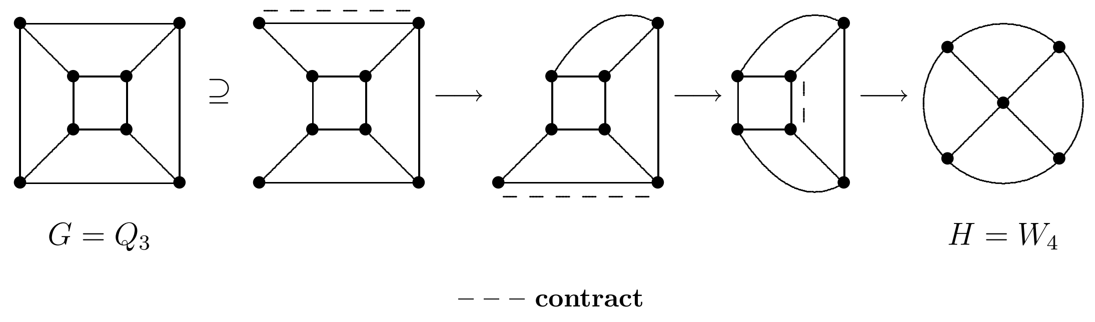

Figure 1

A concept related to minor containment is *topological* containment. We say that a graph G is a *subdivision* of a graph H if G may be obtained by subdividing edges of H (an edge $\{u, v\}$ is subdivided by replacing $\{u, v\}$ with a path with ends u and v and whose internal vertices are new). We say that G topologically contains H if G contains a subgraph that is a subdivision of H. Thus topological containment is a special case of minor containment (we can only contract edges at least one of whose endpoints have degree two). Observe that W_4 is not topologically contained in Q_3.

Topological containment has been heavily studied by graph theorists. Perhaps the most famous theorem in this regard is Kuratowski's [Ku]: a graph is planar if and only if it does not topologically contain K_5 or $K_{3,3}$. We note here that these two graphs are *minimally* nonplanar, that is, every graph topologically (and properly) contained in either of them is planar.

For the sake of exposition, let us view this theorem in terms of minors. Clearly, every minor of a planar graph is also planar. That is, the class of planar graphs is *closed* in the minor order. Consequently, no planar graph contains a K_5 or $K_{3,3}$ minor. Moreover, every proper minor of either of these two graphs is planar, and neither one contains the other as a minor. But can there be other minimal excluded minors? The answer is negative, for if G were such a purported graph, then G would be nonplanar, and thus it would contain, topologically (and therefore as a minor), either K_5 or $K_{3,3}$. In summary, a graph is planar if and only if it does not contain a K_5 or $K_{3,3}$ minor.

We note in passing two other points of interest concerning planarity. One is that planarity can be tested in polynomial time (in fact in linear time [HT]). The other is that a problem of natural interest is to try to extend Kuratowski's theorem to higher surfaces. (A surface is obtained from the sphere by "gluing" onto it a finite number of "handles" and/or "crosscaps" [Ma].) A graph can be embedded on a given surface if it can be drawn on that surface without crossings. Given a surface S, can we characterize those graphs embeddable in S by a finite list of excluded graphs in the topological order? In the 1930s, Erdös conjectured that the answer is yes. No results were obtained on this conjecture until much later, first with a proof for the case when S is the projective plane [Ar], and then for the case when S is non-orientable [GHW].

Obstructions. Kuratowski's theorem may be regarded as a characterization of planarity by means of excluded graphs, henceforth termed obstructions. Characterizations of this nature abound in combinatorial mathematics and optimization. Some familiar examples include the max-flow min-cut theorem, Seymour's description of the clutters with the max-flow min-cut property and Farkas' lemma. In all these, the presence of a desired feature is characterized by the absence of some obstruction. Besides being aesthetically pleasing, theorems of this type are desirable because such characterizations provide evidence of "tractability" of the problem at hand, giving hope for a polynomial-time test for the desired feature.

The graph minors project contains several such theorems, many of which turn out to be at the heart of both proofs and applications. As expected, there are algorithmic aspects to these theorems. As a very introductory example, one can test in polynomial time if any graph can be embedded on the torus.

Well-quasi-orders. A class Q, equipped with a transitive and reflexive relation $\leq$, is called a *quasi-order*. For example, the class of all graphs is a quasi-order, where $\leq$ is the minor containment relation. There has been some confusion as to the difference between quasi-orders and partial-orders; it suffices to look at graph minors to understand this difference. It is convenient to regard isomorphic copies of a given graph as different entities and so, for distinct graphs G and H, we can simultaneously have $G \leq H$ and $H \leq G$. Thus $\leq$ is not a partial-order, because the minor relation is not anti-symmetric.

A quasi-order with class Q and relation $\leq$ is a *well-quasi-order* if (1) for every infinite sequence $a_1, a_2, \ldots$ of elements of Q, there exist integers $1 \leq i < j$ such that $a_i \leq a_j$, and (2) there exists no infinite descending chain $b_1 > b_2 > \ldots$ of distinct elements of Q.

Example 2.0.1 Let Q be the set of all closed intervals of the real line, with nonnegative integer endpoints; i.e., $Q = \{[a, b] : 0 \leq a \leq b \text{ and } a, b \text{ integer}\}$. For $I = [a, b]$, $J = [c, d]$, we write $I \leq J$ if either J contains I and $a = c$ or if I and J are disjoint with $b < c$. Clearly $(Q, \leq)$ is a quasi-order. To see that it is a well-quasi-order, note first that (2) is satisfied. To prove that (1) holds, consider any sequence $S = I_1, I_2, \ldots$ with the property that there do not exist integers $1 \leq i < j$ such that $I_i \leq I_j$. Let $I_1 = [a, b]$. Clearly S contains no members of the form $[c, d]$ with $b < c$. It is also clear that S contains finitely many members of the form $[c, d]$ with $d \leq b$. Finally, for each integer x with $a \leq x \leq b$, S contains finitely many members of the form $[x, y]$. For if $I_{j(x)} = [x, y^*]$ is the first such member, then all remaining members of the form $[x, y]$ satisfy $y < y^*$. We conclude that S is finite, as desired.

We use this example to illustrate that, given a well-quasi-order, we in general do not have an absolute bound on the size of an antichain (set of pairwise incomparable elements), we merely know it must be finite. In particular, "finite" does not necessarily imply "small." A result of relevance is Kruskal's proof [Kr] of a conjecture of Vázsonyi, namely, that trees form a well-quasi-order under topological containment.

2.1 The Graph Minor Theorem and Some of Its Consequences

We can now state the Graph Minor Theorem and some of the most important consequences arising from its proof. Very little progress had been made on this result, formerly a conjecture attributed to K. Wagner, until the work of Robertson and Seymour.

The Graph Minor Theorem The class of all graphs is a well-quasi-order under the minor relation.

Corollary 2.1.1 Let C be a class of graphs closed under minors. Then C can be characterized by a finite list of minor obstructions.

To see that this follows from the theorem, let S be the set of minor-minimal graphs not in C. Then S is an antichain, and thus it is finite. Hence $G \in C$ if and only if G does not contain as a minor any graph isomorphic to a member of S.

