Bounded Combinatorial Width and Forbidden Substructures

by

Michael John Dinneen
B.S., University of Idaho, 1989
M.S., University of Victoria, 1992

A Dissertation Submitted in Partial Fulfillment
of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

We accept this dissertation as conforming
to the required standard

Dr. Michael R. Fellows, Supervisor (Department of Computer Science)

Dr. Hausi A. Müller, Department Member (Department of Computer Science)

Dr. Jon C. Muzio, Department Member (Department of Computer Science)

Dr. Gary MacGillivray, Outside Member (Department of Math. and Stats.)

Dr. Arvind Gupta, External Examiner (School of Computing Science, Simon Fraser University)

Supervisor: M. R. Fellows

Abstract

A substantial part of the history of graph theory deals with the study and classification of sets of graphs that share common properties. One predominant trend is to characterize graph families by sets of minimal forbidden graphs (within some partial ordering on the graphs). For example, the famous Kuratowski Theorem classifies the planar graph family by two forbidden graphs (in the topological partial order). Most, if not all, of the current approaches for finding these forbidden substructure characterizations use extensive and specialized case analysis. Thus, until now, for a fixed graph family, this type of mathematical theorem proving often required months or even years of human effort. The main focus of this dissertation is to develop a practical theory for automating (with distributed computer programming) this classic part of graph theory. We extend and (more importantly) implement a variation of the seminal work done by Fellows and Langston regarding computing finite-basis characterizations.

The recently celebrated Robertson–Seymour Graph Minor Theorem establishes that many natural graph families are characterizable by a *finite* set of graphs. In particular, if a graph family is closed under the three basic minor operations (i.e., isolated vertex deletions, edge deletions, and edge contractions) then there exists (by a nonconstructive argument) a finite set of forbidden graphs. Two examples are the well-known k-Vertex Cover and k-Feedback Vertex Set graph families. In this dissertation, we characterize, for the first time, these parameterized families, among others, for small k.

Our forbidden graph computations use a restricted search space consisting of graphs of bounded combinatorial width (where pathwidth and treewidth are two important metrics). Using an algebraic enumeration scheme for graphs, we have implemented a terminating algorithm that *will* find all minor-order forbidden graphs for each fixed pathwidth. For a targeted graph family, this algorithm requires a mathematical description given in one of many acceptable forms (or combinations thereof), such as a finite-index congruence or a set of automaton-generating tests. Our main assumption is that an upper bound on the pathwidth (or treewidth) of the largest forbidden graph of a particular graph family is more readily available than its order (or size).

A byproduct of our bounded width approach is that we give practical linear time membership algorithms in the form of dynamic programs (over parsed graph structures of bounded width) for several graph families (e.g., k-MAXIMUM PATH LENGTH and OUTER PLANAR).

Examiners:

Dr. Michael R. Fellows, Supervisor (Department of Computer Science)

Dr. Hausi A. Müller, Department Member (Department of Computer Science)

Dr. Jon C. Muzio, Department Member (Department of Computer Science)

Dr. Gary MacGillivray, Outside Member (Department of Math. and Stats.)

Dr. Arvind Gupta, External Examiner (School of Computing Science, Simon Fraser University)

Contents

List of Figures

List of Tables

Acknowledgements

I acknowledge my thesis advisor Dr. Mike Fellows for introducing the topic for this dissertation and being enthusiastic about the results. Special thanks are due to my friend Dr. Kevin Cattell who contributed several years of C++ programming and research towards the realization of our software project. Without him, many of the following results would still be on the *waiting list*. I thank my Los Alamos mentor Dr. Vance Faber for providing me with an opportunity to do research in a productive atmosphere (including access to the vast computer resources at *The Laboratory*). I thank Dr. Arvind Gupta for being my external examiner and for his detailed comments. I also thank all of my friends at the University of Victoria (notably, Todd Wareham) and in the United States for the relaxing times away from research. Lastly, I appreciate my family in Idaho for enduring my transient lifestyle.

Kevin Cattell assisted me in the following areas:

- He programmed all of the database code, the t-parse canonicity algorithm, some of the primitive t-parse algorithms (e.g., our edge-contraction algorithm on page 61), and several shell scripts.
- He contributed valuable research towards our vertex cover characterizations [CD94] (which includes some of the results of Chapter 4), vigorously proof-read our [CDF95a] paper, and helped prove that our pathwidth algorithm runs in linear time [CDF95b].
- He monitored (jointly) several obstruction set searches, and often fine tuned the software for better efficiency. (Some of our searches required many time-consuming restarts and took several months of CPU time to complete.)

Chapter 1

Introduction

One of the most famous results in graph theory is Kuratowski's characterization of planar graphs: a graph is planar if and only if it "contains" neither the complete bipartite graph $K_{3,3}$ nor the complete graph K_5. The *obstruction set* for planarity thus consists of these two graphs, which are displayed below in Figure 1.1. These graphs are "almost" planar (or the "smallest" non-planar).

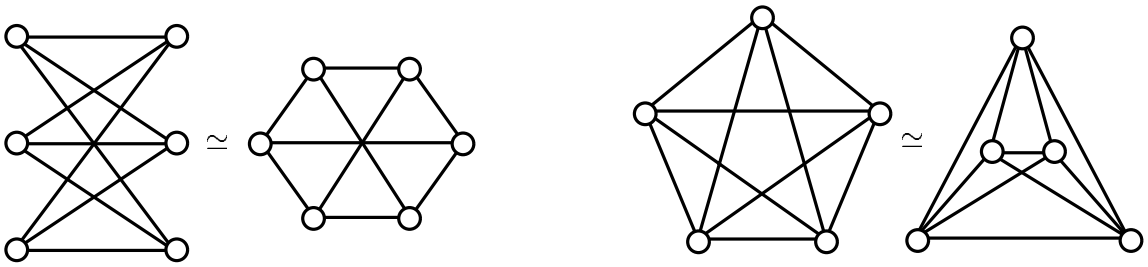


Figure 1.1: Kuratowski's planar obstructions.

This example indicates the form of all obstruction set characterizations of graph families; for some fixed graph family, denoted by $\mathcal{F}$, a graph G is a member of $\mathcal{F}$ if and only if G does not contain (as a substructure) any member of some set of graphs $\mathcal{O}(\mathcal{F}) = \{O_1, O_2, \ldots\}$. The precise meaning of "substructure" may vary. For these types of characterizations, graphs are related by some *partial order* as illustrated in Figure 1.2 by a Hasse diagram where each node (representing some graph) is structurally contained in the ones connected above it. Here, any non-family graph

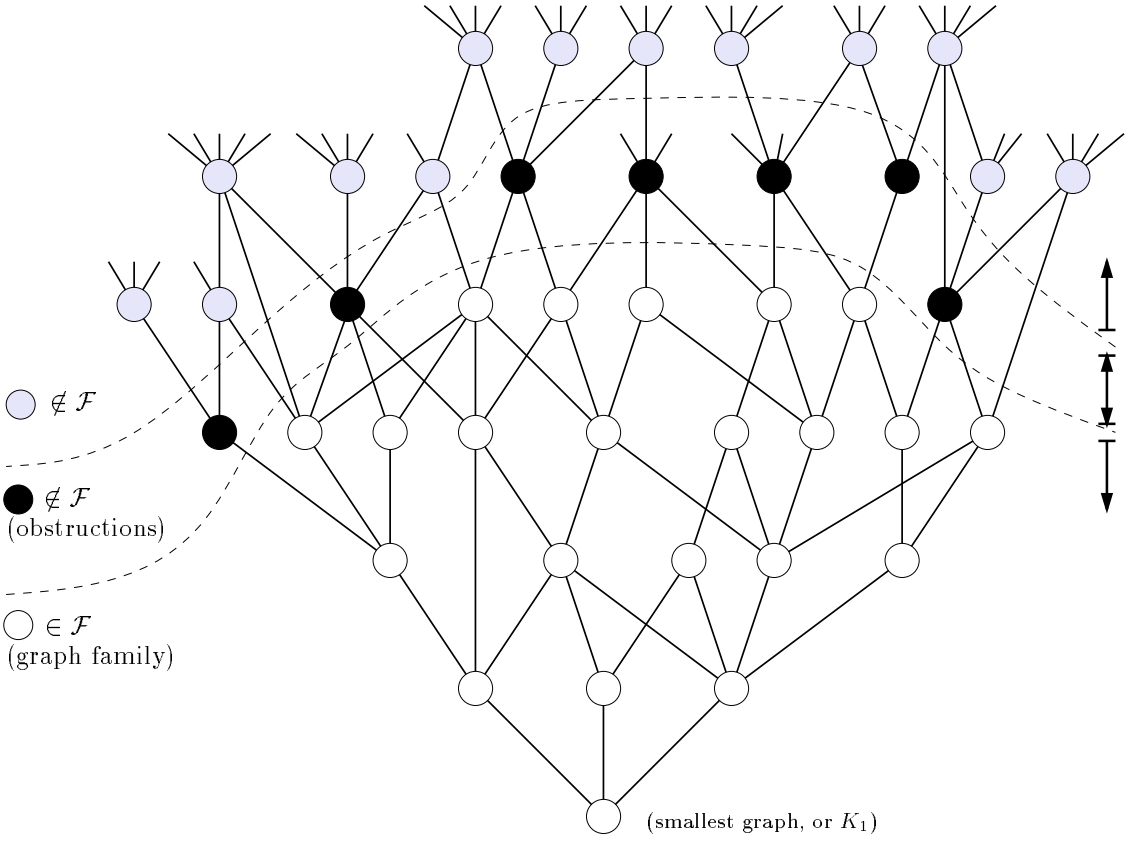


Figure 1.2: Illustrating a partial order of graphs and a family's obstructions $\mathcal{O}(\mathcal{F})$.

above the lower dashed line (i.e., any graph represented by a black or gray node) contains at least one obstruction (black node). Notice that, by definition, these types of orderings of graphs are transitive in the upward direction even though all relationships are not shown in the figure. Each edge in Figure 1.2 can be viewed as one primitive graph reduction between a bigger graph and a smaller graph. Different combinations of these reductions on a given graph may yield the same graph (e.g., note that edges can cross over the obstruction set boundary).

Classifying graphs by forbidden substructures has been an important part of research in graph theory ever since its inception, and there are many characterization theorems of this kind. Perhaps a familiar example is the set of acyclic graphs (i.e., those graphs without a self-connecting path of edges). Here, the forbidden subgraphs are the individual cycles of various lengths $\{C_i \mid i \geq 3\}$. If we add more structure to this subgraph partial order then there is just one forbidden substructure, namely

the smallest cycle C_3. This is accomplished by adding subdivided edge relationships on graphs (i.e., a graph with a vertex inserted on some edge is structurally above the original graph). We now present two famous examples based on two different partial orders. (These orders are formally defined later in Chapter 3.)

The first graph embedding analog to Kuratowski's characterization of planar graphs was presented in [GHW79b] for the real projective plane (Möbius band + lid). This simplest surface after the plane (sphere) can be viewed as a plane with a circular boundary where diagonally opposite boundary points are identified (e.g., see [FG93, Sti93]). Below in Figure 1.3 we easily show graph embeddings (without edge crossings) of both the planar obstructions $K_{3,3}$ and K_5; this indicates that the family of projective-planar graphs is larger than the family of planar graphs. For this non-orientable surface, an obstruction set of 103 irreducible graphs of [GHW79a] in the topological (homeomorphic) partial order was proven complete in [Arc80]. (From these 103 obstructions there are 35 minor-order forbidden graphs [Arc95, Mah80].)

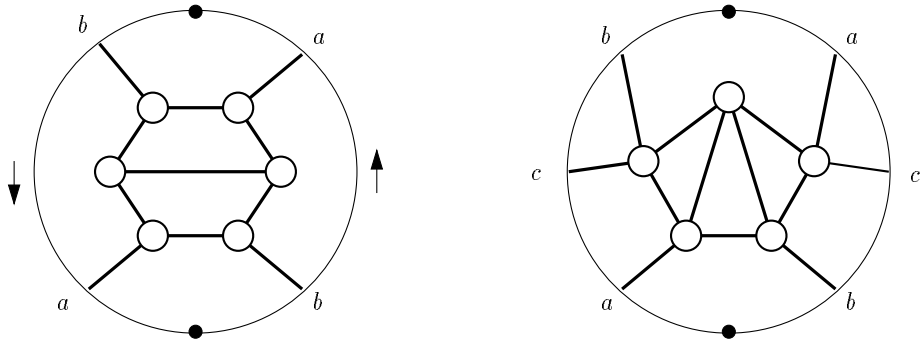


Figure 1.3: Projective plane embeddings of Kuratowski's planar obstructions.

Our next example is drawn from computational biology. We consider a graph to be *chordal* if all induced cycles of length four or greater contain an internal cross edge (chord). An independent triple of vertices is called an *asteroidal triple* if between every pair in the triple there exists a path that avoids the neighborhood of the third. The four forbidden asteroidal triples (see Figure 1.4) and the simple cycle C_4 prevent a graph from being represented as an intersection of interval line segments [Weg67, Fel89]. This finite classifying set of minimal graphs is obtained from the general

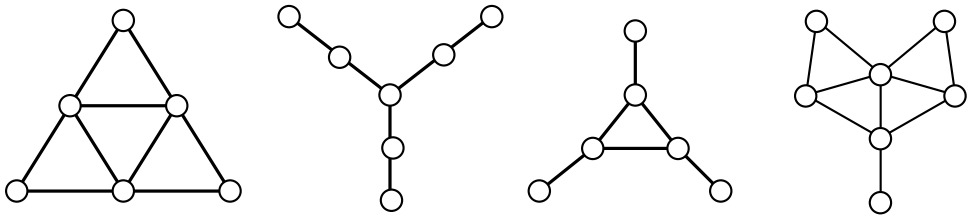


Figure 1.4: Induced forbidden chordal graphs (asteroidal triples).

infinite family of asteroidal triples; this earlier result (based on the vertex-induced subgraph partial order) showed that *interval graphs* are characterized as graphs that are chordal and free of asteroidal triples [LB62, Mir94].

Several other examples of classifying graphs by forbidden substructures can be found in [APC87, CKK72, Fis85, LPS95, MS83, Pro93, RS90c]. Most of the known obstruction set characterizations were difficult to prove, and proven solely by classical paper-and-pencil methods (e.g., see [KL94, GH79]). This situation is now changing as computers are being incorporated into the research process. For example, the elusive target of finding the obstruction set for the family of torus embeddable (doughnut's surface embeddable) graphs now appears to be closer by the new theories of automated mathematical theorem-proving.

This dissertation is motivated by the recent results of Robertson and Seymour [RS84a] on the well-partial-ordering of graphs under the minor (and other) orders. They have non-constructively established that many natural graph properties have "Kuratowski-type" characterizations. That is, these graph families can be characterized by finite obstruction sets in an appropriate partial order. (The first few papers of their comprehensive graph minors series may be found in [RS83]–[RSf].) Our work has focused on finding obstruction sets for these types of characterizable graph families.

The following sections survey our computational approach for finding obstruction sets. We begin with a review of the minor partial order for graphs. Then in Section 1.2 we introduce two kinds of k-parameterized graph families that form the basis of many of our graph family characterizations. Next in Sections 1.3 and 1.4 we explain how to efficiently search for obstruction sets within a bounded combinatorial width domain. At the end of this introduction we outline the remaining chapters of this dissertation.

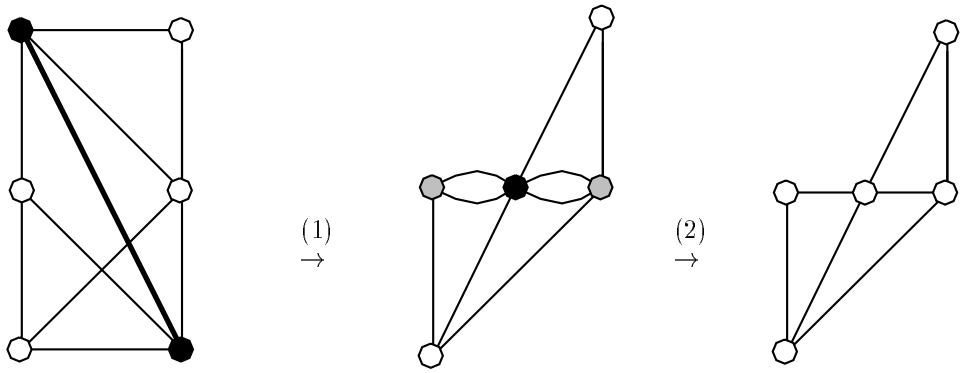
1.1 The Graph Theory Setting

The reader is assumed to have a familiarity with graph theory, as may be obtained from any one of these introductory graph theory texts [BM76, CL86, HR90, Tru93, Wil79]. We now introduce a few of our basic graph-theoretical conventions.

We are primarily interested in finite simple undirected graphs (i.e., those finite graphs without loops and multi-edges). A *graph* $G = (V, E)$, in our context, consists of a finite set of *vertices* V and a set of *edges* E, where each edge is an unordered pair of vertices. We use the variables n (sometimes $|G|$) and m to denote the *order* (number of vertices) and *size* (number of edges) of a graph. A graph is *connected* if there exists a path between every two vertices. A *tree* is a connected graph with $m = n - 1$, that is, it contains no cycles but has a unique path between every two vertices. Other formal graph-theoretic definitions are presented later as needed.

This dissertation focuses on a partial order based on graph minors. Specifically, a graph H is a *minor* of a graph G, denoted $H \leq_m G$, if H can be obtained by contracting some (possibly zero) edges in a subgraph of G. Figure 1.5 illustrates the edge contraction operation. Recall that any partial order $\leq$ of graphs, including the minor order $\leq_m$, satisfies three key properties. It is reflexive ($G_1 \leq G_1$), transitive (if $G_1 \leq G_2$ and $G_2 \leq G_3$ then $G_1 \leq G_3$), and anti-symmetric (if $G_1 \leq G_2$ and $G_2 \leq G_1$ then $G_1 = G_2$).

The most important application of the celebrated *Graph Minor Theorem* (GMT) by Robertson and Seymour, formerly known as Wagner's Conjecture, is that many graph families are easily shown to have polynomial time membership algorithms. Here, the GMT guarantees that many graph families have only a finite number of obstructions. These membership algorithms are based on an $O(n^3)$ algorithm that can check if a fixed graph is a minor of another graph [RS85b]. Hence, in theory, to check for membership in an applicable graph family $\mathcal{F}$, one just runs this polynomial time minor checking algorithm once for each obstruction in $\mathcal{O}(\mathcal{F})$. An input graph G is a member of $\mathcal{F}$ if and only if G has none of these obstructions as a minor. Thus, once the "promised" finite set of obstructions $\mathcal{O}(\mathcal{F})$ is found for $\mathcal{F}$, a constructive membership algorithm exists. However, it is still open whether these new algorithms can be made practical. This is because there are astronomically large hidden constants



(1) First collapse edge while preserving adjacencies.
(2) Then remove any incident multi-edges that occur.

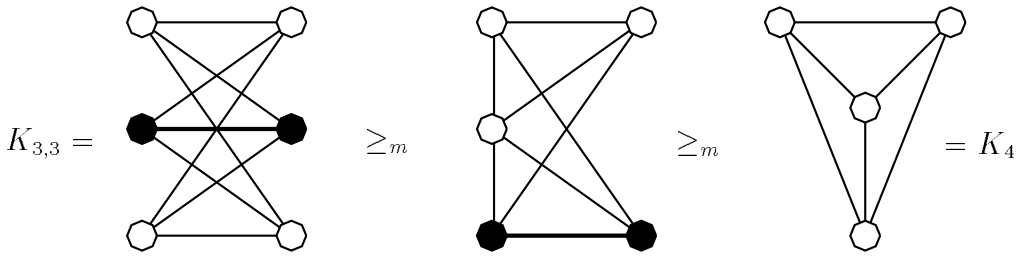


Figure 1.5: Demonstrating the 'edge contraction' operation for the minor order.

in this cubic time minor-containment complexity result [FRS87]. Also the number of obstructions may be prohibitively large.

To indicate what types of graphs interest us, we investigate a contrived (but non-trivial) graph family. Recall that the *independence* of a graph is the maximum number of mutually non-adjacent vertices (i.e., the cardinality of the largest independent set). It is easy to show that any minor of a graph with independence + size $\leq k$ also satisfies this inequality. For example, after deleting an edge from a graph the independence can increase by at most one, implying that the sum will not increase. For any fixed k, let us call this graph family k–Edge Bounded IndSet. If one does not observe that this family has a finite number of members, recognizing graphs in k–Edge Bounded IndSet may seem to require a running time of $O(n^{k-m})$. That is, where the size m has been previously computed, an algorithm could check to see if any of the vertex subsets of cardinality $k - m$ is an independent set. An application

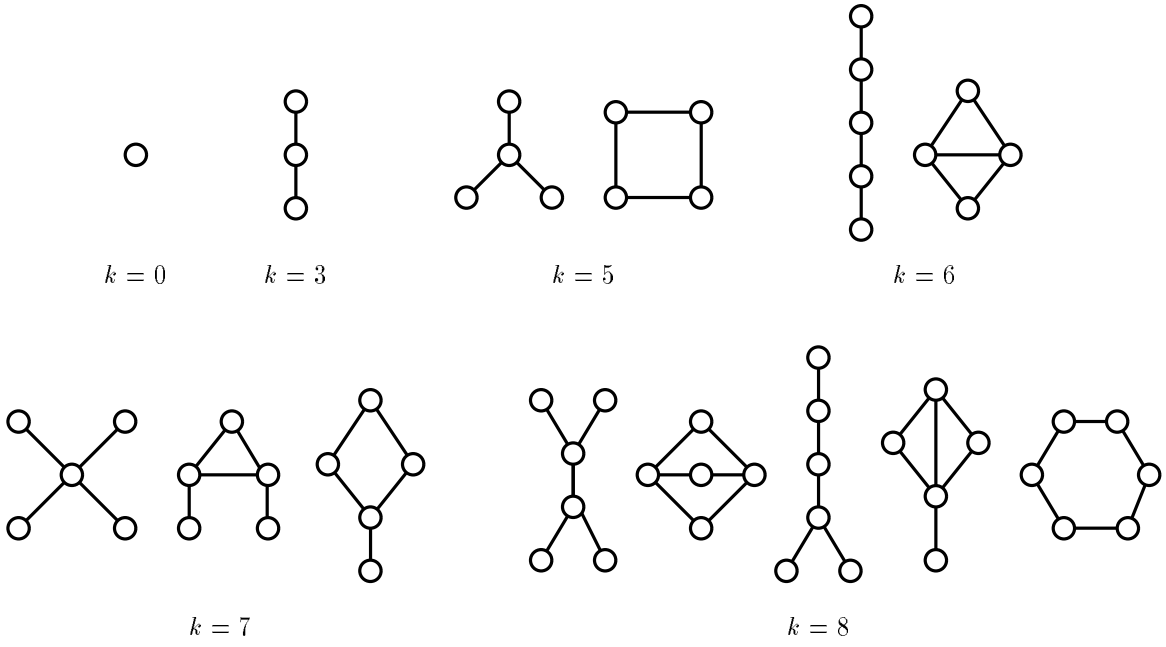


Figure 1.6: All connected minor-order obstructions for k–Edge Bounded IndSet, for $k = 0, 1, \ldots, 8$.

of the GMT improves this brute force algorithm to $O(n^3)$ time, based on checking a finite set of forbidden minors.

For each small fixed k, an obstruction set for k–Edge Bounded IndSet is easily found by the following logic: We know that $k+1$ isolated vertices is an obstruction, so $k+1$ also bounds the order of the largest obstruction. We display all the connected obstructions for 0–Edge Bounded IndSet through 8–Edge Bounded IndSet in Figure 1.6. One of our general results of Chapter 4 shows how disconnected obstructions are obtained. For example, the union $P_2 \cup K_1$ of a path of length 2 and an isolated vertex is an obstruction for 4–Edge Bounded IndSet.

We can sometimes exploit other properties of graph families to reduce the above mentioned polynomial time complexity. We use the popular notions of treewidth and pathwidth, also introduced by Robertson and Seymour [RS83, RS84c], to combinatorially bound certain subsets of a graph family. Informally, graphs of width at most k are subgraphs of tree-like or path-like graphs where the vertices form cliques of size at most $k+1$. (These combinatorial widths are discussed in detail in Chapter 2.) When restricted to graphs of treewidth (or pathwidth) at most k, there exists a linear

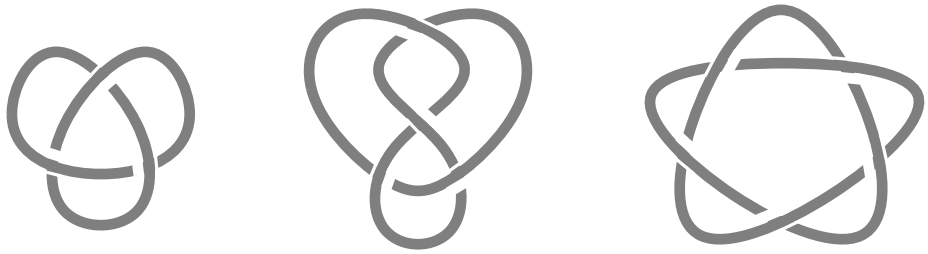


Figure 1.7: Some non-trivial knots.

time algorithm to check if a fixed graph is a minor of the input [ALS91]. That is, these algorithms run in time $O(f(k) \cdot n)$ where f is some computable function. A well-noted example, recently improved by Bodlaender, is that all finitely characterizable (minor order) graph families that exclude at least one planar graph have linear time membership algorithms [Bod93a]. That is, for a fixed planar graph P, either a graph has P as a minor or has treewidth at most some function of $|P|$ [RS90a].

For a concrete example of a guaranteed linear time membership algorithm, consider our k–Edge Bounded IndSet families. We see that all of these families have a planar obstruction (e.g., the planar graph consisting of $k+1$ isolated vertices). Thus, by Bodlaender's result there exist linear time recognition algorithms for these graph families. Also known is a linear time independent set algorithm for graphs of bounded treewidth (e.g., see [Bod93c]).

As mentioned earlier, a constant time membership algorithm is known for each k–Edge Bounded IndSet graph family since each has a finite (function of k) number of graph members. In particular, if more than k edges are read for an input graph G then we know that G is not a member of k–Edge Bounded IndSet. So the yes-instance input must be of constant size.

A remarkable consequence of the GMT is that for some graph families that were not even known to be decidable, the GMT can be applied to prove the existence of a polynomial time recognition algorithm (see [Fel89] for a nice survey). For example, consider the set of graphs that can be embedded in 3-dimensional space such that every cycle is *not* knotted (in the everyday sense as depicted by the knots in Figure 1.7). It is not hard to see that this family of *knotless graphs*, which contains the planar graph family, is closed under minors (i.e., taking a subgraph or contracting an edge

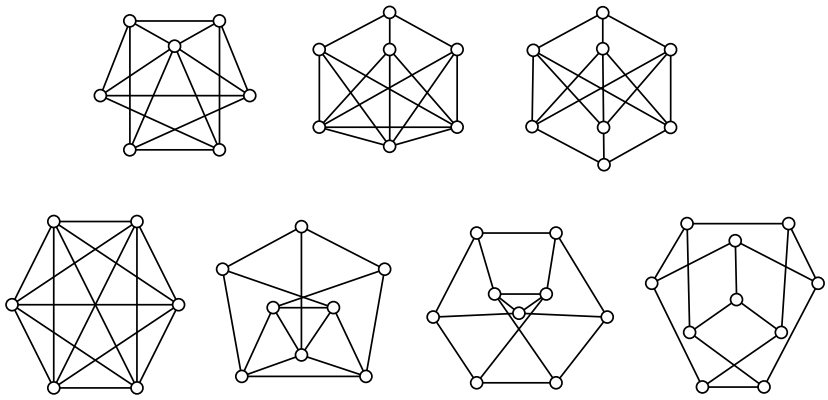


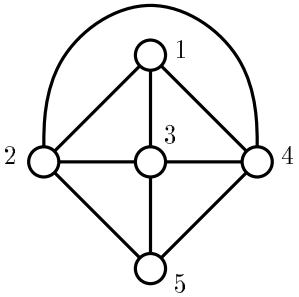
Figure 1.8: The minor-order obstructions for the linkless graph family.

can not create a knot with respect to a knotless embedding). The complete graph K_7 is an example of a graph that contains a least one nontrivial knot no matter how it is embedded in 3-space [CG83]. However, it is not known if the 4-partite graph $K_{3,3,1,1}$ has a knotless embedding [Tho95]. The promised finite obstruction set for this family is still unknown. However, a related "knot theory" family of graphs, those graphs with a *linkless 3-space embedding*, has recently been characterized by the seven forbidden minors shown in Figure 1.8 [MRS88, RST95a, RST95b, RST95c].

1.2 Some Finitely Characterizable Graph Families

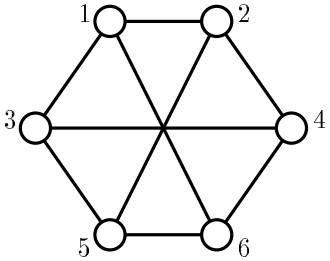
As a consequence of the Graph Minor Theorem, any family of graphs $\mathcal{F}$ that is *closed* under minors (i.e., if $H \leq_m G$ and $G \in \mathcal{F}$ then $H \in \mathcal{F}$) has an obstruction set $\mathcal{O}(\mathcal{F})$ of finite cardinality. More generally, a family of graphs $\mathcal{F}$ that is closed relative to a partial order is called a *lower ideal*.

Like the k–Edge Bounded IndSet families, many minor-order lower ideals are parameterized by an integer k. The previously mentioned planar and toroidal graph families are surface-embeddable families for fixed genus (i.e., the number of sphere handles) $k = 0$ and $k = 1$, respectively [CL86, Tru93]. In fact, our principle graph families (e.g., k–Vertex Cover and k–Feedback Vertex Set), for which our computational technique is illustrated, fall into this framework. A k-parameterized



VC = $\{2, 3, 4\}$

FVS = $\{2, 3\}, \{2, 4\}$ or $\{3, 4\}$



VC = $\{1, 4, 5\}$ or $\{2, 3, 6\}$

FVS = $\{1, 4\}, \{1, 5\}, \{2, 3\},$
$\{2, 6\}, \{3, 6\}$ or $\{4, 5\}$

Figure 1.9: Two graphs that are members of both the 3–Vertex Cover and 2–Feedback Vertex Set graph families.

lower ideal $\mathcal{F}$ is defined by some type of graph invariant function λ that maps graphs to integers such that whenever $H \leq_m G$ we also have $\lambda(H) \leq \lambda(G)$. In this general framework, a graph family $\mathcal{F}$ equals $\{G \mid \lambda(G) \leq k\}$ for some integer constant k.

Figure 1.9 illustrates two members of both the graph families 3–Vertex Cover and 2–Feedback Vertex Set by displaying each minimum vertex cover (VC) and feedback vertex set (FVS). The family 3–Vertex Cover is the set of graphs that have a VC of cardinality at most 3 (i.e., every edge is incident to at least one of the three "cover" vertices VC). Likewise, a graph G is in 2–Feedback Vertex Set if there exists an FVS with two vertices u and v such that $G \setminus \{u, v\}$ is acyclic (i.e., this subgraph contains no cycles).

Given a minor-order lower ideal $\mathcal{F}$, often (but not always) another class $\{k\text{–}\mathcal{F} \mid k > 0\}$ of parameterized and finitely-characterizable graph families is obtained by defining lower ideals that are within k vertices (or edges) of $\mathcal{F}$. That is, for a fixed family $\mathcal{F}$, the following parameterized families are lower ideals (e.g., see [FL88b] for proof):

$$k\text{–}\mathcal{F}_v = \{G = (V, E) \mid G \setminus V' \in \mathcal{F} \text{ where } V' \subseteq V \text{ and } |V'| \leq k\} \quad .$$

And the following families may also be lower ideals:

$$k\text{–}\mathcal{F}_e = \{G = (V, E) \mid G \setminus E' \in \mathcal{F} \text{ where } E' \subseteq E \text{ and } |E'| \leq k\} \quad .$$

Table 1.1: Number of connected and disconnected minor-order obstructions for k–Edge Bounded IndSet, for $k = 0, 1, \ldots, 12$.

$k =$	0	1	2	3	4	5	6	7	8	9	10	11	12
connected	1	0	0	1	0	2	2	3	5	12	21	42	86
disconnected	0	1	1	1	2	2	4	7	10	17	31	58	105
total obstructions	1	1	1	2	2	4	6	10	15	29	52	100	191

For example, in Chapter 9 we study the lower ideal 1–Outer Planar that is based on those graphs that are "within one vertex" of the family of outer-planar graphs. (Unfortunately, the corresponding family of graphs that are at most one edge away from being outer-planar is not a minor-order lower ideal.) In Chapter 7, we look at the lower ideals k–Feedback Edge Set that are those graphs that are "within k edges" of acyclic (i.e., each graph in this family has a set of at most k edges when removed produces a *forest*).

There is a tendency for the number of obstructions for natural parameterized families to grow explosively as a function of the parameter k. For example, the number of minor-order obstructions for k–Pathwidth (i.e., graphs with pathwidth at most k) is 2 for $k = 1$, 110 for $k = 2$, and provably more than 60 million for $k = 3$ [KL94]. Even looking at our small k–Edge Bounded IndSet graph families, the counts given in Table 1.1 indicate exponential growth in the number of obstructions. One goal of our research goal is to explore the following working hypothesis:

Natural forbidden substructure theorems of feasible size are feasibly computable.

We suspect that the toroidal obstruction set, which has a projected size of about 3000 graphs, is feasibly computable by the methods of this dissertation.

Of course, if a lower ideal is not parameterized, the feasibility of computing its obstruction set will have to be based on other criteria. The family of knotless graphs is one such interesting example where we need an alternate method to predict the size of the obstruction set. Again, in any of these non-parameterized cases, if the obstruction set is small (and in conjunction with a few other family-specific

ingredients), we conjecture that it should be feasibly computable.

1.3 A Historical Perspective on Computing Obstructions

Most of the previous techniques for computing obstructions have been specific to a particular lower ideal. In particular, several Ph.D. dissertations focus on characterizing just one graph family and use strictly mathematical case analysis (see for example [Arc80, Kin89, Sar89]). For some of these characterizations computers are used, for the most part, to mainly aid in the verification process, usually by weeding through a manageable known set of potentially minimal forbidden graphs.

The basic results for our general-purpose computational approach originated from the foundational paper by Fellows and Langston [FL89a], which used the Graph Minor Theorem to prove termination of a finite-state search procedure. This approach for computing obstruction sets requires additional problem-specific results. These results are nontrivial, but seem to be generally available (in one form or another) for virtually every natural lower ideal $\mathcal{F}$. The Fellows–Langston approach for computing obstructions requires the following three ingredients:
(i) a decision algorithm for $\mathcal{F}$,
(ii) a finite-index congruence that refines the canonical congruence for $\mathcal{F}$, and
(iii) a bound on the maximum treewidth of the obstructions for $\mathcal{F}$.
The above mentioned "regular language" canonical congruence is introduced in the next section.

In general, we know that having only the first ingredient is not sufficient to compute the obstruction set for an arbitrary lower ideal $\mathcal{F}$ (e.g., see [vL90, FL89b]). That is, from only a decision algorithm for $\mathcal{F}$, there is *no* algorithm to compute $\mathcal{O}(\mathcal{F})$. We conjecture that having the first two ingredients may be sufficient [CDDF95]. This dissertation uses these three basic ingredients but in a more feasible way than the "can be computed" results of [FL89a].

We now mention two other successful efforts concerning the theory of computing obstructions. Lagergren and Arnborg in [LA91] constructively show (i.e., without

the GMT) that every bounded-treewidth minor-order lower ideal with a finite-index family congruence has a finite number of obstructions. Also for bounded treewidth, Courcelle has indirectly proved that the obstructions for any lower ideal defined by a second-order monadic logic formula can be computed; he showed a method of producing the above mentioned finite-index congruence from this formula [Cou90a]-[Cou92b]. We do not believe that either of these theoretical methods has been explored in the implementation sense.

One of the goals for this dissertation is to bridge the gap between the theory of obstruction set computations and their practical implementations.

1.4 An Overview of Our Computational Technique

This dissertation describes a basic theory for computing obstruction sets, which is partially automatable for any minor-order lower ideal. We have successfully used our technique to find obstruction sets for several types of graph families and these are presented in Part II of this dissertation.

We now sketch our method for computing obstruction sets. To find minor-order obstructions it is sufficient to find the obstructions for a slightly relaxed partial order called the *boundary minor order* (for graphs with a constant number of labeled vertices denoting the "boundary"). It is easy to extract ordinary obstructions from the boundaried obstructions.

We have an enumeration scheme that generates all of the boundaried graphs for a specified maximum pathwidth (or treewidth). Our tree-like enumeration scheme has an important property that allows us to prune at dead-ends during the search process. We have a "prefix property" on boundaried graphs with respect to our enumeration order that ensures that it is safe to prune the search space this way. Thus only fruitful branches to boundaried obstructions are taken, thereby reducing the overall computation time. Our enumeration tree is obtained from a partial order of boundaried graphs. This order is different than the "substructure" partial order (in this case, the boundary minor order) that was illustrated earlier in Figure 1.2.

Figure 1.10 shows the beginning of an enumeration tree (and partial order) for

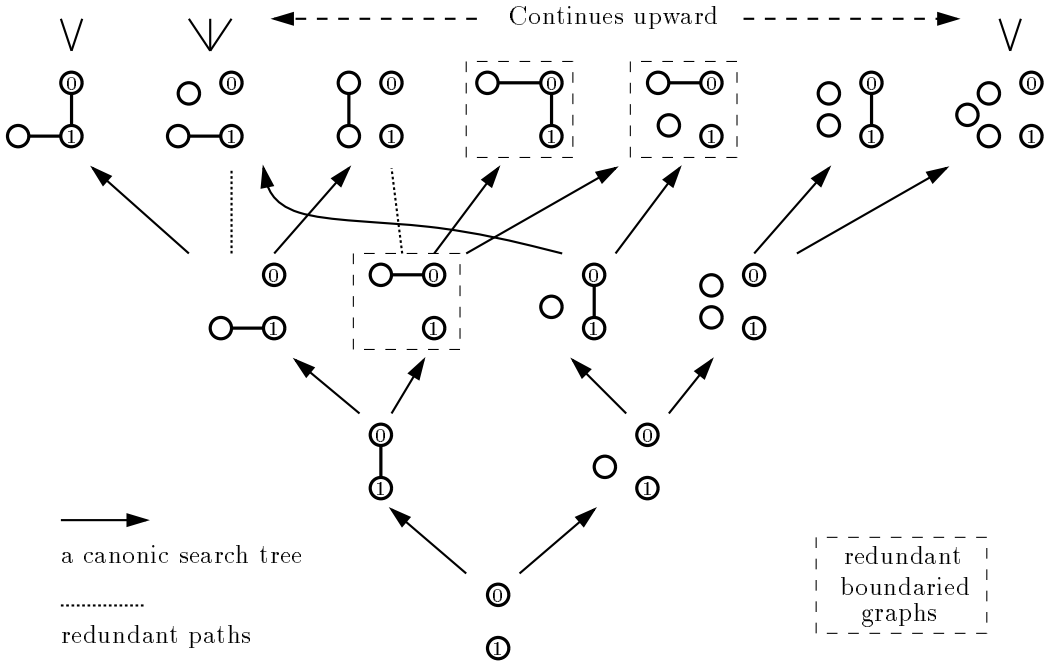


Figure 1.10: Example of our enumeration order for graphs of pathwidth 1.

graphs of pathwidth 1. Within this order we grow the search tree by following the "canonic" paths upward. Also, a graph may be assigned a "redundant" label if every obstruction above that graph is obtainable by expanding out from another graph. This initial (underlying) search tree is independent of any specific lower ideal that we are trying to characterize.

For our main prefix pruning process, we require one or more family-specific algorithms. These easily obtainable requirements are used to determine "graph minimality." Here, with respect to a finite-index congruence for a lower ideal $\mathcal{F}$, a boundaried graph G is *minimal* if no boundaried minor H of G is in the same equivalence class as G. Our enumeration tree of boundaried graphs has the *prefix property* that every parent of a minimal graph is also minimal.

In order to define graph minimality, we use an equivalence class on boundaried graphs (of the same boundary size) with respect to a lower ideal $\mathcal{F}$ as follows. First we "glue together" two boundaried graphs with the binary operator $\oplus$, by coalescing boundary vertices with the same label. Then two graphs G and H are in the same equivalence class of this *canonical congruence* if and only if for every boundaried

graph Z (called an *extension*),

$$G \oplus Z \in \mathcal{F} \iff H \oplus Z \in \mathcal{F} \quad .$$

For the set of graphs of bounded treewidth (or pathwidth), this canonical congruence is finite-index for any minor-order lower ideal (e.g., see [AF93]).