Let v denote a vertex in a graph G. Let the edges incident on v be $\{v, u_i\}$, $1 \leq i \leq p$, and $\{v, w_j\}$, $1 \leq j \leq q$, where $2 \leq p, q$. Let H be the graph obtained by replacing v with two new vertices, u and w, replacing the edges incident on v with $\{u, u_i\}$, $1 \leq i \leq p$, and $\{w, w_j\}$, $1 \leq j \leq q$, and adding a new edge $\{u, w\}$. We say H is obtained from G by *splitting* v.

Corollary 2.1.2 [RSVIII] Let C be a class of graphs closed under minors. Then C can be characterized by a finite number of topological obstructions. Moreover, each such obstruction can be obtained from a minor obstruction with vertex splittings.

Corollary 2.1.3 [RSVIII] Let S be any surface. Then the class of graphs embeddable in S can be characterized by a finite number of topological obstructions.

This follows from the last corollary since embeddability in S is closed under minors.

We remark here that the proof of Erdös' Conjecture (Corollary 2.1.3) does not require a solution to Wagner's Conjecture. Rather, the tools used to settle the latter are a superset of those used for the former. Also, the number of topological obstructions can be quite large (indeed, for the projective plane there are 103, and for higher surfaces there are many more).

Theorem 2.1.4 [RSXIII] Let H be a fixed graph. Then there is a polynomial-time algorithm for testing, for any input graph G, whether G contains H as a minor.

The running time of the algorithm is $O(n^3)$, where $n = |V(G)|$. The constant hidden in the big Oh is a very rapidly growing function of the size of H.

Corollary 2.1.5 Let C be a class of graphs closed under minors. Then membership in C

can be tested in polynomial time.

With regards to Corollary 2.1.5, the testing algorithm would make use of Theorem 2.1.4, by testing minor containment of all obstructions. Thus the proof of this corollary is intrinsically *nonconstructive*. We know the desired polynomial-time algorithm exists, but we cannot implement it unless we have the list of obstructions, a task towards which the graph minors project provides no clues. Moreover, even if the obstructions were available, the resulting algorithm would be very impractical. Means of avoiding these problems in many general cases have been developed by Fellows and Langston, and are discussed in Section 6.

Another way to interpret Corollary 2.1.5 is to regard minor closure as a certificate of tractability. Once a given graph property is found to be closed, then an effort can be launched to find an efficient, direct algorithm for testing that particular property.

Corollary 2.1.6 Let S be a given surface. Then graph embeddability in S can be tested in polynomial time.

Of special interest are the consequences towards the disjoint paths problem: given vertices s_i and t_i $(1 \leq i \leq k)$, not necessarily distinct, in a graph G, find pairwise vertex-disjoint paths between s_i and t_i $(1 \leq i \leq k)$. This problem is $\mathcal{NP}$-hard for general k [Ka]. For $k = 1$, the problem is trivial. For $k = 2$, a complex solution has been known for some time [Se, Sh]. But the techniques used in [RSxiii] to obtain Theorem 2.1.4 yield the following.

Theorem 2.1.7 The disjoint paths problem can be solved in polynomial time for every fixed k.

The algorithms of Theorems 2.1.4 and 2.1.7 are constructive. The disjoint paths problem is addressed in more detail in Section 5.

This concludes our outline of some of the major results that stem from the graph minors project. In the sequel, we provide more information on topics as promised above, and discuss the important graph parameters treewidth and pathwidth.

3 Treewidth

Treewidth plays a critical role in the graph minors project. It may be said that it measures the complexity of a graph, in the sense that a graph of small treewidth can be recursively decomposed, by removing a few vertices, into two graphs of roughly equal size. A consequence of this is that many $\mathcal{NP}$-hard problems can be efficiently solved in graphs of small treewidth with dynamic programming.

A *tree decomposition* of the graph G consists of a pair (T, X), where T is a tree (which is not part of G, but merely another graph) and $X = \{X_t : t \in V(T)\}$ is a family of subsets of $V(G)$, one for each vertex of T, satisfying the following properties:

(1) For every edge $\{u, v\}$ of G, there exists $t \in V(T)$ with $u, v \in X_t$, and

(2) For every pair y, z of vertices of T, if w is any vertex in the path between y and z in T, then $X_y \cap X_z \subseteq X_w$.

The *width* of (T, X) is $max\{|X_t| - 1 : t \in V(T)\}$. The *treewidth* of G is the minimum integer w such that there is a tree decomposition of G of width w. Graphs of treewidth at most k have also been called *partial k-trees*.

These definitions may be restated in a way that is more familiar to some readers. Condition (2) states that, for every vertex v of G, the set of vertices $t \in V(T)$ such that $v \in X_t$ forms a subtree of T, say T_v. Condition (1) then states that for every edge $\{u, v\}$ of G, the subtrees T_u and T_v must intersect. Now consider the graph H with vertex set $V(G)$, and such that $\{a, b\}$ is an edge of H whenever T_a and T_b intersect. Then H is chordal, and every chordal graph can be obtained as such an intersection graph of subtrees of a tree [Ga]. Furthermore, the clique number of H is precisely the width of (T, X) plus one. Consequently, we arrive at an equivalent definition of treewidth: it is the minimum, over all chordal supergraphs H of G, of the clique number of H minus one.

It is $\mathcal{NP}$-hard to compute treewidth [AP]. But the family of graphs with treewidth at most k is minor-closed for every fixed k, and hence polynomial-time recognizable.

Example 3.0.1 Series-parallel graphs. These may be iteratively defined as follows: the graph consisting of a single edge is series-parallel, and given a series-parallel graph, we obtain a new one by adding an edge in parallel to an existing edge, or by subdividing an existing edge. Series-parallel graphs have treewidth at most 2, which may be shown inductively. For example, let G be series-parallel, and let (T, X) be a tree decomposition of G of width at most 2. Let $\{u, v\}$ be an edge of G. Suppose we subdivide $\{u, v\}$ by introducing a new vertex w, to obtain a new graph G'. Let r be a vertex of T such that $u, v \in X_r$. Now define a new tree T' to consist of T, together with the edge $\{r, q\}$, where q is a new vertex. Set $Y_t = X_t$, for $t \in V(T)$, and $Y_q = \{u, v, w\}$. It is seen that (T', Y) is a tree decomposition of G', of width at most 2.

Example 3.0.2 Grids. Let $m > 1$ be an integer. The m-grid is the graph with vertex set $\{(i, j) : 1 \leq i \leq m, 1 \leq j \leq m\}$ and edge set $\{\{(i, j), (i + 1, j)\} : 1 \leq i \leq m - 1, 1 \leq j \leq m\} \cup \{\{(i, j), (i, j + 1)\} : 1 \leq i \leq m, 1 \leq j \leq m - 1\}$. It can be shown that the m-grid has treewidth m. To see that the treewidth is at most m, consider the tree decomposition (T, X), where T is the path with vertices $1, 2, \ldots, m^2 - m$, and, for $1 \leq i \leq m - 1, 1 \leq j \leq m, X_{m(i-1)+j} = \{(i, k) : j \leq k \leq m\} \cup \{\{(i + 1, k) : 1 \leq k \leq j\}$. It is easily verified that this is indeed a tree decomposition of width m. The lower bound (treewidth at least m) is more difficult to establish [RSx].

Grids play an important role in the graph minors project, for it can be shown that for every $n \geq 1$, any planar graph with n vertices is a minor of the m-grid, with $m = O(n^2)$. Moreover, grids are useful in proving the following results.

Theorem 3.0.3 [RSv] Let H be a planar graph. Then there exists a number $w(H)$, such that any graph with no minor isomorphic to H has treewidth at most $w(H)$.

Corollary 3.0.4 [RSiv] Given an infinite list $G_1, G_2, \ldots$ of graphs, with G_1 planar, there exist $i < j$ such that G_i is a minor of G_j.

Of course this corollary follows from the Graph Minor Theorem even without G_1 being planar. But using Theorem 3.0.3 its proof can proceed as follows: assuming that G_1 is not a minor of $G_i, i > 1$, it follows that G_i has bounded treewidth. Thus, G_i has "simple" structure (think of it as a "thickened" tree) and now a generalization of the proof technique in Kruskal's theorem finishes the job. The main point here is, once more, that small treewidth implies simple structure. We underscore the planarity requirement in Theorem 3.0.3. Is there some "structure theorem" if this requirement is dropped? This question is studied in [RSxv]–[RSxvii], and the result can be informally described as follows.

Let us first restate the above concerning the exclusion of planar graphs. Let H be a fixed planar graph. If G does not contain a minor isomorphic to H, then G can be obtained by pasting together small graphs in a tree-like structure. Now suppose H is not planar. We obtain a corresponding structure theorem for graphs with no H-minor by replacing the phrase "small graphs" with the phrase "graphs with simple structure." This simple structure is in fact rather complex to state precisely. It is convenient to view it as parameterized by H itself; since H is fixed, this is loosely termed "simple." We obtain a graph of simple structure by starting with a graph of "small" genus, up to a "small" number of troublesome vertices (that is, a graph that can be embedded on a surface of low genus after removing only a few vertices). We attach to this graph a "small" number of necklace-like graphs. In these necklaces, the beads are "small" graphs, and the beads are attached to one another in a ring-like fashion.

We have obviously used the word "small" rather liberally. In the case of genus, small means smaller that the genus of H. In the other cases (the number of troublesome vertices, the number of necklaces and the sizes of the beads in the necklaces), it is again helpful to think of these numbers as parameters of H. Thus, for a fixed H, we would be able to write done explicit upper bounds for all parameters instead of the word "small." The proof techniques involved in obtaining these results are quite complex and interesting in their own right.

It is worth pointing out one further structure theorem. In [AST] it is shown that for any fixed graph H, if G has no minor isomorphic to H, then G satisfies a "separator theorem" that generalizes Lipton and Tarjan's [LT] planar separator theorem.

3.1 Bounded Treewidth and Efficient Algorithms

Suppose we are given a tree decomposition (T, X) of a graph G, whose width is bounded by some small constant w. It is often the case that we can exploit the structure of (T, X) to derive polynomial-time, dynamic-programming algorithms for solving problems that are $\mathcal{NP}$-hard on general graphs.

The generic approach would be as follows: first direct all edges of T away from a given root r. Next, for any vertex t of T, let T_t denote the subtree of T rooted at t. The key point is to notice that $\bigcup\{X_v : v \in V(T_t)\}$ can "interact" with the rest of G only through X_t, which is "small." Thus one seeks to generate a dynamic-programming strategy that stores all relevant information about the subgraph of G induced by $\bigcup\{X_V : v \in V(T_t)\} = Y_t$, by listing all possible "states" of X_t. We may assume that, without loss of generality, T is binary (degree at most 3), and thus it is easy for the recursion to move up the tree T. This approach, which frequently yields polynomial-time algorithms, has been taken by many authors, and it is essentially impossible to list all papers on this topic.

Let us discuss some folklore examples. The algorithms we sketch are not the best possible. We simply want to illustrate how polynomiality is achieved. Using notation as before, for any vertex t of T, we denote by G_t the subgraph of G induced by Y_t.

Example 3.1.1 The vertex cover problem. Given a graph G we seek a subset of vertices C, with minimum cardinality, such that every edge of G has at least one end in C. Now suppose we are given a width-w tree decomposition (T, X) of G. Let t be a vertex of T and, for each subset Z of X_t, let $f(Z, t)$ denote the smallest cardinality of a vertex cover of G_t, with the added restriction that we use Z in the cover. Thus a table with 2^{w+1} entries can be used to record this information for each t. It is not difficult to see how to update the table as we move up the tree T. Furthermore, $\min\{f(Z, r) : Z \subseteq X_r\}$ solves the vertex cover problem. The run time of the algorithm is exponential in w, but linear in $|E(G)|$.

Example 3.1.2 The traveling salesman problem in graphs. Let G be a graph with weights on the edges, where we seek a Hamiltonian cycle of minimum length. One possible recursion is based on the following states. Let Z be a subset of X_t, and consider (for each $0 \leq m \leq |X_t|$) $2m$ distinct vertices $o_i, d_i, 1 \leq i \leq m$, contained in $X_t - Z$. Let P be the set of pairs $\{o_i, d_i\}$. Then we denote by $f(t, Z, P)$ the minimum total length of a family of vertex-disjoint paths $R_i, 1 \leq i \leq m$, in Y_t, such that R_i has ends o_i and d_i, and $\bigcup\{V(R_i) : 1 \leq i \leq m\} = Y_t - Z$.

Example 3.1.3 The vertex coloring problem. Given a graph G, we seek to partition the vertices of G into a minimum number of independent sets (the chromatic number of G).

Here the recursion works as follows: for each partition π of X_t, we denote by $s(t, \pi)$ the minimum cardinality of a partition of $Y(t)$ into independent sets, such that the intersection of X_t with the color classes yields the partition π.

Many graph problems are not amenable to this generic dynamic programming approach. For example, any problem that is $\mathcal{NP}$-hard on trees cannot be solved this way (trees have treewidth 1). A partial characterization of the solvable problems, together with a prototype algorithm, can be found in [ALS].

The fact that the algorithms described above exist is not, of course, a consequence of Robertson and Seymour's work; nevertheless, it is clearly related to the concepts of treewidth and tree decomposition, and so it seems appropriate to touch on this topic here.

3.2 Branchwidth, Tangles and Graph Searching

An interesting graph parameter that is related to treewidth is *branchwidth*. Branchwidth may be computationally more tractable than treewidth (at least in terms of approximation). Further, concepts related to branchwidth play a crucial role in the development of the graph minors project, and so it is useful to review them here.

Given a graph G, a *branch decomposition* of G consists of a pair (T, f), where T is a binary tree, and f is an injective map from $E(G)$ to the leaves of T. Notice that if e is an edge of T, then there is a partition of $E(G)$ into two classes (A, B) that corresponds to e — this is defined by the partition of the leaves of T into the two subtrees of $T - e$. The *order* of e is defined as the number of vertices of G that have incident edges both in A and in B. The width of (T, f) is the maximum order of any edge in T. See Figure 2.