During our search, any graph above a nonminimal graph in the enumeration order (recall Figure 1.10) is not generated. Thus, many graphs both in and out of the targeted lower ideal are avoided altogether. Because of our prefix property and the GMT, we know that only a finite number of graphs will be enumerated for each bounded-width search. Thus, if the largest width of any obstruction is known (for a targeted lower ideal) then we can find the complete obstruction set. For example, we know that the obstructions for the k–Vertex Cover graph family have pathwidth at most $k+1$.

For a flavor of what our search trees looks like, we now give a concrete example. The exact details of what follows may not be totally clear at this juncture. Our enumeration tree for the 5–Edge Bounded IndSet family is shown in Figure 1.11. The "#node" labels are indices into a search database and are sequenced in enumeration order. The white nodes, both the circles (in family) and squares (out of family), represent minimal graphs (as defined above). That is, the five white squares denote the boundaried minor-order obstructions to 5–Edge Bounded IndSet. The two minor-order obstructions that where shown earlier in Figure 1.6 are #36 = $K_{1,3}$ and #65 = C_4. These can be extracted by following the "(#node, operator)" labels up the tree. The other three boundaried obstructions represent the same disconnected obstruction $P_2 \cup 2 \cdot K_1$ but with different boundaries. The root #0 represents the smallest boundaried graph consisting of three isolated boundary vertices. The missing "#node" numbers in this tree are either irrelevant (e.g., disconnected) or non-canonic/redundant graphs. The only nonminimal family graph, #35 = P_3, is congruent to one of its edge deleted minors ($K_1 \cup P_2$). Lastly, we point out that graph #24, the cycle C_3, is an example of a minimal graph that does not lead to some boundaried obstruction in the search tree.

Next we survey our four methods for determining graph minimality with respect to the canonical congruence for a lower ideal $\mathcal{F}$. A main contribution of this dis-

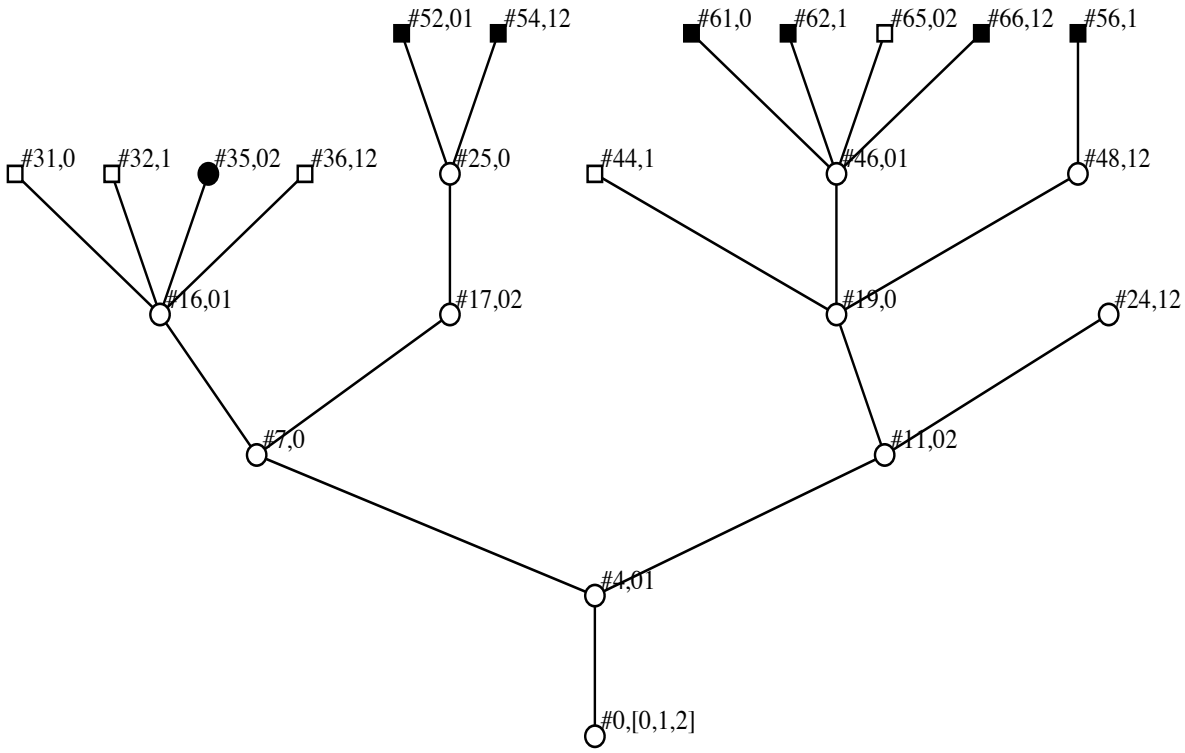


Figure 1.11: Actual search tree for the 5–Edge Bounded IndSet graph family's obstructions, pathwidth ≤ 2.

sertation is to combine these methods to make obstruction set computations more feasible.

Direct nonminimality checking: After testing for some structural or graph invariant property, we sometimes know when a graph has a congruent minor with respect to the canonical congruence. Often these properties are quickly computed. For example in the family k–Edge Bounded IndSet, we can show that a graph G with an isolated K_3 is nonminimal since the minor obtained by deleting one of its edges is congruent to G.

A finite-state dynamic program: Given an algebraic representation for a boundaried graph (of bounded width), we use a finite-state dynamic program to determine whether a graph G and a minor G' are congruent. These algorithms are designed so that whenever two input graphs fall in the same computation state, they are con-

gruent. By the nature of the dynamic program, both $G \oplus Z$ and $G' \oplus Z$ will fall in the same state for any boundaried graph Z. Unfortunately, in most cases, if two graphs fall into different states, we can not infer that they are not congruent. For our k–EDGE BOUNDED INDSET example, a congruence that can be implemented as a dynamic program is now presented. For a boundaried graph G and each vertex subset S of the boundary ∂ define the following partial function:

$$IndSet_G[S] = \max(|I| \leq k+1 \ : \ I \cap \partial = S \text{ and } I \text{ is an independent set})$$

Two boundaried graphs G and H are in the same equivalence class (in this refinement of the canonical congruence for k–EDGE BOUNDED INDSET) if

$$\min(|E(G)|, k+1) = \min(|E(H)|, k+1) \quad \text{and}$$

$$IndSet_G[S] = IndSet_H[S] \quad \text{for all } S \subseteq \partial \quad .$$

We later present (and prove correct) a linear time dynamic program for the independent set component of this congruence (see Chapter 4).

A random extension search: A random boundaried graph Z may be a witness to G not being congruent to a minor G'. That is, we search for an extension Z such that $G \oplus Z$ is not in $\mathcal{F}$ while $G' \oplus Z$ is in $\mathcal{F}$. Very little computer resources are needed for these types of checks. We repeat these random extension searches several times until each minor G' of G is shown to be in a different equivalence class than G's equivalence class (or until a "give-up" threshold is reached). Since $\mathcal{F}$ is a lower ideal, it is beneficial to find a random boundaried graph Z such that $G \oplus Z$ is not in $\mathcal{F}$ but for any minor Z' of Z, $G \oplus Z'$ is in $\mathcal{F}$. (This is done before checking if an extended minor $G' \oplus Z$ is in $\mathcal{F}$.)

Boundaried graph tests: We use a finite subset of boundaried graphs, called a *testset*, that is sufficient to determine if two graphs are in the same canonical equivalence class. That is, if two graphs G and H pass the same set of tests via $\oplus$ then they are congruent (for all extensions), otherwise they are not. A graph G *passes* a test Z whenever $G \oplus Z$ is in $\mathcal{F}$. (The reader may recall the Myhill–Nerode Theorem regarding regular languages.) Usually a testset for a lower ideal $\mathcal{F}$ is quite large. Thus, determining minimality or nonminimality for a particular graph may require many family membership checks. We can often reduce the cardinality of a testset

by observing a few properties of $\mathcal{F}$. For the family k–Edge Bounded IndSet, we know that any "useful" test requires at most k edges. Furthermore, for this lower ideal, any component C of a test Z without any boundary vertices can be replaced with $Independence(C) + |E(C)|$ isolated vertices. (This means that all edges of each test for this family should belong on some path to a boundary vertex.)

We use various selections of the above mentioned minimality/nonminimality methods (in Part II of this dissertation) to characterize several interesting graph families. As alluded to earlier, finding obstruction set characterizations is (in principle) one route for establishing constructive versions of many of the known applications of the Robertson–Seymour results.

1.5 A Survey of The Dissertation

The primary and secondary research contributions in this dissertation are:

1. Finding obstruction set characterizations for various families of graphs using newly developed computational techniques.
   - Prove usable combinatorial width bounds for lower ideals.
   - Introduce practical graph enumeration schemes (using algebraic representations of graphs).
   - Use compact finite-index family congruences.
   - Provide canonical congruences in the form of testsets.
   - Utilize a randomized obstruction-set search strategy.
2. Developing efficient algorithms for graphs of bounded combinatorial width.
   - Give dynamic programs for several graph families.
   - Prove a simple linear time algorithm for finding path decompositions of small width.
   - Show how family-membership automata for graphs can be constructed from bounded-width finite-index congruences.

For the first research area (i.e., our more glamorous and developed area), we implemented an automated technique for finding obstruction sets. In the next four chapters we describe some of our practical experience in realizing this goal (e.g., what new theory was needed). In the four chapters that follow this general theory, we illustrate our approach by characterizing several graph families. We also present several required family-specific results that were needed for the completion of these obstruction set computations,

The second research area is more applied in nature. It has been known in several research circles that efficient graph algorithms often exist when the input is restricted to graphs of bounded combinatorial width. This is contrasted with the general case where the same computational problems appear to be intractable. Many automated techniques based on formal problem descriptions exist, in theoretical form only, for producing these types of bounded-width algorithms (e.g., see [Bor88, Cou87]). Until these approaches are developed, we rely on the successful dynamic-programming methods (e.g., see [ALS91, Bod88a, CSTV92]). This dissertation contributes new practical algorithms to this list. That is, for each of our obstruction set computations, we also give a corresponding linear time membership algorithm for parsed graphs of bounded pathwidth.

A survey of the remaining chapters of this three part dissertation is given below. At this time, we point out that some of the material of this dissertation has appeared elsewhere, notably Chapters 6, 7 and 12 in the papers [CD94], [CDF95a] and [CDF95b], respectively. After a concluding Chapter 13, an annotated bibliography is given for the sake of completeness. An index is also included at the end.

The first part of this dissertation contains our basic theory for computing obstruction sets based on a bounded combinatorial width search space. Chapter 2 shows how graphs can be algebraically represented in this bounded width domain. Chapter 3 provides the set-theoretic background needed for finding minor-order obstructions. Chapter 4 contains our main theory for finding obstructions, with implementation aspects in mind. Chapter 5 concludes Part I with a brief description of our software and a selection of future research topics.

The second part of this dissertation illustrates our computational technique for finding obstruction sets. This coverage spans several graph families. We begin in Chapter 6 by characterizing the well-known families k–VERTEX COVER, for $k = 1, 2, \ldots, 5$. Next in Chapter 7, we illustrate the use of Myhill–Nerode testsets (consisting of boundaried graph tests) to compute obstruction sets for two types of "almost tree" families of graphs (i.e., k–FEEDBACK VERTEX SET and k–FEEDBACK EDGE SET). In the third chapter in this series, namely Chapter 8, we generalize the lower ideals of the previous two chapters by studying those graphs with small path covers and cycle covers. To round out our presentation we give in Chapter 9 some initial results on computing obstructions for surface embedable graph families, with the main focus on the family 1–OUTER PLANAR.

The third part of this dissertation presents several applications-oriented topics concerning finite state automata and bounded combinatorial width. Each of these chapters is independent from the remaining parts of this dissertation and can be digested without a full understanding of our obstruction set computation theory. Chapter 10 contains a link between computational biology and VLSI layout problems in terms of bounded pathwidth. Chapter 11 develops some complexity results regarding minimizing testsets for automata and shows how automata can be used to compute obstruction sets. Lastly, Chapter 12 presents a very simple linear time algorithm for finding small path decompositions of a graph (or determines that a graph has large pathwidth).

Part I

The Basic Theory

Chapter 2

Bounded Combinatorial Width

We are interested in certain subsets of the set of all finite graphs. In particular, two popular graph families that we utilize have "bounded combinatorial width." Here the width of a graph is defined in many possible ways, often influenced by the graph problems that are being investigated. For example, for VLSI layout problems we may define the width of a planar graph (which represents a circuit) as the least surface area needed over all layouts, where vertices and edge connections are laid out on a grid. Two sets of graphs that we utilize in this dissertation are graphs of bounded *pathwidth* and *treewidth*. These width metrics and the corresponding graph families that are bounded by some fixed width are important for a variety of reasons:

1. These families are minor-order lower ideals.
2. They play a fundamental role in the proof of the Graph Minor Theorem.
3. Many popular classes of graphs are subsets of these families.
4. These widths are used to cope with intractability.
5. There are many natural applications.

The first statement is particularly important for our obstruction set computations. Here if H is a minor of a graph G then the pathwidth (treewidth) of H is at most the pathwidth (treewidth) of G.

Regarding the development of graph algorithms, we want to emphasize that there is a common trend in that practical algorithms often exist for restricted families of graphs (such as trees or planar graphs), while the general problems remain hard to solve for arbitrary input. This tendency for efficient algorithms includes instances where the problem's input is restricted to graphs of bounded pathwidth or bounded treewidth.

2.1 Introduction

The use of bounded combinatorial width is important in such diverse areas as computational biology, linguistics, and VLSI layout design [KS93, KT92, Möh90]. For problems in these areas, the graph input often has a special tree-like (or path-like) structure. A measure of width of these structures is formalized as the treewidth (or pathwidth). It is easy to find examples of families of graphs with bounded treewidth such as the sets of series-parallel graphs, Halin graphs, outer-planar graphs, and graphs with bandwidth at most k (see Section 2.1.3 and [Bod86a]). One also finds other natural classes of graphs with bounded treewidth in the study of the reliability of communication networks and in the evaluation of queries on relational databases [MM91, Bie91, Arn85]. The treewidth parameter also arises in places such as the efficiency of doing Choleski factorization and Gaussian elimination [BGHK91, DR83].

The pathwidth and treewidth of graphs have proven to be of fundamental importance for at least two distinct reasons: (1) their role in the deep results of Robertson and Seymour [RS83, RS86a, RS91a, RSc], and (2) they are "common denominators" leading to efficient algorithms for bounded parameter values for many natural input restrictions of $\mathcal{NP}$-complete problems. To further explain this second phenomenon, consider the classic $\mathcal{NP}$-complete problem of determining if a graph G has a vertex cover of size k, where both G and k are part of the input. It is known that if k is fixed (i.e., not part of the input) then all yes instances to the problem have pathwidth at most k. This restricted problem can be solved in linear time for any fixed k (see Chapter 6). For more examples, besides the bounded-width families studied in this dissertation, see [Bod86a, FL92, Möh90]. In general, many problems, such

as determining the independence of a graph, can be solved in linear time when the input also includes a bounded-width path decomposition (or tree decomposition) of the graph (see [Arn85, AP89, Bod88a, CM93, WHL85] and [Bod86a] for many further references). Examples of this latter type are different from the vertex cover example because a width bound for the yes instances is not required (e.g., for every fixed k there exists a graph of independence k with arbitrarily large treewidth).

Another well-known fact about the set of graphs of pathwidth or treewidth at most k is that each set is characterizable by a set of forbidden minors. We denote these parameterized lower ideals by k–PATHWIDTH and k–TREEWIDTH. For example some recently discovered obstruction sets for 1–PATHWIDTH, 2–TREEWIDTH, 3–TREEWIDTH are shown in Figures 2.3 and 2.4 (on page 55) at the end of this chapter [APC87]. Besides the single obstruction K_3 for 1–TREEWIDTH, a set of 110 obstructions for 2–PATHWIDTH (see [Kin89]) is the only other known characterization for bounded pathwidth or treewidth.

2.1.1 Treewidth and k-trees

Informally, a graph G has treewidth at most k if its vertices can be placed (with repetitions allowed) into sets of order at most $k+1$ and these sets arranged in a tree structure, where (1) for each edge (u, v) of G both vertices u and v are members of some common set and (2) the sets containing any specific vertex induces a subtree structure. Figure 2.1 illustrates several graphs with bounded treewidth (pathwidth). Below is a formal definition of treewidth.

Definition 1: A *tree decomposition* of a graph $G = (V, E)$ is a tree T together with a collection of subsets T_x of V indexed by the vertices x of T that satisfies:

1. $\bigcup_{x \in T} T_x = V$.
2. For every edge (u, v) of G there is some x such that $u \in T_x$ and $v \in T_x$.
3. If y is a vertex on the unique path in T from x to z then $T_x \cap T_z \subseteq T_y$.

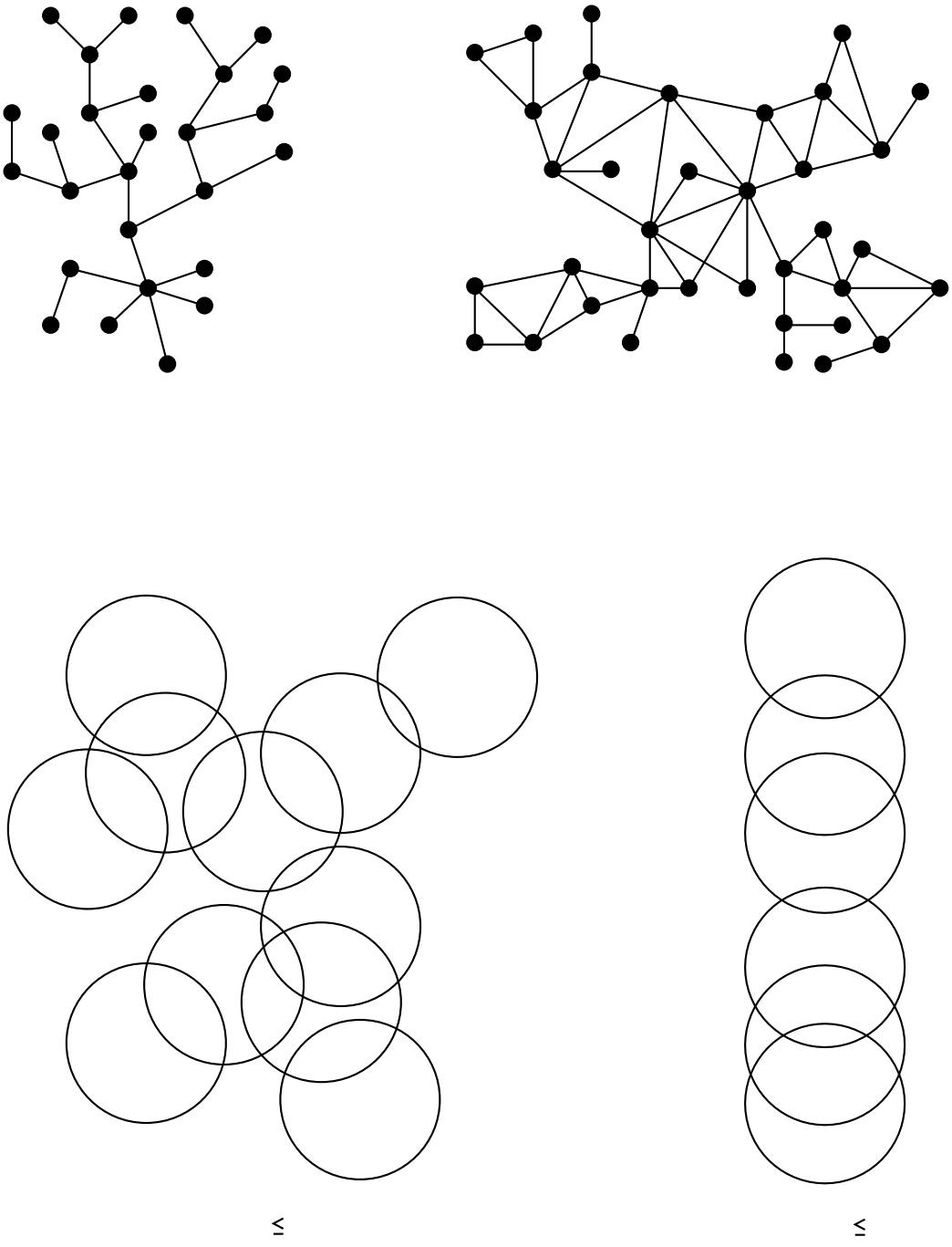


Figure 2.1: Demonstrating graphs with bounded treewidth (and pathwidth).

The *width* of a tree decomposition is the maximum value of $|T_x| - 1$ over all vertices x of the tree T. A graph G has *treewidth* at most k if there is a tree decomposition of G of width at most k. *Path decompositions* and *pathwidth* are defined by restricting the tree T to be simply a path (see Section 2.1.2 below).

Definition 2: A graph G is a *k-tree* if G is a k-clique (complete graph) K_k or G is obtained recursively from a k-tree G' by attaching a new vertex w to an induced k-clique C of G', such that the open neighborhood $N(w) = V(C)$. A *partial k-tree* is any subgraph of a k-tree.

Alternatively, a graph G is a k-tree if

1. G is connected,
2. G has a k-clique but no $(k+2)$-clique, and
3. every minimal separating set is a k-clique.

The above result was taken from one of the many characterizations given in [Ros73]. The next useful result appears in many places [ACP87, RS86a, Nar89].

Theorem 3: The set of partial k-trees is equivalent to the set of graphs with treewidth at most k

2.1.2 Pathwidth and k-paths

For our representations of graphs of bounded width we use graphs that have a distinguished set of labeled vertices. These are formally defined next.

Definition 4: For a positive integer k, a *k-simplex* S of a graph $G = (V, E)$ is an injective map $\partial_S : \{1, 2, \ldots, k\} \to V$. A *$k$-boundaried graph* $B = (G, S)$ is a graph G together with a k-simplex S for G. Vertices in the image of ∂_S are called *boundary* vertices (often denoted by ∂). The graph G is called the *underlying graph* of B.

We now define a family of graphs where each member has a linear structure. As in Theorem 3, a graph has pathwidth at most $k-1$ if and only if it is a subgraph of an underlying graph in the following set of k-boundaried graphs. Our definition of a k-path is consistent (except for the base clique) with other definitions that use a set of boundary vertices with one fewer vertex (see for example [BP71]). Later in Lemmas 9 and 11 we justify these two statements.

Definition 5: A graph G is a $(k-1)$-*path* if there exists a k-boundaried graph $B = (G, S)$ in the following family $\mathcal{F}$ of recursively generated k-boundaried graphs.

1. $(K_k, S) \in \mathcal{F}$ where S is any k-simplex of the complete graph K_k.
2. If $B = ((V, E), S) \in \mathcal{F}$ then $B' = ((V', E'), S') \in \mathcal{F}$ where $V' = V \cup \{v\}$ for $v \notin V$, and for some $j \in \{1, 2, \ldots k\}$:
   (a) $E' = E \cup \{(\partial_S(i), v) \mid 1 \leq i \leq k \text{ and } i \neq j\}$, and
   (b) S' is defined by $\partial_{S'}(i) = \begin{cases} v & \text{if } i = j \\ \partial_S(i) & \text{otherwise} \end{cases}$.

A *partial k-path* is subgraph of a k-path.

Item 2 in the above definition states that a new boundary vertex v can be added as long as it is attached to any current $(k-1)$-simplex within the active k-simplex S. The set of boundary vertices of B' is the closed neighborhood $N[v]$ of the new vertex v.

The definition of a tree decomposition is easily specialized for this path-structured case.

Definition 6: A *path decomposition* of a graph $G = (V, E)$ is a sequence $X_1, X_2, \ldots, X_r$ of subsets of V that satisfy the following conditions:

1. $\bigcup_{1 \leq i \leq r} X_i = V$.
2. For every edge $(u, v) \in E$, there exists an X_i, $1 \leq i \leq r$, such that $u \in X_i$ and $v \in X_i$.

3. For $1 \leq i < j < k \leq r$, $X_i \cap X_k \subseteq X_j$.

The *pathwidth* of a *path decomposition* $X_1, X_2, \ldots, X_r$ is $\max_{1 \leq i \leq r} |X_i| - 1$. The *pathwidth* of a *graph* G, denoted $pw(G)$, is the minimum pathwidth over all path decompositions of G.

The next definition makes the development of bounded pathwidth algorithms easier to understand and prove correct.

Definition 7: A path decomposition $X_1, X_2, \ldots, X_r$ of pathwidth k is *smooth*, if for all $1 \leq i \leq r$, $|X_i| = k + 1$, and for all $1 \leq i < r$, $|X_i \cap X_{i+1}| = k$.

It is evident that any path decomposition of minimum width can be transformed into a smooth path decomposition of the same pathwidth. There is a corresponding notion of a *smooth tree decomposition*. See [Bod93a] for a simple procedure that converts any tree decomposition into a smooth tree decomposition. From a smooth decomposition of width k we can easily embed a graph in a k-path or k-tree (e.g., see proof of Lemma 11).

To initiate the reader (and for completeness), we now give two basic results regarding the notion of pathwidth [RS91a]. We use part of the second result when we prove that our algebraic graph representation (given in Section 2.2) is correct.

Lemma 8: If a graph G has components $C_1, C_2, \ldots, C_r$ then

$$pw(G) = \max_{1 \leq i \leq r} \{pw(C_i)\} \quad .$$

Proof. For $1 \leq i \leq r$, let $X_1^i, X_2^i, \ldots, X_{s_i}^i$ be a path decomposition for component C_i with minimum width $w_i = pw(C_i)$. The sequence of sets

$$X_1^1, X_2^1, \ldots, X_{s_1}^1, X_1^2, X_2^2, \ldots, X_{s_2}^2, \ldots, X_1^r, X_2^r, \ldots, X_{s_r}^r$$

is a path decomposition of G since (1) $\bigcup_{i=1}^r V(C_i) = V(G)$, (2) $\bigcup_{i=1}^r E(C_i) = E(G)$, and (3) $X_i^a \cap X_j^b = \emptyset$ for $1 \leq i \leq s_a$, $1 \leq j \leq s_b$, and $1 \leq a < b \leq r$. The width of this decomposition is $\max_{i=1}^r \{w_i\}$, so $pw(G) \leq \max_{i=1}^r \{pw(C_i)\}$.

Let $X_1, X_2, \ldots, X_s$ be a path decomposition of G of minimum width. For any component C_i of G, let $D_j = X_j \cap V(C_i)$ for $j = 1, 2, \ldots s$. We claim that $D_1, D_2, \ldots, D_s$ is a path decomposition of C_i. Since for all $v \in V(G)$ there is an X_k such that $v \in X_k$, $\bigcup_{1 \leq j \leq s} D_j = V(C_i)$. Likewise, since for every edge $(u, v) \in E(G)$ there is an X_k such that $u \in X_k$ and $v \in X_k$. So if $(u, v) \in E(C_i)$, then both $u \in D_k$ and $v \in D_k$. Finally, for $1 \leq i < j < k \leq s$, $X_i \cap X_k \subseteq X_j$ implies $D_i \cap D_k \subseteq D_j$. Therefore, since $|D_k| \leq |X_k|$ for $1 \leq k \leq s$, $pw(C_i) \leq pw(G)$ for any component C_i of G. □

The following well-known lemma shows that the pathwidth of a partial k-path is at most k. Thus we can justifiably think of partial k-paths as either subgraphs or as minors of k-paths.

Lemma 9: If G is a k-path then $pw(G) = k$. Further, if H is a minor of G, $H \leq_m G$, then $pw(H) \leq k$.

Proof. We first prove that any k-path G obtained from a defining $(k+1)$-boundaried graph (G, S) has a path decomposition $X_1, X_2, \ldots, X_r$ of width k where $X_r = image(S)$ (i.e., X_r is the set of boundary vertices). For the base case of G a $(k+1)$-clique, the hypothesis holds since $X_1 = \{1, 2, \ldots, k+1\} = V(G)$ is a path decomposition of width k. For the inductive step, assume $X_1, X_2, \ldots, X_r$ is a path decomposition for G. Let G' be the k-path built from some (G, S) by applying case 2 of Definition 5. If we set $X_{r+1} = image(S')$ then $X_1, X_2, \ldots, X_r, X_{r+1}$ is the required path decomposition of G'. [(1) The new vertex v is in X_{r+1}, (2) all new edges (u, v) in $E(G') \setminus E(G)$ have $u \in X_{r+1}$ and $v \in X_{r+1}$, and (3) $X_i \cap X_{r+1} \subseteq X_i \cap X_r$ implies $X_i \cap X_{r+1} \subseteq X_j$ for $1 \leq i < j \leq r$.]

To see that any k-path G has a lower bound of k for its pathwidth, first notice that G contains a clique C of size $k+1$. For any path decomposition $X = X_1, X_2, \ldots, X_r$ of G, let $Y_i = X_i \cap V(C)$ for $1 \leq i \leq r$. If any $|Y_i| > k$ then the width of X is at least k. Suppose $|Y_i| \leq k$ for all Y_i. There must then be a Y_i and a Y_j, $1 \leq i < j \leq r$, such that $u \in Y_i \setminus Y_j$ and $v \in Y_j \setminus Y_i$ for different vertices u and v. Since $(u, v) \in E(C)$, there must be some Y_k, $1 \leq k \leq r$, such that $u \in Y_k$ and $v \in Y_k$. Now $\{u, v\} \not\subseteq Y_i \cap Y_j$ so either $k < i$ or $k > j$. But $v \notin Y_k$ for $k \leq i$, and $u \notin Y_k$ for $k \geq j$. This contradicts X being a path decomposition (i.e., it fails the edge covering requirement). Therefore, the pathwidth of any k-path is k.

For the second part of the theorem, we start with a path decomposition $X = X_1, X_2, \ldots, X_r$ of width k for a k-path G. It is sufficient to show that any one-step minor H of G has a path decomposition of width at most k. For edge deletions, $H = G \setminus \{(u,v)\}$, the original path decomposition X is a path decomposition of the minor H. For isolated vertex deletions, $H = G \setminus \{v\}$, the path decomposition $X_1 \setminus \{v\}, X_2 \setminus \{v\}, \ldots, X_r \setminus \{v\}$ is a path decomposition of the minor H. For edge contractions, $H = G \,/\, (u,v)$, let

$$Y_i = \begin{cases} X_i & \text{if } X_i \cap \{u,v\} = \emptyset \\ X_i \cup \{w\} \setminus \{u,v\} & \text{otherwise} \end{cases}$$

for $1 \leq i \leq r$, where w is the new vertex created by the contraction. The sequence $Y = Y_1, Y_2, \ldots, Y_r$ is a path decomposition of the edge contracted minor H. The widths of these path decompositions for the three types of one-step minors is at most k.

We justify the edge contraction case and leave the other two simpler cases to the reader. By definition, $\bigcup_{1 \leq i \leq s} Y_i = V(H)$. Let $(a,b) \in E(H)$. If $\{a,b\} \cap \{u,v\} = \emptyset$ then some $Y_i = X_i$ contains both a and b. Otherwise, consider any edge $(a,b) = (x,w)$. An edge incident to vertex w implies that either $(x,u) \in E(G)$ or $(x,v) \in E(G)$. Thus some X_i contains x and either one or both of u and v. And so, $Y_i = X_i \cup \{w\} \setminus \{u,v\}$ contains both x and w. Finally, let i be the minimum index such that $w \in Y_i$ and k be the maximum index such that $w \in Y_k$. Without loss of generality, assume $u \in X_i$. If $u \in X_k$ then $Y \simeq X \setminus \{v\}$ and Y is a path decomposition of H. Otherwise, $v \in X_k$. Since $(u,v) \in E(G)$, there exists a j, $i \leq j \leq k$, such that both $u \in X_j$ and $v \in X_j$. Since X is a path decomposition, $u \in X_m$ for $i \leq m \leq j$, and $v \in X_m$ for $j \leq m \leq k$. So $w \in Y_m$ for $i \leq m \leq k$, and $w \notin Y_m$ for $m < i$ or $m > k$. Therefore, for $1 \leq i < j < k \leq s$, we have $Y_i \cap Y_k \subseteq Y_j$. □

Corollary 10: Any graph of order n and pathwidth k has at most $\binom{k+1}{2} + (n-k-1) \cdot k$ edges.

Proof. We easily prove that equality holds for k-paths by induction on the number of vertices. Since the smallest k-path is the complete graph $G = K_{k+1}$, our base case holds since $|E(K_{k+1})| = \binom{k+1}{2}$. If a k-path G has $n > k+1$ vertices, then there

exists a vertex v that was added to a smaller k-path $G' = G \setminus \{v\}$ (using the recursive definition of k-paths). When vertex v was added to G', there were exactly k new edges added too. So G must have $\binom{k+1}{2} + ((n-1)-k-1) \cdot k + k = \binom{k+1}{2} + (n-k-1) \cdot k$ edges. □

From the above corollary we see that graphs of bounded pathwidth have at most a linear number of edges (with respect to n). It is not difficult to show that graphs of treewidth k also have this *same* bound. So, since the input size is linear in the number of vertices (for our bounded width algorithms), there is no confusion when we talk about having a *linear time* algorithm.

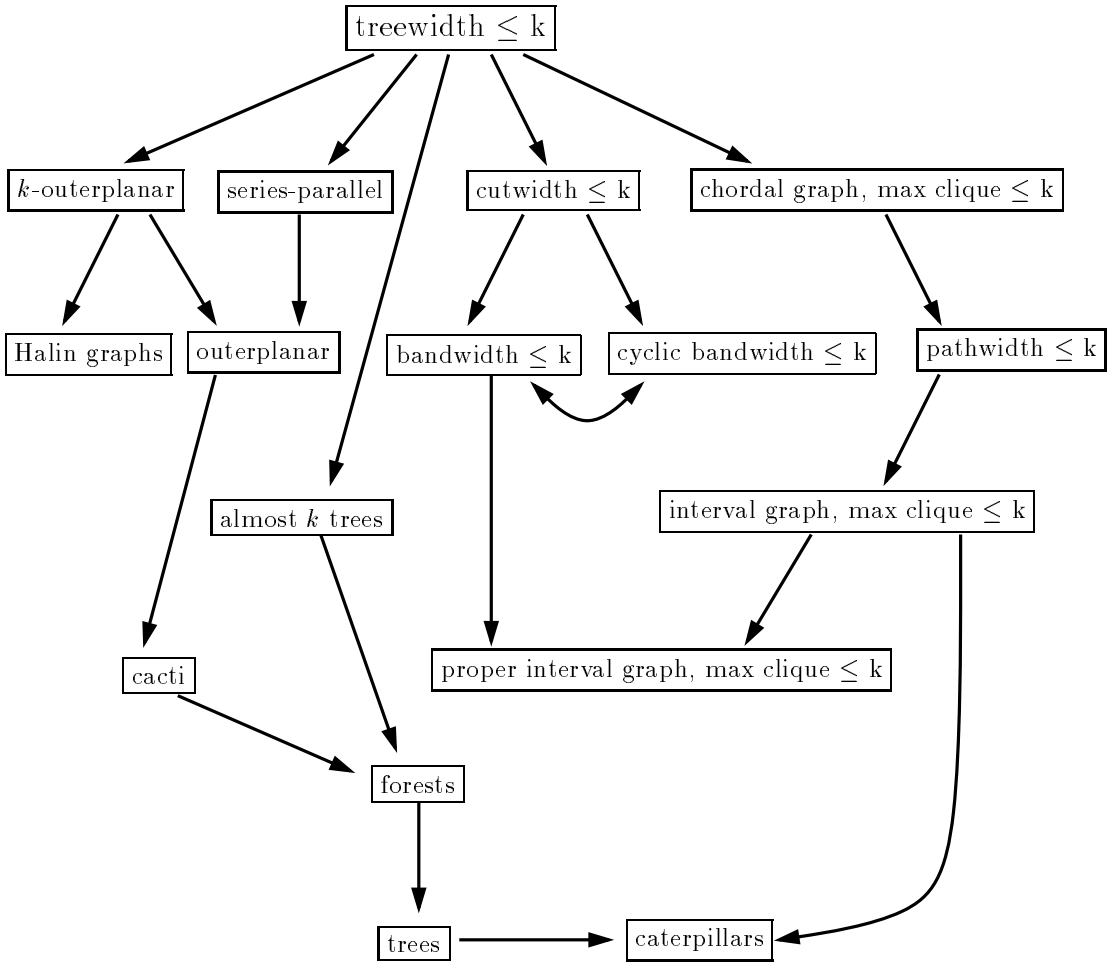
We now complete this subsection by proving the converse of Lemma 9.

Lemma 11: Any graph G of pathwidth at most k is a subgraph of a k-path.

Proof. If G has at most $k+1$ vertices we can embed G in the base clique K_{k+1}, the smallest k-path. Otherwise, let $X_1, X_2, \ldots, X_r$ be a smooth path decomposition of G of width k. We can map the vertices of X_1 into the base clique K_{k+1}. By induction, assume that we have embedded the vertices of $\cup_{j=1}^{i} X_j$ in a k-path where X_i is the set of boundary vertices of the $(k+1)$-boundaried graphs $B_i = (G_i, S_i)$, generated by the rules of Definition 5. We can construct the next $(k+1)$-boundaried graph $B_{i+1} = (G_{i+1}, S_{i+1})$ by adding the unique vertex $v \in X_{i+1} \setminus X_i$ in step 2 of Definition 5 and dropping the unique vertex $v' \in X_i \setminus X_{i+1}$ from the simplex S_i of the previous k-path G_i. Since each X_i, $1 \leq i \leq r$ is mapped to a $k+1$-clique, every edge of G is mapped to an edge of the k-path $G_r \supseteq G$. □

2.1.3 Other graph-theoretical widths

There are actually many types of combinatorial width metrics for graphs. Often a set of graphs bounded by one type of combinatorial width is a subset of a class of graphs with another bounded parameter. A relationship between treewidth and several graph widths is given by Bodlaender in [Bod88b]. In addition, Wimer's dissertation [Wim87b] explores the relationships between 2-terminal recursive families (e.g., series-parallel graphs, outer-planar graphs, and cacti). This work is now highlighted in Figure 2.2.



Note that each constants k denotes a different constant. For example, the set of all graphs of bandwidth k is contained in the set of graphs of cutwidth $k' = (k^2 + k)/2$.

Figure 2.2: A partial order of several bounded-width families of graphs.

We briefly describe two of the bounded width families, given in Figure 2.2, that originated from VLSI linear layout problems. A linear layout of a graph $G = (V, E)$ is a bijection $f : V \to \{1, 2, \ldots, |V|\}$, that is, an assignment of integers 1 to $|V|$ to the vertices. The bandwidth of a layout is the maximum value of $|f(i) - f(j)|$ over all edges $(i, j) \in E$, that is, the maximum distance any edge has to span in the layout. The cutwidth of a layout is the maximum number of edges (i, j) such that $f(i) \leq k < f(j)$ over all $1 \leq k < |V|$, that is, the maximum number of lines that can be cut between any layout position k and $k + 1$. The bandwidth (cutwidth) of the graph is the minimum bandwidth (cutwidth) over all linear layouts. Both of these "min of max" definitions are similar to the pathwidth and treewidth definitions that we saw earlier. In fact, the pathwidth of a graph is equivalent to the vertex separation of a graph which is also defined as a linear layout problem (see Chapter 10).

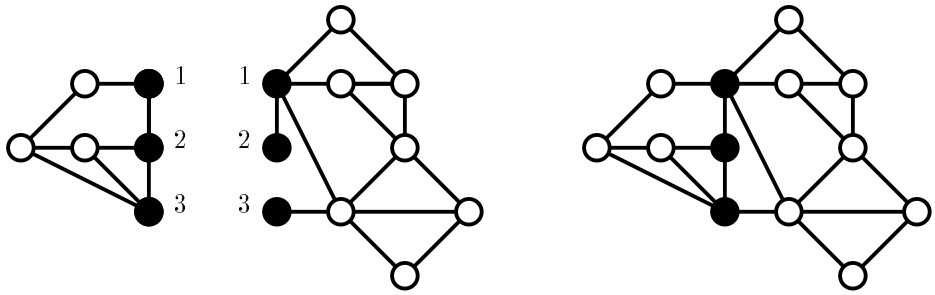
The definition of interval graphs may be found in most introductory graph theory books; they are also mentioned in Chapter 10. We conclude this section with a description for two of the other less known bounded-width families. A *Halin graph* is a planar graph $G = T \cup C$, where T is a tree of order at least 4 with no degree two vertices and C is a simple cycle connecting all and only the leaves of T. A *cactus* (member of the *cacti* family) is a connected outer-planar graph that has single edges and simple cycles as its blocks.

2.2 Algebraic Graph Representations

Recall that we are mainly interested in finite simple graphs. However, some of our graphs have a vertex boundary of size k, meaning that they have a distinguished set of vertices labeled $1, 2, \ldots, k$. We sometimes use 0 (instead of k) as a boundary label. One important operation on k-boundaried graphs is given next.

Definition 12: Two k-boundaried graphs (each with a boundary of size k) can be "glued together" with the $\oplus$ operator, called *circle plus*, that simply identifies vertices with the same boundary label.

Example 13: The binary operator $\oplus$ on two 3-boundaried graphs A and B is illustrated below. Note that common boundary edges (in this case, the edges between boundary vertices 1 and 2) are replaced with a single edge in $G = A \oplus B$.



2.2.1 Operator sets and t-parses

In this section we show that $(t+1)$-boundaried graphs of pathwidth at most t are generated exactly by strings of unary operators from the following *operator set* $\Sigma_t = V_t \cup E_t$:

$$V_t = \{\textcircled{0}, \ldots, \textcircled{t}\} \quad \text{and}$$
$$E_t = \{\boxed{i\ j} \mid 0 \leq i < j \leq t\}.$$

To generate the graphs of treewidth at most t, the additional binary operator $\oplus$ is added to Σ_t. The semantics of these operators on $(t+1)$-boundaried graphs G and H are as follows:

$G\ \textcircled{i}$	Add an isolated vertex to the graph G, and label it as the new boundary vertex i.
$G\ \boxed{i\ j}$	Add an edge between boundary vertices i and j of G (and ignore if the edge already exists).
$G \oplus H$	Take the disjoint union of G and H except that boundary vertices of G and H that are labeled the same are identified (i.e., the circle plus operator).

Definition 14: A *parse* is a sequence (or tree, if $\oplus$ is used) of operators $[g_1, g_2, \ldots, g_n]$ in Σ_t^* that has vertex operators $\textcircled{i}$ and $\textcircled{j}$ occurring in $[g_1, g_2, \ldots, g_n]$ before the first edge operator $\boxed{i\ j}$, $0 \leq i < j \leq t$.

Definition 15: A *t-parse* is a parse $[g_1, g_2, \ldots, g_n]$ in Σ_t^* where all the vertex operators $\textcircled{0}$, $\textcircled{1}$, ..., $\textcircled{t}$ appear at least once in $[g_1, g_2, \ldots, g_n]$. That is, a t-parse is a parse with $t+1$ boundary vertices.

For clarity, we say that a *treewidth t-parse* is any t-parse containing at least one $\oplus$ operator and a *pathwidth t-parse* is any t-parse without $\oplus$ operators.

Intuitively, t-parses represent $(t+1)$-boundaried graphs that have pathwidth (or treewidth) at most t. We justify this statement later in this section. These graphs are constructed by using the rules defined for the unary vertex and edge operators and the binary circle plus operator. The simplex S of a t-parse (i.e., the corresponding boundaried graph) is informally defined to be $\partial_S(i) = \textcircled{i}$, $0 \leq i \leq t$, where each $\textcircled{i}$ is the last occurrence in the parse. The symbol ∂ is used to denote the set of boundary vertices of a boundaried graph.

As we proceed with our obstruction set computation theory, we use the phrase "a t-parse G" in one of three ways:

- as a $(t+1)$-boundaried graph (also called G),
- as an implied path (or tree) decomposition of a graph, or
- as a data structure (e.g., for dynamic programs).

Thus, a t-parse G is used with standard object-oriented terminology (i.e., a t-parse G "is a" boundaried graph and G "has a" decomposition), where any operations on graphs may also be applied to a t-parse G.

Definition 16: Let $G = [g_1, g_2, \ldots, g_n]$ be a t-parse and $Z = [z_1, z_2, \ldots, z_m]$ be any sequence of operators over Σ_t. The *concatenation* $(\cdot)$ of G and Z is defined as

$$G \cdot Z = [g_1, g_2, \ldots, g_n, z_1, z_2, \ldots, z_m].$$

The t-parse $G \cdot Z$ is called an *extended t-parse*, and $Z \in \Sigma_t^*$ is called an *extension*. For the treewidth case, G and Z are viewed as two connected subtree factors of a parse tree $G \cdot Z$ instead of two parts of a sequence of operators.

A vertex operator $g_j = \textcircled{i}$, for any $0 \leq i \leq t$, in a t-parse $G_n = [g_1, g_2, \ldots, g_n]$ is called a *boundary vertex* if $g_k \neq \textcircled{i}$ for $j < k \leq n$.

Definition 17: A graph G is *generated* by Σ_t if there is a $(t+1)$-boundaried graph $(B, S) = G_n = [g_1, g_2, \ldots, g_n]$ such that $B \simeq G$ for some t-parse G_n. That is, G is isomorphic to an *underlying graph* of a t-parse.

For ease of discussion throughout the remaining part of this chapter, we limit ourselves to graphs of bounded pathwidth while leading up to our obstruction set search theory. In certain situations, where required, we provide additional information pertaining to graphs of bounded treewidth.

Definition 18: Let $G_n = [g_1, g_2, \ldots, g_n]$ be a boundaried graph represented as some parse over Σ_t. A *prefix graph (of length m)* of G_n is $G_m = [g_1, g_2, \ldots, g_m]$, $1 \leq m \leq n$.

Lemma 19: Every t-parse $G_n = [g_1, g_2, \ldots, g_n]$ represents a graph with a path decomposition of width t.

Proof. Without loss of generality, let $G_n = [g_1, g_2, \ldots, g_n]$ be a $(t+1)$-boundaried graph. Let v_i denote the index of the i-th vertex operator in the t-parse G_n. Let I be the set of these v_i and for $i \in I$,

$$X_i = \{j \mid g_{v_j} \text{ is a boundary vertex of } G_{v_i}\}$$

We claim that $X = X_1, X_2, \ldots, X_{|I|}$ is a path decomposition of width t for G_n.

For any vertex operator g_k, define $label(g_k)$ to be i if g_k is the i-th vertex operator in G_n. By definition of the X_i's, $\bigcup X_i = \{1, 2, \ldots, |I|\} = V(G_n)$. For any edge operator $g_k = \boxed{i\ j}$ of G_n, let $b_1(g_k)$ and $b_2(g_k)$ denote the indices of the preceding vertex operators $\textcircled{i}$ and $\textcircled{j}$, respectively. For each edge operator $g_k = \boxed{i\ j}$ of G_n, let $u = label(b_1(g_k))$ and $v = label(b_2(g_k))$. Since edge operators add edges to the current boundary of a prefix graph, either X_u or X_v must contain both u and v. Assuming $i < j$, $X_i \cap X_j$ represents the boundary vertices of G_{v_j} that did not change from G_{v_i}. So for any k, $i < k < j$, we have $X_i \cap X_j \subseteq X_i \cap X_k$. This implies that $X_i \cap X_j \subseteq X_k$.

Finally, since only (and exactly) the vertex operators $\textcircled{0}$, $\textcircled{1}$, ..., $\textcircled{t}$ appear in G_n we have $\max_{1 \leq i \leq |I|} \{|X_i|\} = t+1$. So X is a path decomposition of width t for G_n. □

And next we prove the result in the other direction.

Lemma 20: Every partial t-path of order at least $t+1$ is represented by some t-parse.

Proof. We first show that every t-path can be represented by some t-parse. We prove inductively from the recursive definition of t-paths. The initial $(t+1)$-clique can be represented by the following sequence of operators over Σ_t.

$$[\textcircled{0}, \textcircled{1}, \boxed{0\ 1}, \textcircled{2}, \boxed{0\ 2}, \boxed{1\ 2}, \textcircled{3}, \boxed{0\ 3}, \boxed{1\ 3}, \boxed{2\ 3}, \ldots, \textcircled{t}, \boxed{0\ t}, \boxed{1\ t}, \ldots, \boxed{t\text{–}1\ t}]$$

For the inductive step, assume that any t-path G with an active $t+1$ simplex S is represented by an operator string $G_m = [g_1, g_2, \ldots, g_m]$ such that ∂_S is defined properly. The recursive operation of "replacing a simplicial vertex with a new vertex v (forming a new clique)" can be modeled by appending the operator sequence of the form

$$[\textcircled{i}, \boxed{0\ i}, \ldots, \boxed{i\text{–}1\ i}, \boxed{i\ i{+}1}, \ldots, \boxed{i\ t}]$$

to G_m depending on which simplicial vertex (boundary vertex i) should be replaced. Thus, any t-path can be represented by a t-parse over Σ_t.

To represent partial t-paths, note that each edge is represented by one or more edge operators. By simply deleting all of these edge operators, any edge in the t-parse is removed. Thus, by repeatedly deleting edges, we get a t-parse for any partial t-path.

□

Note that many t-parse representations may exist for a partial t-path since many path decompositions are generally possible. Finally, combining the previous two lemmas we have the following theorem.

Theorem 21: The family of graphs (t-parses) generated by Σ_t equals the family of partial t-paths of order at least $t+1$.

Corollary 22: A graph G has pathwidth t if and only if it is generated by Σ_t but not $\Sigma_{t\text{-}1}$.

Proof. If a graph G is generated by $\Sigma_{t\text{-}1}$ then it has pathwidth $t-1$. If G has pathwidth t then it is a partial t-path and by the above theorem it has a t-parse representation over Σ_t. □

For a treewidth analog to Theorem 21, we have the following result.

Theorem 23: The set of treewidth t-parses represents the set of graphs of order at least $t+1$ and treewidth at most t.

Proof. We first show that any t-parse G has a tree decomposition $(T, \{T_x\})$ of width t where the current boundary of G is in some T_x, $x \in T$. We prove this by induction on the number of operators in G and whether or not G has any $\oplus$ operators. For the base cases of no $\oplus$ operators, we have by Lemma 19, a path decomposition $X = X_1, \ldots, X_r$ of width t. Thus G has a tree decomposition $(P_r, \{X_i \mid i \in V(P_r)\})$, where P_r denotes a path of length r. Note that by our constructive proof the set X_r contains the current boundary of G.

Now assume G contains at least one $\oplus$ operator. We have three cases. If $G = G_1 \oplus G_2$ then let $(T', \{T'_x\})$ and $(T'', \{T''_x\})$ be tree decompositions for G_1 and G_2, respectively. Since both G_1 and G_2 have fewer operators than G, there exists sets T'_x and T''_x that contain the boundary of G_1 and G_2 (or of G itself). A tree decomposition of G of the desired form is constructed by identifying T'_x and T''_x in the two subtree decompositions. If $G = G_1 \cdot [\boxed{i\ j}\ ]$ then let $(T', \{T'_x\})$ be a tree decomposition of the desired form for G_1. Since both boundary vertices i and j are in some vertex set T'_x for some $x \in T'$, $(T', \{T'_x\})$ is the required tree decomposition for G. Similarly, if $G = G_1 \cdot [\textcircled{i}]$, let $(T', \{T'_x\})$ be a tree decomposition of the desired form for G_1. Let T be the tree constructed by attaching a vertex i to T' at vertex x, where T'_x contains all the boundary vertices of G_1. Set $T_i = T'_x \setminus \{i'\} \cup \{i\}$ where i' represents the old boundary vertex. This proves that every t-parse has treewidth at most t.

Now we show that any t-tree G with at least $t+1$ vertices contains a t-parse representation. As in the proof of Lemma 20, any partial t-tree is representable

by removing the appropriate edge operators. Our proof is based on a smooth tree decomposition $(T, \{T_x\})$ of G. In particular, $|T_x| = t+1$ for all $x \in T$ and $|T_u \cap T_v| = t$ whenever u and v are adjacent in T. We recursively build a t-parse for G by arbitrarily picking a root vertex r of T and having T_r as the final t-parse boundary.

We first require a function $\phi_{i,j}$ that takes a t-parse H and returns a t-parse H' where the boundary labels i and j have been interchanged. (The reader should observe that the function $\phi_{i,j}$ is easy to construct.) For the proof below we assume that the underlying graph G is labeled $1, 2, \ldots, |G|$, and that any partial t-parse will have boundary vertex i less than boundary j, whenever the underlying label for i is less than the underlying label for j. This *boundary order* is needed to keep things aligned when we use the $\oplus$ operator.

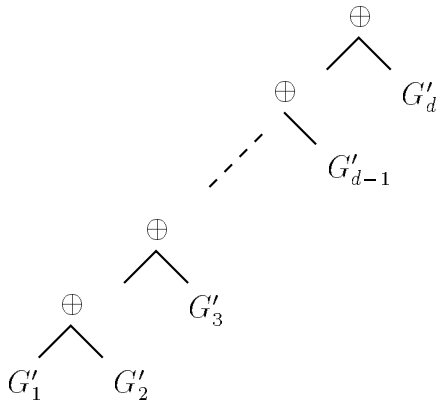
If G has $t+1$ vertices (i.e., a clique K_{t+1}) then the construction given in Lemma 20 for the t-path case is sufficient. We now recursively build a t-parse for G based on the degree $d \geq 1$ of a root vertex r. Let $v_1, \ldots, v_d$ be the neighbors of vertex r in T. For each $1 \leq i \leq d$ consider the subtree decomposition $(T^i, \{T^i_x\})$ induced by the subtree of T rooted at v_i. By induction we have t-parses $G_1, \ldots, G_d$ for these pieces of G with boundary sets $T_{v_1}, \ldots, T_{v_d}$, respectively.

If $d = 1$, we have the following t-parse, where i denotes the boundary vertex representing the single vertex in $T_{v_1} \setminus T_r$, for the original t-parse G (using the proof method of Lemma 20).

$$G_1 \cdot [\textcircled{i}, \boxed{0\ i}, \ldots, \boxed{i{-}1\ i}, \boxed{i\ i{+}1}, \ldots, \boxed{i\ t}\,]$$

If the new boundary vertex i is out of boundary order with respect to T_r then we can apply the $\phi_{i,j}$ function (possibly several times with different vertices i and j).

If $d > 1$ then we apply one vertex operator for each subtree parse and a total of $d-1$ circle plus operators to combine the pieces. The vertex operator is chosen the same way as in the $d = 1$ case and we may have to permute with the $\phi_{i,j}$ function. Let G'_i denote a particular t-parse, for $1 \leq i \leq d$. The t-parse for G is then created by the following parse.

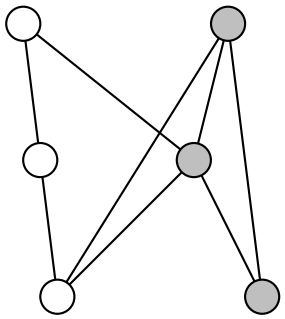


Note that multi-edges created by the repeated $\oplus$ operators are ignored. □

2.2.2 Some t-parse examples

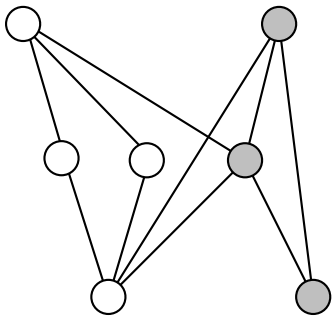
The next three examples illustrate our algebraic representation of t-boundaried graphs of bounded width. The first two t-parse examples are graphs of pathwidth 2 which are then combined with the circle plus operator to form a graph of treewidth 2.

Example 24: A t-parse with $t = 2$, and the graph it represents. (The shaded vertices denote the final boundary.)



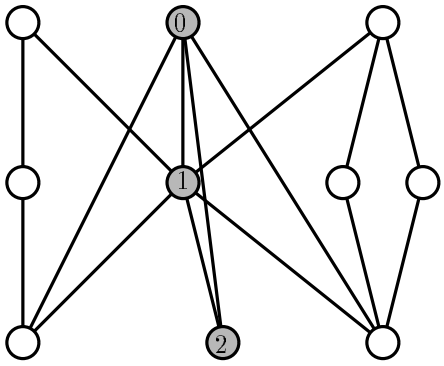
[(0), (1), (2), [0 1], [1 2], (1), [0 1], [1 2], (0), [0 1], [0 2], (2), [0 2], [1 2]]

Example 25: Another 2-parse and the graph it represents.



$[\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \boxed{1\ 2}, \textcircled{1}, \boxed{0\ 1}, \boxed{1\ 2}, \textcircled{1}, \boxed{0\ 1}, \boxed{1\ 2}, \textcircled{0}, \boxed{0\ 1}, \boxed{0\ 2}, \textcircled{2}, \boxed{0\ 2}, \boxed{1\ 2}]$

Example 26: We demonstrate the circle plus operator $\oplus$ with the 2-boundaried graphs (2-parses) given in the previous two examples. The second graph is reflected and glued onto the first graph's boundary. In general, the pathwidth of a t-parse usually increases when the binary operator $\oplus$ is used (although not in this example).



2.2.3 Other operator sets

Our t-parse representation for graphs of pathwidth (or treewidth) at most t is not unique. We now present two different operator sets for representing graphs of bounded combinatorial width. Many other algebraic representations are available, sometimes in disguised form, by other sources (e.g., see [Bor88, BC87, CM93, Wim87b]). The following two examples illustrate two extremes regarding the number of operators required to represent graphs of pathwidth (or treewidth) of at most k. The first set shows that only a constant number (4) of operators is needed, independent of the width k [Fel]. The second set uses a polynomial number of operators per boundary

size, but generates k-boundaried graphs for pathwidth k (instead of graphs with boundary size $k+1$). We point out that another "middle ground" operator set based on a combination of these two sets is given in [Lu93].

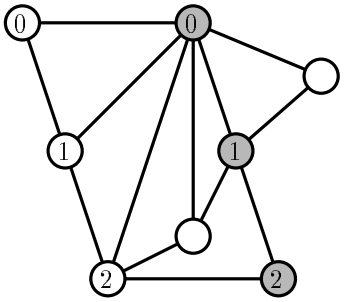
A constant sized operator set Ψ_k, for graphs with pathwidth at most k, consists of two boundary permutation operators p_1 and p_2 and the two t-parse graph building operators $\textcircled{0}$ and $\boxed{0\ 1}$. The domain for these operators is again $(k+1)$-boundaried graphs. The permutation operators are used to relabel the boundary. These permutations, given in the standard cyclic form, are $p_1 = (0,1)$ and $p_2 = (0,1,\ldots,k)$. These two permutations generate the symmetric permutation group S_{k+1} and hence, by applying in succession, arbitrary relabelings of the boundary are possible. It is easy to see that Ψ_k, with its 4 operators, generates exactly the same set of boundaried graphs as our pathwidth t-parse operator set Σ_k. We also observe the following:

Observation 27: A graph G has treewidth at most k if and only if it is obtainable by using the 5 operators $\Psi_k \cup \{\oplus\}$.

A big pathwidth operator set, called Ω_k, for k-boundaried graphs of pathwidth at most k, contains the t-parse operator set Σ_{k-1} and has the following two additional operator types. In the definitions below G denotes any k-boundaried graph.

$G\ \boxed{\text{i}}$	Add a pendent vertex to the current boundary vertex i of G, and label it as the new boundary vertex i.
$G\ \boxed{\{b_1, b_2, \ldots, b_p\}}$	Add a new interior vertex and attach it to all of the boundary vertices $b_1, b_2, \ldots, b_p$ of G, $1 \leq p < k$.

Example 28: Below we illustrate the operator set Ω_3 in generating a 3-boundaried graph of pathwidth 3.



$[\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \boxed{1\ 2}, \boxed{0}, \boxed{0\ 1}, \boxed{0\ 2}, \textcircled{1}, \boxed{\{0,1,2\}}, \boxed{0\ 1}, \boxed{2}, \boxed{1\ 2}, \boxed{\{0,1\}}]$

We now prove that this operator set that uses a smaller boundary size can be used to represent graphs of bounded pathwidth.

Theorem 29: The operator set Ω_k generates precisely the set of graphs of pathwidth at most k.

Proof. Clearly if a graph has $n \leq k$ vertices (which means it has pathwidth at most $k-1$) then it is representable by a sequence that starts with n distinct vertex operators and an edge operator for each edge.

We next show, without loss of generality, that every (well-defined) sequence

$$S = [\textcircled{0}, \textcircled{1}, \ldots, \textcircled{k\text{-}1}, s_1, s_2, \ldots, s_r]$$

of Ω_k operators represents a graph of pathwidth at most k. We do this by showing how to build a t-parse G, $t = k$, for representing the underlying graph. For each prefix $S_i = [\ldots, s_1, s_2, \ldots, s_i]$, we have a corresponding prefix t-parse $G_{\phi(i)}$ of G, where ϕ is some increasing integer function. The construction uses a "spare" (but variably labeled) boundary vertex of Σ_k for simulating the $\boxed{\{b_1, b_2, \ldots, b_p\}}$ and $\boxed{i}$ operators. For the following discussion we use $\pi(j)$ to denote a injective map from the boundary of S_i to the boundary of $G_{\phi(i)}$ and the integer v to denote the unique boundary label not in the image of π. We start by setting $G_{\phi(0)} = [\textcircled{0}, \textcircled{1}, \ldots, \textcircled{k\text{-}1}]$, π to be the identity map, and $v = k$. Now we have four cases for the inductive steps:

1. If $s_r = \textcircled{i}$ then for $i' = \pi(i)$, $G_{\phi(r)} = G_{\phi(r-1)} \cdot [\textcircled{i'}]$.
2. If $s_r = \boxed{i\ j}$ then $G_{\phi(r)} = G_{\phi(r-1)} \cdot [\boxed{\pi(i)\ \pi(j)}]$.
3. If $s_r = \boxed{\mathrm{i}}$ then $G_{\phi(r)} = G_{\phi(r-1)} \cdot [\textcircled{v}, \boxed{\pi(i)\ v}]$. Swap the values of $\phi(i)$ and v.
4. If $s_r = \boxed{\{b_1, b_2, \ldots, b_p\}}$ then

$$G_{\phi(r)} = G_{\phi(r-1)} \cdot [\textcircled{v}, \boxed{\pi(b_1)\ v}, \boxed{\pi(b_2)\ v}, \ldots, \boxed{\pi(b_p)\ v}].$$

Thus, since any pathwidth t-parse, $t = k$, has pathwidth at most k so does any k-boundaried graph generated by Ω_k.

We now show that any k-path G is representable by a string of Ω_k operators. Again this is sufficient since we can remove edge operators $\boxed{i\ j}$ or replace $\boxed{\{b_1, b_2, \ldots, b_p\}}$ operators to obtain any partial k-path. Let $X = X_1, X_2, \ldots X_r$ be a smooth path decomposition of G. We constructively build a partial parse P_i using Ω_k operators for each vertex induced k-path defined by $\bigcup_{1 \le j \le i} X_j$. The current boundary of P_i will be $\partial(P_i) = X_i \cap X_{i+1}$, for $1 \le i < r$. For each vertex $u \in G$ that is on the current boundary of P_i, define $\partial_i(u)$ to be the corresponding boundary label in $0, 1, \ldots, k-1$. The initial K_{k+1} clique is parsed by

$$P_1 = [\,\textcircled{0}, \textcircled{1}, \boxed{0\ 1}, \textcircled{2}, \boxed{0\ 2}, \boxed{1\ 2}, \textcircled{3}, \boxed{0\ 3}, \boxed{1\ 3}, \boxed{2\ 3}, \ldots,$$

$$\textcircled{k\text{-}1}, \boxed{0\ k\text{–}1}, \boxed{1\ k\text{–}1}, \ldots, \boxed{k\text{–}2\ k\text{–}1}, \boxed{\{0, 1, \ldots, k\text{–}1\}}\,]$$

where $\partial_1(u)$ is assigned arbitrarily for the vertices $X_2 \setminus X_1$. Let old_i denote the unique vertex in $X_i \setminus X_{i+1}$ and new_i denote the unique vertex in $X_{i+1} \setminus X_i$, for $1 \le i < r$. We have two cases to consider when constructing P_{i+1} from P_i, for $1 \le i \le r-2$.
If $new_i = old_{i+1}$ then

$$P_{i+1} = P_i \cdot [\ \boxed{\{0, 1, \ldots, k\text{–}1\}}\ ]$$

else (i.e., $new_i \neq old_{i+1}$) for $j = \partial_i(old_i)$

$$P_{i+1} = P_i \cdot [\boxed{\text{j}}, \boxed{0\ j}, \ldots, \boxed{j\text{–}1\ j}, \boxed{j\ j\text{+}1}, \ldots, \boxed{j\ k\text{–}1}\,]$$

then $\partial_{i+1}(new_i) = j$. The final parse for G is simply $P_r = P_{r-1} \cdot [\ \boxed{\{0, 1, \ldots, k\text{–}1\}}\ ]$ to end with a K_{k+1} clique (assuming $r > 1$). □

One of the reasons why we picked our t-parse operator set Σ_t over other possible sets is that we want each unary operator to add something significant (but not too complex) to its $(t+1)$-boundaried graph argument. That is, we want a smooth and rapid path, via these graph building operators, to the obstructions (but do not want to overshoot the graph family too far by adding complex graph pieces). That is, we believe that the search tree is smaller using our choice of operators. An analogy from the computer architecture world is that we would prefer a RISC chip over one with a full and powerful instruction set (or, in the other extreme, one with only tedious nano-code primitives). Another reason for our choice is that we want a prefix property of the parse strings. This property, for obstruction set searching, is elaborated in Chapter 4.

2.3 Simple Enumeration Schemes

To help tackle the apparently intractable task of finding obstruction sets, we need to limit the domain of the search. If we know that a particular obstruction set $\mathcal{O}$ is finite, any graph invariant over members of $\mathcal{O}$ will have an upper bound. To take advantage of a particular bounded invariant, a method is needed to generate all these restricted graphs. This set of graphs must be substantially smaller than the set of graphs of order at most $\max\{|O_i| \ : \ O_i \in \mathcal{O}\}$. As indicated by the topic of this chapter, the two main invariants that we exploit are the pathwidth and the treewidth metrics.

As seen by the examples in Section 2.2, it is easy to represent (with a computer) graphs of pathwidth or treewidth of at most t by strings of unary operators or by trees with the additional binary $\oplus$ operator. This suggests a natural way of enumerating all these bounded width graphs: Just enumerate all possible valid combinations of t-parse strings (or t-parse trees). Unfortunately, many different t-parses correspond to the same underlying graph. To reduce our search process we need to know when two t-parses represent the same graph. We formalize this concept next.

Definition 30: Two k-boundaried graphs $B_1 = (G_1, S_1)$ and $B_2 = (G_2, S_2)$ are *free-boundary isomorphic*, denoted $B_1 \simeq_{\partial_f} B_2$, if there exists a graph isomorphism ϕ between G_1 and G_2 such that

$$\{\phi(\partial_{S_1}(i)) \mid 1 \leq i \leq k\} = \{\partial_{S_2}(i) \mid 1 \leq i \leq k\} \quad .$$

That is, boundary vertices of G_1 are mapped under ϕ to boundary vertices of G_2. And B_1 and B_2 are *fixed-boundary isomorphic* if there exists an isomorphism ϕ such that

$$\phi(\partial_{S_1}(i)) = \partial_{S_2}(i) \text{ for } 1 \leq i \leq k \quad .$$

That is, each boundary vertex i of G_1 is mapped under ϕ to boundary vertex i of G_2.

Our goal is to find an enumeration scheme that generates each free-boundary isomorphic t-parse at most once.

2.3.1 Canonic pathwidth t-parses

One simple way of generating each free-boundary isomorphic t-parse is to define equivalence classes of t-parses and generate one representative for each class of free-boundary isomorphic graphs. A t-parse is said to be *canonic*, with respect to some linear ordering of t-parses, if it is a minimum t-parse within its equivalence class. For an enumeration scheme, these minimum representatives should be defined so that extending the set of canonic representatives of order n generates a set (or superset) of the canonic representatives of order $n+1$. Here, if we generate a non-canonic t-parse of order $n+1$ we discard it before generating the canonic representatives of order $n+2$. We call an enumeration order (scheme) *simple* if this property holds.

The simplest linear ordering of t-parses is a lexicographical order of the parse strings. We define the *lex-canonic order* between two t-parses by comparing operators at equal indices (from the beginning of each string) until a difference is found. The individual operators in Σ_t are related as follows:

1. Vertex operator $\textcircled{i}$ is less than any edge operator $\boxed{j\ k}$.
2. Vertex operator $\textcircled{i}$ is less than any vertex operator $\textcircled{j}$ whenever $i < j$.
3. Edge operator $\boxed{i\ j}$, where $i < j$, is less than edge operator $\boxed{k\ l}$, where $k < l$, whenever $i < k$, or $i = k$ and $j < l$.

A canonic t-parse in the lex-canonic order $<_l$ (called a *lex-canonic t-parse*) is any t-parse G such that $G <_l H$ for any free-boundary isomorphic t-parse $H \simeq_{\partial_f} G$ where $H \neq G$. Since any prefix of a lex-canonic t-parse is also lex-canonic, a simple enumeration scheme is possible using this form of canonicity. (This prefix property is easily seen by assuming that a prefix P of a lex-canonic t-parse $G = P \cdot S$ is not lex-canonic then a contradiction arises regarding G being lex-canonic. That is, consider a lexicographically less t-parse $P' \cdot S' <_l G$ that is free-boundary isomorphic to G, where $P' <_l P$ and $P' \simeq_{\partial_f} P$.)

Example 31: Several t-parses are listed below in increasing lex-canonic order. The fourth t-parse is not lex-canonic since it is free-boundary isomorphic to the second

t-parse. Likewise, the fifth t-parse is free-boundary isomorphic to the first (which is also lex-canonic).

$$[\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \textcircled{0}, \boxed{0\ 1}]$$
$$[\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \boxed{0\ 2}, \textcircled{0}, \boxed{0\ 1}, \boxed{0\ 2}]$$
$$[\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \boxed{0\ 2}, \textcircled{0}, \boxed{0\ 1}, \boxed{0\ 2}, \textcircled{1}, \boxed{0\ 1}]$$
$$[\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \boxed{1\ 2}, \textcircled{1}, \boxed{0\ 1}, \boxed{1\ 2}]$$
$$[\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \boxed{1\ 2}, \textcircled{2}]$$

We now present an alternate canonic representation for pathwidth t-parses. The main benefit of this scheme, based on a non-lexicographical order, is that most non-canonic t-parses are easily determined (by checking for required canonicity properties of the parse string). For any t-parse P, let $vseq(P)$ be the vertex subsequence of the parse, that is, just the vertex operators $\textcircled{i}$ for some i, and let $vpos(P)$ be the positions of the vertex operators in the original t-parse.

Example 32: For the t-parse

$$G = [\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \boxed{0\ 2}, \textcircled{0}, \boxed{0\ 1}, \boxed{0\ 2}, \textcircled{1}, \boxed{0\ 1}, \textcircled{2}]$$

we have

$$vseq(G) = (\textcircled{0}, \textcircled{1}, \textcircled{2}, \textcircled{0}, \textcircled{1}, \textcircled{2})$$

and

$$vpos(G) = (1, 2, 3, 6, 9, 11) \quad .$$

Definition 33: For two t-parses P_1 and P_2 of the same free-boundary graph, we define a *linear order* $<_c$ as follows where the symbol $<$ denotes the lexicographical order on integer sequences and $<_l$ denotes the lex-canonic order on Σ_t operator sequences.

1. If $vpos(P_1) < vpos(P_2)$ then $P_1 <_c P_2$.

2. Else if $vseq(P_1) <_l vseq(P_2)$ then $P_1 <_c P_2$.
   [Here $vpos(P_1) = vpos(P_2)$.]

3. Else if $P_1 <_l P_2$ then $P_1 <_c P_2$.
   [Here $vpos(P_1) = vpos(P_2)$ and $vseq(P_1) = vseq(P_2)$.]

The order $<_c$ is a linear order because $<_l$ is a linear order (i.e., if case 3 is ever reached then either $P_1 <_c P_2$ or $P_2 <_c P_1$ holds).

For the remainder of this section, a t-parse P of a free-boundary graph G is termed *canonic* if there is no other parse P' such that $P' \simeq_{\partial_f} P$ and $P' <_c P$. The idea behind this t-parse order is that we want vertex operators to come earlier in the parse. For the linear order $<_c$ we note that any vertex operator $\textcircled{i}$ and any edge operator $\boxed{j\ k}$ do not need to be compared lexicographically.

We now state some useful facts about canonic t-parses (using the $<_c$ order).

Lemma 34: Let $G_n = [g_1, g_2, \ldots, g_n]$ be a canonic t-parse. If $g_{k_1} = \textcircled{i}$ and $g_{k_2} = \textcircled{j}$ are consecutive vertex operators, then for any edge operator $g_k = \boxed{a\ b}$, $k_1 < k < k_2$, either $a = j$ or $b = j$.

Proof. Assume that, for some $k_1 < k < k_2$, $g_k = \boxed{a\ b}$ where $a \neq j$ and $b \neq j$. Clearly the following parse, G'_n, represents the same free-boundary graph.

$$G'_n = [g_1, g_2, \ldots, g_{k_1} = \textcircled{i}, \ldots, g_{k-1}, g_{k+1}, \ldots, g_{k_2} = \textcircled{j}, g_k = \boxed{a\ b}, g_{k_2+1}, \ldots, g_n]$$

But $vpos(G'_n)$ is lexicographically less than $vpos(G_n)$. This can not happen since G_n is canonic. So, we must have either $a = j$ or $b = j$ in order to prevent the possible edge shift. □

The previous lemma states that any edge operators that immediately precedes a vertex operator $\textcircled{i}$ must be adjacent to the previous $\textcircled{i}$. The next result regarding the position of the boundary edges follows easily from the previous lemma.

Lemma 35: Let G_n be a canonic t-parse. If $g_m = \boxed{i\ j}$ is a boundary edge of G_n, $1 \leq m < n$, then there are no vertex operators in positions $m+1, m+2, \ldots, n$.

Proof. If not, the next vertex operator would have to be $\textcircled{i}$ or $\textcircled{j}$. □

The following lemma states that any prefix of a canonic t-parse is also canonic. This means that using $<_c$ to define t-parse canonicity is suitable for a simple t-parse enumeration scheme.

Lemma 36: If G_n is a canonic t-parse then the prefix G_{n-1} is a canonic t-parse.

Proof. If this is not true, then there exists $H_{n-1} \simeq_{\partial_f} G_{n-1}$, with $H_{n-1} <_c G_{n-1}$. Let $\sigma : V(G_{n-1}) \to V(H_{n-1})$ be a free-boundary isomorphism mapping between G_{n-1} and H_{n-1}. Define

$$h_n = \begin{cases} \textcircled{\sigma_i} & \text{if } g_n = \textcircled{i} \\ \boxed{\sigma_i\ \sigma_j} & \text{if } g_n = \boxed{i\ j} \end{cases} .$$

Now consider $H_n = H_{n-1} \cdot h_n$. The parse H_n is isomorphic to G_n with $H_n <_c G_n$. So we must have $H_{n-1} \not<_c G_{n-1}$. □

We now turn to the problem of determining when an extension of a canonic t-parse is also canonic.

Lemma 37: Let G_n be a canonic t-parse. If $Z = [\boxed{i\ j}]$ is a single edge operator extension, then $G_{n+1} = G_n \cdot Z$ can be tested for canonicity with a constant number of isomorphism calls.

Proof. Suppose G_{n+1} is not canonic. Then there exists a free-boundary isomorphic t-parse $H_{n+1} \simeq_{\partial_f} G_{n+1}$ with $H_{n+1} <_c G_{n+1}$. Since both G_{n+1} and H_{n+1} contain boundary edges we know from Lemma 35 that the last vertex operator has index $k \leq n$ (that is, $k = n + 1 -$ "number of boundary edges"). An isomorphism mapping $\sigma : V(G_{n+1}) \to V(H_{n+1})$ shows that the t-parses H_k and G_k are free-boundary isomorphic. This is seen by noting that for any edge (a, b) of G_{n+1}, $G_{n+1} \setminus \{(a, b)\} \simeq_{\partial_f} H_{n+1} \setminus \{(\sigma_a, \sigma_b)\}$. Since G_n is canonic the prefix G_k is also canonic (likewise H_k), so $H_k \simeq_{\partial_f} G_k$ implies $H_k = G_k$.

Thus, if G_{n+1} is not canonic then its canonic t-parse representation is identical to G_{n+1} except for the boundary edges at the end of the parse. If there are i boundary edges then we have at most $\binom{\binom{t+1}{2}}{i}$ possible t-parses to consider. □

Example 38: Below is an instance of where a constant number of isomorphism calls would tell us that the extended t-parse H is not canonic. This example is created by adding an edge between a pendent vertex and an isolated boundary vertex of the

canonic t-parse G. The t-parse K is free-boundary isomorphic to H and canonic.

$$G = [\textcircled{0}, \textcircled{1}, \textcircled{2}, \textcircled{3}, \boxed{0\ 1}, \textcircled{0}, \boxed{0\ 2}]$$
$$H = [\textcircled{0}, \textcircled{1}, \textcircled{2}, \textcircled{3}, \boxed{0\ 1}, \textcircled{0}, \boxed{0\ 2}, \boxed{1\ 3}]$$
$$K = [\textcircled{0}, \textcircled{1}, \textcircled{2}, \textcircled{3}, \boxed{0\ 1}, \textcircled{0}, \boxed{0\ 1}, \boxed{2\ 3}]$$

We can easily eliminate an isomorphism check for several of the possible t-parses mentioned in the previous lemma. For a prospective t-parse H_{n+1} to be free-boundary isomorphic to G_{n+1}, the degree sequences of the boundaries must be identical. These degree sequences can be refined to include both the number of boundary and non-boundary incident edges. That is, these two "ordered pair" degree sequences (one for G_{n+1} and one for H_{n+1}) must coincide.

Observation 39: Let G_n be a canonic t-parse. If $Z = [\textcircled{i}]$, a single vertex operator extension, then $G_n \cdot Z$ is non-canonic if Lemma 34 is violated, which is likely and easy to check.

The next lemma helps us detect other non-canonic situations for any t-parse of length n that ends with a vertex operator.

Lemma 40: With $m < n$ being the smallest index of any edge operator of a canonic t-parse G_n, there are no consecutive vertex operators g_i and g_{i+1} in G_n, for $m < i < n$.

Proof. First consider two identical vertex operators $\textcircled{i}$ consecutive in G_n. One of these can be replaced by a $\textcircled{0}$ and moved to the first of the string since the semantics of $[\ldots,\textcircled{i},\textcircled{i}]$ causes an internal isolated vertex. Now a suffix of G_n with two different consecutive vertex operators can be rewritten as (assuming G_{n-1} is canonic)

$$G_{n-1} \cdot [\textcircled{j}] = [\ldots, \boxed{i\ k}, \boxed{i\ j}, \textcircled{i}, \textcircled{j}] >_c [\ldots, \boxed{i\ j}, \textcircled{j}, \boxed{i\ k}, \textcircled{i}]$$

that is less in the $<_c$ order. If there are more than two consecutive vertex operators, $[\ldots, \boxed{a\ b}, \textcircled{i}, \textcircled{j}, \textcircled{k}, \ldots]$ then we can also shift one of these earlier. Here, the vertex that can be shifted before the edge operator $\boxed{a\ b}$ is determined by $\{i,j,k\} \setminus \{a,b\}$. □

One thing that is not quite resolved is how to efficiently handle the not so easy vertex operator cases. If both Lemmas 34 and 40 are not violated we still do not know if a t-parse $G_n \cdot [\textcircled{i}]$ is canonic. We believe that a non-canonic t-parse can pass both these lemmas for only a very few special cases (maybe not even enough times to worry about). Even without a fast canonic algorithm for this case, we still have a fast method for generating all bounded pathwidth graphs. Since the set of graphs of pathwidth at most t is obtained from the set of canonic t-parses (and we generate a superset), the above statements provide us with an efficient means of generating each such partial t-path with a constant sized boundary exactly once. If one wishes, a free-boundary isomorphism algorithm can be used to check for redundancies [SD76].

2.3.2 Canonic treewidth t-parses

It is known that both free and rooted tree generation can be done efficiently (in constant amortized time) with one of the algorithms given in [Wil89, WROM86, BH80]. To algebraically represent a graph of bounded treewidth (i.e., a generalized "bounded-width" tree) we model its underlying tree decomposition structure as an equivalent tree (of maximum degree three) using our treewidth operator set Σ_t (see Theorem 23). Again, since many tree decompositions of minimum width exist for a given graph we strive for a canonic representation. In the discussion given below, we incorporate the additional binary operator $\oplus$ within the pathwidth lexicographical canonic scheme (i.e., we expand the domain for the $<_l$ order).

We view a treewidth t-parse T as a rooted parse tree where the current set of active boundary vertices are inferred from the operator semantics.

The first simplification for our enumeration scheme is to restrict the t-parse arguments for the $\oplus$ operator. We simply require that no boundary edges occur in both t-parses G_1 and G_2 before $G_1 \oplus G_2$ is enumerated. Any edge that needs to be adjacent to two boundary vertices is added with edge operators after (or above) the circle plus.

To compare two t-parses that have different underlying tree decomposition trees, we use a ranking method similar to the one commonly used for ranking rooted trees [BH80]. The idea for our new linear order (details to come shortly) is to define another

set of t-parse equivalence classes (with respect to each tree decomposition structure) and then linearly order these classes.