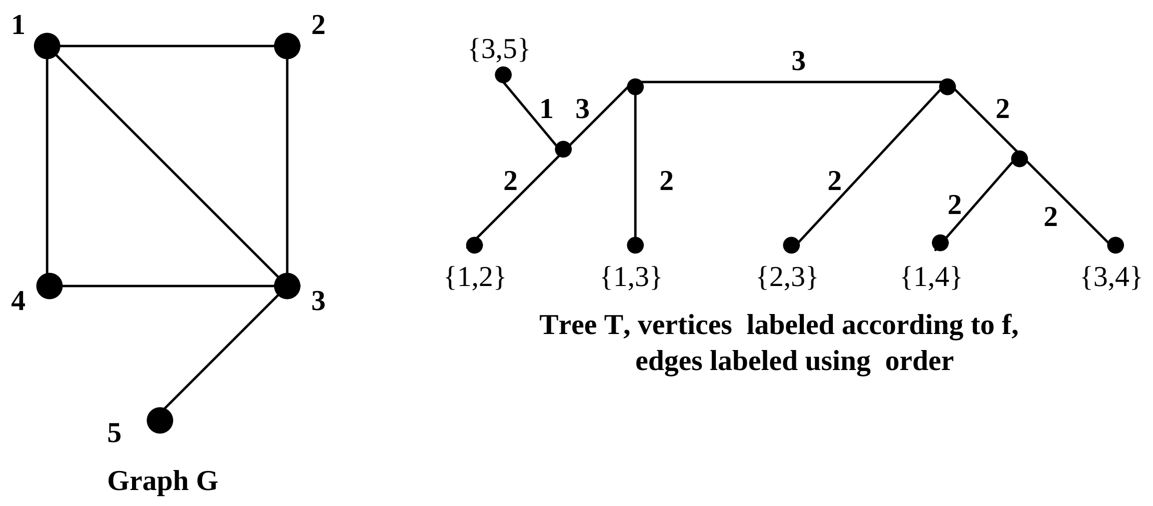
1
2
4
3
5
Graph G
{3,5}
1
3
3
2
2
2
2
2
2
{1,2}
{1,3}
{2,3}
{1,4}
{3,4}
Tree T, vertices labeled according to f,
edges labeled using order

Figure 2

The branchwidth of G is the minimum width of any branch decomposition of G. Computing the branchwidth of a graph is $\mathcal{NP}$-hard. It is not very difficult to show the following.

Theorem 3.2.1 [RSx] The branchwidth and treewidth of any graph differ by at most a factor of 3/2.

Does this theorem help in approximating treewidth? Historically, it first appeared that the answer to this question should be positive, since in [RSx] a min-max characterization of branchwidth was given, while no such characterization of treewidth was yet known. However, the complete picture is more complex.

Theorem 3.2.2 [ST2] The branchwidth of a planar graph can be computed in polynomial time.

In any case, the tools developed in [RSx] were extremely useful towards completing the proof of the Graph Minor Theorem. Moreover, the ideas behind the min-max characterization of branchwidth were later adapted to yield a similar characterization of treewidth (and also of pathwidth, a concept we cover later). These min-max formulae were then used to improve some of the original theorems in the graph minors project.

It is enlightening to describe the relationship of treewidth to a class of graph searching games. Regard a graph as a system of roads. A fugitive resides in the vertices and can travel along edges. We wish to capture the fugitive (whose position is always known) using a fixed number of guards, who always occupy vertices and travel using helicopters. In one time unit, some of the guards can move to a different subset of vertices. During the move the fugitive can scurry, infinitely fast, to a new vertex, traveling along any path that is not blocked by an unmoving guard. Our objective is to corner the fugitive in such a way that no escape is possible.

The minimum number of guards needed for this purpose is called the *search number* of the graph. Thus, for example, a tree has search number 2 (place a guard at any vertex and observe which subtree is occupied by the fugitive, then corner the fugitive into smaller and smaller subtrees). The graph in Figure 3 has search number 3. To see this, notice that with at most 2 guards the fugitive can always occupy a vertex in $V(G) - \{s, t\}$. But using three guards, with two guards we can first isolate the fugitive in one of the paths with ends s, t, and with the third guard we then capture the fugitive.

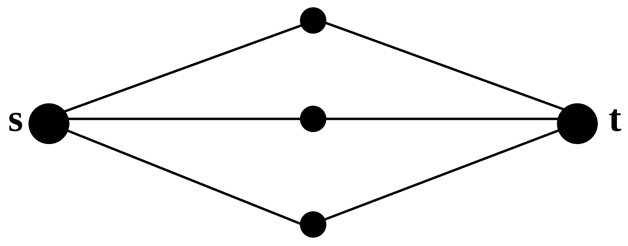

Figure 3

Theorem 3.2.3 [ST1] For any graph G, the treewidth of G equals the search number of G minus 1.

Intuitively, a tree decomposition (T, X) of G yields a search strategy of G involving a number of guards equal to the width of (T, X) plus one: we corner the fugitive into smaller and smaller subgraphs of the form G_t (recall the notation of Section 2.1) by placing guards in the subset X_t. The definition of tree decomposition shows that this strategy indeed works and requires the desired number of guards. This argument yields one of the two bounds needed to prove Theorem 3.2.3.

The proof of the other bound also yields a min-max formula for treewidth, as follows. Let k be an integer. A *haven of order* k is a function g that assigns to each subset Y of k or fewer vertices, the vertex set of one of the components of $G - Y$, denoted $g(Y)$. The function g, in addition, satisfies that whenever X and Y are subsets of vertices with $|Y| \leq k$ and $X \subseteq Y$, then $g(Y) \subseteq g(X)$. The *tangle number* of G is the maximum order of a tangle. As examples, the tangle number of a tree is 1, and the tangle number of the complete graph K_n is $n - 1$.

Theorem 3.2.4 For any graph G, the tangle number of G equals the treewidth of G.

Intuitively, think of a tangle of order k as the escape plan of the fugitive in case k guards are being used: if the guards occupy Y, the fugitive hides in $g(Y)$. The proof of Theorem 3.2.4 and the formalization of the preceding argument are quite complex, and are not given here. These tools are related to some of the concepts in [RSx].

Given that it is $\mathcal{NP}$-hard to compute treewidth, Theorem 3.2.4 is at first glance surprising. But observe that (for arbitrary k) the complete description of a tangle requires exponential time. Nevertheless, this theorem and its connection to graph searching may turn out to be useful (from a computational point of view) towards estimating treewidth.

Graph searching games similar to the one given above have previously been considered by researchers in the computer science community. They are of interest in that they provide a worst-case scenario for the process of immunizing a network against a computer virus. We return to this topic later.

3.3 Treewidth and Signal Routing Problems

A problem that frequently arises in communications networks is the following: suppose we are given nodes $1, 2, \ldots, n$, and an $n \times n$ traffic matrix M (where m_{ij} is the traffic rate between i and j). We wish to design a binary tree T, with leaves precisely $1, 2, \ldots, n$, to carry the traffic. Notice that given such a tree T, each edge e of T will carry a certain total amount of traffic or *congestion* (namely, the traffic between nodes separated in T by e). The tree T is to be chosen so that the maximum such congestion is minimized. The resulting optimal congestion level is called the *carvingwidth* of M [ST2]. We remark that the use of trees as communications networks is widespread and natural in many applications because of their simplicity and ease of fabrication. We are essentially designing a tree (a very simple structure) to realize a more complex pattern (the traffic requirements).

Not surprisingly, it is $\mathcal{NP}$-hard to compute carvingwidth. However, if M is planar (that is, if the graph G with vertices $\{1, \ldots, n\}$ and an edge $\{i, j\}$ whenever $m_{ij} > 0$ is planar) then carvingwidth can be solved in polynomial time. In particular, there is a nice min-max characterization of carvingwidth in this special case [ST2]. Further, the tools and proof techniques are essentially the same as those used to prove Theorem 3.2.2.