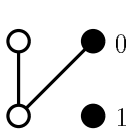
Definition 41: Let $v(G)$ denote the number of vertex operators in a t-parse G. A *signature* $s(G)$ for a t-parse G is an integer sequence defined recursively with respect to the root's operator type:

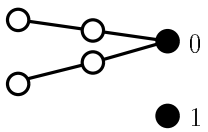
1. If $G = G_1 \cdot [\boxed{i\ j}\,]$ then $s(G) = s(G_1)$.
2. If $G = G_1 \cdot [\textcircled{i}\,]$ then $s(G) = [v(G), s(G_1)]$.
3. If $G = G_1 \oplus G_2$, where without loss of generality $s(G_1) \leq s(G_2)$, then $s(G) = [s(G_1), s(G_2)]$.

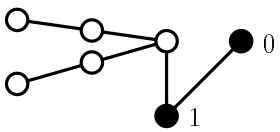
A signature s_1 is less than a signature s_2 if $|s_1| < |s_2|$, or $|s_1| = |s_2|$ and $s_1 < s_2$ in lexicographic order.

For any two t-parses G_1 and G_2 that have the same signature, let $(B_1^1, B_2^1, \ldots)$ and $(B_1^2, B_2^2, \ldots)$ be the pathwidth t-parse branches of G_1 and G_2, respectively, obtained from a post-order traversal of the structural tree. These vertex and edge operators are sequenced from leaves to root. Two t-parses G_1 and G_2 (not necessarily free-boundary isomorphic) can be compared lexicographically by comparing B_i^1 with B_i^2, starting at $i = 1$, and increasing i until a difference is found.

Example 42: A treewidth 1-parse is shown below for $S(K_{1,3})$, a subdivided $K_{1,3}$. (This pathwidth one obstruction is also shown in Figure 2.3 (a).)







$A = [\textcircled{0}, \textcircled{1}, \boxed{0\ 1}, \textcircled{0}, \boxed{0\ 1}, \textcircled{1}]$ $\qquad A \oplus A$ $\qquad (A \oplus A) \cdot [\boxed{0\ 1}, \textcircled{0}, \boxed{0\ 1}\,]$

With the far right edge operator $\boxed{0\ 1}$ being the root, the signature of this 2-boundaried graph is $[9, 4, 3, 2, 1, 4, 3, 2, 1]$.

Definition 43: A treewidth t-parse G is *structurally canonic* if it has the smallest signature over all t-parse representations of graphs free-boundary (fixed-boundary) isomorphic to G. In addition, the t-parse G is *treewidth canonic* if it is the lexicographic minimum t-parse over all structurally canonic representations within the free-boundary (fixed-boundary) isomorphic equivalence class. We call these orderings of treewidth t-parses the free-boundary (or fixed-boundary) *lex-rank order*.

It is understood from the context whether we are talking about the free or fixed boundary cases, with the latter case being more common.

Lemma 44: If a graph G of treewidth t also has pathwidth t then the structurally canonic (and treewidth canonic) t-parse for G has no $\oplus$ operators.

Proof. This follows from the fact that a t-parse with more $\oplus$ operators also has more vertex operators (the circle plus operator absorbs $t+1$ vertices). Since the length of a signature for a t-parse equals the number of vertex operators, any signature without $\oplus$ operators (i.e., a pathwidth t-parse) is less in the treewidth t-parse comparison order than any non-trivial treewidth t-parse. □

The above lemma allows us to do computations (enumerations) for pathwidth t-parses and then reuse (if using the lex-canonic pathwidth scheme) any partial results for these t-parses when computing within the treewidth t-parse domain. For example, for our obstruction set searches we would keep any proofs of graph minimality or nonminimality from a search that was restricted to pathwidth t-parses.

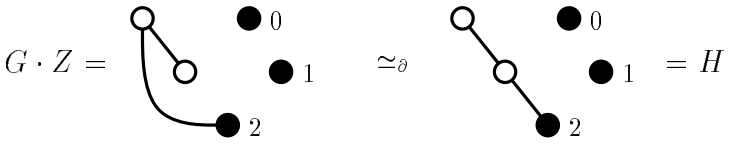
Our treewidth analog to Lemma 36's "prefix of canonic is canonic" is given below.

Lemma 45: For the fixed-boundary case, any rooted induced subtree S of a treewidth canonic t-parse T is also treewidth canonic.

Proof. By induction on the number of operators, it suffices to look at prefixes with one less operator. Let g_n denote the last operator of T. The validity of this statement for the pathwidth operators ($g_n = \textcircled{i}$ or $g_n = \boxed{j\ k}$) follows from the left associative semantic interpretation of these unary operators. That is, if there exists a prefix S with a more canonic parse, then applying the unary operator g_n to S contradicts

T being treewidth canonic. For the treewidth operator case, $g_n = \oplus$, assume $T = G_1 \oplus G_2$ is a treewidth canonic t-parse. By definition of treewidth canonic, we can also assume $G_1 \leq G_2$ (in lex-rank order). If there exists a t-parse H_1 that is fixed-boundary isomorphic to G_1 that has a smaller rank, then the fixed-boundary isomorphic t-parse $T' = H_1 \oplus G_2$ contradicts the fact that T is treewidth canonic. This contradiction can take place either in or outside T's structurally canonic equivalence class. That is, if $s(H_1) < s(G_1)$ then we can find a better structural equivalence class for T. The same argument holds for the child parse G_1 replaced with G_2. □

As a consequence of the above lemma there is a simple enumeration scheme (as was given in Section 2.3.1 for the pathwidth t-parses) for generating all fixed-boundaried t-parses of treewidth at most t. Note that like our free-boundary pathwidth case, extending a canonic fixed-boundaried t-parse G may yield a non-canonic t-parse $G \cdot Z$, but any prefix (subtree) of a fixed-boundary canonic H is canonic. An example of the former problem is given below.



$$[\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \boxed{0\ 2}, \textcircled{0}] \cdot [\textcircled{1}] \simeq_\partial [\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \textcircled{0}, \boxed{1\ 2}, \textcircled{1}]$$

The subtree property of Lemma 45 does not hold for the free-boundary treewidth case since the semantics of the binary operator $\oplus$ require fixed-boundary vertices. For example, consider the free-boundary canonic 1-parse $A \oplus B$, where A is taken from Example 42 and $B = [\textcircled{0}, \textcircled{1}, \boxed{0\ 1}, \textcircled{1}, \boxed{0\ 1}, \textcircled{0}]$. Here both subtrees A and B are free-boundary isomorphic but B is not treewidth canonic for the free-boundary case.

We end this section with an interesting open computational problem regarding how "linear" a tree decomposition can be for a given graph. The structurally canonic t-parses, in some sense, have the simplest underlying treewidth structure for $(t+1)$-boundaried graphs (see proof of Lemma 44).

Problem 46: Closeness to Pathwidth t

Input: A graph G of treewidth t.

Question: How close is G to pathwidth t based on how many $\oplus$ operators are needed for a t-parse representation? (That is, how many degree three vertices of T are needed over all smooth tree decompositions $(T, \{X_i \mid i \in T\})$ of width t for G?)

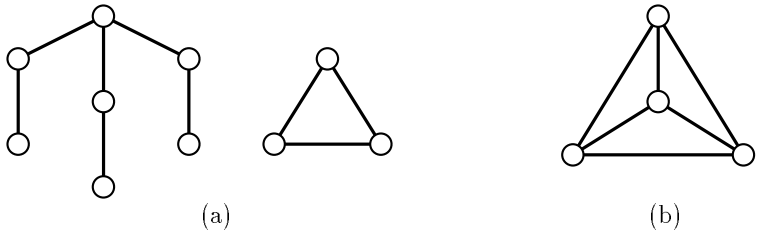


Figure 2.3: The obstructions to (a) pathwidth 1 and (b) treewidth 2.

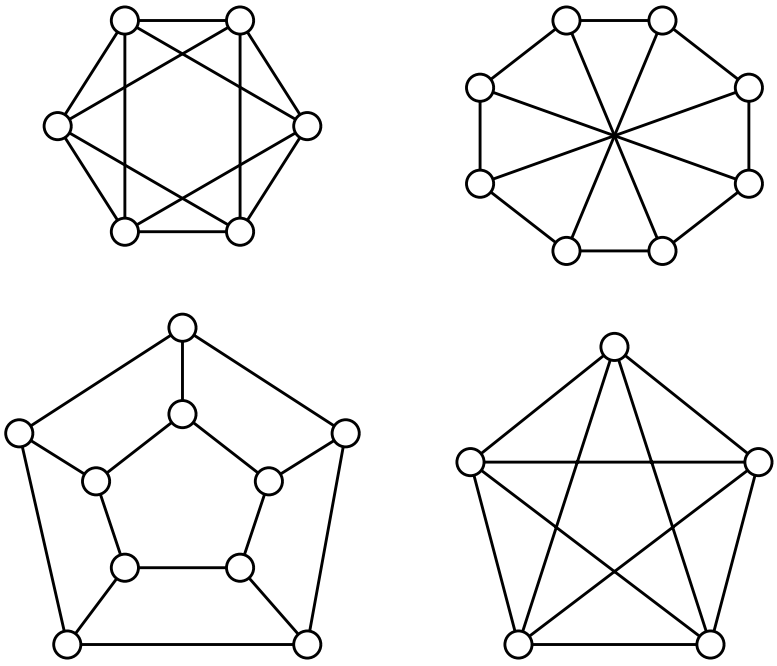


Figure 2.4: The minor-order obstructions to treewidth 3.

Chapter 3

Graph Minors and Well-Quasi-Orders

In this chapter, we formally introduce the notion of minor-order forbidden graphs, which we have been calling obstructions. We also explain why for many classes of graphs (i.e., the minor-order lower ideals) there exists a finite number of these family-characterizing graphs. Lastly, we define a closely related partial order for k-boundaried graphs. This order is the mathematical foundation for our obstruction set search theory that is presented in Chapter 4.

3.1 Preliminaries

We begin with a review of a few definitions from set theory that are significant for characterizations of graph families by obstruction sets.

Definition 47: A *quasi-order (preorder)* is a reflexive and transitive binary relation. A *partial order* is an anti-symmetric quasi-order. A partially ordered set is called a *poset*.

A simple example of a quasi order $\leq_q$ of graphs that is not a partial order is the order defined by comparing the number of vertices of the graphs (i.e., $G \leq_q H$ if and only if $|G| \leq |H|$).

Definition 48: A set S with a quasi-order $\leq$ is a *well-quasi-order* if for any countable sequence $(s_1, s_2, \ldots)$ of members of S, there are indices $1 \leq i < j$ such that $s_i \leq s_j$.

Recall that a *subgraph* H of a graph G is a subset of the vertices and edges of G such that whenever a vertex $v \in V(G) \setminus V(H)$ then any edge of $E(G)$ incident to v is not in $E(H)$. We (combinatorially) *contract* an edge $e = (u, v)$ in G to form a edge contracted graph $H = G/e$ by first deleting e and then replacing vertices u and v with a new vertex w such that any edge incident to u or v is now incident to w. Also recall the following graph order that is of special interest to this dissertation.

Definition 49: For two graphs G and H, the graph H is a *minor* of the graph G, written $H \leq_m G$, if a graph isomorphic to H can be obtained from G by taking a subgraph and then contracting (possibly zero) edges. A *minor operation* is any edge deletion, edge contraction, or isolated vertex deletion.

Observation 50: The minor-order relation $\leq_m$ defines a partial order on the family of finite simple graphs.

Figure 3.1 shows a Hasse diagram (part of a minor-order poset) for all edge contractions of a graph with 5 vertices.

Definition 51: A *lower ideal* $(I, \leq)$ of a quasi-order $(S, \leq)$ is a subset $I \subseteq S$ such that for any $x \in I$, if $y \leq x$ then $y \in I$.

For a special case, we also say that a family $\mathcal{F}$ of graphs (within the set of all simple graphs) is *closed* under the minor order if whenever G is in $\mathcal{F}$ and H is a minor of G, $H \leq_m G$, together imply that H is also in $\mathcal{F}$. This equivalent to saying that $(\mathcal{F}, \leq_m)$ is a lower ideal. For minor-order lower ideals, the forbidden graphs are defined as follows.

Definition 52: The *obstruction set* $\mathcal{O}(\mathcal{F})$ for a minor-closed family $\mathcal{F}$ of graphs is the set of minimal graphs (in the minor order) in the complement $\overline{\mathcal{F}}$. Specifically, G is an obstruction for $\mathcal{F}$ (i.e., $G \in \mathcal{O}(\mathcal{F})$) if and only if G is in $\overline{\mathcal{F}}$ (i.e., $G \notin \mathcal{F}$) and for any proper minor H of G, H is in $\mathcal{F}$ (i.e., $H \notin \overline{\mathcal{F}}$).

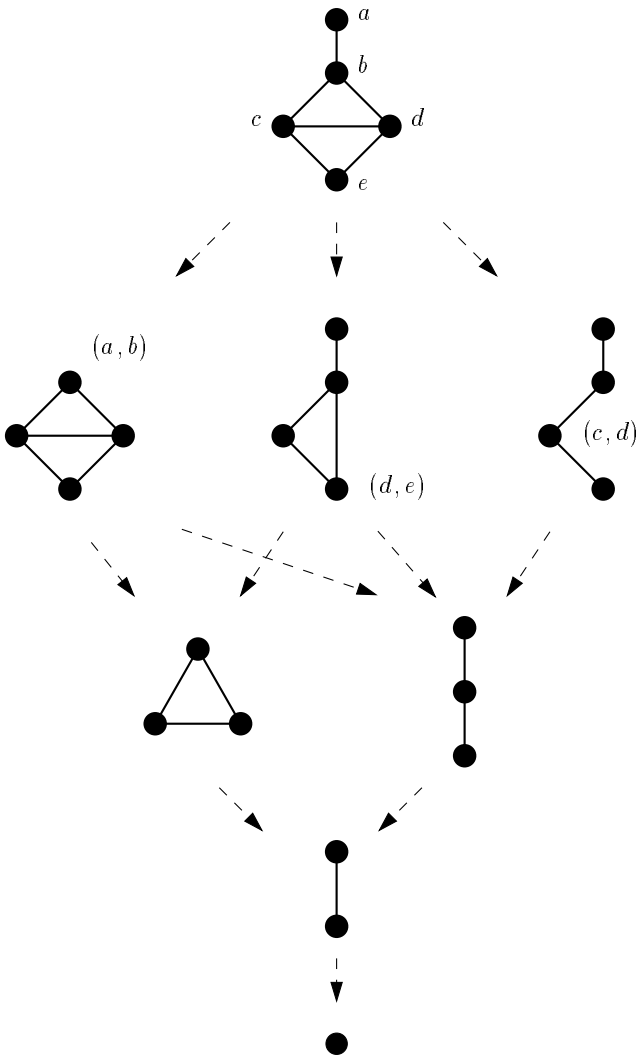


Figure 3.1: Illustrating an edge contraction poset.

For example, the obstructions for the family of planar graphs are K_5 and $K_{3,3}$ since any graph that is not planar must contain one of these two as a minor (i.e., these two obstructions are the minimal non-planar graphs within the minor-order poset of all graphs). In general, any graph in the complement of a lower ideal $\mathcal{F}$ must contain at least one obstruction as a minor.

3.2 The Graph Minor Theorem

How are obstruction sets used to characterize families of graphs that are closed under the minor order? It is not obvious that the set of obstructions for a minor-order lower ideal must be finite. The recently celebrated Graph Minor Theorem (stated below) by Robertson and Seymour guarantees a finite characterization for all minor-order lower ideals [RS85b]. This statement was known as Wagner's Conjecture for a long time [Wag37, RSc].

Theorem 53: (Robertson–Seymour) The minor order is a well-partial-order.

Corollary 54: There are only a finite number of minor-order minimal graphs in any set of graphs. In particular, any minor-order lower ideal of graphs has a finite number of obstructions.

Proof. Let $S = (G_1, G_2, \ldots)$ be any countable sequence of distinct minor-order minimal graphs. Since the minor-order is a well-partial-order, there must be an index $i < j$ such that $G_i \leq_m G_j$. However, this contradicts the fact that G_j is minimal. So there must be a finite number of minor-order minimal graphs. □

The bad news regarding this graph theoretical breakthrough is that Robertson and Seymour's proof is nonconstructive. (For more information see [FRS87].) This means that for any set of graphs we only know that the set of minor-order minimal graphs is finite but currently have no general method for obtaining it. To tackle this predicament, for our obstruction set computations, we use the following variation of the minor order. The elements of this partial order are k-boundaried graphs.

Definition 55: Let G be a k-boundaried graph. A k-boundaried graph H is a *∂-minor (boundaried minor)* of G, denoted $H \leq_{\partial m} G$, if H is a minor of G such that no boundary vertices of G are deleted by the minor operations, and the boundary vertices of H are the same as the boundary vertices of G.

Definition 56: Let G be a k-boundaried graph. A k-boundaried graph H is a *one-step ∂-minor* of G if H is obtained from G by a single minor operation: one isolated vertex deletion, one edge deletion, or one edge contraction (and $H \leq_{\partial m} G$).

Lemma 57: The *∂-minor order* $\leq_{\partial m}$ on k-boundaried graphs is a partial order.

Proof. Assume $K \leq_{\partial m} H$ and $H \leq_{\partial m} G$ for the k-boundaried graphs K, H and G. If we do the one-step ∂-minor operations used to show $K \leq_{\partial m} H$ after the one-step ∂-minor operations used to show that $H \leq_{\partial m} G$, we see that transitivity holds (i.e., $K \leq_{\partial m} G$). Also, since each one-step ∂-minor of a graph reduces either the number of edges or the number of vertices, the ∂-minor order is anti-symmetric. □

Lemma 58: If a family $\mathcal{F}$ of graphs is a lower ideal under the minor order $\leq_m$ then the family $\mathcal{F}' = \{(G, S) \mid G \in \mathcal{F}$ and S is a k-simplex of $G\}$ of k-boundaried graphs is a lower ideal in the ∂-minor order $\leq_{\partial m}$.

Proof. This follows from the fact that the number of graph-reducing minor operations for the ∂-minor order is a subset of the minor-order operations. □

For our obstruction set searches concerning this ∂-minor order, the width of each k-boundaried graph is bounded by a fixed pathwidth or treewidth. Specifically, we restrict the search to the set of t-parses ($t = k - 1$).

Lemma 59: If a family $\mathcal{F}$ of graphs is a lower ideal under the minor order then the family $\mathcal{F}'$, consisting of $\mathcal{F}$ restricted to t-parses, is a lower ideal under the ∂-minor order $\leq_{\partial m}$.

Proof. Follows from Lemma 58 and the next result (Lemma 60), which shows that the set of t-parses is closed under the ∂-minor order. □

The next two results show that every ∂-minor of a t-parse has a representation as a t-parse. We know that the family of bounded pathwidth graphs is closed under the minor order. However, this fact does not guarantee that the boundary vertices are preserved by the minor operations. We actually implemented a different but similar ∂-minor algorithm from the one used in the proof below.

Lemma 60: Let G be a pathwidth t-parse, and suppose H is a ∂-minor of G. Then H has a representation as a pathwidth t-parse.

Proof. Assume $G = [g_1, g_2, \ldots, g_n]$ is the pathwidth t-parse. We show how to construct a t-parse for any one-step ∂-minor of G. The pathwidth t-parse for any minor H of G can then be constructed by repeating these one-step ∂-minor procedures.

1. **Delete an isolated non-boundary vertex**:
   If operator $g_i = \textcircled{u}$, $1 \leq i < n$ is both a non-boundary vertex (i.e., $g_j = \textcircled{u}$ for some $i < j \leq n$) and an isolated vertex (i.e., $g_k \neq \boxed{u\ v}$ for $u \neq v$ and $i < k < j$), then $[g_1, g_2, \ldots, g_{i-1}, g_{i+1}, \ldots, g_n]$ represents a "deleted isolated vertex" minor $G \setminus \{u\}$. (Note that we could have just as easily deleted g_j.)

2. **Delete an edge**:
   If $g_i = \boxed{u\ v}$, $1 \leq i \leq n$ is any edge operator, then $[g_1, g_2, \ldots, g_{i-1}, g_{i+1}, \ldots, g_n]$ represents a "deleted edge" minor $G \setminus \{(u, v)\}$.

3. **Contract a non-boundary edge**:
   If $g_i = \boxed{u\ v}$, $1 \leq i < n$ is any edge operator such that, for $i < c \leq n$, $g_c = \textcircled{u}$ or $g_c = \textcircled{v}$ then do the following steps to create a "contracted edge" minor $G/(u, v)$. First, without loss of generality, G is represented in one of the following two forms

   $$[g_1, \ldots, g_a = \textcircled{u}, \ldots, g_b = \textcircled{v}, \ldots, g_i = \boxed{u\ v}, \ldots, g_c = \textcircled{v}, \ldots, g_n] \quad (*)$$

   or

   $$[g_1, \ldots, g_a = \textcircled{u}, \ldots, g_b = \textcircled{v}, \ldots, g_i = \boxed{u\ v}, \ldots, g_c = \textcircled{u}, \ldots, g_n],$$

   where $g_j \neq \textcircled{u}$ for $a < j < c$ and $g_j \neq \textcircled{v}$ for $b < j < c$. Note that the vertex operator $\textcircled{v}$ may occur between g_a and g_b.

For the first form, use the t-parse

$$[g_1, \ldots, g_a, \ldots, g_{b-1}, g'_{b+1}, \ldots, g'_{i-1}, g'_{i+1}, \ldots, g'_{c-1}, g_c, \ldots, g_n],$$

where for $b < k < c$,

$$g'_k = \begin{cases} \boxed{u\ x} & \text{if } g_k = \boxed{v\ x} \text{ and } x \neq v \\ g_k & \text{otherwise} \end{cases},$$

to represent the minor G/g_i.

For the second form, first create from G a t-parse $[g'_1, g'_2, \ldots, g'_{c-1}, g_c, \ldots, g_n]$, where for $1 \leq k < c$,

$$g'_k = \begin{cases} \boxed{u\ x} & \text{if } g_k = \boxed{v\ x} \text{ and } x \notin \{u, v\} \\ \boxed{v\ x} & \text{if } g_k = \boxed{u\ x} \text{ and } x \notin \{u, v\} \\ \textcircled{u} & \text{if } g_k = \textcircled{v} \\ \textcircled{v} & \text{if } g_k = \textcircled{u} \\ g_k & \text{otherwise} \end{cases},$$

to guarantee that the boundary will be preserved. The new t-parse satisfies (*) so we can now use the previous construction. The preprocessing step is justified since the operator $g_c = \textcircled{u}$ in the prefix graph $G_c = [g_1, g_2, \ldots, g_c]$ tells us which boundary vertex to preserve; that is, the contracted edge between vertices represented by g_a and g_b becomes the active boundary vertex v in $G_c/(u, v)$.

(Recall that the semantics of the edge operator $\boxed{i\ j}$ prevent multi-edges from appearing.)

One can check that all three of these one-step ∂-minor procedures will preserve G's boundary. □

Observation 61: Any ∂-minor of a treewidth t-parse is a treewidth t-parse.

Proof. Deletion of an isolated non-boundary vertex and deletion of an edge is similar to that in the proof of Lemma 60. An isolated vertex may originate from an $\textcircled{i}$ appearing above or below a circle plus $\oplus$ in the t-parse tree. In this case we remove the vertex operator closest to the root of the parse tree just above the isolated vertex; there

must be one since we are trying to remove a non-boundary vertex. This operation is correct because the $\oplus$ operator merges together two previous isolated boundary vertices (with the same label) from different subtree parses. For the edge deletion case with respect to $\oplus$, we also have to be careful about two $\boxed{i\ j}$ operators being used to represent the same edge. However, removing all instances is easy.

Contracting a non-boundary edge is also possible by using a modified version of the pathwidth t-parse contraction algorithm. Let $e = (u, v)$ be an edge that we want to contract. For these two vertices u and v, we remove the corresponding vertex operator closer to the root instead of the lower indexed one. Again, we may have to permute the boundary to get the t-parse branches to satisfy (*); this happens when two vertex operators nearest the edge operator are both $\textcircled{v}$ (see proof of Lemma 60).

□

Definition 62: A t-parse (boundaried graph) G is a *∂-obstruction (boundaried obstruction)* of a lower ideal $\mathcal{F}$ if every proper ∂-minor of G is in $\mathcal{F}$ while G is not in $\mathcal{F}$.

For any minor-order lower ideal $\mathcal{F}$ and an obstruction $O \in \mathcal{O}(\mathcal{F})$, there exists a ∂-obstruction G in Σ_t^* such that $O \leq_m G$ trivially whenever $t \geq pw(O)$. In this case, the boundary of G (with $t+1$ vertices) can be any set of vertices X_1 from a smooth path decomposition $X_1, X_2, \ldots, X_r$ of width t for O. Let $B^t = \{(G_i^t, S_i^t) \mid i = 1, 2, \ldots, r_t\}$ denote the set of ∂-obstructions of width t for $\mathcal{F}$. If t is the pathwidth of the largest obstruction for a lower ideal $\mathcal{F}$ then

$$\mathcal{O}(\mathcal{F}) \subseteq \bigcup_{0 \leq j \leq t} \left(\bigcup_{1 \leq i \leq r_j} G_i^j \right) \quad .$$

The previous discussion also holds for obstructions of bounded treewidth. That is, for any smooth tree decomposition $(T, \{T_x\})$ of a graph G of width t, there exists a treewidth t-parse with the boundary of G's representation being the vertices of any set T_x of T. So, the set of underlying boundaried obstructions (for bounded treewidth) is a superset of the non-boundary obstructions.

Lemma 63: If $\mathcal{F}$ is a lower ideal in the minor order then there are a finite number of ∂-obstructions (in the ∂-minor order) for $\mathcal{F}$.

Proof. It suffices to show that any minor-order obstruction $O \in \mathcal{O}(\mathcal{F})$ is below at most a finite number of ∂-obstructions in the minor order. Let $O \leq_m B$ for some ∂-obstruction B. There is a sequence of length at most t of edge contractions between two boundary vertices and/or isolated boundary vertex deletions to produce O from B. If any other one-step minor operations were possible then B would not be a ∂-obstruction. For example, deleting any edge can be done first, which is a valid ∂-minor operation showing that B has a ∂-minor not in $\mathcal{F}$. Therefore, any ∂-obstruction above O can have at most $|O| + t$ vertices. □

Even though Lemma 63 guarantees that there are only a finite number of boundaried obstructions for any minor-order lower ideal, we have the following analog to the Graph Minor Theorem for t-boundaried graphs, which was first cited (without proof) and used in [FL89a]. The following simple proof uses Robertson and Seymour's more recent but less-known result that the set of finitely edge-colored graphs is also well-quasi-ordered under the minor order, denoted by $\leq_{cm}$. (The proof should appear in [RSf].)

Theorem 64: (Fellows–Langston) The ∂-minor order is a well-partial-order (on t-boundaried graphs and t-parses).

Proof. Given any countable sequence $G_1, G_2, \ldots$ of t-boundaried graphs we need to show that there exist indices $1 \leq i < j$ such that $G_i \leq_{\partial m} G_j$. To do this we first define a bijection ϕ, as illustrated in Figure 3.2, from t-boundaried graphs to edge-colored graphs using $(t+1)$ colors. For each boundary vertex labeled k, the function ϕ adds a single 'k'-colored pendent edge while removing the boundary label. The original noncolored edges in the domain graph of ϕ retain the 'blank' color in the image graph. Thus, this map produces a total $t+1$ colors in the image.

Since the range of ϕ is a well-partial-order, we consider the following sequence of $t+1$ edge-colored graphs: $\phi(G_1), \phi(G_2), \ldots$. Here, there must exist a $\phi(G_i) \leq_{cm} \phi(G_j)$ for $1 \leq i < j$. To prove our theorem, we show that

$$H' = \phi(H) \leq_{cm} \phi(K) = K' \;\Rightarrow\; H \leq_{\partial m} K \quad .$$

Let $M = m_1, m_2, \ldots, m_r$ be the sequence of one-step minor operations that shows $H' \leq_{cm} K'$. Since both H' and K' have exactly one edge of color 1 through t, none

of these colored edges were contracted or deleted within M. Also since none of the colored edges $1, 2, \ldots, t$ in H' are adjacent, there are no edge contractions between any edge $(u, v) \in E(K')$ where u and v are both incident to colored edges of K'. Thus, M constitutes a valid sequence of boundary minor operations on the underlying boundaried graph $\phi^{-1}(K') = K$, showing that $H \leq_{\partial m} K$. □

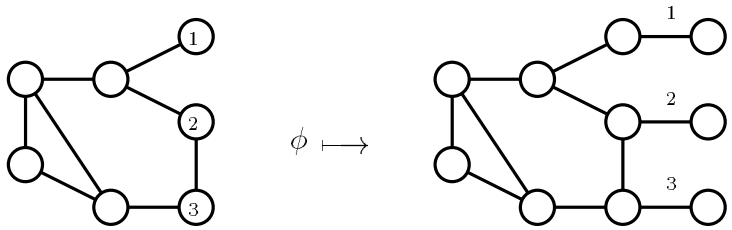


Figure 3.2: A map ϕ from boundaried graphs to edge-colored graphs.

3.3 Other Graph Partial Orders

This chapter's final section summerizes some other partial orders for graphs that appear in the field of graph theory. We remind the reader that the obstruction set characterizations presented in later chapters of this dissertation are all based on the minor order.

Perhaps the most natural partial order for graphs is the *subgraph* partial order. Here a graph H is below another graph G, often denoted $H \subseteq G$, if it is a subgraph. A slight variation of this order is call the (vertex) *induced subgraph* partial order. Here a graph H is an induced subgraph of a graph G, often denoted $H \leq G$, if there exists a subset V' of $V(G)$ such that $H = G \setminus V'$. We presented in Chapter 1 the induced subgraph obstructions (known as asteroidal triples) that prevent chordal graphs from being interval graphs.

It is easy to see that the subgraph and induced subgraph partial orders are not well-partial-orders since, for example, the set of cycles $\{C_i \mid i \geq 3\}$ is an infinite anti-chain (i.e., a countable set of incomparable graphs).

There are two well-known intermediate partial orders of graphs that fall between the structured minor order and the simple subgraph order. The first order is the

topological order. We define the topological (partial) order as a restricted minor order where only subgraphs and edges incident to a degree two vertex may be contracted. Other papers alternately define the topological order on graphs based on the following relation (e.g., see [Wil87]). Here a graph G topologically contains H, often denoted $H \leq_t G$, if some subgraph of G is isomorphic to a subdivision of H, where to 'subdivide' a graph is to replace its edges by paths that are vertex disjoint except at endpoints. The second intermediate order is the *induced minor order*. Here a graph H is an *induced minor* of a graph G, often denoted $H \leq_{im} G$, if H can be obtained by contracting edges from a induced subgraph of G.

For the topological order, Kruskal proved the following well-known result [Kru60].

Theorem 65: (Kruskal's Tree Theorem) The topological order is a well-quasi-order for the set of finite trees.

A stronger result stating that all finite rooted trees are well-quasi-ordered for the topological order (with a simple proof) is due to Nash-Williams [NW63]. Since the minor order is a stronger relation, Kruskal's Tree Theorem immediately implies that the minor order is a well-partial-order for trees.

The *immersion order* $\leq_i$ on graphs is defined by three types of graph reductions: deleting vertices, deleting edges, or lifting edges. Two adjacent edges (a, b) and (b, c) are *lifted* from a graph if they are replaced with a single edge (a, c). See Figure 3.3 (a). In addition to the Graph Minor Theorem, Robertson and Seymour have recently proven that the immersion order on graphs is a well-partial-order. (The proof of this fact will appear in [RSf] of the graph minor series.)

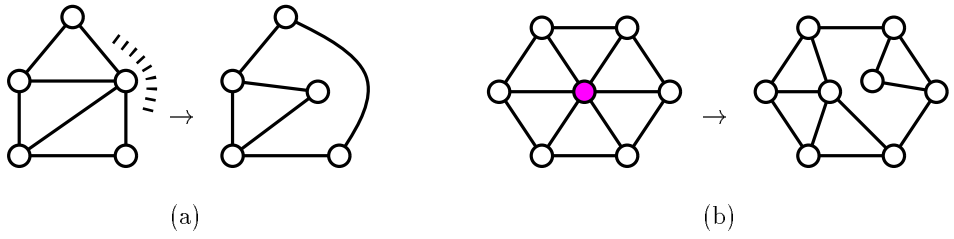


Figure 3.3: Illustrating the (a) lift and (b) fracture graph operations.

A related partial order termed the *weak immersion order* $\leq_{wi}$ is defined by allowing graphs to be related by repeated uses of these four graph operations: delete an isolated vertex, delete an edge, remove a subdivision, or *fracture* a vertex. A vertex v is fractured by replacing it with two new vertices v_1 and v_2 and partitioning the edges incident on v into two classes, making these incident, respectively, on v_1 and v_2 [Fel89]. See Figure 3.3 (b).

Makedon and Sudborough showed that the family of graphs with cutwidth at most 2 is characterized by a set of five forbidden weak immersion order (multi-) graphs [MS83]; these graphs are displayed in Figure 3.4 below. The family k–CUTWIDTH, graphs with cutwidth at most k, is also a lower ideal for the immersion order (e.g., see [Ram94]).

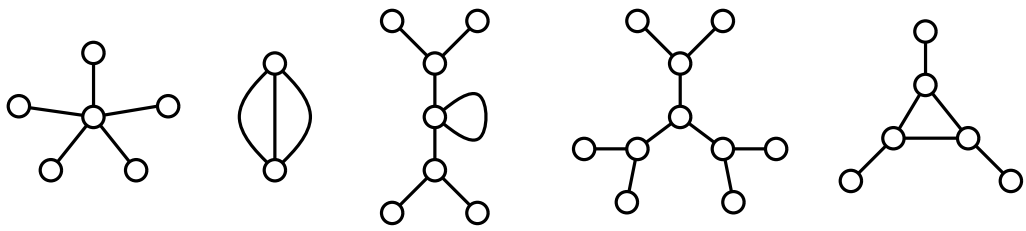


Figure 3.4: The weak immersion order obstruction set for 2–CUTWIDTH.

Chapter 4

Finding Forbidden Minors

We search for obstructions within the set of graphs of *bounded pathwidth* (or *bounded treewidth*). We now describe how to do this efficiently with our algebraic t-parse representation. For ease of exposition throughout this chapter, we limit ourselves to bounded pathwidth graphs in the obstruction set search theory, and only point out further information regarding a bounded treewidth search, as needed.

4.1 Key t-parse Properties

Recall that our concatenation partial order over t-parses is based on strings (or trees) of t-parse operators. This ordering of graphs is used during our enumeration of the graphs of bounded combinatorial width. As explained below, this enumeration process can be restricted to a finite search space consisting of the minimal graphs in the boundary minor order intersected with any finite-index family (lower ideal) congruence.

As we saw in Chapter 2, both the k–Pathwidth and k–Treewidth families of graphs are lower ideals in the minor order, so a ∂-minor H of a t-parse G is representable as a t-parse. One should keep in mind that these minor-order algorithms can actually operate on t-parses directly, bypassing any unnecessary conversion to and from the standard graph representations.

The following sequence of definitions and results form our theoretical basis for computing minor-order obstruction sets. We first recall a definition for the more familiar and general setting of k-boundaried graphs (e.g., see [FL89a]).

Definition 66: Let $\mathcal{F}$ be a fixed graph family and let G and H be k-boundaried graphs. We say that G and H are congruent with respect to the *canonical congruence* $\sim_{\mathcal{F}}$ if for each k-boundaried graph Z,

$$G \oplus Z \in \mathcal{F} \iff H \oplus Z \in \mathcal{F} \quad .$$

A k-boundaried graph Z is a *distinguisher* for $G \not\sim_{\mathcal{F}} H$ if $G \oplus Z \in \mathcal{F}$ and $H \oplus Z \notin \mathcal{F}$ (or vice versa).

For our setting of graphs of bounded combinatorial width (t-parses) we have the following analogous definition.

Definition 67: Let $\mathcal{F}$ be a fixed graph family and let G and H be t-parses. We say that G and H are *$\mathcal{F}$-congruent* (also written $G \sim_{\mathcal{F}} H$) if for each extension $Z \in \Sigma_t^*$,

$$G \cdot Z \in \mathcal{F} \iff H \cdot Z \in \mathcal{F} \quad .$$

If G is not congruent to H, then we say G is *distinguished* from H (by Z), and Z is a *distinguisher* for G and H. Otherwise, G and H *agree* on Z.

We call the above family congruence the *bounded-width canonical congruence* since it resembles the general canonical congruence for $\mathcal{F}$ on $(t+1)$-boundaried graphs. However, we also use the phrase "canonical congruence" loosely to mean either case depending on the context. The only difference here is that we restrict each Z, in the latter case, to be a parse string representing a graph of bounded combinatorial width. Recall that the operator $(\cdot)$ is called the t-parse *concatenation* operator.

The next definition provides a means for distinguishing graphs in different equivalence classes of the canonical congruence $\sim_{\mathcal{F}}$.

Definition 68: A set $T \subseteq \Sigma_t^*$ (or a set T of boundaried graphs) is a *testset* if $G \not\sim_{\mathcal{F}} H$ implies there exists a $Z \in T$ that distinguishes G and H with respect to $\mathcal{F}$.

As shown later, a testset is only useful for finding obstruction sets if it has finite cardinality. We are interested in canonical congruences for our obstruction set computations because of the following consequence of the GMT (see [AF93] and [Cou90a]–[Cou92b]).

Observation 69: The canonical congruence for any minor-order lower ideal $\mathcal{F}$ is of finite index for the set of t-parses.

Proof. This result follows by using the following two facts: (1) the GMT implies that the family $\mathcal{F}$ has a finite number of obstructions, and (2) there exists a finite state algorithm for each minor-containment problem for input graphs of bounded treewidth (by Courcelle and others). The second fact means that, for the family of graphs that excludes a single obstruction, there is a finite state automaton that accepts t-parses in that family. Since regular languages are closed under intersections, the bounded-width canonical congruence $\sim_{\mathcal{F}}$ is of finite index. (See Section 5.3.2 and Chapter 11 for further information.) □

The above observation can be improved to show that the canonical congruence $\sim_{\mathcal{F}}$ is of finite index for the set of k-boundaried graphs. This is because all obstructions of the lower ideal $\mathcal{F}$ have bounded treewidth. A *finite* testset for $\mathcal{F}$ is constructed by taking a finite union of finite testsets of boundary size k, for each width t up to the treewidth bound.

Definition 70: A t-parse G is *nonminimal* if G has a proper ∂-minor H such that $G \sim_{\mathcal{F}} H$. Otherwise, we say G is *minimal*. A t-parse G is a *∂-obstruction* if G is minimal and is not a member of $\mathcal{F}$.

In general, if a graph family $\mathcal{F}$ is a minor-order lower ideal and a t-parse G is minimal, then for each ∂-minor H of G, there exists an extension Z such that

1. $G \cdot Z \notin \mathcal{F}$ and
2. $H \cdot Z \in \mathcal{F}$.

That is, there exists a distinguisher (in this direction) for each possible ∂-minor H of the t-parse G.

The obstruction set $\mathcal{O}_{\mathcal{F}}$ for a family $\mathcal{F}$ is obtainable from the boundaried obstruction set $\mathcal{O}^{\partial}_{\mathcal{F}}$ (set of ∂-obstructions) by contracting boundary edges or deleting isolated boundary vertices, whenever the search space of width $\partial - 1$ is large enough. We will use the symbol $\mathcal{O}_{\mathcal{F}}$, or $\mathcal{O}(\mathcal{F})$, to also denote $\mathcal{O}^{\partial}_{\mathcal{F}}$ since the type of the obstructions will be clear from the context.

In our search for $\mathcal{O}^{\partial}_{\mathcal{F}}$, we must prove that each t-parse generated is minimal or nonminimal. The following two results drastically reduce the computation time required to determine these proofs.

Lemma 71: A t-parse G is minimal if and only if G is distinguished from each *one-step* ∂-minor of G. Or equivalently, G is nonminimal if and only if G is $\mathcal{F}$-congruent to a *one-step* ∂-minor.

Proof. We prove the second statement. Let G be nonminimal and suppose there exists two minors K and H of G such that $K \leq_{\partial m} H$ and $K \sim_{\mathcal{F}} G$. It is sufficient to show $H \sim_{\mathcal{F}} G$.

For any extension $Z \in \Sigma_t^*$, if $G \cdot Z \in \mathcal{F}$ then $H \cdot Z \in \mathcal{F}$ since $H \cdot Z \leq_{\partial m} G \cdot Z$ and $\mathcal{F}$ is a ∂-minor lower ideal. Now let Z be any extension such that $G \cdot Z \notin \mathcal{F}$. Since $K \sim_{\mathcal{F}} G$, we have $K \cdot Z \notin \mathcal{F}$. And since $K \cdot Z \leq_{\partial m} H \cdot Z$, we also have $H \cdot Z \notin \mathcal{F}$. Therefore, G is $\mathcal{F}$-congruent to H. □

Lemma 72 (Prefix Lemma) If $G_n = [g_1, g_2, \ldots, g_n]$ is a minimal t-parse then any prefix t-parse G_m, $m < n$, is also minimal.

Proof. Assume G_n is nonminimal. It suffices to show that any extension of G_n is nonminimal. Without loss of generality, let H be a one-step ∂-minor of G_n such that for any $Z \in \Sigma_t^*$,

$$G_n \cdot Z \in \mathcal{F} \iff H \cdot Z \in \mathcal{F} \quad .$$

Let $g_{n+1} \in \Sigma_t$ and $G_{n+1} = G_n \cdot [g_{n+1}]$. Because of the semantics of the operator set Σ_t, $H' = H \cdot [g_{n+1}]$ is a one-step ∂-minor of G_{n+1} such that for any $Z \in \Sigma_t^*$,

$$G_{n+1} \cdot Z = G_n \cdot ([g_{n+1}] \cdot Z) \in \mathcal{F} \iff H \cdot ([g_{n+1}] \cdot Z) = H' \cdot Z \in \mathcal{F} \quad .$$

Thus, any extension of G_n is nonminimal. □

The above two lemmas also hold when the circle plus operator $\oplus$ is included in Σ_t (i.e., treewidth t-parses). For illustration consider the Prefix Lemma: If G is a nonminimal t-parse with an $\mathcal{F}$-congruent minor G', and Z is any t-parse, then $(G \oplus Z)'$ is an $\mathcal{F}$-congruent minor of a nonminimal $G \oplus Z$, where we use the prime symbol to denote the corresponding minor operation done to the G part of $G \oplus Z$. The awkward notation is needed since $G' \oplus Z$ may equal $G \oplus Z$ when common boundary edges exist in both G and Z.

4.2 A Simple Procedure for Finding Obstructions

The Prefix Lemma implies that every minimal t-parse is obtainable by extending some minimal t-parse; this provides a finite tree structure for the search space. (This guarantee of termination is proven at the end of this section.) Also since a $(t+1)$-boundaried graph may have many t-parse representations, we can further reduce the size of the search tree by enforcing a canonical structure on the t-parses enumerated. To do this we ensure that every prefix of every canonic t-parse is also canonic (recall Chapter 2). Combining these two ideas, we can search for minor-order obstructions by using the following simple procedure:

1. Start with the empty t-parse $\emptyset = [\textcircled{0}, \textcircled{1}, \ldots, \textcircled{t}]$ as the initial minimal t-parse. Since there are no one-step boundary minors this t-parse is vacuously minimal. Add the t-parse $\emptyset$ to a grow set $Grow$.

2. If there exists a t-parse G in the grow set $Grow$ then enumerate all one-operator extended t-parses $E_1, E_2, \ldots, E_{r \leq |\Sigma_t|}$ of G that are canonic (and relevant). Otherwise, stop.

3. For each t-parse E_i, $1 \leq i \leq r$, either prove it minimal or nonminimal.

4. Remove G from $Grow$ and add each minimal t-parse E_i, $1 \leq i \leq r$, to $Grow$. Return to step 2.

See the sample search tree given in Figure 4.1. We used the term *relevant* above to mean that we might only be interested in obstructions for a restricted subset

of t-parses. For example, three popular classes of relevant t-parses (in addition to just the free-boundary isomorphic graphs) are: connected graphs, bounded-degree graphs, and planar graphs. (Each class is intersected with the graphs of bounded combinatorial width t.)

This search procedure bypasses many graphs in a lower ideal $\mathcal{F}$ (usually of infinite cardinality) when branching out towards the finite number of obstructions (minimal leaves of the search tree). Recall that the set of minimal t-parses contains the set of boundaried obstructions of width t (i.e., the non-family minimal t-parses are precisely the boundaried obstructions).

Step 3 in the above procedure is the hardest to compute and we will dedicate most of the remainder part of this chapter to the issue of proving t-parses minimal or nonminimal. Also notice what happens when $r = 0$ at step 2. For this situation, we do not expand the search tree at a minimal t-parse; this is why there is a white circle leaf in our sample search tree of Figure 4.1. For example, for our canonic t-parse enumeration scheme, it is easy to see that the clique K_{t+1} is a canonic dead-end, for $t > 1$.

Now that we have presented our basic method for finding obstruction sets, we use the Prefix Lemma (as promised above) to prove that the procedure will terminate.

Theorem 73: For a lower ideal $\mathcal{F}$ with an available decision algorithm for computing $\sim_{\mathcal{F}}$ (i.e., for proving t-parse minimality), there exists a terminating algorithm (e.g., the one given above) to compute $\mathcal{O}(\mathcal{F})$ for each pathwidth t.

Proof. It suffices to prove that we will generate only a finite number m of minimal t-parses. The Prefix Lemma allows us to prune at each of at most $O(m)$ nonminimal t-parses. Our search tree has bounded out-degree consisting of the number of operators in Σ_t (i.e., a function of the width t). Recall that $\sim_{\mathcal{F}}$ has finite index. If there is not a finite number m of minimal t-parses, then there must be an infinite set S of distinct t-parses in some $\sim_{\mathcal{F}}$ equivalence class. But since the boundary minor order is a well-partial-order, there must exist two t-parses G_1 and G_2 in S such that $G_1 \leq_{\partial m} G_2$. This implies that the t-parse G_2 has G_1 as a congruent minor. This contradicts the fact that G_2 was minimal. Therefore, our search will stop shortly after all of the finite number of minimal (and canonic) t-parses have been enumerated. □

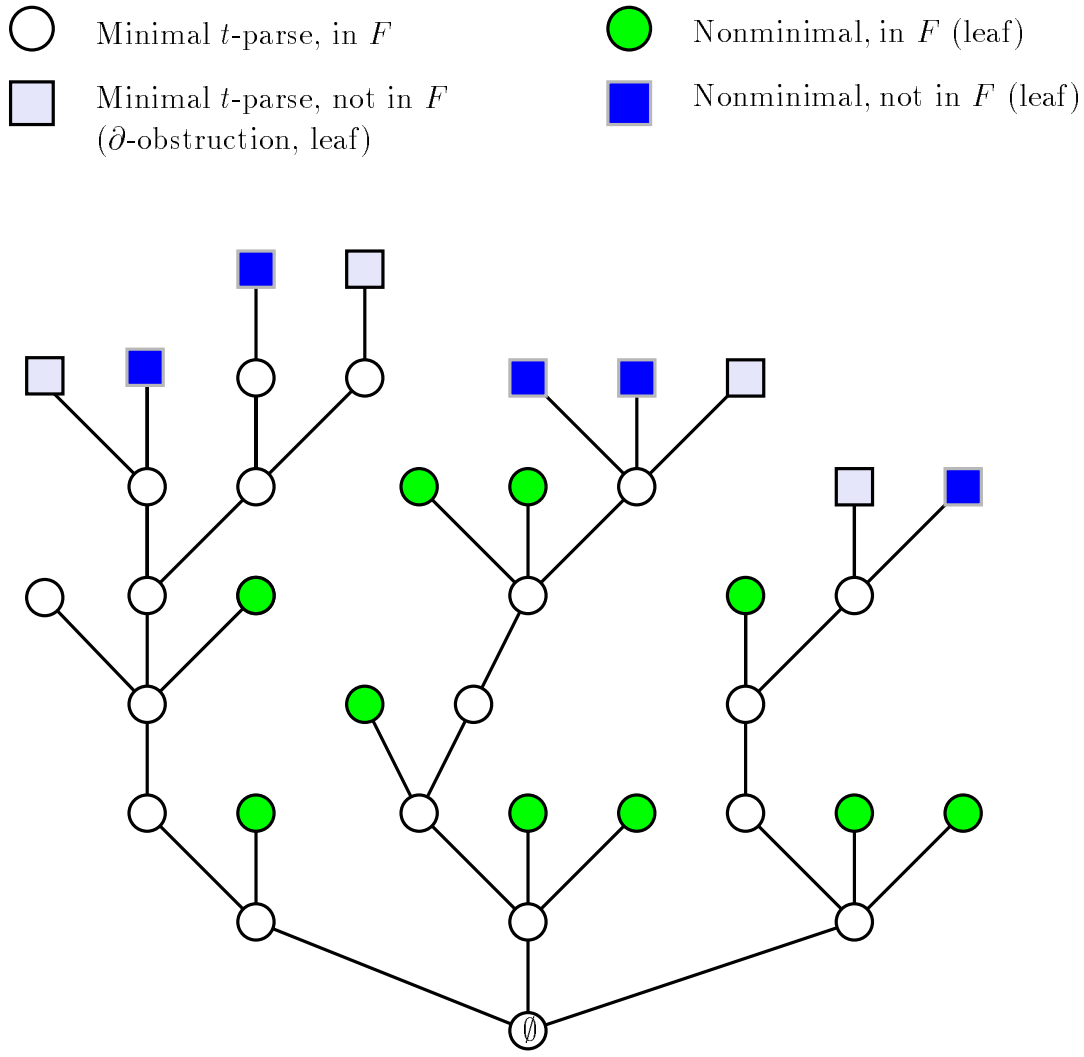


Figure 4.1: A typical t-parse search tree (each edge denotes one operator).

We point out that for treewidth t-parse searches we also have a easy proof of termination. However, in this case, the out-degree of the search tree changes for each minimal t-parse G. This is because concatenation $(\cdot)$ also includes the use of the $\oplus$ operator. Here $G \oplus G'$ is enumerated for every other treewidth canonic operand G', such that G' is less than G in the lex-rank order (see Definition 43 and Section 2.3.2).

4.3 Proving t-parses Minimal or Nonminimal

We currently use the following four techniques to prove that a t-parse in the search tree is minimal or nonminimal.

1. Direct nonminimality checks.
2. Dynamic programming congruences.
3. Random extension searches.
4. Canonical congruence testsets.

These steps are listed in the order that they are attempted (if available); if one succeeds, the remainder do not need to be performed. The first three of these may not succeed, though the fourth method always will.

Note that a boundaried obstruction G is the only type of minimal t-parse that has an empty extension $Z = [\,]$ that distinguishes each minor $G' \leq_{\partial m} G$ from G. In these trivial cases we do not need to use any of the above four proof methods.

By default, the third method of doing random extension searches is always available whenever we have a decision algorithm for a lower ideal $\mathcal{F}$. All of the above techniques are *optional* in the sense that we may only need (or want) to use just one method. We now list three common situations as examples. If we have the canonical congruence $\sim_{\mathcal{F}}$ for $\mathcal{F}$, implemented as a dynamic program, then the second method is sufficient (e.g., see our k–Vertex Cover characterizations). If we have an algorithm that detects *all* nonminimal graphs then the first method is sufficient (e.g., see

our k–FEEDBACK EDGE SET characterizations). Lastly, if we are only interested in computing a partial set of obstructions (with a good chance of finding the complete set), we can use only the third method.

A typical obstruction set search will use all four proof techniques, which are explained in more detail in the following subsections.

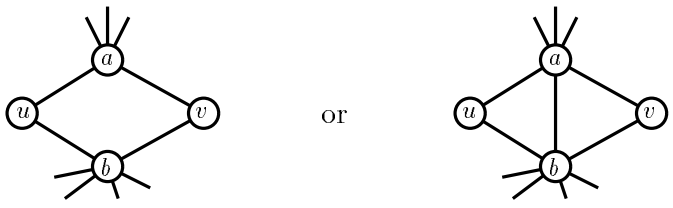
4.3.1 Direct proofs of nonminimality

There exist easily observable properties of t-parses, called *pretests*, that imply nonminimality. For any $(t+1)$-boundaried graph (t-parse) we have to assume that the boundary ∂ is completely connected (or has the potential to be). This is because we do not want to eliminate out any prefix that may be extended to an obstruction.

For an example of a pretest consider the k–FEEDBACK VERTEX SET family of Chapter 7. The existence of a degree one vertex (in the interior) is a trivial example of such a nonminimal property. In fact many pretests that we use are easy to prove. To illustrate this point further we have the following pretest.

Lemma 74: Any graph with respect to the k–FEEDBACK VERTEX SET family that contains a 4-cycle with two degree two vertices is nonminimal.

Proof. We know that if a graph G has adjacent degree two vertices then it is nonminimal except for the C_3 cases (in fact, this is another simple pretest). This is because the minor (G/e), created by contracting the edge e between these degree two vertices, still requires a minimum feedback vertex set of the same size. Alternatively stated, $FVS(G) = FVS(G/e)$. So we just need to consider a graph G with a 4-cycle that looks like:



For either case we claim that the edge contracted minor $G' = G/(a,u)$ is congruent to G (i.e., a witness to nonminimality). This is easily seen since any feedback vertex

set V' of G (or any extension of G) that contains vertex u or v can be replaced with $V'' = (V' \setminus \{u\}) \cup \{b\}$ or $V'' = (V' \setminus \{v\}) \cup \{b\}$. This set V'' is also a feedback vertex set for G'. Likewise, if W' is a feedback vertex set for G' then W' is also a feedback vertex set for G. □

If there exists an efficient algorithm for a pretest then we may opt to test it on each enumerated t-parse before invoking our canonicity algorithm. The reason is that if we get an easy nonminimal proof for a t-parse then we do not care whether it is canonic or not (i.e., it does not matter whether we have found a nonminimal dead-end in our search tree or a redundant representation). Our current system does the canonicity testing in two parts:

1. Check for the easy non-canonic properties, using the results of Chapter 2 (i.e, one can think of this step as a "non-canonic pretest").

2. Use a slower algorithm (e.g., a brute force back-tracking method) that tests for t-parse canonicity.

For each newly enumerated t-parse, we then try all the "fast" nonminimality pretests between these two canonic stages.

For our t-parse enumeration scheme, a related "pretest" type pruning is presented below in Section 4.4.1. Here, for certain lower ideals, it is possible to prune the search tree at the disconnected t-parses, regardless of their minimal/nonminimal status.

4.3.2 Proofs based on a dynamic programming algorithm

We can sometimes use a dynamic programming algorithm $\mathcal{A}$ as a *finite refinement* $\sim$ of the canonical congruence $\sim_{\mathcal{F}}$ (for a fixed width). We can view each of these algorithms as a finite state (linear/tree) automaton for $\mathcal{F}$, by simulation. This means that there is a mapping from the dynamic programming states of $\mathcal{A}$ (equivalence classes of $\sim$) onto the equivalence classes of $\sim_{\mathcal{F}}$. The input to a bounded width t *finite-state algorithm* $\mathcal{A}^t$ is any t-parse over the alphabet Σ_t. These algorithms always run in linear time, where the size of the input is the number of t-parse operators, but

may contain large hidden constants that are a function of both t (width) and k, where k is a possible parameterized integer (i.e., k is some index in a series of families $\{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_k, \ldots\}$).

For such a finite-state algorithm $\mathcal{A}^t$ for $\mathcal{F}$, if both a t-parse G and a one-step ∂-minor G' of G end up in the same computation state, then $G \sim_{\mathcal{F}} G'$. (This is because $G \sim G' \Rightarrow G \sim_{\mathcal{F}} G'$.) Then by definition, the t-parse G is nonminimal. However, if G and G' have distinct final states, no conclusion can be reached. If every one-step ∂-minor G' is not in the same final state as G then we usually proceed with the next step (given in Section 4.3.3), the exception being when the states of $\mathcal{A}^t$ correspond in a one-to-one fashion with the equivalence classes of the canonical congruence $\sim_{\mathcal{F}}$. In this latter case, we call $\mathcal{A}^t$ a *minimal finite-state algorithm* for $\mathcal{F}$ (for width t).

If we are fortunate to have a minimal finite-state algorithm then we can also prove minimality. That is, we can use the finite-state algorithm to provide both minimal and nonminimal proofs. This is because distinct final states imply the existence of an extension that distinguishes the two states (and their t-parse representatives). More formally, a finite-state algorithm $\mathcal{A}^t$ is minimal (with respect to a lower ideal $\mathcal{F}$) if for any two t-parses G and H we have $\sim = \sim_{\mathcal{F}}$, or specifically

$$(\text{for all } Z \in \Sigma_t^* \,, \quad G \cdot Z \in \mathcal{F} \iff H \cdot Z \in \mathcal{F}) \quad \Rightarrow \quad G \sim H$$

where $G \sim H$ means that G and H end up in the same state of $\mathcal{A}^t$. An example of such a minimal finite-state algorithm is presented in Chapter 6 for the k–Vertex Cover graph families.

Now consider the rare case that a finite-index congruence (implemented as some type of dynamic program) requires substantial computational effort to update a t-parse's state. (Here we are considering the time to append a single operator to the t-parse.) We have some possible remedies when using this type of dynamic program as a congruence for our obstruction set computations:

1. We can keep a state lookup table for each canonic t-parse. Thus during the enumeration of t-parses, we do one dynamic programming step with a single operator applied to a previous state (of a prefix t-parse). This previous state is obtained from the lookup table, instead of recomputing several states for the

length of a t-parse. One unfortunate consequence of this is that we may have to determine states for some nonminimal t-parses. These graphs occur from the set of one-step ∂-minors of a given t-parse. For these cases we update the states (for several operators) starting from a minimal t-parse that is a prefix.

2. From the dynamic-programming congruence $\mathcal{A}^t$ for a lower ideal $\mathcal{F}$, we can initially build a deterministic finite state automaton $\mathcal{M}$ that accepts t-parses in $\mathcal{F}$. This automaton provides fast (single-step) state transitions. Alternatively, we can use a canonical congruence $\sim_{\mathcal{F}}$ implemented as a testset to automatically create $\mathcal{M}$. (See below for more information and Sections 11.3–11.4 for how to build these automata.) The only flaw with either of these constructions is that the generated automaton may be too large to store in memory (even after being minimized).

We end this subsection with a short case study. We show how to construct dynamic-programming congruences for the k–Edge Bounded IndSet lower ideals. Recall that members of these families have independence + size $\leq k$. The reader is invited to skip to the next t-parse minimality/nonminimality proof method (Section 4.3.3) if these details are not of interest.

We first consider a dynamic program that determines the independence of a path-width t-parse. This algorithm $\mathcal{I}$-d.p. is presented in Figure 4.2 and proven correct below in Lemma 75. For the related k–Edge Bounded IndSet family, a boundaried graph congruence can be specified as follows. For a t-parse G and each subset S of the boundary ∂ define the following substates (whenever possible):

$$IndSet_G[S] = \max(|I| \leq k+1 \ : \ I \cap \partial = S \text{ and } I \text{ is an independent set}) \quad .$$

Two t-parses G and H are in the same equivalence class if

$$\min(|E(G)|, k+1) = \min(|E(H)|, k+1) \quad \text{and}$$

$$IndSet_G[S] = IndSet_H[S] \quad \text{for all } S \subseteq \partial \quad .$$

For our dynamic-programming implementation, we use a special symbol `void` to denote that there is no independent set containing some $S \subseteq \partial$. That is, the following

Initial Conditions:

For the prefix $G_{t+1} = [\textcircled{0}, \ldots, \textcircled{t}]$ of a t-parse $G_n = [g_1, g_2, \ldots, g_n]$ set

$$IndSet[S] = |S|, \quad \text{for each } S \subseteq \partial \quad .$$

Loop:

For each operator g_k, $k = t+2, \ldots, n$, do the following (over all $S \in 2^{\partial}$):

Case 1: For new (replacement) boundary vertex $g_k = \textcircled{i}$ and $i \in S$,

$$IndSet[S]' = \max(IndSet[S], IndSet[S \setminus \{i\}]) + 1 \quad .$$

Case 2: For new boundary vertex $g_k = \textcircled{i}$ and $i \notin S$,

$$IndSet[S]' = \max(IndSet[S], IndSet[S \cup \{i\}]) \quad .$$

Case 3: For new boundary edge $g_k = \boxed{i\ j}$ with $i \in S$ and $j \in S$,

$$IndSet[S]' = \texttt{void} \quad .$$

Case 4: For new boundary edge $g_k = \boxed{i\ j}$ with $i \notin S$ or $j \notin S$,

$$IndSet[S]' = IndSet[S] \quad .$$

On Exit:

Return maximum value of any $IndSet[S]$ over all $S \subseteq \partial$.

Figure 4.2: A general independence algorithm $\mathcal{I}$-d.p. for pathwidth t-parses.

simple alteration to the above substates is used:

$$IndSet_G[S] = \begin{cases} \texttt{void} & \text{if } S \text{ is not an independent set of } G, \text{ else} \\ \max(|I| \leq k+1 \ : \ I \cap \partial = S \text{ and } I \text{ is an independent set}) \end{cases} .$$

Next we present a proof that the (infinite-state) dynamic program $\mathcal{I}$-d.p. correctly updates these $IndSet[S]$ substates in the t-parse setting.

Lemma 75: Consider the independence algorithm $\mathcal{I}$-d.p. given in Figure 4.2. For any pathwidth t-parse G and extension operator $g \in \Sigma_t$, the $IndSet[S]$ substates of $\mathcal{I}$-d.p. are correctly updated for $G \cdot [g]$.

Proof. By definition of $IndSet[S]$ for $S \subseteq \partial$, the state $IndSet[S] = |S|$ for the empty t-parse $G = [\textcircled{0}, \textcircled{1}, \ldots, \textcircled{t}]$. The four dynamic-programming cases handle: (1) whether g is a vertex or edge operator, and (2) the specific index $S \subseteq \partial$. In the case analysis below recall that a witness I for $IndSet[S]$ is an independent set of the t-parse such that $I \cap \partial = S$. The notation $IndSet[S]'$ denotes the substate of $G \cdot [g]$, while $IndSet[S]$ denotes the previous state of G.

Case 1: $g = \textcircled{i}$ where $i \in S$

Let I and I_i be maximum independent sets for $IndSet[S]$ and $IndSet[S \setminus \{i\}]$ such that $I \cap \partial = S$ and $I_i \cap \partial = S \setminus \{i\}$. For a new isolated boundary vertex i, we see that $IndSet[S]' \geq \max(IndSet[S], IndSet[S \setminus \{i\}]) + 1$, because either one of the sets I or I_i, plus one additional vertex, is a witness. To show that equality holds we consider the following two cases when $IndSet[S]'$ is at least two greater than both $IndSet[S]$ and $IndSet[S \setminus \{i\}]$. If I' is a witness for $IndSet[S]'$ then consider $I'' = I' \setminus \{i'\}$, where i' denotes the new boundary vertex of $G \cdot [\textcircled{i}]$. First, if I'' contains the boundary vertex i of G then I'' is a witness for $IndSet[S]$ of larger cardinality. Second, if I'' does not contain the boundary vertex i of G then I'' is a witness for $IndSet[S \setminus \{i\}]$ of larger cardinality. Both these contradictions show that this dynamic step is correct.

Case 2: $g = \textcircled{i}$ where $i \notin S$

Now let I and I_i be independent set witnesses for both $IndSet[S]$ and $IndSet[S \cup \{i\}]$, respectively. Since $i \notin S$, both I and I' are also maximal independent sets with respect to $IndSet[S]'$. Thus $IndSet[S]' \geq \max(IndSet[S], IndSet[S \cup \{i\}])$. Again

(as in case 1), equality must hold otherwise we can contradict one of the previous maximum values $IndSet[S]$ or $IndSet[S \cup \{i\}]$.

Case 3: $g = \boxed{i\ j}$ with $i \in S$ and $j \in S$

There is no possible independent set when both vertices i and j are adjacent, so the assignment $IndSet[S]' = \texttt{void}$ is correct. In future steps, we define $\max(n, \texttt{void}) = n$ for any integer n, that is, `void` is less than any integer n in this linear order.

Case 4: $g = \boxed{i\ j}$ with $i \notin S$ or $j \notin S$

Since a witness independent set I for $IndSet[S]$ is also a witness for $IndSet[S]'$ and the independence of a graph does not increase when edges are added, no change is needed for this situation.

□

From the above lemma, it follows that we can determine in *linear time* the maximum independent set of a t-parse G by running the $\mathcal{I}$-d.p. algorithm. The independence of the graph is then the largest $IndSet[S]$ value. To actually find an independent set of maximum cardinality, a history of witness vertex sets (i.e., one independent set I for each $IndSet[S]$) is kept and updated during the dynamic steps.

For each fixed parameter k, it is easy to convert $\mathcal{I}$-d.p. into a finite-state algorithm $\mathcal{A}_k^t$ that can determine if a t-parse has an independent set of size greater than k. This is done by simply assigning $k+1$ to any $IndSet[S]$ that ever exceeds $k+1$. We do not know whether this finite-state algorithm $\mathcal{A}_k^t$ is the smallest in terms of the number of computation states required. This implied graph family "independence greater than k" is not a lower ideal in the minor order. However, we can adapt each finite-state algorithm $\mathcal{A}_k^t$ into a dynamic-programming congruence for each k–Edge Bounded IndSet lower ideal.

Lemma 76: Let $\mathcal{F} = k$–Edge Bounded IndSet. For each boundary size t, an upper bound on the number of equivalence classes for $\sim_{\mathcal{F}}$ is $(k+2)^{2^t+1}$.

Proof. Consider the k–Edge Bounded IndSet congruence given earlier. For each subset of the boundary S, $IndSet[S]$ is assigned either one of the integers $\{0, \ldots, k+1\}$ or the special symbol `void`. Since there are 2^t possible choices for S and each graph

in the lower ideal can have at most k edges (again an integer $\{0, \ldots, k+1\}$), the bound follows. □

We close this section by mentioning that the above bound is probably not very strong for k–EDGE BOUNDED INDSET. If we know that a boundaried graph has k' edges then we should be able to restrict to the substates of the finite-state independence algorithm for $\mathcal{A}^t_{k-k'}$.

4.3.3 Proofs obtained by a randomized search

We can prove that a t-parse G is minimal by finding, for each one-step ∂-minor G' of G, a distinguisher Z, that is, a parse extension Z such that $G' \cdot Z$ is a graph in $\mathcal{F}$ while $G \cdot Z$ is not a graph in $\mathcal{F}$. (Recall Lemma 71.)

In the experimental results described in Part II of this dissertation, a large fraction of the minimality proofs in the search trees were obtained by a randomized algorithm. This randomized algorithm consists of the following high-level steps for a t-parse G:

1 Initialize $L = \{G' \mid G' \text{ is a one-step } \partial\text{-minor of } G\}$.
2 Until $L = \emptyset$ or until a time limit is exceeded do:
  2.1 Choose $G' \in L$.
  2.2 Randomly generate a distinguisher Z for G and G'.
  2.3 Pick a minor $Z' \leq_{\partial m} Z$ such that Z' is a distinguisher for G and G', but for every proper minor $Z'' \leq_{\partial m} Z'$, Z'' does not distinguish G and G'.
  2.4 Remove G' and then any other $G'' \in L$ such that $G'' \cdot Z' \in \mathcal{F}$.

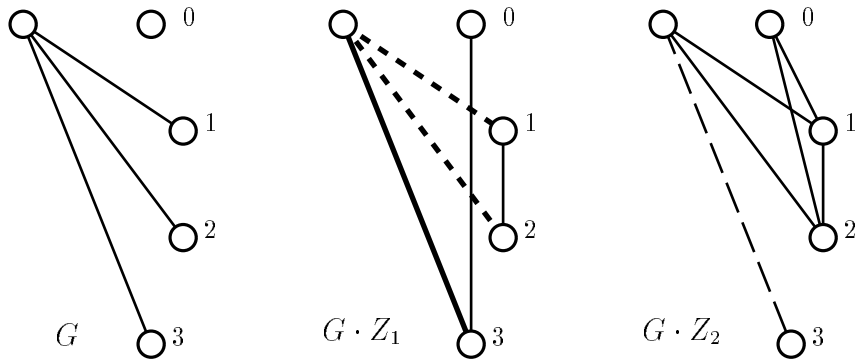
The goal of this algorithm is to find a distinguisher for each one-step ∂-minor. In particular, we can use a *different* distinguisher for *each* one-step ∂-minor in a proof of minimality. Step 2.1 chooses a one-step ∂-minor as a target. Step 2.2., which is the most computationally intensive step, may not succeed in the given time limit. If it is successful then steps 2.3 and 2.4 extract as much useful information as possible from this distinguisher. In practice we found that many one-step ∂-minors of G could be distinguished from G by the same extension Z. The minimization of Z in creating

Z' in step 2.3 increases the chances that the extension distinguishes G from other one-step ∂-minors.

Example 77: Consider the family $\mathcal{F}$ = 2–Vertex Cover. The figure below depicts a t-parse G, (for $t = 3$) along with two randomly generated extended graphs $G \cdot Z_1$ and $G \cdot Z_2$ that provides a proof of minimality for G.

The t-parse $G = [\textcircled{0}, \textcircled{1}, \textcircled{2}, \textcircled{3}, \boxed{0\ 1}, \boxed{0\ 2}, \boxed{0\ 3}, \textcircled{0}] \simeq K_{1,3} \cup K_1$.

The two proof extensions are $Z_1 = [\boxed{1\ 2}, \boxed{0\ 3}]$ and $Z_2 = [\boxed{0\ 2}, \boxed{1\ 2}, \boxed{0\ 1}]$.



The compact visual representation of the one-step minors consist of: a bold edge is an edge contracted minor, a dashed edge is an edge deleted minor, and a bold-dashed edge indicates two different minors, each one obtained by an edge deletion or an edge contraction. The extension figure in the middle shows five one-step minors of G, and an extension Z_1 that gives the desired minimality proof for all those five minors. The extension figure on the right finishes the t-parse minimality proof (using Z_2) for the remaining single edge deleted minor.

We end this subsection with two remarks concerning possible uses of random distinguishers.

1. In our experimental work, we found that a careful examination of those cases where a randomized search for a proof of minimality failed (and hence, for which this failure provides circumstantial evidence of nonminimality), often provided structural insights that could be translated into new nonminimality pretests.

2. Often good candidates for distinguishers can be obtained from partial sets of computed obstructions. Here a "piece" (boundaried subgraph) of an obstruction has minimality properties for the current lower ideal. Trivially the single edges (via a single edge operator extension) are pieces of obstructions. Thus, for any interesting lower ideal, we could start with these edge pieces even before we find our first obstruction.

4.3.4 Proofs based on a testset congruence

We can use a complete testset (see Definition 68) to determine if a t-parse G is distinguished from each of its one-step ∂-minors (recall Lemma 71). If so, the t-parse G is minimal by a testset proof. Alternatively, a t-parse G is nonminimal if and only if it has a one-step ∂-minor G' such that G and G' agree on every test. Thus after a testset run we have a proof that a graph is either minimal or nonminimal.

It helps to have a testset of minimum size and a fast membership algorithm for $\mathcal{F}$. Unfortunately, it is $\mathcal{NP}$-hard to find the minimum sized testset for the minimal automaton or canonical congruence for $\mathcal{F}$. (See Chapter 11 for a proof.) However, there exist heuristics that to help find redundant tests in a testset. For example, we can remove one test and see if the remaining tests define the same congruence. That is, we can compare the minimal automata generated by these two testsets. (See Section 11.3 for instructions.)

All of our testsets, that are presented in Part II, were proven complete for the set of all boundaried graphs. It is easy to see that if two graphs are congruent for the set of all $(t+1)$-boundaried graphs then they are also congruent for the set of t-parses.

We point out that testsets exist for other "regular language" graph families besides the bounded-width minor-order lower ideals. The set $\mathcal{F}_{HC}$ of t-boundaried graphs with a Hamiltonian cycle is a classic example. In Figure 4.3 we illustrate a testset for $\mathcal{F}_{HC}$ for graphs of boundary size 3. Recall that a graph G is *Hamiltonian* if there exist a cycle of length $|G|$.

The proof that these tests are sufficient is done by a case analysis regarding how a Hamiltonian cycle may pass across the boundary. We provide a simple proof below

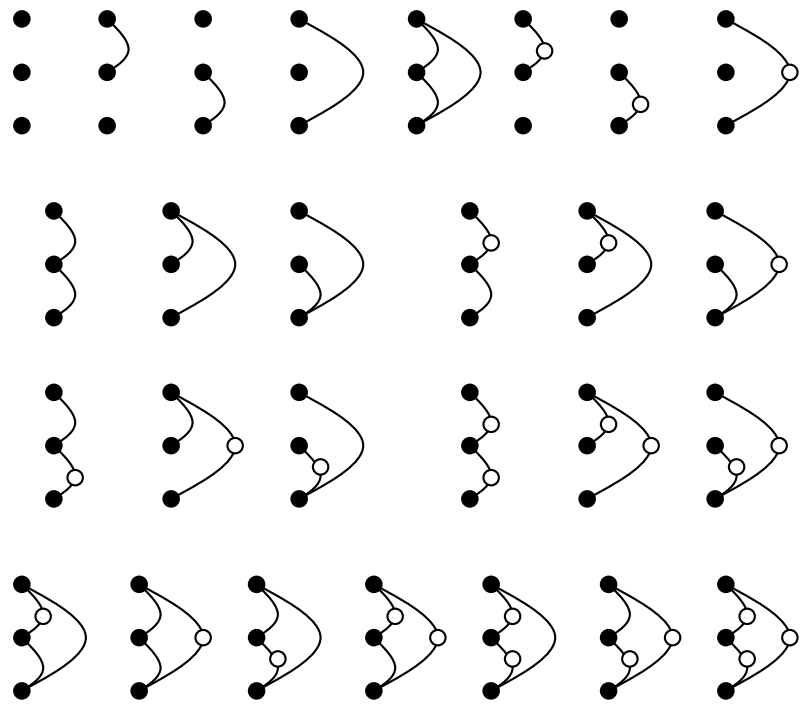


Figure 4.3: The set T^3_{HC} of 3-boundaried graph tests for Hamiltonicity.

in order to illustrate how one generally constructs testsets for an arbitrary family $\mathcal{F}$. In establishing most of our testsets for lower ideals we use this basic proof technique.

Lemma 78: The set T^3_{HC} is a Hamiltonian cycle testset for the set of 3-boundaried graphs.

Proof. Let $\sim_{HC}$ denote the canonical Hamiltonian-cycle family congruence on graphs of boundary size 3. Recall that $G \sim_{HC} H$ if and only if for every 3-boundaried graph extension Z,

$$G \oplus Z \in \mathcal{F}_{HC} \iff H \oplus Z \in \mathcal{F}_{HC} \quad .$$

If $G \not\sim_{HC} H$ then, without loss of generality, we have some Z such that $G \oplus Z \in \mathcal{F}_{HC}$ and $H \oplus Z \notin \mathcal{F}_{HC}$. We just need to show that this distinguishing Z can be replaced with one of the 27 tests in T^3_{HC} (defined in Figure 4.3). Let C be any Hamiltonian cycle of $G \oplus Z$. Clearly the 3-boundaried graph $Z' = Z \cap C$ also distinguishes G and H. This is because non-Hamiltonicity is closed under edge deletions; note that $V(Z') = V(Z)$. Now Z' must consist of a set of connecting paths (or unique cycle) between the boundary vertices. Again, since non-Hamiltonicity is closed under edge

contractions between adjacent degree two vertices, Z' can be replaced with one of the reduced tests in T^3_{HC}. □

4.4 Making the Theory Practical

This section contains a selection of results that help speed up the process of computing obstruction sets.

One computational improvement deals with determining whether a t-parse G is minimal or nonminimal when G is not in a lower ideal $\mathcal{F}$. By definition of the canonical congruence $\sim_{\mathcal{F}}$ for $\mathcal{F}$, we see that G is minimal if and only if for every ∂-minor H of G, the minor H is in $\mathcal{F}$. Otherwise, if H is not in $\mathcal{F}$ then $H \sim_{\mathcal{F}} G$. Thus, minimality is easily determined by checking that each one-step ∂-minor of an out-of-family t-parse is in the lower ideal $\mathcal{F}$.

4.4.1 Pruning at disconnected t-parses

Often for a lower ideal $\mathcal{F}$ we are only interested in computing the connected obstructions since (as explained in this section) the disconnected ones are easy to derive. Below we indicate a general setting where we can prune the search tree whenever a disconnected boundaried graph is enumerated. Here "disconnected" means that there is a connected component containing only non-boundary (interior) vertices.

Let λ be a function that maps graphs to non-negative integers such that:

1. For graphs G_1 and G_2, $\lambda(G_1 \cup G_2) = \lambda(G_1) + \lambda(G_2)$,
2. For any minor H of G, $\lambda(H) \leq \lambda(G)$, and
3. For any graph G there exists a minor H such that $\lambda(H) \geq \lambda(G) - 1$.

The family of graphs $\mathcal{F}[k] = \{G \mid \lambda(G) \leq k\}, k \geq 0$ has an obstruction set since it is a lower ideal (from property 2 above) in the minor order. Also $\mathcal{O}(\mathcal{F}[k])$ has finite cardinality because of the Graph Minor Theorem. A concrete example is $\lambda(G) = genus(G)$, where $genus(G)$ denotes the smallest genus of all orientable

surfaces on which G can be embedded (property 1 follows from [BHKY62]). In fact, there are many examples of lower ideals that satisfy all three properties such as all of the "within k vertices" families characterized in this dissertation.

Observation 79: If $\mathcal{F}$ is a lower ideal such that

$$G_1 \in \mathcal{F} \text{ and } G_2 \in \mathcal{F} \iff (G_1 \cup G_2) \in \mathcal{F}$$

then the function

$$\lambda(G) = \min(|V'| \ : \ G \setminus V' \in \mathcal{F} \text{ where } V' \subseteq V)$$

can be used to define the above parameterized lower ideals $\mathcal{F}[k]$, $k \geq 0$.

Proof. Clearly, $\lambda(G_1 \cup G_2) \leq \lambda(G_1) + \lambda(G_2)$ since if $G_1 \setminus V_1 \in \mathcal{F}$ and $G_2 \setminus V_2 \in \mathcal{F}$ then $(G_1 \cup G_2) \setminus (V_1 \cup V_2) \in \mathcal{F}$. Likewise, $\lambda(G_1) + \lambda(G_2) \leq \lambda(G_1 \cup G_2)$ since if $(G_1 \cup G_2) \setminus V_{1,2} \in \mathcal{F}$ then $G_1 \setminus (V(G_1) \cap V_{1,2}) \in \mathcal{F}$ and $G_2 \setminus (V(G_2) \cap V_{1,2}) \in \mathcal{F}$. So property 1 holds for λ. Property 2 follows from the fact that $\mathcal{F}[k]$ is defined as "a within k vertices" family of a lower ideal (see [FL88b]). Property 3 follows from the fact that if H is any minor obtained from G by deleting a single vertex then $\lambda(H)$ can be at most one less than $\lambda(G)$. □

For example, our k–Vertex Cover and k–Feedback Vertex Set parameterized lower ideals are defined from the base lower ideals $\mathcal{F}_{VC} = \{\text{graphs with no edges}\}$ and $\mathcal{F}_{FVS} = \{\text{graphs with no cycles}\}$, respectively. Here the families $\mathcal{F}_{VC}$ and $\mathcal{F}_{FVS}$ are closed under graph unions, so Observation 79 is valid.

Theorem 80: If $G = C_0 \cup C_1$ is an obstruction for $\mathcal{F}[k]$, $k \geq 0$, then each C_i is an obstruction for $\mathcal{F}[\lambda(C_i) - 1], i = 0, 1$.

Proof. First note that $\lambda(C_i) \neq 0$ for otherwise removing that component from G is a contradiction to the fact that G is an obstruction. Thus, we claim that each C_i is an obstruction to a smaller family (i.e., some $\mathcal{F}[k']$ where $k' < k$).

Assume that C_i is not an obstruction for $\mathcal{F}[\lambda(C_i)-1]$, the smallest family not containing C_i. Not being an obstruction implies that there is a minor C' of C_i such that $\lambda(C') = \lambda(C_i)$. But this implies that for the minor $G' = C' \cup C_{1-i}$ of G,

$$\begin{aligned} \lambda(G') &= \lambda(C' \cup C_{1-i}) = \lambda(C') + \lambda(C_{1-i}) \\ &= \lambda(C_1) + \lambda(C_2) = \lambda(C_1 \cup C_2) \\ &= \lambda(G) = k+1 \quad . \end{aligned}$$

The existence of this minor contradicts the assumption that G is an obstruction for $\mathcal{F}[k]$. So the connected graph C_i must also be an obstruction. □

Corollary 81: If $G = \bigcup_{i=0}^{r} C_i$ is an obstruction for $\mathcal{F}[k]$, $k \geq 0$, then each C_i is an obstruction for $\mathcal{F}[\lambda(C_i)-1], i = 0, 1, \ldots, r$.

Proof. This follows by recursively applying the above theorem. □

Many obstruction set characterizations are based on k-parameterized lower ideals. It would be surprising if the growth rate in the number of obstructions per each k is slow. To conclude this subsection we present two observations to substantiate this claim.

Observation 82: If the number of connected obstructions f_k^c of $\mathcal{F}[k]$ is at least one, for all $k \geq 0$, then the total number of obstructions f_k of $\mathcal{F}[k]$ is greater than or equal to the number of integer partitions of $k+1$.