The above results suggest that there is a deep connection between treewidth and carvingwidth. Let us consider the case where M is a $\{0, 1\}$-matrix; i.e., M is the adjacency matrix of G. Then computing carvingwidth corresponds to finding good graph embeddings [HMR]. Let *congestion*(G) denote the carvingwidth of M.

Theorem 3.3.1 [Bi] If G has treewidth k and maximum degree d, then $\Omega(max\{k, d\}) \leq$

$congestion(G) \leq O(kd)$.

Thus, for graphs of bounded degree, treewidth and congestion are of the same order of magnitude.

There is another parameter that arises in routing problems and is related to treewidth. Consider a binary tree T with leaves labeled $\{1, 2, \ldots, n\}$ as above. If $m_{ij} > 0$, then it is desirable that the path in T between i and j be short. The dilation of T is the maximum length of any such path, and $dilation(G)$ is the minimum dilation over all binary trees T.

Theorem 3.3.2 [Bi] If G has treewidth k and maximum degree d, then $\Omega(\log k + \log d) \leq dilation(G) \leq O(\log k + \log^* n \log d)$.

Thus, approximating treewidth is tantamount to approximating dilation within a very small additive error ($\log^* n$ is an extremely slowly growing function of n).

3.4 On Computing Treewidth

Given that the concepts associated with treewidth are extremely useful in a wide range of applications, and that computing treewidth (and branchwidth, carvingwidth, etc.) is $\mathcal{NP}$-hard, what can one say about computability of treewidth? There are at least three ways of approaching this problem: (1) approximation algorithms, (2) testing for small treewidth, and (3) experimental results.

A polynomial-time approximation algorithm that does not depend on fixing the treewidth has very recently been devised by Robertson, Seymour and Thomas.

Theorem 3.4.1 [Th] There is a polynomial-time algorithm that, given a graph G and an integer k, either proves that G has treewidth at least k or provides a tree decomposition of G of width at most $k^2 + 2k - 1$.

The proof of this result relies on a minimax characterization from [ST$_{\text{I}}$] and the decomposition method of [RS$_{\text{XIII}}$]. We stress that in this theorem the parameter k is not fixed, but is instead part of the input. The algorithm is fairly reasonable (its run time does not involve excessive constants or very high degrees) and its main application lies in testing whether the treewidth of a given graph is small. This is important since many of the above applications require bounded treewidth.

Are there sharper approximation algorithms? Ideally, one seeks a polynomial-time algorithm that approximates treewidth up to a constant factor. Until branchwidth was shown to be $\mathcal{NP}$-hard, a natural approach was to seek an exact algorithm for this parameter instead. In any case, approximating treewidth remains a crucial open problem. Moreover, it seems likely that the tools required for this task would also be of use towards other $\mathcal{NP}$-hard problems involving cuts (such as graph bisection) for which no constant-factor approximation algorithms are known.

Next we turn to the problem of testing for small treewidth. Recall that the property (for any given k) of having treewidth at most k is closed under minors. Thus, according to Corollary 2.1.5, there is a polynomial-time algorithm to compute (exactly) the treewidth of a graph known to have small treewidth. But this approach is nonconstructive and perhaps not useful. Another approach would be to use a dynamic-programming scheme as described in Section 3.1. However, such an algorithm may be quite unreasonable. A third approach is the recent result of Reed.

Theorem 3.4.2 [Re] For each fixed k, we can test whether G has treewidth at most k in $O(n \log n)$ time.

In terms of experimental results concerning the computation of treewidth, no major results are available. An intriguing possibility is the use of integer programming to compute tangles. It is easy to see that the existence of a tangle (of a given order) can be described by a system of equations in 0-1 variables (but an exponential number of those, unfortunately). A possible research problem of interest would be to describe the polyhedral structure of the convex hull of tangles.

4 Pathwidth and Cutwidth

In the development of the graph minors project, treewidth was preceded by another graph parameter, *pathwidth*. The pathwidth of a graph can be much larger than its treewidth. Several important applications of pathwidth arose well before the graph minors project.

The definition of pathwidth is similar to that of treewidth, except that we restrict ourselves to tree decompositions (T, X) where T is a *path* (such tree decompositions are called *path decompositions*). Thus if (T, X) is a path decomposition of G, then every vertex v of G is mapped into a subpath P_v of T (i.e., each vertex essentially is mapped into an interval), so that whenever $\{u, v\}$ is an edge of G, then P_u and P_v intersect. The width of the path decomposition is the maximum number of subpaths P_v that are incident with any vertex of T minus one. There is a connection similar to that between treewidth and chordal graphs: pathwidth equals the smallest clique number over all *interval* supergraph of G minus one [Go]. For example, paths have pathwidth 1, and a complete binary tree on $m > 1$ levels has path width $\lceil m/2 \rceil$.

In terms of graph minors, the most important theorem involving pathwidth is an analogue to Theorem 3.0.3.

Theorem 4.0.1 [RSI] For every forest F there is a number $p(F)$, such that if a graph G does not have a minor isomorphic to F, then G has pathwidth less than $p(F)$.

The original proof of Theorem 4.0.1 employed a function p that was very rapidly growing in $|V(F)|$. This result has been improved [BRST] to show that $p(F) = |V(F)| - 1$, which is best possible.

Recall that treewidth is related to graph searching and embedding problems. The same is true for pathwidth, and again these connections chronologically preceded those for treewidth.

First we consider graph searching. There are two versions of this game that have been known in the literature for some time. Here the main difference is that the guards do not know where the fugitive is. In one version, called *edge searching*, the portion of the graph "secured" by the guards can be extended by sliding a guard along an edge leading out of this portion. In the other, called *node searching*, an edge is cleared by placing a guard at each end, simultaneously. For either kind of game one can define the search number of a graph, as previously, to be the minimum number of guards needed to catch the fugitive. It is shown in [KP] that the edge-search number and the node-search number never differ by more than 1, and that the node-search number always equals pathwidth plus 1. A different version of the game, called *mixed searching*, is considered in [BS]. In mixed searching, moves from both edge and node searching are allowed. This enables one to obtain short proofs for the monotonicity of both edge and node searching (monotonicity here means that no search strategy of a graph need ever repeat the same step).

With regards to graph embedding problems, the connection here is via the $\mathcal{NP}$-hard cutwidth problem, defined as follows. Given a graph G on n vertices, suppose we label the vertices with the integers $1, 2, \ldots, n$. The width of this labeling is the maximum, over $1 \leq h \leq n-1$, of the number of edges $\{i, j\}$ with $i \leq h$ and $h < j$. The objective is to find a labeling with minimum width (defined as the cutwidth of G). This problem originally arose in the design of linear arrays, an early form of printed circuit. In [MS] it is shown that if G has pathwidth p and maximum degree d, then the cutwidth of G is $\Omega(max\{p, d\})$ and $O(pd)$, a result similar to Theorem 3.3.1. Thus, for graphs of bounded pathwidth, there is a polynomial-time algorithm that approximates cutwidth up to a constant factor.