Proof. Because of property (3) of $\mathcal{F}[k]$, we know that any obstruction $O \in \mathcal{O}(\mathcal{F}[k])$ has $\lambda(O) = k+1$. By Corollary 81 there exist disconnected obstructions $O_1 \cup O_2 \cup \cdots \cup O_m$, where each O_i is connected and $\sum_{i=1}^{m} \lambda(O_i) = k+1$. There is a one-to-one correspondence with these (non-isomorphic) obstructions and the number of integer partitions of $k+1$. □

Example 83: We can easily generate the number of integer partitions for various n. Hence, the following table gives lower bounds on the total number of obstructions for $\mathcal{F}[k]$.

n =	1	2	3	4	5	6	7	8	9	10	11	12	13	...	20	30
counts =	1	2	3	5	7	11	15	22	30	42	56	77	101		627	5604

n =	40	50	60	70	80	90	100
counts =	37338	204226	966467	4087968	15796476	56634173	190569292

To lead up to our next observation, let $\mathcal{F}_k = \{G \mid \gamma(G) \leq k\}$ be any k-parameterized lower ideal that is defined by some integer function $\gamma(G) \leq p(|G|)$, that is bounded by some polynomial function p. Also let $p_\gamma(G,k)$ denote the corresponding graph problem that determines if $\gamma(G) \leq k$ where both G and k are part of the input.

Observation 84: If $p_\gamma(G,k)$ is $\mathcal{NP}$-complete then $f_k = |\mathcal{O}(\mathcal{F}_k)|$ must be super-polynomial else the polynomial time hierarchy ($\mathcal{PH}$) collapses to Σ_3^P.

Proof. Yap (see [Yap83]) has shown that the following are equivalent for $i > 0$:
(a) $\Pi_i^P \subseteq \Sigma_i^P/\text{poly}$ or $\Sigma_i^P \subseteq \Pi_i^P/\text{poly}$
(b) $\Sigma_i^P/\text{poly} = \Pi_i^P/\text{poly}$
(c) $\mathcal{PH}/\text{poly} = \Sigma_i^P/\text{poly}$ or $\mathcal{PH}/\text{poly} = \Pi_i^P/\text{poly}$

Now if f_k is bounded by a polynomial then co-$p_\gamma(G,k) \in \mathcal{NP}/\text{poly}$ by having polynomial advice consisting of the obstruction sets for all $1 \leq k \leq p(|G|)$. That is, we can nondeterministically verify in polynomial time (for input G and k) that some obstruction $O \in \mathcal{O}(\mathcal{F}_k)$ is a minor of G. Since $p_\gamma(G,k)$ is $\mathcal{NP}$-complete, we get

$$\text{co-}\mathcal{NP} = \Pi_1^P \subseteq \mathcal{NP}/\text{poly} = \Sigma_1^P/\text{poly} \quad .$$

An application of Yap's result, given above, shows that

$$\Sigma_1^P/\text{poly} = \Pi_1^P/\text{poly} = \mathcal{PH}/\text{poly} \quad .$$

And this implies $\Sigma_3^P = \Pi_3^P$ by another theorem of Yap that states that for $i > 0$,

$$\Sigma_i^P/\text{poly} = \Pi_i^P/\text{poly} \Rightarrow \Sigma_{i+2}^P = \Pi_{i+2}^P \quad .$$

So using the well-known fact that if $\Sigma_i^P = \Pi_i^P$ for any $i > 0$ then the polynomial time hierarchy collapses to level i, we get $\mathcal{PH} = \Sigma_3^P = \Pi_3^P$ unless there are a super-polynomial number of obstructions for $\mathcal{F}_k$. □

This observation can be applied to most k-parameterized lower ideals. For example, our k–VERTEX COVER and k–FEEDBACK VERTEX SET graph families satisfy the preconditions of the above result. Also since determining the genus of a graph is $\mathcal{NP}$-complete, the lower ideal k-GENUS $= \{G \mid genus(G) \leq k\}$ also belongs in the above class.

4.4.2 Searching via universal distinguishers

One approach for speeding up an obstruction set search is to try find a smaller search tree. In fact, we can do this by defining a variation of our previously discussed "minimal t-parse" search tree.

Definition 85: An extension Z is a *universal distinguisher* for a t-parse (boundaried graph) G if for every one-step ∂-minor G' of G, we have

$$G \cdot Z \notin \mathcal{F} \quad \text{and} \quad G' \cdot Z \in \mathcal{F} \quad .$$

A t-parse G is *universal minimal* if there exists a universal distinguisher for it.

We have found several initial lists of obstructions using this universal distinguisher approach (e.g., see Chapter 9). The search tree restricted to universal minimal graphs is smaller because of the following observation.

Observation 86: With respect to a canonical congruence $\sim_{\mathcal{F}}$, the set of universal minimal t-parses is a subset of the set of minimal t-parses.

The next lemma guarantees that this search approach is a viable one.

Lemma 87: The Prefix Lemma (Lemma 72) holds for universal distinguisher searching. That is, any prefix of a universal minimal t-parse is universal minimal.

Proof. First observe that every boundaried obstruction has a canonic t-parse representation. If $G \cdot Z$ is a boundaried obstruction then G is universal distinguisher minimal via Z. □

The above prefix property means that our underlying obstruction set search software can be used here with only a few alterations.

It seems to be harder (in the computational sense) to determine when a t-parse G is universal minimal (or universal nonminimal). The good news is that we can use a slightly altered version of our random distinguisher search to show that a t-parse is universal minimal. Here for a t-parse G, we simply try to find one extension Z such that $G \cdot Z$ is not in $\mathcal{F}$ while for each minor $G' \leq_{\partial m} G$, $G' \oplus Z$ is in $\mathcal{F}$. As done for the randomized algorithm given in Section 4.3.3, we should minimize Z with respect to G and $\mathcal{F}$ before checking if it is an universal distinguisher.

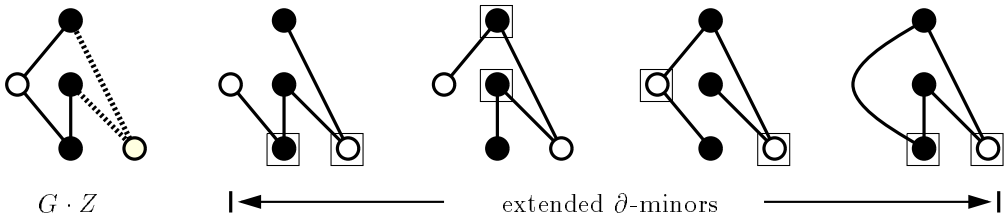
The proof of Lemma 87 tells us that if we run enough random distinguisher searches (see Section 4.3.3) then we will eventually find all the boundaried obstructions for $\mathcal{F}$. By using only this type of proof technique, there is a simple approximation search scheme that has the potential of finding all the obstructions.

Next we give a concrete example of a universal minimal t-parse.

Example 88: Let us reconsider the 2–VERTEX COVER family of graphs that is restricted to t-parses, for $t = 3$. The short t-parse

$$G = [\,(0), (1), (2), \boxed{0\ 1}, \boxed{1\ 2}, (1), \boxed{1\ 2}\,]$$

is universal minimal by the distinguisher $Z = [\,(2), \boxed{0\ 2}, \boxed{1\ 2}\,]$. If H is any one of G's four non-isomorphic one-step ∂-minors, we can find a vertex cover of size two for $H \cdot Z$ (see the squared vertices below). As can be seen, the 3-boundaried graph G is a prefix of the obstruction C_5.



We further explore the idea of universal distinguisher searching in the next chapter under the more general framework of using second-order congruences.

4.4.3 Using other finite-index congruences

One may wonder if we really need to use the canonical (family) congruence $\sim_{\mathcal{F}}$ or can we use another "easy to program" congruence for a lower ideal $\mathcal{F}$. The main result of this section answers this question in the affirmative. First let us formally define what is meant by a finite-state algorithm for any lower ideal $\mathcal{F}$.

Definition 89: A *finite-state algorithm $\mathcal{A}_{\mathcal{F}}$ (congruence $\sim$)* for any lower ideal of graphs $\mathcal{F} \subseteq \Sigma_t^*$ (i.e., a subset of the t-parses) satisfies these four properties:

1. The algorithm $\mathcal{A}_{\mathcal{F}}$ is a mapping of t-parses to a finite set of states, which correspond to the integers $\{0, 1, 2, \ldots, r\}$, for some $r \geq 0$.
2. If G and H are two t-parses that are not $\mathcal{F}$-congruent, $G \not\sim_{\mathcal{F}} H$, then $\mathcal{A}_{\mathcal{F}}(G) \neq \mathcal{A}_{\mathcal{F}}(H)$.
3. If G and H are two t-parses that are not members of $\mathcal{F}$ then $\mathcal{A}_{\mathcal{F}}(G) = \mathcal{A}_{\mathcal{F}}(H)$, that is, there exists an unique out-of-family state.
4. If G and H are two t-parses such that $\mathcal{A}_{\mathcal{F}}(G) = \mathcal{A}_{\mathcal{F}}(H)$ then for any $z \in \Sigma_t$, $\mathcal{A}_{\mathcal{F}}(G \cdot [z]) = \mathcal{A}_{\mathcal{F}}(H \cdot [z])$.

The states of any finite-state algorithm $\mathcal{A}_{\mathcal{F}}$ represent a refinement of the canonical (family) congruence $\sim_{\mathcal{F}}$. Often these algorithms are written as dynamic programs (recall Section 4.3.2). Also in the above definition, the symbol $\sim$ denotes the implicit congruence (refinement of $\sim_{\mathcal{F}}$) obtained from the states of $\mathcal{A}_{\mathcal{F}}$.

We can define t-parse minimality in terms of $\sim$. For instance, a t-parse G is *minimal* if for every proper minor $H \leq_{\partial m} G$, $H \not\sim G$ (or $\mathcal{A}_{\mathcal{F}}(G)$ and $\mathcal{A}_{\mathcal{F}}(H)$ are different states).

Lemma 90: If a t-parse obstruction set search with $\sim$ (substituted for $\sim_{\mathcal{F}}$) terminates then the obstruction set for $\mathcal{F}$ (of width t) has been found.

Proof. Because of property 2 above, any obstruction O of $\mathcal{O}(\mathcal{F})$ is minimal. Also since $\sim$ has only one out-of-family state (property 3), all minimal out-of-family

t-parses are boundaried obstructions. The Prefix Lemma also holds for this definition of t-parse minimality since (a) extensions are supergraphs and (b) the family $\mathcal{F}$ is assumed to be a lower ideal. Therefore, every prefix of O is minimal and will not be pruned. □

Notice that for the above lemma, termination of the search process does not rely on the fact that the algorithm $\mathcal{A}_{\mathcal{F}}$ has a finite number of states. That is, it does not use property 1. So it turns out that an infinite state algorithm (or at least one not proven to be finite state) satisfies the premise (and result) of the lemma.

For these non-canonical congruences, one handy result is missing. This is the ability to prove minimality via one-step minors (see Lemma 71). It is possible, in fact, to do a tree-like search for any generic congruence and redefine minimality. That is, in this context, a t-parse G is said to be minimal if and only if each one-step ∂-minor is distinguished from G. However, doing so may make it easier for the algorithm not to terminate, or at the very least, generate a larger than needed search tree. An experimental compromise is to try alternative minimality definitions for various minor steps, up to some $k > 1$.

4.4.4 Finding uses of randomization

The methods of the previous two sections have something in common in that we may want to end a search before everything is proven. Let us call an *approximation run* for a lower ideal $\mathcal{F}$ any computation that yields a search tree for $\mathcal{O}(\mathcal{F})$ that still has t-parses that are not proven minimal or nonminimal. That is, we may terminate the program early where "unknown" t-parses remain. We reuse proofs obtained on a previous approximation run when we restart an obstruction set search for the same lower ideal and width. The topic of this section concerns how randomization can be used to improve the chances that repeated runs progress towards finishing (i.e., we want to reach a point where all t-parses have been proven either minimal or nonminimal). Here we want subsequent approximation runs to do different work (via nondeterminism). Doing this provides a better chance of finding new proofs.

Depending on how hard it is to for the computer to obtain a proof, we store

the status of a t-parse (e.g., minimal, nonminimal, noncanonic, or irrelevant) in a centralized database to be accessed in future searches. For any t-parse that has an "unknown" status (e.g., after a time-out canonic attempt), the search must assume that the t-parse is on a path to an obstruction (e.g., in this case, the t-parse is assumed to be canonic).

We have already seen randomization used in the random extension searches of Section 4.3.3. There are other subtle places that randomness may help. One example is for the algorithm that determines whether a t-parse is canonic or not. It is evident that we do not want to do duplicate work during repeated searches. For example, repeated CPU time should be avoided if a t-parse canonic run is inconclusive during the first time. There are many open problems regarding how we can use randomization to help make our obstruction search theory more practical.

Chapter 5

The Implementation and The Future

In this chapter we describe our software for computing obstruction sets and discuss some research that is currently being explored. Our automated approach is continually improving by this new theory.

5.1 Using Distributed Programming

We believe that, in some sense, the most practical aspect of our obstruction set search theory is that it permits a distributive programming approach. In the following paragraphs we explain our efforts in exploiting this theme. Figure 5.1 at the end of this section gives a visual perspective of the many types of processes that are involved. These key processes are discussed below.

Manager: This process is the main search-tree engine of the system. The manager controls the database of status and proof information regarding t-parses. We run the manager on a computer that contains a dedicated local disk for the database (e.g., we have access to a Sparc-2 research machine at the University of Victoria for this purpose). It is also preferable to have a large swap space because a skeleton of the search tree is kept in memory for the duration of

the search. This tree includes record pointers to the database for all t-parses enumerated. The manager tells another process, called the dispatcher, what work needs to be done and waits for results. The manager grows the search tree in a greedy fashion as work becomes available. This strategy is dependent on the sequence order of proofs returned by the dispatcher. These are proofs of t-parse minimality or nonminimality. Also, the manager is the only process that writes to the database. Another process called the **locker** issues read/write access tokens to processes wanting access to the database.

Dispatcher: The dispatcher is the process that distributes t-parse work to the worker clients. It maintains a queue of work to be done along with a queue of idle worker processes. The work (consisting of enumerated t-parses with each t-parse initially having unknown status) is sent to a worker using the standard TCP/IP socket communication protocols. The worker processes can be anywhere on the internet, and not necessarily of the same computer architecture (e.g., we currently support Sparcs, Cray Y-MPs, and IBM-6000s). For communicating processes running on a machine with the same operating system as the dispatcher (usually SunOS), asynchronous I/O is used, otherwise synchronous I/O is used.

Workers: The workers consume most of the CPU resources during an obstruction set search. The main task for these processes is to determine whether a t-parse is minimal or not using the methods of Section 4.3. Some workers may also check whether a t-parse is canonic or not. Since we have workers compiled and installed on multiple hardware platforms, we use a special binary I/O stream object (a C++ class) to facilitate communication between the dispatcher and a worker. The number of workers possible for an obstruction set search is practically unlimited; however, our largest searches use only about 40 machines (mostly Sparcs).

Extractors: These processes are the user's interface to the distributive system. The extractors (and **browsers** for interactive viewing) are tools that retrieve information such as:

- The status of an individual t-parse (e.g., a partial set of one-step minors that have been distinguished).
- A view of the current search tree.
- A dump of the boundaried (or ordinary) minor-order obstructions found. The output is produced in either text, PostScript, or binary form.
- The current statistics about how many minimal and nonminimal nodes per depth of the search tree. This is used quite frequently in order to predict how close a current obstruction set search is to completing.
- An archive or backup of the current search, which is later used in a restart. For example, this is used before a scheduled power outage.

We have another type of extractor (or more accurately called a controller) that allows us to peek at the status of all system worker processes. This "backdoor" into the dispatcher allows us to monitor the search at the internet level. We can manually kill off frozen connections through this backdoor.

5.2 Software Summary

During our quest for a general-purpose system for computing obstruction sets, we have redeveloped several key software components. In fact, the current system was restarted from scratch four years ago. The first version was dropped after one year of effort when we discovered that the automaton constructed by the methods given in Section 11.3 were too big to be practical.

Some of our initial goals for a software project is reflected in our system's acronym: *VLSI Automated Compilation System* (VACS). The VACS research group at the University of Victoria (with Kevin Cattell and Mike Fellows as the other two members) envisioned a system that could generate approximating automaton for real-world engineering problems. A method of using obstructions for doing this task is mentioned later in Section 5.3.2.

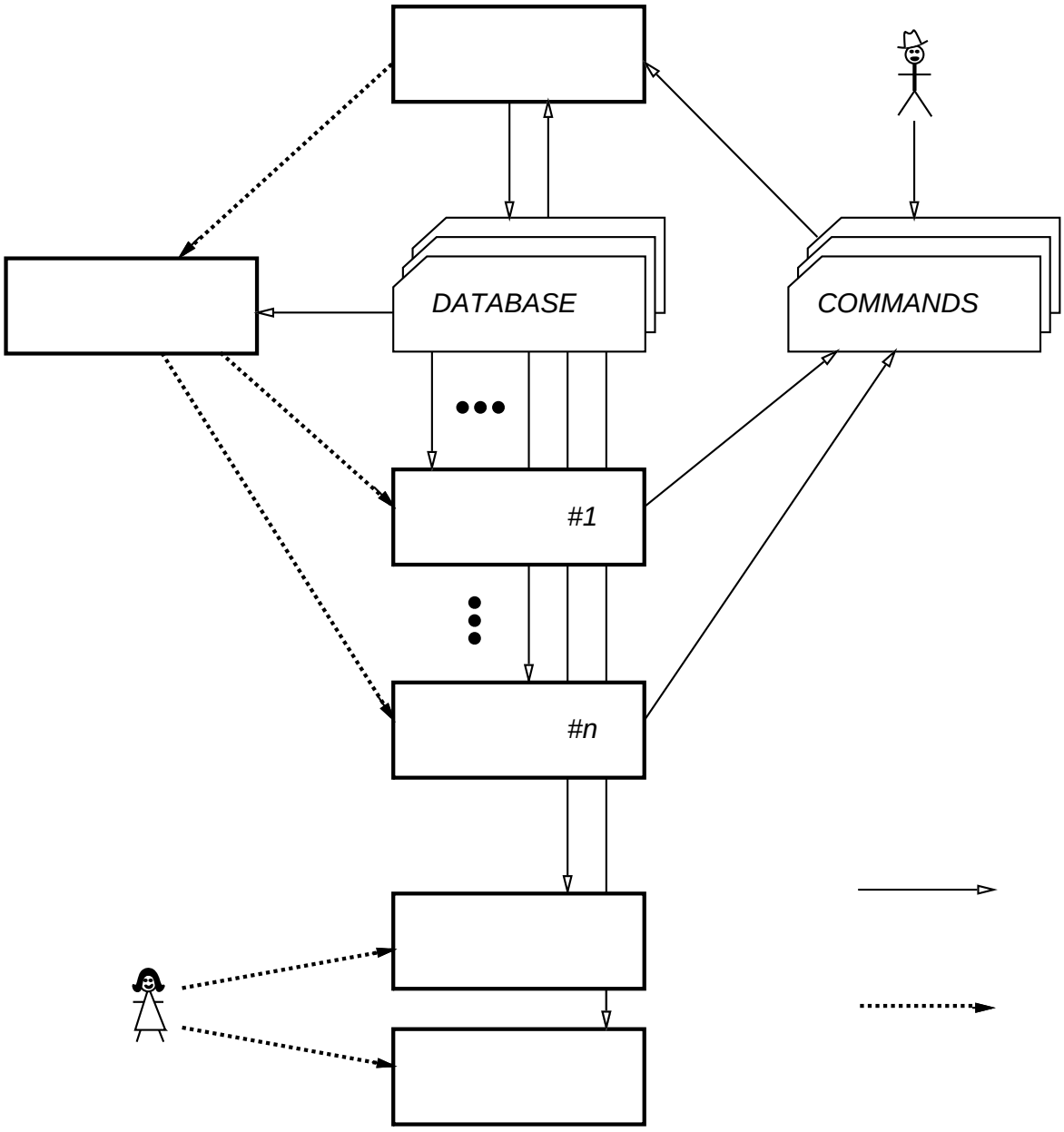


Figure 5.1: Schematic view of our distributed obstruction set software.

We now illustrate the user's steps required to find minor-order obstructions using our software. We assume that the user is interested in a particular lower ideal $\mathcal{F}$. Refer back to Section 5.1 for more information on the various processes mentioned below.

1. Create a problem file directory for the lower ideal $\mathcal{F}$ and the desired t-parse search width.

2. Add source code for the lower ideal $\mathcal{F}$ that consists of a family membership algorithm and optionally any (or all) of the following: nonminimal pretests, a finite-index congruence, or a testset. There are general C++ mechanisms in place for adding each of these components. If none of these proof techniques is provided, the system will only use random distinguisher searches to prove t-parses minimal (and may only compute a partial set of obstructions).

3. Make a control file that contains key-word descriptions of the problem. This should consists of at least the required run-time parameters that are listed in Figure 5.2.

4. Start a database *locker* daemon on the machine containing the problem directory. This needs to be started only once and resides for repeated runs.

5. Invoke the "search" script. This `csh` script first starts the *manager* process which in turn spawns a *dispatcher* process. The dispatcher will create an internet socket and store the current port number (chosen by the operating system) in an external file.

6. Start as many *worker* processes as needed. If the machine does not have access to the problem directory then the internet port number has to be manually copied to a "clone" directory (without the database) on the remote machine.

7. Monitor the progress of the search with various *extractors* and *browsers*.

8. When the search completes, run extractors to retrieve the obstructions (for the given search width) and statistics of the current search. The search is gracefully terminated by the manager telling the dispatcher, which in turn notifies all of the workers to abort.

```
const ProbInfo::ParseTableEntry ProbInfo::parseTable[ ] =
{
  { "problem name",          Required, String,  0  },
  { "problem number",        Required, Integer, 0  },
  { "directory",             Required, Subpath, 0  },
  { "keyword",               Required, String,  0  },
  { "pathwidth",             Required, Integer, 0  },
  { "membership flag",       Optional, Integer, "0"  },
  { "use congruence",        Optional, Boolean, "true"  },
  { "congruence flag",       Optional, Integer, "0"  },
  { "connected",             Optional, Boolean, "true"  },
  { "tree search",           Optional, Boolean, "false"  },
  { "conservative grow",     Optional, Boolean, "true"  },
  { "extn edge weight",      Optional, Integer, "80"  },
  { "extn runs",             Optional, Integer, "1"  },
  { "extn tries",            Optional, Integer, "10"  },
  { "extn max length",       Optional, Integer, "50"  },
  { "extn skip",             Optional, Integer, "0"  },
  { "save nonmin proofs",    Optional, Boolean, "true"  },
  { "congruence is tight",   Optional, Boolean, "false"  },
  { "use testset",           Optional, Boolean, "true"  },
  { "tilde minor search",    Optional, Boolean, "false"  },
  { "max equal iso level",   Optional, Integer, "-1"  },
  { "workers run canonic",   Optional, Boolean, "false"  },
  { "load all nonminimal",   Optional, Boolean, "false"  },
  { "limit to prefix",       Optional, String,  " "  },
  { "send PMP",              Optional, Boolean, "true"  },
  { "univ dist search",      Optional, Boolean, "false"  },
  { "pretest extensions",    Optional, Boolean, "false"  },
  { "canonic timeout",       Optional, Integer, "0"  }
};
```

Figure 5.2: Some available run-time options for obstruction set computations.

We now briefly explain the optional run-time parameters, given in Figure 5.2, for our obstruction set searches. The "membership flag" option is used to specify a run-time option to the membership algorithm, mainly used for testing purposes (or to provide additional information about the current lower ideal). The "use congruence" option tells the workers that we have implemented a refinement of the canonical congruence and to use it. The "congruence flag" option is available for different workers to run the dynamic-programming congruence in different modes. Using this option and the membership flag, the user can quickly change aspects of the search without having to recompile the program. The "connected" option notifies the manager to prune the search tree at the disconnected t-parses (see Section 4.4.1). The "tree search" option allows one to restrict the domain of the search to trees (i.e., treewidth one graphs of bounded pathwidth). Using "conservative grow" as an option tells the manager to finish the search tree at level h (all t-parse strings of length $h + t + 1$) before enumerating t-parses at level $h + 1$. The five options "extn edge weight", "extn runs", "extn tries", "extn max length" and "extn skip" allow the user to control the random distinguisher searches. Here one specifies the edge density (as a percentage), limits the attempts for proving that a t-parse G is minimal, bounds how many times the search tries to distinguish a specific minor G' from G, specifies the maximum length (number of operators) for each extension, and states how many operators are appended to Z before $G \cdot Z$ is tested for membership in $\mathcal{F}$. The "save nonmin proofs" option allows the user to save in the family database all nonminimal t-parse proofs. These proofs are not needed when the search uses only a random distinguisher searches as the only t-parse proof technique (see Section 4.3.3). The "congruence is tight" option notifies the workers that the implemented congruence is in fact the canonical congruence for $\mathcal{F}$. The "use testset" option tells the workers that a testset for $\sim_{\mathcal{F}}$ is available (and can be either loaded from disk or enumerated at run time). The "tilde minor search" option turns off the one-step minor mode for proving t-parses minimal. The "max equal iso level" option is used only during a restart of the search. Here the proofs from a backup database are loaded unconditionally for all t-parses shorter in length than this "iso level" (any t-parse not in the database, up to this level, is assumed to be non-canonic). The "workers run canonic" option allows the manager to request (to the dispatcher)

that the workers should help prove whether a t-parse is canonic or not (recall the two canonic stages mentioned in Section 4.3.1). The `"load all nonminimal"` option is used to cancel the default mode of retrying random distinguisher searches for "assumed nonminimal" t-parses (this is used in a restart of an approximation run). The `"limit to prefix"` option allows the search to start at any subtree of the search tree (we used this for our 2–FEEDBACK VERTEX SET computations). The `"send PMP"` option tells the workers to send back all "Partial Minor Proofs" for each t-parse G that they handle (e.g., a proof is partial if all of the one-step minors of G have not been distinguished from G). The `"univ dist search"` option turns on the universal distinguisher search method that was presented in Section 4.4.2. The `"pretest extensions"` option is for the random distinguisher searches. Here each extension Z must fail every direct nonminimal pretest that is installed for the current lower ideal $\mathcal{F}$. Finally, the `"canonic timeout"` option allows the impatient user to specify the maximum amount of CPU time that can be spent on trying to prove that a t-parse is non-canonic. If the timeout is reached then the system assumes that the t-parse is canonic for the integrity of the search.

For our obstruction set searches, we ran our manager and dispatcher processes on a Sparc-2 workstation. This machine has a relatively small amount (230M) of dedicated local disk space. This drive was mainly used to store the search database. However, this machine has 64M RAM of internal memory which allows for less disk swapping during our larger obstruction set searches. We concurrently ran workers at both the University of Victoria (only Sparcs) and Los Alamos National Laboratory (Sparcs, Cray Y-MPs, and IBM-6000s).

The following Table 5.1 lists our main software directories with the line counts for the source code. Not included is a few thousand lines of miscellaneous code (e.g., special source files to interface the LEDA graph library). The size of the executable program is 1344K after being stripped of debugger code; this executable was produced by a Sparc AT&T CC-3.02 compiler. We also have over one thousand lines of shell script code, where the average length of each script is about thirty lines. These scripts were written in `Perl`, `csh`, or `ksh`. In fact, an ASCII version of Table 5.1 was generated by a simple `csh` script that calls the UNIX `sed` and `awk` utilities.

Table 5.1: Source code breakdown by area for our VACS software (subdirectories).

Header Files		
.c lines*	.h lines	Directory
	287	inc
1392	1559	inc/array
	170	inc/automata
	1433	inc/tparse
	21	inc/tparse/i-o
	847	inc/tparse/algorithm
	455	inc/browser
	891	inc/family
	123	inc/family/testset
577	2661	inc/general
	2529	inc/graph
	699	inc/graph/algorithm
	330	inc/graph/i-o
	181	inc/isomorphism
	2015	inc/search
1469	1227	inc/set
	1141	inc/vacs

* C++ template code is included here.

Source Files	
.c lines	Directory
75	src/automata
2057	src/browser
3435	src/tparse
3594	src/tparse/algorithm
425	src/tparse/i-o
2204	src/family
983	src/family/testset
1044	src/filesystem
2999	src/general
4452	src/graph
4637	src/graph/algorithm
717	src/graph/generate
1503	src/graph/i-o
2444	src/graphplay
909	src/isomorphism
1185	src/i-o
154	src/locker
765	src/process
2378	src/search
5184	src/searchmain
1560	src/searchnode
212	src/special

.c lines = 46354, .h lines = 16569, and (.c + .h) lines = 62923.

The main engine for the VACS project is built and in operation. The computational results of this dissertation are our initial achievements regarding our systematic method for characterizing lower ideals. However, like in any major software project, there are several rough spots that need to be smoothed out. The next section mentions a few areas of research that we are currently pursuing.

5.3 Future Research Goals

We now begin with some obvious research topics that we want to explore first. These areas of research all deal with making obstruction set computations more practical (i.e., continuing the theme of Section 4.4).

- Implement the bounded treewidth search. One of our first tasks is to complete the proof for our 2–Feedback Vertex Set obstruction set. The easiest mechanical way of doing this is to patch the current system with a treewidth t-parse search, although Chapter 7 gives strong arguments that there may be an alternate proof. We note that searching through graphs of bounded treewidth is not straightforward. For example, our enumeration scheme presented in Chapter 2 is not as efficient for the treewidth case as it was for the pathwidth case. Thus, we may need results from the following research area to quickly purge t-parse redundancies.

- Develop better canonic algorithms for graphs of bounded combinatorial width. Currently, we have a practical canonicity algorithm for small t-parses. For some of our larger lower ideals we noticed that some t-parses, with as few as 13 vertices, caused the algorithm to stall. This is why a canonic time-out mechanism was installed. Since graph isomorphism is polynomial time when restricted to graphs of bounded treewidth (pathwidth), we also suspect there is a guaranteed polynomial time algorithm to test if a t-parse is canonic. We can not rule out the possibility that another operator set is more suitable for uniquely representing the set of partial t-trees (t-paths).

- Use a "virtual" boundary size with respect to the boundary minor order. We have noticed that our current search trees have several long branches that are the result of the boundary minor order being weaker than the minor order. For example, we find boundary minor obstructions with isolated boundary vertices even though we are pruning the search at disconnected t-parses. (See Section 4.4.1.) To reduce the size of such a search tree, one obvious idea is to redefine the boundary minor order. Suppose we consider graphs with a variable-sized boundary (up to some maximum size). Here the deletion of an isolated boundary vertex could be taken as a minor. Also, edge contractions between boundary vertices would be legal minors. Since there are more minors for a given graph G in this new framework, the chances of finding a minor congruent to G (with respect to $\sim_{\mathcal{F}}$) improves. Thus, there will be fewer minimal t-parses in the search tree. Using virtual boundaries is promising except for the fact that our prefix property (see Lemma 72) may no longer be valid.

- Reuse minimal t-parses for higher search widths. If one has an appropriate canonic scheme, one can reuse minimal boundaried graphs at width t for a width $t+1$ search (perhaps by simply inserting a new vertex operator at the beginning of each minimal t-parse). This idea follows from the observations: (1) minimality is satisfied if and only if all minors are distinguished by some extension, and (2) the set of graphs with a larger width is a bigger pool (superset) of potential distinguishers. It should be easy to implemented this computation saving feature.

- Do recursive minor checks. For any finite-index congruence $\sim$ it is possible to show that a t-parse G is nonminimal by finding a congruent boundaried minor that is more than a one-step minor of G. Here we construct a poset of boundaried minors (similar to Figure 3.1) with G at the top and the smallest t-parse $[\textcircled{0}, \textcircled{1}, \ldots, \textcircled{t}]$ at the bottom. We check the equivalence classes of all proper minors against G's equivalence class, beginning at the top and proceeding down in a breadth-first fashion. (Using recursive minor checks for $\sim = \sim_{\mathcal{F}}$ is superfluous by Lemma 71.)

- Adapt our search technique to handle other well-partial-orders. So far we have concentrated on the minor order. There are other interesting graph orders, such as the immersion order (see Section 3.3), that our search process may be adaptable. However, there are some details that need to be resolved. For example, we have to handle edge lifts across the boundary in the "boundary immersion" order with respect to a lower ideal's canonical congruence.

Two more proposed and on-going research areas, which are based on additional theoretical ideas, are presented in the following subsections.

5.3.1 Second-order congruence research

Another technique to reduce the search space for obstruction set computations is to use a second-order congruence (in replace of the canonical family congruence $\sim_{\mathcal{F}}$). Recall that the quantifiers $\exists$ and $\forall$ are read "there exists" and "for all", respectively.

Definition 91: The *canonical second-order congruence* for a lower ideal $\mathcal{F}$, denoted by $\approx_{\mathcal{F}}$, is defined on finite sets of t-boundaried graphs by: if S_1 and S_2 are t-boundaried graphs then $S_1 \approx_{\mathcal{F}} S_2$ if and only if for every t-boundaried graph Z:

$$(\exists X_1 \in S_1,\ X_1 \oplus Z \notin \mathcal{F}) \iff (\exists X_2 \in S_2,\ X_2 \oplus Z \notin \mathcal{F}) \quad .$$

A (non-canonical) second-order congruence for $\mathcal{F}$ is an equivalence relation $\sim$ defined on finite subsets of t-boundaried graphs for which $S_1 \sim S_1$ implies $S_1 \approx_{\mathcal{F}} S_2$.

We now point out that second-order congruences are related to our universal distinguisher search method that was presented in Section 4.4.2. Let $G' <_{\partial m} G$ denote a proper ∂-minor of a t-boundaried graph G (i.e., $G' \leq_{\partial m} G$ and $G' \neq G$). For any t-boundaried graph G consider these two sets $S_1 = \{G\}$ and $S_2 = \{G' \mid G' <_{\partial m} G\}$. We claim that G is *nonminimal*, with respect to a universal distinguisher search, if and only if $S_1 \approx_{\mathcal{F}} S_2$. That is, for every t-boundaried graph Z:

$$G \oplus Z \notin \mathcal{F} \ \Rightarrow\ (\exists G' <_{\partial m} G,\ G' \oplus Z \notin \mathcal{F}) \quad .$$

Or, in a more familiar form, there does not exist a t-boundaried graph Z such that:

$$G \oplus Z \notin \mathcal{F} \text{ and } (\forall G' <_{\partial m} G,\ G' \oplus Z \in \mathcal{F}) \quad .$$

And because $\mathcal{F}$ is a lower ideal, we easily note, in the other direction, for any t-boundaried graph Z,

$$(\exists G' <_{\partial m} G,\ G' \oplus Z \notin \mathcal{F}) \ \Rightarrow\ G \oplus Z \notin \mathcal{F} \quad .$$

The above logic leads us to the next observation.

Observation 92: If we have a second-order congruence for a minor-order lower ideal $\mathcal{F}$ then we can easily run a universal distinguisher search.

One of our motivations for using second-order congruences is the possibility of detecting when to stop the search without a prior knowledge of the largest combinatorial width of an obstruction. In fact, the following main result of [CDDF95] uses stopping conditions along with a factoring result for bounded pathwidth similar to the one given in Theorem 181 of Chapter 12.

Theorem 93: Suppose the following are known for a minor-order lower ideal $\mathcal{F}$:
(1) A decision algorithm for $\mathcal{F}$.
(2) A decision algorithm for a "terminating" second-order congruence for $\mathcal{F}$.
Then the obstruction set $\mathcal{O}(\mathcal{F})$ can be computed by an algorithm that uses the above two algorithms as subroutines.

We point out that if a second-order congruence is not known to satisfy our "terminating" definition, we still may be able to detect when to stop the search. Furthermore, we would like to answer this question: How can we guarantee that a signal to stop will be detected soon after the last obstruction has been computed? We would like to stop the search after the minimal graphs of width $t+1$ have been searched whenever the largest obstruction has width t.

5.3.2 Approximation algorithms based on partial obstruction sets

This dissertation has primarily been interested in computing complete characterizations of lower ideals (by forbidden minors). We realize, however, that there are some potential uses for partial lists of obstructions. We think that there is a future in building graph-theoretic approximation algorithms from a selected subset of the obstructions. We have already built a good software foundation for such an automated algorithm compiler.

It is suspected that for any minor-order lower ideal $\mathcal{F}$, most of the non-family graphs are above a small number of the obstructions $\mathcal{O}(\mathcal{F})$. It is well-known that the family of planar graphs is almost the same as the set of graphs that excludes just the one Kuratowski obstruction $K_{3,3}$. For example, we can prove the following known result:

Observation 94: Any three-connected non-planar graph with at least six vertices contains $K_{3,3}$ as a minor.

Another promising example was recently exhibited in the VLSI design area by the work of Langston and Ramachandramurthi [LR92, Ram94]. Their simple method of using just one (or a few) of the 3-track gate matrix layout obstructions (or equivalently, the 2–PATHWIDTH obstructions) for a membership algorithm has out performed the other best-known heuristics for the problem. Note that this research is based on a small percentage of the complete set of the 110 obstructions, recently proven complete by Kinnersley [KL94].

These two observations suggest that good approximating automata may be constructed by using partial obstruction sets.

One may wonder what obstructions are good candidates for approximating any specific lower ideal. (Actually, this technique may work for "almost" lower ideals too.) We know of three classes of obstructions that may be useful:

1. The obstructions with the smallest number of vertices (or edges).

2. All of the obstructions that are bounded by a small combinatorial width.

3. The obstructions that occur frequently via our random distinguisher searches.

It would be interesting to do some experimental computing to see how close, in practice, a family of graphs defined by excluding one of these classes is to a targeted lower ideal. The benefit of using the second class of obstructions (i.e., those of bounded combinatorial width) is that we do not have to compute the complete set of obstructions. Also, related to the third class, it is natural to assume that if a distinguisher Z for a t-parse G is easily found then the obstructions below $G \cdot Z$ are good candidates. It is expected that the above three classes of obstructions overlap.

The easiest way to use a partial obstruction set for some family $\mathcal{F}$ is to write quick minor-order tests $\{M_i\}$ for each obstruction O_i, $i = 1, 2 \ldots, r$. The approximating algorithm for an input graph G would return 'no' if any obstruction O_i is found as a minor (using M_i). Otherwise, it would return 'yes' that G is a member of $\mathcal{F}$. Note that this algorithm has only one-sided errors. This happens whenever it returns 'yes' when the real answer is 'no'.

Unfortunately the above approach is not very automated. This is because there is no general efficient algorithm that tests if any fixed graph O_i is a minor of another. (Recall that the Robertson–Seymour $O(n^3)$ algorithm has large hidden constants.) However when the input is restricted to graphs of bounded combinatorial width there is hope. In this case, we can build a minor testing automaton for a fixed graph O_i using the methods of Section 11.3. A testset T_i for O_i consists of boundaried graphs obtained from "pieces" of subdivided O_i. Here the constructed automaton M_i accepts t-parses that contain O_i as a minor.

Another method for generating approximating automata is to use pieces of all the obstructions as testsets. Using this approach we build one automaton for all of $\cup_{i=1}^{r} O_i$ for a lower ideal $\mathcal{F}$, instead of one automaton for each obstruction. Since obstructions usually share the same substructures, this testset $T = \cup_{i=1}^{r} T_i$ should be of manageable size. However, it may be easier to build a nondeterministic automaton M that accepts graphs in the union of the minor-containment families $\uparrow (\cup_{i=1}^{r} O_i)$ [or the intersection of the single obstruction lower ideals $\downarrow (\cap_{i=1}^{r} O_i)$] from the deterministic automata M_i, $i = 1, 2, \ldots, r$. See Figure 5.3. One may even convert M to a deterministic

automaton, and then minimize.

For the bounded width domain we may elect to use the set of easily proven minimal t-parses as an approximating testset since these, in most cases, are already "pieces" of obstructions.

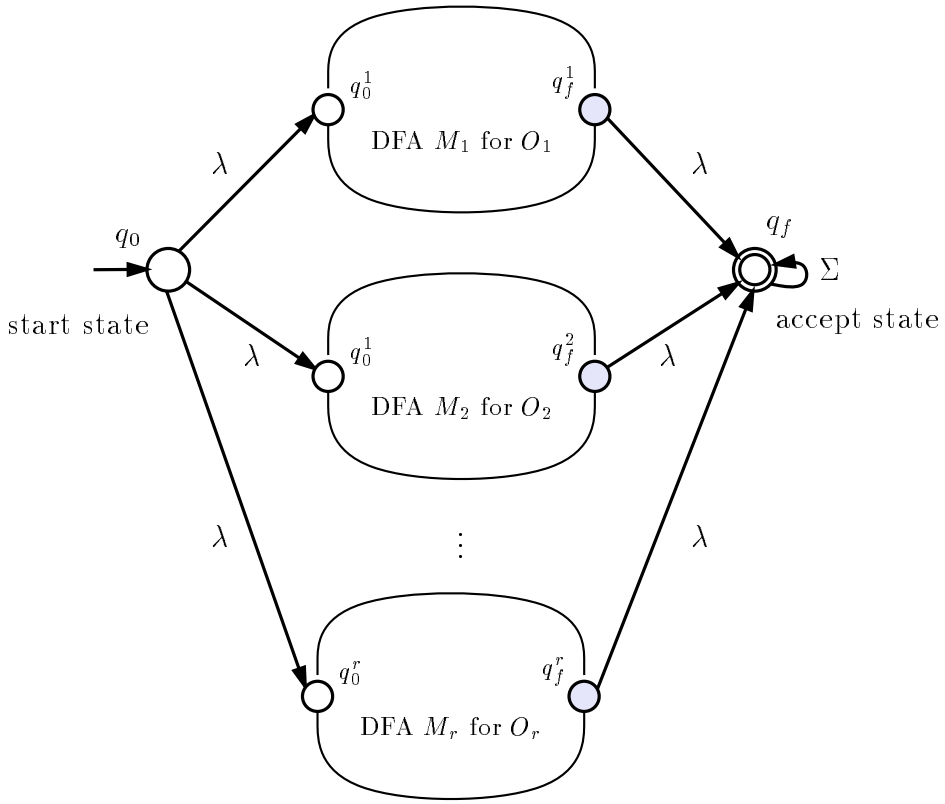


Figure 5.3: Building an NFA that accepts the union of minor-containment families.

Part II

Obstruction Set Characterizations

Chapter 6

Vertex Cover

We make two contributions in this chapter. First, we present a linear time algorithm that determines the size of the minimum vertex cover for graphs of bounded path-width (partial t-paths). This algorithm has the important property of being minimal (defined in Chapter 4 and reviewed later in Section 6.2). It is this property which allows us to compute the second contribution of this chapter: the obstruction sets for the first five parameterized family instances of the vertex cover problem.

6.1 Introduction

The general problem of determining if a graph has a vertex cover of size k, with k part of the input, is well known to be $\mathcal{NP}$-complete [GJ79]. However, several $\mathcal{NP}$-complete problems have polynomial time parameterized versions. In any of these instances a problem-specific integer k is held constant. The vertex cover problem is one such example where we are interested in determining if an input graph has a vertex cover of size at least k, where k is not part of the input. The brute force approach of checking all k subsets of the vertices gives a crude $O(n^{k+2})$ algorithm. Alternatively, if a tree or path decomposition of bounded width is available, determining the minimum vertex cover of a graph can be done linear time [ALS91]. In addition to our obstruction set characterizations, we present a practical, finite-state algorithm for the set of graphs with available path decompositions of bounded width. Furthermore, by

the observations (1) path decompositions for fixed k can be found in linear time (see [Bod93c, Klo93a]) and (2) a pathwidth bound exists for the family (see Theorem 103), we have a linear time algorithm. Interestingly, a direct parameterized algorithm is presented in [DF95] with the same complexity.

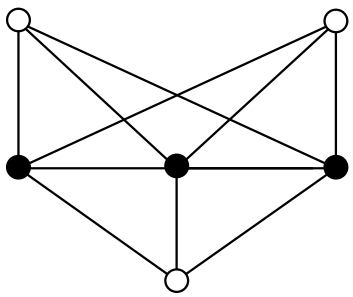
The rest of this chapter is organized as follows. The remaining part of this section formally defines the vertex cover problem, and introduces results and notation used in this chapter. Section 6.2 presents our general vertex cover algorithm and its minimal, finite-state variation for bounded pathwidth graphs. Section 6.3 contains results that reduce the amount of work needed to compute the obstructions sets. Section 6.4 presents the obtained obstructions sets and we conclude in Section 6.5 with some comments regarding the related k-IndependentSet and k-Clique families.

The general vertex-cover decision problem is defined as follows (see [GJ79]):

Problem 95: Vertex Cover (VC)
Input: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Question: Is there a subset $V' \subseteq V$ with $|V'| \leq k$ such that V' contains at least one vertex from every edge in E?

A set V' in the above problem is called a *vertex cover* for the graph G. The family of graphs that have a vertex cover of size at most k will be denoted by k–Vertex Cover. For a given graph G, let $VC(G)$ denote the least k such that G has a vertex cover of cardinality k.

Example 96: A graph in the 3–Vertex Cover family is displayed below. Notice that no edges remain when the set of black vertices of the example (or indeed, any vertex cover) is removed from the graph.



Finding the smallest vertex cover of a graph is important for many applications. For a simple example, consider a communications system that is modeled as a graph where vertices are computers and edges are the transmission lines. A minimum vertex cover of the graph indicates the smallest number of computers that need special monitoring software for all of the connections of the system.

The next result, stating that these parameterized vertex cover families can be characterized by forbidden minors, is well-known.

Lemma 97: The graph family k–VERTEX COVER is a lower ideal in the minor order.

Proof. Assume a graph $G(V, E) \in k$–VERTEX COVER has a minimal vertex cover $V' \subseteq V$. If $H = G \backslash \{(u, v)\}$ for some $(u, v) \in E$ (edge deletion), then V' is also a vertex cover for H. Likewise, if $u \in V$ is an isolated vertex of G, V' also covers $H = G \setminus \{u\}$ (vertex deletion). For any edge $(u, v) \in E$, observe that $|\{u, v\} \cap V'| \geq 1$. Let w be the new vertex created from u and v in $H = G/(u, v)$ (edge contraction). Clearly, $V'' = (V' \cup \{w\}) \setminus \{u, v\}$ is a vertex cover of H with cardinality at most k. Since any minor of G can be created by repeating the above operations, k–VERTEX COVER is a lower ideal. □

6.2 A Finite State Algorithm

In this section we give a practical, finite-state algorithm for the vertex cover problem on graphs of bounded pathwidth in t-parse form. This linear time algorithm is a dynamic program that makes a single left to right scan of the input t-parse $G_n = [g_1, g_2, \ldots, g_n]$. The computational process resembles a finite state automaton in that it accepts words over the operator alphabet Σ_t. Let m be the current scan position of the algorithm on input G_n. The *state table* at operator g_m is indexed by each subset S of the boundary ∂. These 2^{t+1} different entries are defined as follows:

$$V_m(S) = \min\{\ |V'|\ :\ V' \text{ is a vertex cover of } G_m \text{ and } V' \supseteq S\}$$

Two important observations about the state table are:

1. For each boundary subset $S \subseteq \partial$, $V_m(S)$ is a non-decreasing sequence of non-negative integers as m increases.

2. For any boundary subset $S \subseteq \partial$ and any boundary vertex $i \notin S$, either $V_m(S) = V_m(S \cup \{i\})$ or $V_m(S) = V_m(S \cup \{i\}) - 1$.

The algorithm, given in Figure 6.1, starts by setting the sizes for the minimal vertex covers on the edgeless graph $G_{t+1} = [\textcircled{0}, \textcircled{1}, \ldots, \textcircled{t}]$, for all subsets S of the initial boundary ∂.

The type of the operator g_{m+1} (a vertex operator or an edge operator) determines how the state table is updated during the scan. The update of an entry for a specific subset of the boundary S is further broken up according to the relationship between S and the operator. These transitions are described in cases 1–4 of Figure 6.1.

When the algorithm reaches the end of the t-parse, it has computed the minimum number of vertices needed for a vertex cover of G_n. This is because the entry $V_n(\emptyset)$ contains the size of the smallest vertex cover that contains the subset $\emptyset$ of the boundary. As this is an empty condition, $V_n(\emptyset)$ is the size of the smallest vertex cover in G_n.

Theorem 98: For any t-parse $G_n = [g_1, g_2, \ldots, g_n]$, the algorithm in Figure 6.1 correctly computes $VC(G_n)$.

Proof. If G is the empty graph then only steps I and III are executed and the correct result of $VC(G) = 0$ is returned. Assume that the algorithm is correct for all (prefix-) graphs of length m and less. We show that cases 1–4 of step II correctly update the state table, $V_{m+1}(S)$ for $S \subseteq \partial$.

Case 1: $g_{m+1} = \textcircled{i}$ and $i \notin S$

Let V' be a witness vertex cover for $V_m(S)$. Since the new vertex created by g_{m+1} does not add any edges, V' is a vertex cover for G_{m+1}. Since $V' \supseteq S$ for G_m and $i \notin S$, $V' \supseteq S$ for G_{m+1}. Therefore, $V_{m+1}(S) \leq V_m(S)$.

I For $m = t + 1$, set for every $S \subseteq \partial$

$$V_{t+1}(S) = |S| \quad .$$

II For $t + 1 < m < n$, do the following cases:

Case 1: vertex operator $\textcircled{i}$ and $i \notin S$

$$V_{m+1}(S) = V_m(S)$$

Case 2: vertex operator $\textcircled{i}$ and $i \in S$

$$V_{m+1}(S) = V_m(S \setminus \{i\}) + 1$$

Case 3: edge operator $\boxed{i\ j}$, where $i \in S$ or $j \in S$

$$V_{m+1}(S) = V_m(S)$$

Case 4: edge operator $\boxed{i\ j}$, where $i \notin S$ and $j \notin S$

$$V_{m+1}(S) = \min\{V_m(S \cup \{i\}), V_m(S \cup \{j\})\}$$

III The size of the minimum vertex cover of G is

$$V_n(\emptyset) \quad .$$

Figure 6.1: A general vertex cover algorithm for t-parses.

Let V' be a witness vertex cover for $V_{m+1}(S)$. Since V' is minimal, the isolated vertex created by g_{m+1} is not in V'. Thus V' is a vertex cover for G_m. Since the property $V' \supseteq S$ is preserved, $V_m(S) \leq V_{m+1}(S)$.

Case 2: $g_{m+1} = \textcircled{i}$ and $i \in S$

Let $S' = S \setminus \{i\}$ and V' be a witness vertex cover for $V_m(S')$. Now $W = V' \cup \{i\}$ is a vertex cover for G_{m+1} such that $W \supseteq S$. So, $V_{m+1}(S) \leq V_m(S') + 1$.

For the other direction, let V' be a witness vertex cover for $V_{m+1}(S)$. Since the new boundary vertex i does not help in any vertex cover of G_m, $V'' = V' \setminus \{i\}$ is a vertex cover for G_m such that $V'' \supseteq S$. Hence $V_{m+1}(S) \geq V_m(S') + 1$.

Case 3: $g_{m+1} = \boxed{i\ j}$ where $i \in S$ or $j \in S$

Let V' be a witness vertex cover for $V_m(S)$. Since the boundary is not changed by the edge operator g_{m+1} and $i \in S$ or $j \in S$, V' also covers the edges of G_{m+1}. Thus, $V_{m+1}(S) \leq V_m(S)$. If V'' is a vertex cover of G_{m+1} with $i \in S$ or $j \in S$, then V'' also covers the edges of G_m. So $V_{m+1}(S) \geq V_m(S)$.

Case 4: $g_{m+1} = \boxed{i\ j}$ where $i \notin S$ and $j \notin S$

Let $S' = S \cup \{i\}$ and V' be a witness vertex cover for $V_m(S')$. Since $V' \supseteq S$ is a vertex cover for G_{m+1}, as vertex i is in V', $V_{m+1}(S) \leq V_m(S')$. Likewise, if $S'' = S \cup \{j\}$, then $V_{m+1}(S) \leq V_m(S'')$. So $V_{m+1}(S) \leq \min\{V_m(S \cup \{i\}), V_m(S \cup \{j\})\}$.

Let V' be a witness vertex cover for $V_{m+1}(S)$. Since (i, j) is an edge, either $i \in V'$ or $j \in V'$. Thus $V' \supseteq S \cup \{i\}$ or $V' \supseteq S \cup \{j\}$. If $V' \supseteq S \cup \{i\}$, then V' is a vertex cover for G_m, and so $V_m(S \cup \{i\}) \leq V_{m+1}(S)$. Otherwise, $V' \supseteq S \cup \{j\}$, and $V_m(S \cup \{j\}) \leq V_{m+1}(S)$. Therefore, $\min\{V_m(S \cup \{i\}), V_m(S \cup \{j\})\} \leq V_{m+1}(S)$. □

The following lemma shows that we can limit the vertex-cover membership algorithm to a finite number of possible configurations when testing for membership in k–VERTEX COVER.

Lemma 99: For any fixed integer k as an upper bound, the algorithm given in Figure 6.1 can be converted to a finite state algorithm $\mathcal{A}_k$.

Proof. We show that for fixed k, there are only a finite number of possible states. Consider the state table entry for a boundary subset S. If $V_i(S)$ becomes $k + 1$ for

some i, then the monotonicity of $V_m(S)$ guarantees that $V_j(S) \geq k+1$ for all $j > i$. As we are only interested in knowing whether or not there exists a vertex cover of size k containing S, we can restrict $V_m(S)$ to be in $\{0, 1, 2, \ldots, k, k+1\}$. As there are 2^{t+1} entries in the state table, the number of states is bounded by $(k+2)^{2^{t+1}}$.

To make the parameterized algorithm $\mathcal{A}_k$ for k–VERTEX COVER, change any update function $V_{m+1}(S) = f(V_m)$ in the four cases with $V_{m+1}(S) = \min(f(V_m), k+1)$. It is straightforward to verify that this modified algorithm correctly computes the same state table except that any entry greater than $k+1$ is replaced by $k+1$. □

We define the *final state of a t-parse* G, denoted by V_G, to be the state table (state vector) $[V_m(\emptyset), \ldots, V_m(S)]$ when the finite-state algorithm $\mathcal{A}_k$ terminates. For each boundary subset S, let $V_G(S)$ denote the S entry of V_G. If G and H are t-parses and $V_G = V_H$, it follows immediately that for any operator string $Z \in \Sigma_t^*$, we have $V_{G \cdot Z} = V_{H \cdot Z}$. This in turn implies that $G \cdot Z \in \mathcal{F} \Leftrightarrow H \cdot Z \in \mathcal{F}$ for all Z. That is, G and H agree on all extensions. When the converse of this property holds, the finite-state algorithm is *minimal*. Formally, minimality is satisfied if

$$(\text{ for all } Z \in \Sigma_t^*,\ G \cdot Z \in \mathcal{F} \Leftrightarrow H \cdot Z \in \mathcal{F} \ ) \ \Rightarrow \ V_G = V_H \quad .$$

To show that our finite-state algorithm is minimal, we need to show that if G and H are t-parses, and G and H agree on all extensions, then $V_G = V_H$. We will show the contrapositive; that is, if $V_G \neq V_H$, then there exists an extension Z such that G and H do not agree on Z (that is, there exists Z such that either $G \cdot Z \in F$ and $H \cdot Z \notin F$, or $G \cdot Z \notin F$ and $H \cdot Z \in F$).

Before proving that this k–VERTEX COVER algorithm is minimal, we need the following lemma that provides us with a boundary vertex for building such a t-parse extension Z.

Lemma 100: If G and H are t-parses such that $V_G(\partial) \neq V_H(\partial)$, then there exists an $S \subset \partial$ (i.e., a proper subset of the boundary) such that $V_G(S) \neq V_H(S)$.

Proof. Assume that $V_G(\partial) \neq V_H(\partial)$ is the only difference in the state table. The following three facts

1. $V_G(\partial \setminus \{i\}) = V_H(\partial \setminus \{i\})$ for all $i \in \partial$,
2. $V_G(\partial \setminus \{i\}) \leq V_G(\partial) \leq V_G(\partial \setminus \{i\}) + 1$ for all $i \in \partial$ and
3. $V_H(\partial \setminus \{i\}) \leq V_H(\partial) \leq V_H(\partial \setminus \{i\}) + 1$ for all $i \in \partial$

imply that

$$V_G(\partial) \leq V_G(\partial \setminus \{i\}) + 1 = V_H(\partial \setminus \{i\}) + 1 \leq V_H(\partial) + 1 \text{ for all } i \in \partial,$$

and

$$V_G(\partial) \geq V_G(\partial \setminus \{i\}) = V_H(\partial \setminus \{i\}) \geq V_H(\partial) - 1 \text{ for all } i \in \partial.$$

After combining the above, $V_H(\partial) - 1 \leq V_G(\partial) \leq V_H(\partial) + 1$. So, without loss of generality, assume $V_G(\partial) = V_H(\partial) - 1 = d$. From this identity and facts 1 and 3 above (also see the partial state tables below), we must have $V_G(\partial) = V_G(\partial \setminus \{i\}) = d$ for all $i \in \partial$.

graph G			graph H	
$V_G(\partial)$	d	$<$	$V_H(\partial)$	$d+1$
$V_G(\partial \setminus \{i\})$	$d-1$ or d	$\}=\{$	$V_H(\partial \setminus \{i\})$	d or $d+1$

This can happen if and only if each of the boundary vertices of G are attached to some non-boundary vertex. If not, then a vertex cover $V' \supseteq \partial$ of G would have a redundant vertex $i \in \partial$. The vertex cover created by eliminating vertex i from V' contradicts the value of $V_G(\partial \setminus \{i\})$. However, such a graph G does not exist since the last vertex operator can only have boundary vertex neighbors. Thus, we can conclude that $V_G(\partial) = V_H(\partial)$ or there exists a $S \subset \partial$ such that $V_G(S) \neq V_H(S)$. □

Theorem 101: For the k–VERTEX COVER family, the finite-state algorithm $\mathcal{A}_k$ is minimal.

Proof. Let G and H be t-parses. As discussed above, we show that if $V_G \neq V_H$, then there exists an extension Z such that G and H do not agree on Z. Note that the theorem holds trivially if either one of G or H is not in $\mathcal{F} = k$–VERTEX COVER

by the empty extension $Z = [\,]$. If both $G \notin \mathcal{F}$ and $H \notin \mathcal{F}$ then $V_G = V_H = [k+1, k+1, \ldots, k+1]$ since $V_G(\emptyset) = k+1$ implies $V_G(S) = k+1$ for all $S \subseteq \partial$.

So suppose that $V_G \neq V_H$. Then without loss of generality, there is a boundary subset S with minimum cardinality such that $V_G(S) < V_H(S) < k+1$. Lemma 100 guarantees that $S \neq \partial$.

Let $\{v_1, v_2, \ldots, v_{|S|}\}$ be the boundary vertices in S. Pick a boundary vertex $i \notin S$ and any other boundary vertex $j \neq i$. Construct an extension Z as follows.

$$Z = [\textcircled{i}, \boxed{i\ v_1}, \textcircled{i}, \boxed{i\ v_2}, \ldots \textcircled{i}, \boxed{i\ v_{|S|}}, \overbrace{\textcircled{i}, \textcircled{j}, \boxed{i\ j}, \ldots, \textcircled{i}, \textcircled{j}, \boxed{i\ j}}^{k - V_H(S) + 1 \text{ times}}]$$

The extension Z essentially forces the boundary vertices S to be covered while adding $k - V_H(S) + 1$ isolated edges. Now, $\mathrm{VC}(H \cdot Z)$ is given by $V_H(S) + (k - V_H(S) + 1)$, which equals $k+1$, and so $H \cdot Z \notin \mathcal{F}$. However, $\mathrm{VC}(G \cdot Z)$ is $V_G(S) + (k - V_H(S) + 1) < V_H(S) + (k - V_H(S) + 1) = k + 1$, and so $G \cdot Z \in \mathcal{F}$. Therefore, the t-parses G and H do not agree on all extensions. □

Example 102: Table 6.1 shows the application of the algorithm $\mathcal{A}_k$ to the t-parse given earlier in Example 24 on page 40. As can been seen by examining the graph in Example 24, a minimum vertex cover has cardinality 3, which equals $V_{14}(\emptyset)$.

6.3 The VC Obstruction Set Computation

For k–Vertex Cover two ingredients suffice to compute their obstruction sets. First, we need to know a bound on the pathwidth (or treewidth) of the obstruction set. In general such a bound always exists since the obstruction set is finite. Given such a bound, we can compute all of the obstructions by restricting our search to a fixed pathwidth. Second, we want a minimal finite-state algorithm that operates on t-parses. An overview of how the algorithm $\mathcal{A}_k$ is used is given in Section 6.4.

For vertex cover, we have both of these ingredients. A minimal finite-state algorithm was described in the previous section, and a pathwidth bound is shown later in this section. (The finite-state algorithm is also used as a membership algorithm.)

Table 6.1: Vertex cover state tables computed (columns) for Example 102.

m	3	4	5	6	7	8	9	10	11	12	13	14
S g_m	–	[0 1]	[1 2]	(1)	[0 1]	[1 2]	(0)	[0 1]	[0 2]	(2)	[0 2]	[1 2]
$\emptyset$	0	1	1	1	2	2	2	2	3	3	3	**3**
$\{0\}$	1	1	2	2	2	2	3	3	3	3	3	3
$\{1\}$	1	1	1	2	2	2	2	2	3	3	3	3
$\{2\}$	1	2	2	2	2	2	2	3	3	4	4	4
$\{0,1\}$	2	2	2	3	3	3	3	3	3	3	3	3
$\{0,2\}$	2	2	2	2	2	2	3	3	3	4	4	4
$\{1,2\}$	2	2	2	3	3	3	3	3	3	4	4	4
$\{0,1,2\}$	3	3	3	3	3	3	4	4	4	4	4	4

In short, the input/output combination used to compute the obstruction set for a particular k–Vertex Cover lower ideal is as follows:

input: • pathwidth t
• minimal finite-state algorithm for k–Vertex Cover
output: • obstructions of pathwidth t

The following sequence of results show the value of t that is needed to get the complete set of obstructions $\mathcal{O}(k$–Vertex Cover$)$ for k–Vertex Cover and why we can restrict our search to connected graphs.

We first need an upper bound on the pathwidth of the obstruction set. That is, we need a result of the form: *if* $G \in \mathcal{O}(k$–Vertex Cover$)$*, then* G *has pathwidth at most* k'. For vertex cover, such a bound is easily obtained. We first show that the family k–Vertex Cover is contained in the family k–Pathwidth. This result is well-known [vL90]. It follows from this that $\mathcal{O}(k$–Vertex Cover$)$ is contained in $(k+1)$–Pathwidth.

Theorem 103: The pathwidth of any member of k–Vertex Cover is at most k.

Proof. For a given graph G of k–VERTEX COVER, let V' be a subset of of the vertices of size k that covers all edges. Denote the order of G by n. Let the vertices V of G be indexed by $1, 2, \ldots, n$ with the vertices $V \setminus V'$ coming first. We claim that $\{X_i \mid 1 \leq i \leq n-k\}$ where $X_i = V' \cup \{i\}$ is a path decomposition of G.

Since every vertex is either in V' or is in $V \setminus V'$ we have $\bigcup_{1 \leq i \leq n-k} X_i = V$. Let (u, v) be an edge of G. Since V' is a vertex cover, without loss of generality assume $u \in V'$. If also $v \in V'$ then any subset X_i contains both u and v. Otherwise, v must be indexed between 1 and $n-k$ and the subset X_v contains both u and v. Finally, note that for any $1 \leq i < j \leq n-k$ we have $X_i \cap X_j = V'$ (i.e., the interpolation property is satisfied). Thus, we have a path decomposition of pathwidth k. □

The above theorem cannot be improved, as the complete graph K_{k+1} with pathwidth k is a member of k–VERTEX COVER.

Corollary 104: If $G \in \mathcal{O}(k\text{–VERTEX COVER})$, then the pathwidth of G is at most $k+1$.

Proof. For any edge $e = (u, v) \in E(G)$, let $G' = G \setminus e$. Since G is an obstruction for k–VERTEX COVER, $G' \in k$–VERTEX COVER and hence $\mathrm{VC}(G') \leq k$ by Theorem 103. Let V' be a witness vertex cover for G'. Now $V = V' \cup \{u\}$ is a vertex cover for G of order at most $k+1$. Therefore, by Theorem 103 again, the pathwidth of G is at most $k+1$. □

The number of obstructions we need to find can be reduced by some straightforward observations. The following observations are special cases of the more general results found in Section 4.4.1 regarding disconnected pruning.

Observation 105: If C_1 and C_2 are any two graphs then $\mathrm{VC}(C_1 \cup C_2) = \mathrm{VC}(C_1) + \mathrm{VC}(C_2)$.

Observation 106: If $O = C_1 \cup C_2$ is an obstruction for k–VERTEX COVER, then C_1 and C_2 are obstructions for k'–VERTEX COVER and k''–VERTEX COVER, respectively, for some $0 < k', k'' < k$, with $k' + k'' = k - 1$.

Hence we can restrict our attention to connected obstructions; any disconnected obstruction O of k–VERTEX COVER has $\mathrm{VC}(O) = k+1$ and is an union of graphs from $\bigcup_{i=0}^{k-1} \mathcal{O}(i\text{–VERTEX COVER})$.

Example 107: Since K_3 is an obstruction for 1–VERTEX COVER, and K_4 is an obstruction for 2–VERTEX COVER, the disconnected graph $K_3 \cup K_4$ is an obstruction for 4–VERTEX COVER.

6.4 The VC Obstructions

In essence, our theory allows us to compute for any t all of the obstructions that have pathwidth at most t. However, tractability problems arise as t increases. As shown in the preceding section, we need to use t-parses, $t = k+1$, to obtain all of the obstructions for k–VERTEX COVER.

A brief description of the k–VERTEX COVER obstruction set computation is now stated. The set of all t-parses can be viewed as a tree (recall Section 4.2), in which the parent of a t-parse G of length n is the length prefix of G of length $n-1$. The root of the tree is the empty graph $[\textcircled{0},\textcircled{1},\ldots,\textcircled{t}]$. The minimality of the finite-state algorithm allows us to directly compute a 'pruning rule' for the tree. That is, a t-parse G is minimal if and only if it is not congruent to any of its one-step minors. Any ∂-obstruction (minimal leaf of this search tree) that has all of its minors ($\leq_m$ order) in k–VERTEX COVER is a member of the obstruction set. (That is, we just check that boundary edge contracted minors fall into the family.)

A summary of our obstruction set computations for various k–VERTEX COVER families is shown in Table 6.2. The total graphs column shows the size of the pruned tree described above. In the minimal graphs column, the number of internal graphs plus obstructions is shown; that is, the leaves that are not obstructions have not been counted. The growth rate of the tree can be seen to be extremely high as k (and hence the pathwidth t) increases. The revalent search tree for the 2–VERTEX COVER run is shown in Figure 6.2 for pathwidth 3.

Besides the single obstruction K_2 for the trivial family 0–VERTEX COVER, the

Table 6.2: Summary of obstruction set computation for vertex cover.

k	Elapsed time	Minimal graphs	Total graphs	Connected obstructions	Total obstructions
1	5 seconds	8	31	1	2
2	25 seconds	42	301	2	4
3	3 minutes	320	3,871	3	8
4	4 hours	4,460	82,804	8	18
5	6 days	121,228	3,195,445	31	56

connected obstructions for k–VERTEX COVER, $1 \leq k \leq 5$, are shown in Figures 6.3–6.6. Some patterns become apparent in these sets of obstructions. One such easily proven observation is as follows.

Observation 108: For the family k–VERTEX COVER, both the complete graph K_{k+2} and the cycle C_{2k+1} are obstructions.

6.5 Independent Set and Clique Families

Now consider the following related families of graphs where n denotes the order of a graph and k is any non-negative integer:

$$\begin{aligned} k\text{–CLIQUE} &= \{ \text{ graphs with maximum clique} \leq k \} . \\ k\text{–INDSET} &= \{ \text{ graphs with maximum independent set} \leq k \} . \\ \neg(k\text{–CLIQUE}) &= \{ \text{ graphs with } n - \text{maximum clique} \leq k \} . \\ \neg(k\text{–INDSET}) &= \{ \text{ graphs with } n - \text{maximum independent set} \leq k \} . \end{aligned}$$

In fact one of these families (i.e., $\neg(k$–CLIQUE)) is the same as our studied parameterized vertex-cover families [GJ79].

Lemma 109: The two graph families $\neg(k$–INDSET) and k–VERTEX COVER are identical.

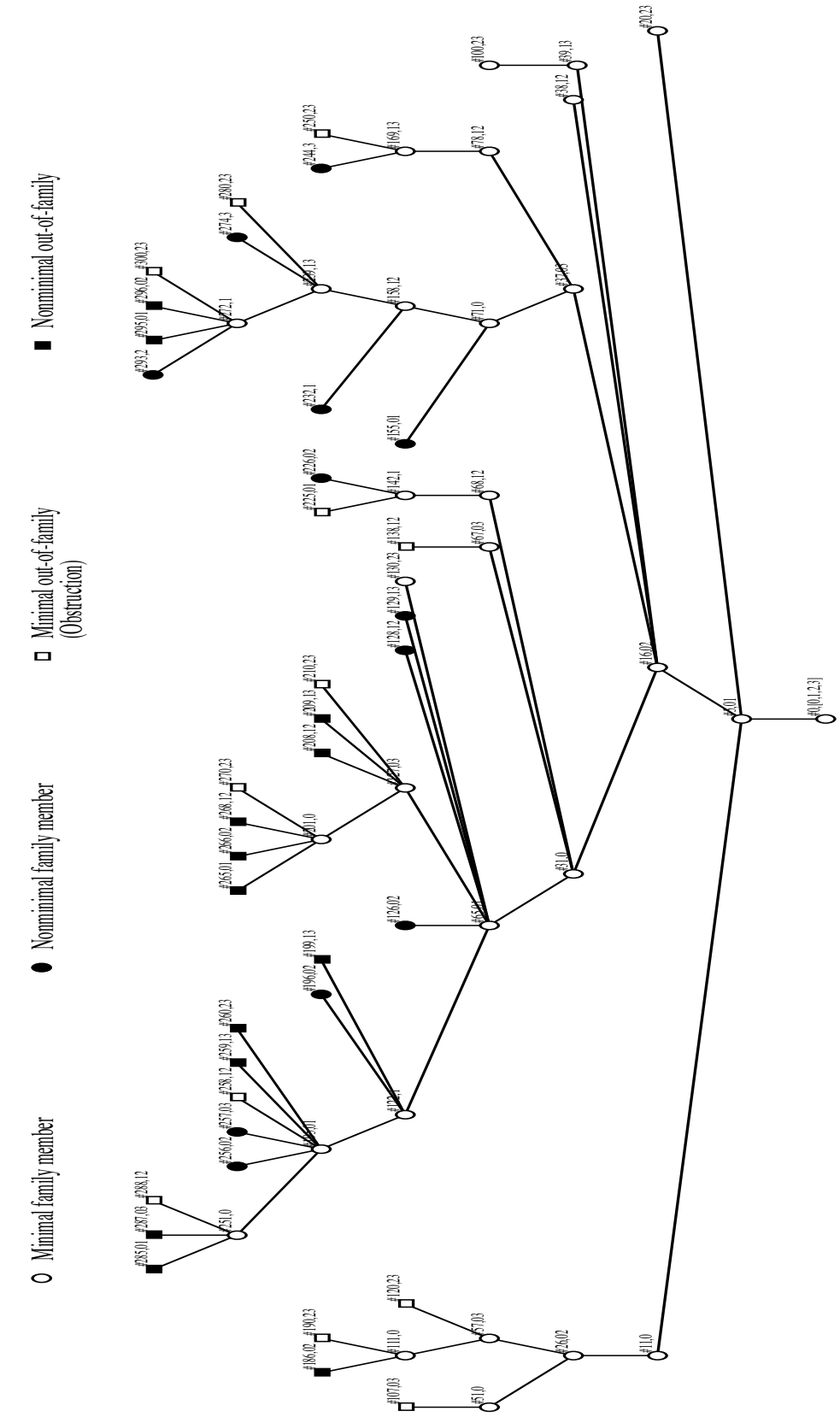


Figure 6.2: Actual search tree for 2–VERTEX COVER with graphs of pathwidth 3.

Proof. It is easy to see that if I is a maximal independent set of a graph $G = (V, E)$ then $V \setminus I$ is a vertex cover. Also, if V' is a minimal vertex cover then $V \setminus V'$ is an independent set. □

Part of the next result has been proven earlier in Lemma 97. This one illustrates another method for proving graph families are lower ideals.

Lemma 110: The family k–VERTEX COVER=$\neg(k$–INDSET) is a lower ideal in the minor order for any $k \geq 0$ while the families k–CLIQUE, k–INDSET, and $\neg(k$–CLIQUE) are *not* lower ideals for all $k \geq 2$, $k \geq 1$ and $k \geq 0$, respectively.

Proof. Let $G = (V, E)$ be a graph in $\neg(k$–INDSET) with maximum independent set I. If v is an isolated vertex of G then it is also in the set I. Thus, deleting v from G preserves the invariant n − maximum independent set. Deleting any edge e can only increase the maximum independent set while n stays fixed, so $(G \setminus \{e\}) \in \neg(k$–INDSET). Let G' be the result of contracting an edge $e = (u, v)$ of G and let I' be a maximum independent set of G'. Also let w denote the new vertex of G' created by the edge contraction. If $|I'| > |I|$ then we get a contradiction to the fact that I was maximum. To see this, consider these two cases. If $w \in I'$ then $(I' \setminus \{w\}) \cup \{u\}$ is a larger maximum independent set for G. If $w \notin I'$ then I' itself is a larger maximum independent set for G. Since I is an independent set, at most one vertex u or v can be in I. So going from G to G', $I' = I \setminus \{u, v\}$ is an independent set for G'. Thus, $|I| - 1 \leq |I'| \leq |I|$. Finally since the value of n decreases by exactly one during an edge contraction, $G' \in \neg(k$–INDSET). We have shown that all minor operations are closed with respect to $\neg(k$–INDSET).

Now consider the family 2–CLIQUE and a member C_4, the cycle of length four. Contracting any edge of C_4 produces $C_3 \simeq K_3$ which is not a member of 2–CLIQUE. Similar examples, such as K_{k+1} with a subdivided edge, show that k–CLIQUE is not a lower ideal for all $k \geq 2$. The only graphs in the family 1–CLIQUE are the isolated graphs (unions of K_1's) and the only minors (by isolated vertex deletions) have a maximum clique bounded above by 1. So 1–CLIQUE is a lower ideal.

As pointed out in the first paragraph, the maximum independent set can increase with edge deletions. It follows that k–INDSET is not a lower ideal for all but the trivial family 0–INDSET (i.e., the empty graph has no deletable edges).

For the family $\neg(k\text{–}\textsc{Clique})$, $k \geq 2$, consider a graph member $G = (k \cdot K_1) \cup K_k$. Note that G has $n = 2k$ vertices and a maximum independent set size of k. Deleting an edge from G causes an increase in the invariant $n -$ maximum clique. For $k = 1$ consider the graph $G = K_1 \cup K_2$. The graph created by deleting the edge from G has $n = 3$ and maximum clique of 1, so $\neg(1\text{–}\textsc{Clique})$ is not a lower ideal in the minor order. Likewise, since K_3 is a member of $\neg(0\text{–}\textsc{Clique})$ but any minor created by a single edge deletion is not, the family $\neg(0\text{–}\textsc{Clique})$ is not a lower ideal. □

In view of the previous results, we can characterize the $\neg(k\text{–}\textsc{IndSet})$ families in terms of the finite k–Vertex Cover obstruction sets. The other remaining few minor-order lower ideals are trivial to characterize. The reader should be able to verify that $\mathcal{O}(0\text{–}\textsc{Clique}) = \{K_1\}$, $\mathcal{O}(1\text{–}\textsc{Clique}) = \{K_2\}$, and $\mathcal{O}(0\text{–}\textsc{IndSet}) = \{K_1\}$.

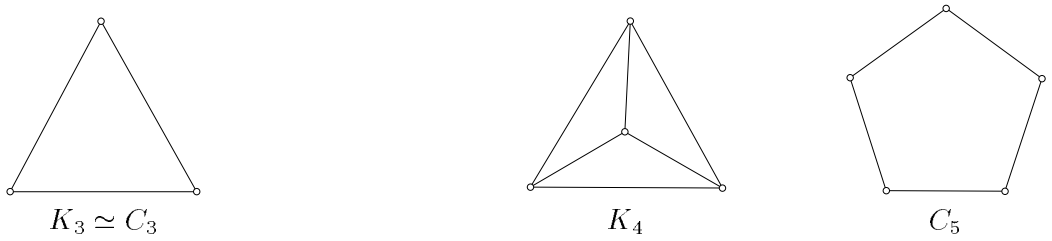


Figure 6.3: Connected obstructions for 1– and 2– VertexCover.

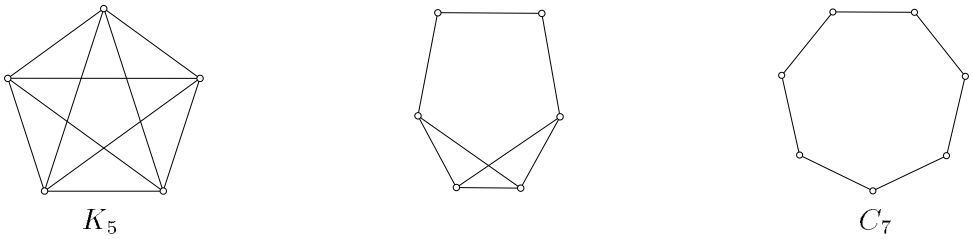


Figure 6.4: Connected obstructions for 3–Vertex Cover.

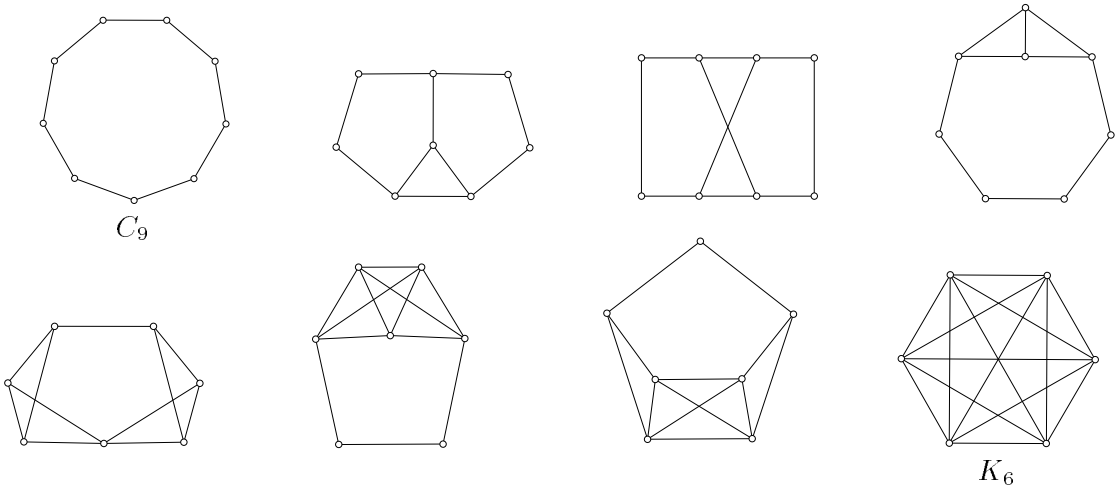


Figure 6.5: Connected obstructions for 4–Vertex Cover.