It also turns out that pathwidth is linear-time equivalent to the gate matrix layout problem [DKL], another problem with application to printed circuits. This problem can be stated as follows. Suppose we are given a $\{0,1\}$ matrix M. Let $M(\pi)$ result from permuting the columns of M according to some permutation π, and suppose we replace the 0 entries of $M(\pi)$ in each row, between the first and last occurrences of 1 in that row, with 1's. The maximum number of 1's in any column of the resulting matrix is called the *width* of π. Then we seek a permutation π of minimum width (this corresponds to laying out devices in a chip so as to minimize the number of wire tracks required to effect desired connections). Call this number the layout width of M. To see the connection with pathwidth, let G denote the clique graph of the transpose of M; i.e., the graph with vertices the rows of M, and a clique arising from the 1's in each column. Then it is easy to verify that the layout width of M is exactly the pathwidth of G (refer to the interval graph interpretation of pathwidth above).

As with treewidth, it is $\mathcal{NP}$-hard to compute the pathwidth of a graph, and approximation algorithms are known only for very special cases [Ya]. Again, there is a min-max formula for pathwidth with corresponding obstructions. These obstructions (an appropriate name might be "linear tangles") are described in detail in [BRST], and it suffices here to say that they are closely related to the tangles of Section 3.2.

Much is known about the nature of obstructions for pathwidth k. For $k = 0$ there is one; for $k = 1$ there are two; for $k = 2$ there are 110 [KL]; and for $k = 3$ there are at least 122 million! Moreover, all tree obstructions are known.

The approximate computation of pathwidth for general graphs is an interesting open problem, and once more we point out the possible use of integer programming techniques in this context. Notice that the existence of a path decomposition of given width corresponds directly to the solvability of a system of linear equations in $\{0,1\}$ variables (as opposed to the treewidth case, where it is easiest to describe the obstructions in this manner).

5 Disjoint Paths

Recall the definition of the disjoint paths problem. We are given, in a graph G, vertices s_i and $t_i (1 \leq i \leq k)$, not necessarily distinct. We seek pairwise vertex-disjoint paths between s_i and $t_i (1 \leq i \leq k)$. In this section we outline how graph minors theory yields an algorithm with complexity $O(n^3)$ for this problem, for each fixed value of k.

It is worthwhile first to compare this problem to that of H-minor containment: given G, test whether it has a minor isomorphic to H. For each fixed H, this problem can be reduced to the disjoint paths problem. The resulting algorithm will, however, have high complexity (the degree depends on $|V(H)|$, still polynomial for fixed H, but perhaps not very appealing).

Similarly, the disjoint paths problem is somewhat reminiscent of the H-minor containment problem, where H consists of k independent edges. In any case, Robertson and Seymour reduced both problems to a more general one, called the Folio problem [RSxiii].

We next briefly outline one of the main ideas in the disjoint paths algorithm. Our intent is not to try to present an accurate description of the algorithm, but rather to illustrate the deep connection between the disjoint paths problem and issues related to graph minors. The argument is most persuasive when restricted to planar graphs. Thus we assume a planar input graph, G.

If G has "not very large" treewidth (a condition that can be tested in polynomial time), then the problem is fairly simple: one can apply a dynamic programming approach as in Section 2.1. Suppose on the other hand that G has very large treewidth. Then, by Theorem 3.0.3, G contains an enormous square grid minor H; i.e., a minor isomorphic to the m-grid where m is very large. For simplicity, assume H is actually a subgraph of G (the exact situation is not very different). Since there are at most $2k$ vertices s_i, t_i, we may even assume that H is "far away" from all the s_i and t_i. (For example, none of the internal faces of H, as embedded in G, topologically contain any of the s_i and t_i. See Figure 4.)

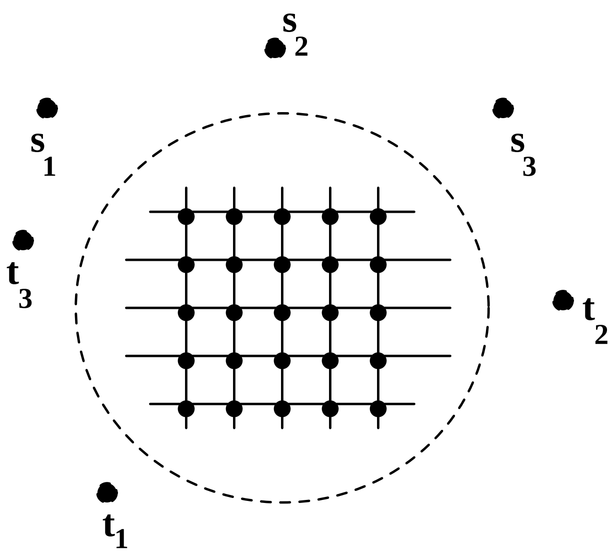

Figure 4

Now let v be a vertex of H located near the middle of H. Then removing v from G should not alter the situation; that is, G contains the desired family of disjoint paths if and only if $G - v$ does. To see this, assume we are given the desired family of disjoint paths, where one of these paths, p_1, contains v. Suppose we perturb slightly p_1 around v. This perturbation will then cause a ripple effect: we will have to move other paths in order to preserve disjointness. But the fact that H is a very large square grid, and far away from all the s_i and t_i, ensures that a global way of shifting the paths does exist, and we can indeed remove v from G without changing the problem, as desired. Consequently, we have now reduced the problem to an equivalent one in a smaller graph. Now we can go back and repeat the treewidth test, and so on until the problem is solved after at most a linear number of vertex removal steps. There remains the algorithmic problem of constructing the square grid minors when needed — but here the fact that G has very high treewidth makes the task easy.

How do we bypass the planarity assumption? The argument that yields H is just as above. But if G is not planar all vertices near the middle of H may be crucial; i.e., always needed in the disjoint paths. Moreover, the "far away" requirement for H may not work out. But in any case, it turns out that one can always find an "irrelevant" vertex. With such a vertex at hand we continue as above. The proof of all this uses several deep structure theorems that are far too complex to describe here. See [RS] for an excellent detailed survey of the disjoint paths algorithm.

5.1 Some New Developments Concerning Disjoint Paths

There are some interesting variants of the disjoint paths problem on planar graphs (in fact, on graphs embedded on surfaces) that have recently been studied. The algorithms and theorems involved do not follow from graph minors theory, but we describe them here for completeness.

Some problems have been solved by Schrijver [Sc]. The problems were initially motivated by certain issues in circuit routing, as follows. Suppose we are given a chip that contains some devices (think of these as right-angle polygons). The chip also contains a system of tracks, forming a grid, for the purpose of routing wires. Our problem is to route wires on this grid so as to realize connections between given pairs of terminals on these devices. These wires cannot touch one another or a device (other than at their ends) and, moreover, we are even given a sketch of how each wire must look; i.e., how the wire must thread its way among the devices. The algorithmic question is: can we find a wire routing that meets all these requirements? A polynomial-time algorithm for this problem was given by Cole and Siegel [CS] and Leiserson and Maley [LM].

The problem can be substantially generalized as follows. We are given a planar graph G, a collection of faces $F_1, F_2, \ldots, F_m$ of G, a collection of vertices s_i, t_i $(1 \leq i \leq k)$ of G, each located in the boundary of some F_j, and a collection of paths q_i $(1 \leq i \leq k)$ between s_i and t_i. Do there exist vertex-disjoint paths p_i $(1 \leq i \leq k)$ between s_i and t_i, such that p_i is homotopic to q_i in $\Re^2 - F_1 \cup F_2 \ldots \cup F_1$?

Schrijver has presented an $O(n^2 \log n)$ algorithm for this problem. We stress here that, unlike the version of the disjoint paths problem discussed before, the parameter k is *not* assumed to be fixed. At first glance this seems surprising, since the (standard) disjoint paths problem is $\mathcal{NP}$-hard for planar graphs. But notice that in this new version we are told how each path must "look like." Reed has improved the algorithm so as to achieve linear run time. The algorithm can also be partially extended to handle disjoint trees (rather than paths) that join specified vertex sets, and also to higher surfaces.

Another area of interest concerns the disjoint paths problem on directed graphs. Ding, Schrijver and Seymour [DSS] have considered the following case: we are given a planar digraph D, vertices $s_i, t_i (1 \leq i \leq k)$ all located on the boundary of one face F, and subsets of edges $A_i (1 \leq i \leq k)$. We seek vertex-disjoint $s_i - t_i$ paths p_i, all of whose edges are contained in $A_i (1 \leq i \leq k)$. They presented a necessary and sufficient condition for the existence of such paths (which extends one given in [RSvi]), together with a polynomial-time algorithm for the problem.

6 Challenges to Practicality

We close this chapter with a discussion of several unusual aspects of algorithms provided by the Graph Minor Theorem. Recall that if F is a minor-closed family of graphs, then we know from the developments already sketched that F can be recognized in polynomial time. Letting n denote the number of vertices in G, the general bound is $O(n^3)$. If F excludes a planar graph, then the bound is reduced to $O(n^2)$. Interestingly, such algorithms suffer from novel shortcomings:

- the algorithms require immense constants of proportionality,
- only the complexity of decision problems is established, and
- there is no general means for finding (or even recognizing) correct algorithms.

We tackle each of these issues in turn, illustrating algorithmic techniques with simple examples. We make no pretense that these examples reflect the state of the art. The interested reader is referred to [FL1, FL2] for more complex methods.

6.1 Constants of Proportionality

The theory developed by Robertson and Seymour proceeds in distinct structural stages. The theorems that employ this structural information introduce stunningly enormous constants of proportionality into polynomial-time decision algorithms. These huge structural constants can sometimes be eliminated by proving problem-specific structural bounds.

Example 6.1.1 Consider the gate matrix layout problem mentioned in the last section. It is known that, for any fixed value of k, there is a surjective map from Boolean matrices to graphs such that all matrices mapped to the same graph have the same layout cost, that the "yes" family of graphs in the image of the map is minor-closed, and that planar obstructions exist. Thus gate matrix layout is decidable for any fixed k in $O(n^2)$ time, but with a gigantic structural constant c_k bounding the treewidth of any graph in F_k entering into the constant of proportionality of the algorithm. (This constant is computed by a nine step procedure that involves several compositions of towers of 2's functions [RSv].) As we have previously noted, however, the family of matrices with gate matrix layout cost k turn out to correspond to the family of graphs with pathwidth $k-1$, which is a proper subset of the family of graphs with treewidth $k-1$. Thus a direct consideration of the needed structural bound allows the constant c_k to be replaced by $k-1$.

A more general approach is to prove structural bounds specific to a particular class of obstructions. These bounds then apply to any family with an obstruction in that class.

Theorem 6.1.2 [FL2] Any minor-closed family that excludes a cycle of length l has treewidth at most $l-2$ and can be recognized in $O(n)$ time.

Example 6.1.3 Reconsider the vertex cover problem, where we seek to determine whether all edges in an input graph G can be covered by at most k vertices, for some fixed k. As discussed in Example 3.1.1, this problem could be solved by finding a tree decomposition and then applying dynamic programming. Both of these steps could require $O(n^2)$ time without special tools. Moreover, the tree decomposition width is the enormous c_k. But the family of "yes" instances is minor-closed and excludes C_{2k+1}, the cycle of length $2k+1$. By applying the technique used in the proof of Theorem 6.1.2, only a (linear-time) depth-first search is needed to obtain a tree decomposition of width at most $2k-1$, followed by a finite number of obstruction tests, each taking linear time. Thus both the structural constant and the time complexity are reduced.

6.2 Decision Problems versus Search Problems

Algorithms based on finite obstruction sets only solve decision problems. In practice, one is usually more concerned with *search* problems, where the goal is to search for evidence that an input is a "yes" instance. For example, a "yes" or "no" response is sufficient to answer the decision version of vertex cover. For the search version, however, we want a satisfying cover (set of k or fewer vertices) when any exist. Fortunately, decision algorithms can be converted into search algorithms for the vast majority of problems amenable to the work of the graph minors project. The general idea is often termed *self-reduction*, whereby the decision algorithm is used as a subprogram by the search algorithm.

Example 6.2.1 In the decision version of the longest path problem, we seek to know whether an input graph contains a simple path of length k or more. The problem is $\mathcal{NP}$-complete in general, but solvable in $O(n)$ time for any fixed k, because the "no" family is minor-closed and excludes a cycle of length $k + 1$. When solving this problem in a practical setting, of course, we are concerned with finding a sufficiently long path when any exist, that is, solving the search version of the problem. To accomplish this, we need only self-reduce as follows. First, accept the input and pose the decision version of the problem. If the response is "no," then halt — no satisfying evidence exists. If the response is "yes," then perform the following series of operations for each edge in the graph:

1. temporarily remove the edge and pose the decision problem again
2. if the new graph is a "no" instance, replace the edge (it is needed in any sufficiently long path)
3. if the new graph is a "yes" instance, permanently remove the edge (some sufficiently long path remains).

Thus, $O(n^2)$ calls to an $O(n)$ decision algorithm suffice, yielding an $O(n^3)$ time search algorithm.

6.3 Nonconstructivity

As mentioned in Section 2, a guarantee of polynomial-time decidability provided by minor-closure is nonconstructive. But need this be the case? To consider such a question, we must decide on a finite representation for an arbitrary minor-closed family. (After all, it would of course be impossible to construct algorithms if the representation were not finite!) A reasonable choice is the Turing machine, the standard model of complexity theory. Unfortunately, a reduction from the halting problem affirms that nonconstructivity cannot be eliminated in a general sense.

Theorem 6.3.1 [**FL**2] There is no algorithm to compute, from a finite description of a minor-closed family represented by a Turing machine that accepts precisely the graphs in the family, the set of obstructions for that family.

So we must settle for something less. In the following, the term *known* refers to an algorithm that can, at least in principle, be coded up and run.

Theorem 6.3.2 [**FL**2] Let P_D denote a decision problem whose "yes" instances are minor-closed. Let P_S denote the corresponding search problem. If algorithms are known to self-reduce P_S to P_D and to check whether a candidate solution satisfies P_S, then an algorithm is known that solves both P_D and P_S.

The proof of this has an interesting wrinkle, in that the resultant (known) algorithms generate and make use of incomplete obstruction sets, yet they cannot be used to generate complete sets or even to check the completeness of proffered sets!

Example 6.3.3 Consider the $\mathcal{NP}$-complete modified cutwidth problem, in which we are given a graph G and a positive integer k, and are asked whether G can be laid out with its vertices along a straight line so that no plane that cuts the line *on* an arbitrary vertex can cut more than k edges. Until recently, the fastest known algorithm for both the decision and the search versions of this problem had time complexity $O(n^k)$. Thus modified min-cut is technically in $\mathcal{P}$ for any fixed value of k. This can be improved on, but nonconstructively, because the family of line graphs of "yes" instances is minor-closed. But modified min-cut is easy to self-reduce and easy to check. Thus the decision and search versions of modified min-cut can be solved in $O(n^3)$ time constructively (with known algorithms).

Acknowledgments

We wish to express our appreciation to Jean Blair, Heather Booth, Rajeev Govindan, Eric Kirsch, Scott McCaughrin and Siddharthan Ramachandramurthi for carefully reviewing an early draft of this chapter. We also wish to thank an anonymous reviewer for many helpful comments.

Postscript

Progress on the topics we have discussed continues apace. By the time this chapter reaches print, we are confident that many more relevant results will have been announced. We apologize in advance to those authors whose recent work has thus been unfortunately omitted from this treatment.

References

[Ar] D. Archdeacon, "A Kuratowski Theorem for the Projective Plane," Ph.D. Thesis, Ohio State University (1980).

[ALS] S. Arnborg, J. Lagergren and D. Seese, "Easy Problems for Tree Decomposable Graphs," *Journal of Algorithms* 12 (1991), 308–340.

[AP] S. Arnborg and A. Proskurowski, "Complexity of Finding Embeddings in a k-Tree," *SIAM Journal on Algebraic and Discrete Methods* 8 (1987), 277–284.

[AST] N. Alon, P. D. Seymour and R. Thomas, "A Separator Theorem for Non-Planar Graphs," to appear.

[Bi] D. Bienstock, "On Embedding Graphs in Trees," *Journal of Combinatorial Theory Series B* 49 (1990), 103–136.

[BM] J. A. Bondy and U. S. R. Murty, Graph Theory with Applications, London, Macmillan, 1976.

[BRST] D. Bienstock, N. Robertson, P. D. Seymour and R. Thomas, "Quickly Excluding a Forest," *Journal of Combinatorial Theory Series B* 52 (1991), 274–283.

[BS] D. Bienstock and P. D. Seymour, "Monotonicity in Graph Searching," *Journal of Algorithms* 12 (1991), 239–245.

[CS] R. Cole and A. Siegel, "River Routing Every Which Way, but Loose," *Proceedings, 25th Annual Symposium on Foundations of Computer Science* (1984), 65–73.

[DKL] N. Deo, M. S. Krishnamoorthy and M. A. Langston, "Exact and Approximate Solutions for the Gate Matrix Layout Problem," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 6 (1987), 79–84.

[DSS] G. Ding, A. Schrijver and P. D. Seymour, "Disjoint Paths in a Planar Graph – a General Theorem," to appear.

[FL1] M. R. Fellows and M. A. Langston, "Nonconstructive Tools for Proving Polynomial-

Time Decidability," *Journal of the ACM* 35 (1988), 727–739.

[FL₂] ———————, "On Search, Decision and the Efficiency of Polynomial-Time Algorithms," *Proceedings, 21st Annual ACM Symposium on Theory of Computing* (1989), 501–512.

[Ga] F. Gavril, "The Intersection Graphs of Subtrees in Trees are Exactly the Chordal Graphs," *Journal of Combinatorial Theory Series B* 16 (1974), 47–56.

[GHW] H. Glover, P. Huneke and C. S. Wang, "103 Graphs That Are Irreducible for the Projective Plane," *Journal of Combinatorial Theory Series B* 27 (1979), 332–370.

[Go] M. O. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, 1980.

[HMR] J. Hong, K. Mehlhorn and A. Rosenberg, "Cost Trade-Offs in Graph Embeddings, with Applications," *Journal of the ACM* 30 (1983), 709–728.

[HT] J. E. Hopcroft and R. E. Tarjan, "Efficient Planarity Testing," *Journal of the ACM* 21 (1974), 549–568.

[Ka] R. M. Karp, "On the Complexity of Combinatorial Problems," *Networks* 5 (1975), 45–68.

[KL] N. G. Kinnersley and M. A. Langston, "Obstruction Set Isolation for the Gate Matrix Layout Problem," Technical Report CS-91-126, Department of Computer Science, University of Tennessee, 1991.

[KP] L. M. Kirousis and C. H. Papadimitriou, "Searching and Pebbling," *Journal of Theoretical Computer Science* 47 (1986), 205–218.

[Kr] J. Kruskal, "Well-Quasi-Ordering, the Tree Theorem, and Vázsonyi's Conjecture," *Transactions of the American Mathematical Society* 95 (1960), 210–225.

[Ku] C. Kuratowski, "Sur le problème des courbes gauches en topologie," *Fundamenta Mathematicae* 15 (1930), 271–283.

[LM] C. E. Leiserson and F. M. Maley, "Algorithms for Routing and Testing Routability of Planar VLSI-Layouts," *Proceedings, 17th Annual ACM Symposium on Theory of Computing* (1985), 69–78.

[LT] R. J. Lipton and R. E. Tarjan, "A Separator Theorem for Planar Graphs," *SIAM Journal on Applied Mathematics* 36 (1979), 177–189.

[Ma] W. S. Massey, Algebraic Topology: An Introduction, New York: Springer, 1967.

[MS] F. Makedon and I. H. Sudborough, "On Minimizing Width in Linear Layouts," *Discrete Applied Mathematics* 23 (1989), 243–265.

[Re] B. Reed, personal communication.

[RS] N. Robertson and P. D. Seymour, "An Outline of a Disjoint Paths Algorithm," in Algorithms and Combinatorics (Korte, Lovász, Prömel and Schrijver, eds.), Springer-Verlag, 1990, 267–292.

[RS$_{\rm I}$] ———, "Graph Minors. I. Excluding a Forest," *Journal of Combinatorial Theory Series B* 35 (1983), 39–61.

[RS$_{\rm IV}$] ———, "Graph Minors. IV. Treewidth and Well-Quasi-Ordering," *Journal of Combinatorial Theory Series B* 48 (1990), 227-254.

[RS$_{\rm V}$] ———, "Graph Minors. V. Excluding a Planar Graph," *Journal of Combinatorial Theory Series B* 41 (1986), 92–114.

[RS$_{\rm VI}$] ———, "Graph Minors. VI. Disjoint Paths Across a Disk," *Journal of Combinatorial Theory Series B* 41 (1986), 115–138.

[RS$_{\rm VIII}$] ———, "Graph Minors. VIII. A Kuratowski Theorem for General Surfaces," *Journal of Combinatorial Theory Series B* 48 (1990), 255-288.

[RS$_{\rm X}$] ———, "Graph Minors. X. Obstructions to Tree Decomposition," *Journal of Combinatorial Theory Series B* 52 (1991), 152–190.

[RS$_{\rm XIII}$] ———, "Graph Minors. XIII. The Disjoint Paths Problem," to appear.

[Th] R. Thomas, personal communication.

[Sc] A. Schrijver, "Decomposition of Graphs on Surfaces and a Homotopic Circulation Theorem," *Journal of Combinatorial Theory Series B* 51 (1991), 161–210.

[Se] P. D. Seymour, "Disjoint Paths in Graphs," *Discrete Mathematics* 29 (1980), 239–309.

[Sh] Y. Shiloach, "A Polynomial Solution to the Undirected Two Paths Problem," *Journal of the ACM* 27 (1980), 455-456.

[ST1] P. D. Seymour and R. Thomas, "Graph Searching and a Minimax Theorem for Treewidth," to appear.

[ST2] ———————, "Call Routing and the Rat-Catcher," to appear.

[Ya] X. Yan, "Approximating the Pathwidth of Outerplanar Graphs," M.S. Thesis, Washington State University, 1989.