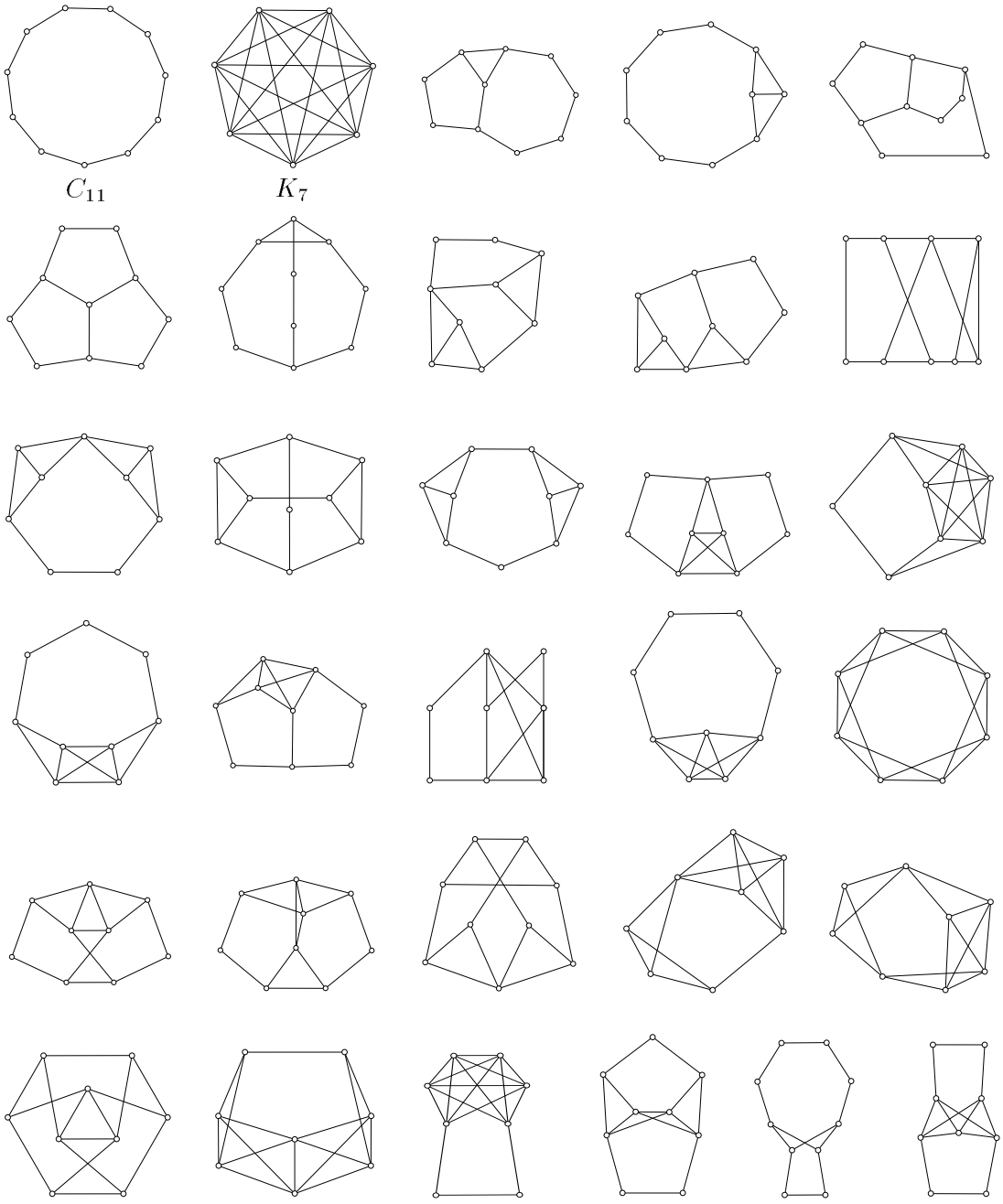


Figure 6.6: Connected obstructions for 5–Vertex Cover.

Chapter 7

Feedback Vertex/Edge Sets

In this chapter we characterize two types of simple graph families. The first family consists of those graphs for which all cycles can be covered with a small set of vertices. The second family consists of those graphs for which all cycles can be covered with a small set of edges. In some limited sense, these families are like the vertex cover families studied in the previous chapter (where a vertex cover can be thought of as a set of vertices that cover all edges). This chapter also illustrates the use of testsets to prove (or disprove) minimality of t-boundaried graphs.

7.1 Introduction

The characterization of graph families based on the following two well-known problems is the focus of this chapter (see [GJ79]).

Problem 111: Feedback Vertex Set (FVS)
Input: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Question: Is there a subset $V' \subseteq V$ with $|V'| \leq k$ such that V' contains at least one vertex from every cycle in G?

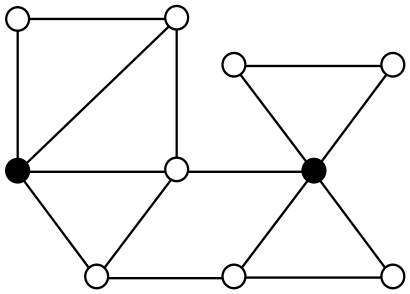
A set V' in the above problem is called a *feedback vertex set* for the graph G. The family of graphs that have a feedback vertex set of size at most k is be denoted by k–Feedback Vertex Set. It is easy to verify that for each fixed k the set of graphs

in k–FEEDBACK VERTEX SET is a lower ideal in the minor order. For a given graph G, let $FVS(G)$ denote the least k such that G has a feedback vertex set of cardinality k. Our second problem of interest is now stated.

Problem 112: Feedback Edge Set (FES)
Input: A graph $G = (V, E)$ and a positive integer $k \leq |E|$.
Question: Is there a subset $E' \subseteq E$ with $|E'| \leq k$ such that $G \setminus E'$ is acyclic?

The edge set E' is a *feedback edge set*. Also for a given graph G, let $FES(G)$ denote the least k such that G has a feedback edge set of cardinality k, and the family k–FEEDBACK EDGE SET $= \{G \mid FES(G) \leq k\}$.

Example 113: Displayed below is a graph in the 2–FEEDBACK VERTEX SET family. Notice that when the black vertices are removed from the example, the graph becomes acyclic (a forest).



The reader should note that the graph in the previous example requires 6 edges in any feedback edge set and thus it is a member of 6–FEEDBACK EDGE SET.

One classic application for minimum feedback sets in (directed) graphs has to do with operating systems. Consider a graph that models processes and system resources as vertices and processor requests or needs as edges. When deadlocks occur in the system a feedback edge set can be picked to refuse selected processor requests or a feedback vertex set can be picked to shut down processors or resources.

For both types of these parameterized families, a standard "slow" polynomial time algorithm exists for membership testing, based on the brute force technique of checking all $\binom{n}{k}$ covering possibilities (where n equals the number of vertices for FVS and

number of edges for FES). We used a somewhat practical $O((2k)^k n^2)$ algorithm for k–FEEDBACK VERTEX SET while debugging our linear time dynamic program (i.e., our finite-index congruence for t-parses). This algorithm by Fellows and Downey (see [DF95]) is based on (1) a quick algorithm by Itai and Rodeh in [IR77] for finding short cycles and (2) the fact that a graph G of minimum degree three with girth at least $2k$ is not in k–FEEDBACK VERTEX SET. For the feedback edge set problem, we show in Section 7.4.1 how to easily check if a graph is a member of k–FEEDBACK EDGE SET.

7.2 The FVS Obstruction Set Computation

We now focus on two problem-specific details for finding the k–FEEDBACK VERTEX SET obstruction sets: a finite-index congruence and a complete testset (i.e., steps 2 and 4 of Section 4.3). We first present a practical, linear time algorithm for the feedback vertex set problem on graphs of bounded pathwidth/treewidth in t-parse form. This general-purpose algorithm is altered to act as a finite-index congruence, that is a refinement of the canonical congruence. We then show how to produce testsets for the graph families k–FEEDBACK VERTEX SET, $k \geq 0$, with respect to any boundary size t.

7.2.1 A finite state algorithm

Throughout the following discussion the boundary size (and width) of a t-parse is fixed. Recall that the current set of boundary vertices of a t-parse G_n is denoted by the ∂ symbol. For any subset S of the boundary ∂, we define the following for all prefixes G_m of G_n, $m \leq n$.

$$F_m(S) = \begin{cases} \text{least } k \text{ such that there is an FVS } V \text{ of } G_m \text{ with } V \cap \partial = S \text{ and } |V| = k \\ \text{otherwise } \infty \text{ whenever } (G \cap \partial) \setminus S \text{ contains a cycle} \end{cases}$$

For any witness set V of G_m consisting of $F_m(S)$ vertices, there is an associated *witness forest* consisting of the trees that contain at least one boundary vertex in $G_m \setminus V$. A witness forest tells us how tight the boundary vertices are held together. Some

of these forests are more concise than others for representing how vertex deletions can break up the boundary.

For two witness forests A and B, with respect to $F_m(S)$, we say $A \leq_w B$ if the following two conditions hold:

1. For any two boundary vertices i and j, i and j are connected in A if and only if i and j are connected in B.

2. If for any t-parse extension Z where there exists some non-boundary vertex b of B such that $(B \setminus \{b\}) \cdot Z$ is acyclic then there exists a non-boundary vertex a of A such that $(A \setminus \{a\}) \cdot Z$ is acyclic.

Also two witness forests A and B are equivalent ($A \equiv_w B$) if $A \leq_w B$ and $B \leq_w A$. A witness forest in reduced form (minimal number of vertices) is called a *park*. The next lemma provides a way of cleaning up a forest to yield a park.

Lemma 114: A witness forest W of G_m may be reduced to a park as follows:
(a) all leafs (end-vertices) not on the boundary may be pruned, and
(b) any non-boundary vertex v of degree two may have an incident edge contracted if the neighborhood $N(v) \not\subseteq \partial$.

Proof. We first show that any "separable" information is not lost after doing either of the above operations.

Let v be a non-boundary end-vertex of W. Since v is not on the boundary of W, there does not exist an extension Z such that $W \cdot Z$ has a cycle containing v. (Vertex v always has degree one.) Thus, all end-vertices of W not on the boundary can not be included with the other witness vertices associated with the witness forest W in any minimal feedback vertex set of any extended G_m.

Now assume v is a non-boundary vertex of degree two of W and $N(v) = \{a, b\}$ where $a \notin \partial$. Let Z be an extension of W such that the removal of vertex v kills some cycles of $W \cdot Z$. Since the degree of v is two, all cycles through v must also pass through a. Thus, vertex a is also a kill vertex for the cycles killed by v in $W \cdot Z$. This shows that we may replace vertex v with vertex a in any feedback vertex set containing v. (Vertex v always has degree two.)

Let W_c be the forest W with edge (a, v) contracted. The vertices v and a of W are replaced with the vertex labeled a in W_c. We now show $W_c \equiv_w W$. For all extensions Z, there is a bijection between cycles in $W_c \cdot Z$ and cycles in $W \cdot Z$. (All cycles that pass through v of $W \cdot Z$ now pass through a cycle with one less edge in $W_c \cdot Z$; all other cycles are identical.) For any cycle killed by vertex v in $W \cdot Z$, the corresponding cycle in $W_c \cdot Z$ is still killed by vertex a. The other vertices of W_c or W still kill the same cycle extensions. Thus $W_c \leq_w W$ and $W \leq_w W_c$.

We now show that the reduced park P derived from W using steps (a) and (b) is minimal.

Let vertex v be a non-boundary vertex of P. Since P is acyclic and contains no end-vertices adjacent to the boundary, vertex v is on some unique path between two boundary vertices i and j. Deleting v disconnects i and j. So $(P \setminus \{v\}) \not\leq_w P$.

Now let P' be the forest P where edge (a, b) is contracted for two non-boundary vertices a and b of degree three or more. Let a_1 and a_2 be two (distinct) boundary vertices connected to vertex a such that vertex b is not on the connecting paths. Likewise, Let b_1 and b_2 be two boundary vertices connected to vertex b such that vertex a is not on the connecting paths. The vertices a_1 and a_2 are distinct from the vertices b_1 and b_2, for otherwise a cycle would contain edge (a, b) in P. Pick a graph extension Z to be the set of boundary vertices $\bar{S}$ with the edges (a_1, a_2) and (b_1, b_2). The graph $(P' \setminus \{a\}) \cdot Z$ is acyclic while the graph $P \cdot Z$ contains two disjoint cycles. This tells us that $(P \setminus \{x\}) \cdot Z$ is cyclic for all x in $P \setminus \partial$. Thus, $P \not\leq_w P'$.

Finally assume P' is the forest P where edge (a, b) is contracted, $a \in \partial$, $b \notin \partial$, and $degree(b) \geq 3$. Let b_1 and b_2 be two boundary vertices connected to vertex b such that the path between b_1 and b_2 passes through vertex b and $a \notin \{b_1, b_2\}$. Pick an extension Z to be the set of boundary vertices $\bar{S}$ with the edges (a, b_1) and (a, b_2). The graph $(P \setminus \{b\}) \cdot Z$ is acyclic. The graph $P' \cdot Z$ contains two cycles which intersects at a. Since the boundary vertex a is not allowed to be deleted, the graph $(P' \setminus \{x\}) \cdot Z$ is cyclic for all x in $P' \setminus \partial$. Thus, $P' \not\leq_w P$. □

There may exist alternative witness forests that preserve minimum-sized feedback vertex sets for all extensions of G_m. A witness forest W is considered to be a park if the above lemma can not be applied to W.

Lemma 115: There are at most $3t-3$ vertices in any park for boundary size t.

Proof. First we consider the degree two non-boundary vertices. For such a vertex v, each of its neighbors must be a boundary vertex. After viewing v and its two incident edges as a single edge between two boundary vertices, we see that at most $t-1$ such vertices can occur. Otherwise, a cycle would exist on the boundary.

Now we consider the remaining non-boundary vertices. Let p be the number of such vertices and e be the edge size of the subpark. Using the fact that the size of a forest must be strictly less than the order, we have $e < t + p$. Since the sum of the vertex degrees is twice the size, we also have $t + 3 \cdot p \leq 2 \cdot e$. Combining these inequalities while solving for p we get

$$\frac{t+3\cdot p}{2} \leq e \leq t+p-1, \text{ or } p \leq t-2.$$

Summing up the boundary (t), the degree two vertices ($t-1$), and the degree three or more vertices ($t-2$), shows that the order of any park can be at most $3t-3$.

□

Corollary 116: There is a finite number of parks with boundary size t.

Proof. Since we have a bound on the number of vertices for a park, we can apply Cayley's Tree Formula (i.e., by counting the number of labeled trees/forests) to get a bound on the total number of distinct parks. There are n^{n-2} labeled trees of order n. □

The result of the previous lemma may be strengthened to arrive at a tighter bound on the number of possible parks. See, for example, the closely related Lemma 120 on page 143. However, this bound is sufficient for our purposes, that is, to see that there is a manageable (constant) number of parks. This means that the following algorithm can be used as an usable finite-index congruence.

For each subset S (with complement $\bar{S} = \partial \setminus S$) of the set of boundary vertices our algorithm keeps track of the related parks in the following sets.

$$P_m(S) = \{P \mid P \text{ is a park of } G_m \text{ with leaves and branches over } \bar{S}\}$$

Now we finally present a linear time dynamic-programming algorithm for the FVS problem which is used as our finite $\mathcal{F}$-congruence for t-parses. This general-purpose algorithm has the same structure as the vertex cover algorithm used in Chapter 6, indicating a standard approach for developing them. The one-pass algorithm simply makes a transition from one state to another for each operator of a t-parse $G_n = [\textcircled{0}, \ldots, \textcircled{t}, g_1, \ldots, g_n]$. Thus, after all the parks $\{P_m(S) \mid S \subseteq \partial\}$ are determined (for G_m), all the parks $\{P_i(S) \mid S \subseteq \partial\}$ for $i < m \leq n$ are never referenced and may be discarded.

Our algorithm, given in Figure 7.1, starts by setting the sizes for the minimal feedback vertex sets on $G_m = G_1$, the edgeless graph with $t+1$ boundary vertices. This is done for all $S \subseteq \partial$. There is only one park associated with $F_1(S)$ at this stage, namely the isolated forest with $t + 1 - |S|$ vertices. We break up the dynamic step into cases depending on what type of operator is at position $m+1$ and the condition (selected in S or not) of any affected boundary vertices of G_m or G_{m+1}. These transitions are described in cases 1-4 of Figure 7.1. When the algorithm reaches the end of the t-parse, it computes the minimum number of vertices needed in any feedback vertex set for G_n by taking the least $F_n(S)$.

For space reasons we leave out the self-evident rules required to update the sets of parks $P_i(S)$ throughout each iteration of step II of the FVS algorithm. This procedure essentially entails extending the parks with the current operator and reducing them by the rules given in Lemma 114 and combining park sets if the two $F_m()$'s are equal in cases 1-2.

Example 117: The following Table 7.1 of values for $F_m(S)$ shows the application of the FVS algorithm with the 2-parse given in Example 25 on page 40. As can been seen by examining the graph in Example 25, a minimum feedback vertex set has cardinality 2 which corresponds to the minimum value in the last column.

Theorem 118: For any t-parse $G_n = [\textcircled{0}, \ldots, \textcircled{t}, g_1, \ldots, g_n]$, the algorithm in Figure 7.1 correctly computes $FVS(G_n)$.

Proof. For part I of the algorithm, we note that $|S|$ vertices are selected from the boundary of G_1 for each $S \subseteq \partial$. Thus the minimum feedback vertex set for such a requirement is initially set, that is, $F_1(S) = |S|$.

I For $m = 1$ and every $S \subseteq \partial$ set

$$F_1(S) = |S| \quad .$$

II For $1 < m < n$ do the following cases:

Case 1: vertex operator $\textcircled{i}$ and $i \notin S$

$$F_{m+1}(S) = \min\{F_m(S), F_m(S \cup \{i\})\}$$

Case 2: vertex operator $\textcircled{i}$ and $i \in S$

$$F_{m+1}(S) = \min\{F_m(S), F_m(S \setminus \{i\})\} + 1$$

Case 3: edge operator $\boxed{i\ j}$ where $i \in S$ or $j \in S$

$$F_{m+1}(S) = F_m(S)$$

Case 4: edge operator $\boxed{i\ j}$ where $i \notin S$ and $j \notin S$

a) If the edge operator creates a cycle on $\bar{S}$ in G_{m+1} or $F_m(S) = \infty$ then

$$F_{m+1}(S) = \infty \quad .$$

b) If there exists a park in $P_m(S)$ such that i and j are in different trees then

$$F_{m+1}(S) = F_m(S)$$

else

$$F_{m+1}(S) = F_m(S) + 1 \quad .$$

III The minimum feedback vertex set order of G is

$$\min\{F_n(S) \mid S \subseteq \partial\} \quad .$$

Figure 7.1: A general feedback vertex set algorithm for t-parses.

Table 7.1: Feedback vertex set state tables computed for Example 117.

S	m / g_m	3 –	4 [0 1]	5 [1 2]	6 (1)	7 [0 1]	8 [1 2]	9 (1)	10 [0 1]	11 [1 2]	12 (0)	13 [0 1]	14 [0 2]	15 (2)	16 [0 2]	17 [1 2]
$\emptyset$		0	0	0	0	0	1	1	1	2	1	1	∞	1	1	∞
{0}		1	1	1	1	1	1	1	1	1	2	2	2	2	2	**2**
{1}		1	1	1	1	1	1	2	2	2	2	2	2	2	2	**2**
{2}		1	1	1	1	1	1	1	1	1	1	1	1	2	2	**2**
{0,1}		2	2	2	2	2	2	2	2	2	3	3	3	3	3	3
{0,2}		2	2	2	2	2	2	2	2	2	2	2	2	3	3	3
{1,2}		2	2	2	2	2	2	2	2	2	2	2	2	3	3	3
{0,1,2}		3	3	3	3	3	3	3	3	3	3	3	3	4	4	4

For a vertex operator $\textcircled{i}$ appended to G_m we relabel vertex i of G_m as i' and label the new vertex in G_{m+1} as i. The correctness for the dynamic step of the algorithm (part II) is now be proved.

Case 1: $F_{m+1}(S) = \min\{F_m(S), F_m(S \cup \{i\})\}$

First, for the $(m+1)$-th operator being $\textcircled{i}$ where $i \notin S$, we show that $F_{m+1}(S) \leq \min\{F_m(S), F_m(S \cup \{i\})\}$. Let K be a witness feedback vertex set of G_m where $S = K \cap \partial$ or $S \cup \{i\} = K \cap \partial$. The graph G_{m+1} resulting from adding an isolated vertex to G_m with new boundary vertex i also has K as a feedback vertex set but with $\partial \cap K = S$.

Now we show that $\min\{F_m(S), F_m(S \cup \{i\})\} \leq F_{m+1}(S)$. Assume K is a minimal feedback vertex set of G_{m+1}. Vertex i is not in K since removing i from K would leave a smaller feedback vertex set for G_{m+1}, contradicting K being minimal. If $i' \in K$ then K is a witness for G_m where $\partial \cap K = S \cup \{i\}$. Likewise, if $i' \notin K$ then K is a witness for G_m where $\partial \cap K = S$. This then shows that either $F_m(S) \leq F_{m+1}(S)$ or $F_m(S \cup \{i\}) \leq F_{m+1}(S)$.

Case 2: $F_{m+1}(S) = \min\{F_m(S), F_m(S \setminus \{i\})\} + 1$

This case assumes that the next operator is a vertex operator $\textcircled{i}$ and $i \in S$. We first show $F_{m+1}(S) \leq \min\{F_m(S), F_m(S - \{i\})\} + 1$. Let K be a witness feedback vertex set of G_m where $S = K \cap \partial$ or $S \cup \{i\} = K \cap \partial$. Adding an isolated vertex to G_m with new boundary vertex i has $K' = K \cup \{i\}$ as a feedback vertex set for G_{m+1} with $\partial \cap K' = S$. Thus, the inequality holds this way.

Now assume that K is a witness feedback vertex set for the graph G_{m+1} and

$$|K| - 1 < \min\{F_m(S), F_m(S \setminus \{i\})\} \quad .$$

If $i' \in K$ then $K' = K \setminus \{i\}$ is a witness for G_m where $\partial \cap K' = S$. Likewise, if $i' \notin K$ then $K' = K \setminus \{i\}$ is a witness for G_m where $\partial \cap K' = S \setminus \{i\}$. Thus, we either have $F_m(S) \leq F_{m+1}(S) - 1$ or $F_m(S \setminus \{i\}) \leq F_{m+1}(S) - 1$.

Case 3: $F_{m+1}(S) = F_m(S)$

The net result of adding an edge with operator $\boxed{i\ j}$ to G_m where either vertex i or vertex j is marked for deletion is the same as if this operator was not present. The edge gets deleted from the graph when the designated selected boundary S is a subset

of the minimal feedback vertex set, with respect to $F_m(S)$, of G_m. If $F_{m+1}(S) < F_m(S)$ then the witness feedback vertex set for G_{m+1} would also be a witness feedback vertex set for G_m (i.e., a contradiction of $F_m(S)$ being minimal).

Case 4: $F_{m+1}(S) = F_m(S)$ or $F_m(S) + 1$ or ∞

If the edge operator $\boxed{i\ j}$ creates a cycle on the non-selected boundary vertices $\bar{S} = \partial \setminus S$ or $F_m(S) = \infty$ then there is no feedback vertex set for G_{m+1}. Thus, $F_{m+1}(S)$ is correctly set to ∞.

We now consider the cases where $F_{m+1}(S)$ is finite. Clearly, $F_m(S) \leq F_{m+1}(S)$ since $G_m \setminus S$ is a proper subgraph of $G_{m+1} \setminus S$.

Assume there is a park for $F_m(S)$ such that boundary vertices i and j are in different trees. This means that there exists a feedback vertex set K of cardinality $F_m(S)$ of G_m with $\partial \cap K = S$ that disconnects the vertices i and j. Adding an edge (i,j) with operator $\boxed{i\ j}$ to $G_m \setminus K$ does not create any cycles. Thus, $G_{m+1} \setminus K = (G_m \cup \{(i,j)\}) \setminus K = (G_m \setminus K) \cup \{(i,j)\}$ is acyclic. In this case $F_{m+1}(S) = F_m(S)$.

If the above case is not true, then all parks have the boundary vertices i and j connected. Since $F_m(S) \neq \infty$, there must be at least one park (witness forest) associated with a minimal feedback vertex set K of G_m. Since operator $\boxed{i\ j}$ does not create a cycle on the non-selected boundary, the unique cycle created in $G_m \setminus K$ by adding the edge (i,j) has at least one non-boundary kill vertex v. Hence, the set $K \cup \{v\}$ is a feedback vertex set of G_{m+1}. We have shown $F_{m+1}(S) \leq F_m(S) + 1$.

We now consider the possibility that $F_{m+1}(S) = F_m(S)$ in this latter case. Let K be a witness for G_{m+1} of cardinality $F_m(S)$. The set K is also a feedback vertex set for G_m. Since K is a minimal feedback vertex set for G_m, the witness forest $G_m \setminus K$ must have vertices i and j connected (by assumption that no park disconnects i and j). However, adding the edge (i,j) to $G_m \setminus K$ causes a cycle. This is a contradiction since $G_{m+1} \setminus K = (G_m \cup \{(i,j)\} \setminus K = (G_m \setminus K) \cup \{(i,j)\}$. (Recall that i and j are not in S.) So $F_{m+1}(S) = F_m(S) + 1$. □

The dynamic program, given in Figure 7.1, for determining the feedback vertex set of a pathwidth t-parse is easily modified to handle treewidth t-parses. All that is needed is to add a case 5 in part II which takes care of the circle plus operator $G_i \oplus G_j$.

This new case is a little messy since the states for the two subtree parses G_i and G_j need to be interweaved. Briefly stated, this is done by checking all combinations (unions) of boundary subsets S_i and S_j of G_i and G_j (resulting in a subset S of $G_i \oplus G_j$) along with checking which best parks from G_i can be glued together with the compatible parks from G_j to form a set of parks for $G_i \oplus G_j$. If the glued parks create any cycles then the value of $F_{\text{“tree index”}}(S)$ needs to be increased to account for additional kill vertices.

Using the same technique that we did for building our finite-state algorithm for k–Vertex Cover, we can convert the above feedback vertex set algorithm to a finite-index congruence for k–Feedback Vertex Set. This is accomplished by restricting the values of $F_m(S)$ to be in $\{0, 1, \ldots, k, k+1\}$; we are only interested in knowing whether or not there exists a feedback vertex set of size at most k containing S. (The value of $k+1$ acts as the value ∞ in the congruence.)

In our application for finding the k–Feedback Vertex Set obstruction sets, we actually used a congruence with slightly fewer states then the one just described. The key idea to this improvement was noticing that if a park P is a minor of a park P' then only the representative P is needed as a witness. We estimate that this allowed us to prove approximately 5% more t-parses nonminimal via the dynamic-programming congruence check. That is, for certain instances we avoided our testset proof method.

7.2.2 A complete FVS testset

A finite testset for the feedback vertex set canonical congruence $\sim_{\mathcal{F}}$ is easy to produce. The individual tests closely resemble the parks described above. The testset that we use consists of forests augmented with isolated triangles (and/or triangles solely attached to a single boundary vertex). Our k–Feedback Vertex Set testset T_t^k consists of all t-boundaried graphs that have the following properties:

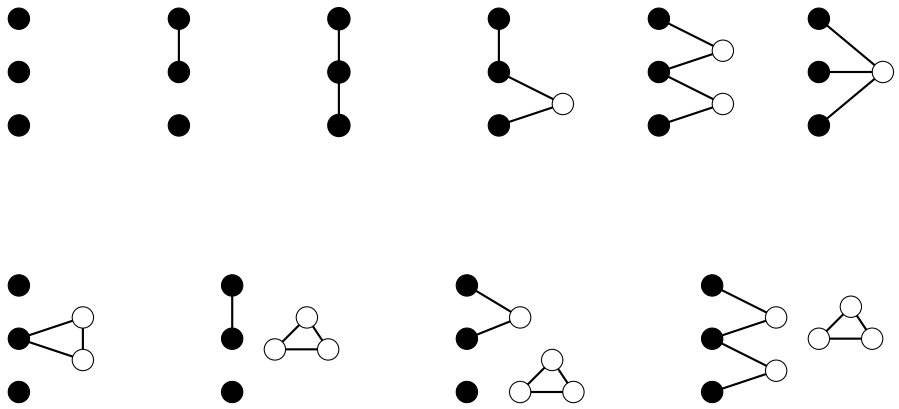
1. Each graph is a member of k–Feedback Vertex Set.
2. Each graph is a forest with zero or more isolated triangles, K_3's.
3. Every tree component has at least two boundary vertices.

4. Every isolated triangle has at most one boundary vertex.

5. Every degree one vertex is a boundary vertex.

6. Every non-boundary degree two vertex is adjacent to two boundary vertices.

Example 119: Some 3-boundaried tests (of $T_{t=3}^{k=1}$) for 1–FEEDBACK VERTEX SET are shown below.



The above restrictions on members of T_t^k gives an upper bound on the number of vertices, as stated in the following lemma. Hence T_t^k is a finite testset.

Lemma 120: The number of vertices for any test $T \in T_t^k$ is at most $3k + 2t - 1$.

Proof. Since $T \in k$–FEEDBACK VERTEX SET there can be at most k isolated triangles, consisting of at most $3k$ non-boundary vertices. We now show by induction that there can be at most $t - 1$ interior forest vertices for boundary size t. Without loss of generality, we may assume the acyclic part of T is a tree (i.e., we can add edges to make another test with the same order.) For a tree with 2 boundary vertices the largest test consists of one interior vertex of degree two. Thus the base case holds. Now assume T is a valid test with t boundary vertices. We consider three cases. If T has a degree one boundary vertex v that is adjacent to another boundary vertex, then $T \setminus \{v\}$ is a valid test for boundary size $t - 1$ containing, by induction, at most $t - 2$ interior vertices. Hence T also has at most $t - 2 < t - 1$ interior vertices. Otherwise, if T has a degree two interior vertex v then $T \setminus \{v\}$ partitions the boundary into two valid tests T_1 and T_2 each with positive boundary

sizes $b_1 + b_2 = t$. By induction, $|V(T_1)| - b_1 \leq b_1 - 1$ and $|V(T_2)| - b_2 \leq b_2 - 1$, so $|V(T)| - t \leq (b_1 - 1) + (b_2 - 1) + 1 = t - 1$. Lastly, if all of T's interior vertices have degree at least 3 then there must be at least twice the number of leaves (boundary vertices). Thus, any acyclic test T can have at most $t - 1$ interior vertices. □

The above bound is tight since the test T consisting of k isolated triangles and $t - 1$ interior degree two vertices, each adjacent to boundary vertex i and $i + 1$, has $3k + 2t - 1$ vertices (see, for example, the last test given in Example 119).

Since these k–FEEDBACK VERTEX SET testsets are based solely on t-boundaried graphs, they are useful for both pathwidth and treewidth t-parse obstruction set computations.

Theorem 121: The set of t-boundaried graphs T_t^k is a complete testset for the graph family k–FEEDBACK VERTEX SET.

Proof. Assume G and H are two t-boundaried graphs that are not $\mathcal{F}$-congruent for $\mathcal{F} = k$–FEEDBACK VERTEX SET. Without loss of generality, let Z be any t-boundaried graph that distinguishes G and H with $G \oplus Z \in \mathcal{F}$ and $H \oplus Z \notin \mathcal{F}$. We show how to build a t-boundaried graph $T \in T_t^k$ from Z that also distinguishes G and H. Let W be a set of k witness vertices such that $(G \oplus Z) \setminus W$ is acyclic. From W, let $W_G = W \cap G$, $W_\partial = W \cap \partial$ and $W_Z = W \cap Z$. Take T' to be $Z \setminus W$ plus $|W_Z|$ isolated triangles, plus $|W_\partial|$ triangles with each containing a single boundary vertex from W_∂. If T' contains any component $C \not\cong K_3$ without boundary vertices, replace it with $FVS(C)$ isolated triangles. Clearly, $G \oplus T' \in \mathcal{F}$ since W_G plus one vertex from each of the non-boundary isolated triangles of T' is a witness set of k vertices. If $H \oplus T' \in \mathcal{F}$ then this contradicts the fact that $H \oplus Z \notin \mathcal{F}$ by using a cover containing W_Z, W_∂ and the interior witness vertices of H (with respect to $H \oplus T'$). Finally, we construct a distinguisher $T \in T_t^k$ by minimizing T' to satisfy the 6 properties listed above. (Note that the extension T is created by not eliminating any cycles in the extension T'.) □

For the graph family 1–FEEDBACK VERTEX SET on boundary size 4, the above testset consists of only 546 tests. However, for 2–FEEDBACK VERTEX SET on boundary size 5, the above testset contains a whopping set of 14686 tests. As can be seen by

the increase in the number of tests, a more compact feedback vertex set testset would be needed (if possible) before we attempt to work with boundary sizes larger than 5. The large number of tests (especially T_5^2) for the FVS families indicates why using the testset step to prove t-parses minimal or nonminimal is the most CPU-intensive part of our obstruction set search (and is why it is attempted last).

7.3 The FVS Obstructions

Our search for the 1–Feedback Vertex Set and 2–Feedback Vertex Set obstructions is now presented. As mentioned in Chapter 4, we need some type of lemma that bounds the search space. The following well-known treewidth bound can be found in [vL90] along with other introductory information concerning the minor order and obstruction sets. We provide a proof in order to suggest how generous the bound probably is for the k–Feedback Vertex Set obstructions, which is a very small subset of the $(k+1)$–Feedback Vertex Set family.

Lemma 122: A graph in k–Feedback Vertex Set has treewidth at most $k+1$.

Proof. Let $G = (V, E)$ be a member of k–Feedback Vertex Set and $V' \subseteq V$ be a set of k witness vertices such that $G' = G \setminus V'$ is acyclic. The remaining forest G' has a tree decomposition T of width 1. Notice that a tree decomposition T' consisting of the vertex sets of T augmented as $T'_i = T_i \cup V'$ is a tree decomposition for G of width $k+1$. □

Corollary 123: An obstruction for k–Feedback Vertex Set has treewidth at most $k+2$.

Proof. Let G be an obstruction and v any vertex of G. By definition, $G' = G \setminus \{v\} \in$ k–Feedback Vertex Set. Since G' has a tree decomposition T of width at most $k+1$, adding the vertex v to each vertex set of T yields a tree decomposition of width at most $k+2$ for G. □

We now consider when the pathwidth of a k–Feedback Vertex Set obstruction G can be larger than the treewidth bound of $k+2$. If we attempt to build a path

decomposition like the tree decompositions in the proof of Lemma 7.3, we see that the forest G' (obtained by deleting an arbitrary vertex v and k witness vertices from G) has to have pathwidth at least 2. From [EST94] we know that the forest contains a subdivided $K_{1,3}$. So, such an obstruction must have at least $1+k+7$ vertices. For pathwidth 3 (see Chapter 12 or [EST94]), the forest has to contain one of the tree obstructions of order 22, and hence G has to have at least $1+k+22$ vertices for pathwidth to be more than the treewidth plus one.

Lemma 124: If O is an obstruction to 2–Feedback Vertex Set and has pathwidth greater than 4, then O either has at least 24 vertices or is also an obstruction to k–Pathwidth, for some $k \geq 4$.

Proof. Assume that the pathwidth of O is 5 and is not a pathwidth obstruction. (If the pathwidth is larger than 5 then we can get a larger vertex bound.) There must then exist a minor G of O with the same pathwidth as O. Since O is a 2–Feedback Vertex Set obstruction, the minor G must have a feedback vertex set V of cardinality 2. If the forest $G' = G \setminus V$ has pathwidth 2 or less, we can build a path decomposition of G of width 4 by adding the two vertices of V to the sets of a path decomposition of G' of width 2. So that leaves us with the case that G' must contain a tree of pathwidth at least 3. Such a tree must have at least 22 vertices so G must have at least 24 vertices. Since O has the same pathwidth as G, the obstruction O of 2–Feedback Vertex Set must also have 24 vertices. □

Any connected obstruction to 2–Feedback Vertex Set does not contain 3 disjoint cycles, or any degree one vertices, or any consecutive degree two vertices, so having 24 or more vertices seems unreasonable. Observe that the graph K_5 is an obstruction to both 2–Feedback Vertex Set and 3-Pathwidth (not pathwidth 4!), and that most of the k-Pathwidth obstructions have pendent vertices (and other nonminimal properties), so it is unlikely that the second case of the lemma is possible. Unfortunately at this time, we have not proven the impossibility of either of these two cases. We hope, with regards to 2–Feedback Vertex Set, that we can find a definitive proof, and avoid a treewidth 4 search.

Besides the single obstruction K_3 for the trivial family 0–Feedback Vertex Set, the connected obstructions for 1–Feedback Vertex Set and the connected obstructions for 2–Feedback Vertex Set (pathwidth ≤ 4) are shown in Figures 7.3–7.5. The two connected obstructions for 1–Feedback Vertex Set were found in about 3 hours of accumulated CPU time when combining 4 worker processes, a database manager process, and a dispatcher process running concurrently. Our pathwidth 4 search for 2–Feedback Vertex Set consumed over 40 thousand hours of CPU time running for about three months in duration while averaging 20 workers (from initially a collection of 15-30 SUN Sparcs, and more recently including a few IBM 6000s and two Cray Y-MPs).

Table 7.2 contains a brief summary of how many proofs our system had to find for 2–Feedback Vertex Set (pathwidth 4). The first column states various starting (or restarting) points in the search. Lack of memory and disk space is the main reason for the separate runs. The second column gives the number of canonic non-boundaried obstructions that have the given prefix. The 'minimal nodes' column gives the number of minimal t-parses that we encountered; these are the internal nodes of our search tree plus any boundaried obstructions. The last column gives the total number of graphs the system had to check. This total includes those t-parses that were proved minimal or nonminimal (or irrelevant). The missing entries in the table represent places that were fast dead-end runs (i.e., small subtrees of the search tree leading only to nonminimal t-parses) and we did not bother keeping the proofs.

We believe that 2–Feedback Vertex Set may be the only feasible family to characterize since there are at least 744 obstructions to 3–Feedback Vertex Set. In fact this count is a very small percentage since we know of an obstruction with order 15 and we have only searched through a subset of the graphs with maximum order 10.

In our display of the two obstruction sets for the "within one/two vertices of acyclic" families, we present only the connected obstructions since any disconnected obstruction O of the lower ideal k–Feedback Vertex Set is a union of graphs from $\bigcup_{i=0}^{k-1} \mathcal{O}(i$–Feedback Vertex Set) such that $\mathrm{FVS}(O) = k+1$. (See Section 4.4.1.)

Table 7.2: Summary of our 2–FEEDBACK VERTEX SET obstruction set computation, pathwidth 4.

Pathwidth Four Prefixes for Feedback Vertex Set 2	Obsts	Extract File	Minimal t-parses	Total proofs
[0,1,2,3,4,01,02,0,03,12]	x			
[0,1,2,3,4,01,02,0,01,13]	x			
[0,1,2,3,4,01,02,03,0,12]	0			
[0,1,2,3,4,01,02,03,12,14]	x			
[0,1,2,3,4,01,02,03,14,24]	x			
[0,1,2,3,4,01,02,03,12,13]	x			
[0,1,2,3,4,01,02,03,04,12]	1 (K5)	none	15	211
[0,1,2,3,4,01,02,03,0,04]	0	none		
[0,1,2,3,4,01,02,0,01,12]	0	A	150	2251
[0,1,2,3,4,01,02,0,03,04]	0	B	233	3271
[0,1,2,3,4,01,02,03,04,0]	10	4 (deg)	5177	74611
[0,1,2,3,4,01,02,03,0,01]	16	C	68634	1013641
[0,1,2,3,4,01,02,0,01,03]	13	D	153772	2286001
[0,1,2,3,4,01,02,0,01,02]	13	E	(prefixed below)	
Prefix(E) + [03,04,0]	9	E13a	105482	1565416
Prefix(E) + [03,04,12]	10	E13b	91976	1359376
Prefix(E) + [03,04,13]	0	E13c	5241	78436
Prefix(E) + [03,04,34]	0	E13d	509	7636
Prefix(E) + [03,12]	10	E12a	35976	532651
Prefix(E) + [03,13]	0	E12b	260	3886
Prefix(E) + [03,14]	0	E12c	45	676
Prefix(E) + [03,34]	0	E12d	41	616
Prefix(E) + [12] E11a	2	E11	10517	157231
Prefix(E) + [13] E11b	0	none	13	151

Example 125: Since K_3 is an obstruction for 0–FEEDBACK VERTEX SET, and K_4 is an obstruction for 1–FEEDBACK VERTEX SET, the graph $K_3 \cup K_4$ is an obstruction for 2–FEEDBACK VERTEX SET.

Some patterns become apparent in these two sets of obstructions such as the following easily-proven observation.

Observation 126: For the family k–FEEDBACK VERTEX SET, the complete graph K_{k+3}, the augmented complete graph $A(K_{k+2})$ which has vertices $\{1, 2, \ldots, k+2\} \cup \{v_{i,j} \mid 1 \leq i < j \leq k+2\}$ and edges

$$\{(i,j) \mid 1 \leq i < j \leq k+2\} \quad \cup$$

$$\{(i, v_{i,j}) \text{ and } (v_{i,j}, j) \mid 1 \leq i < j \leq k+2\} \quad ,$$

and the augmented cycle $A(C_{2k+1})$ are obstructions.

7.4 The FES Obstruction Set Computation

We now focus on two problem-specific areas for computing the k–FEEDBACK EDGE SET obstruction sets: a direct minimality test and a complete testset (i.e., steps 1 and 4 of Section 4.3).

7.4.1 A direct nonminimal FES test

We first describe a simple graph-theoretical characterization for the graphs that are within a few edges of acyclic. This trivial result also shows that Problem 112 (i.e., determining the minimum feedback edge set of a graph) has a linear time decision algorithm.

Theorem 127: A graph $G = (V, E)$ with c components has $FES(G) = k$ if and only if $|E| = |V| - c + k$.

Proof. For $k = 0$ the result follows from the standard result for characterizing forests. If $FES(G) = k$ then deleting the k witness edges produces an acyclic graph and thus $|E| = |V| - c + k$. Now consider a graph G with $|V| - c + k$ edges for some $k > 0$. Since G has more edges than a forest can have, there exists an edge e on a cycle. Let $G' = (V, E \setminus \{e\})$. By induction $FES(G') = k - 1$. Adding the edge e to a witness edge set E' for G' shows that $FES(G) = k$. □

Unlike the k–Feedback Vertex Set lower ideals, it is not obvious that the family k–Feedback Edge Set is a lower ideal in the minor order. However, with the above theorem one can easily prove this.

Corollary 128: For each $k \geq 0$, the family of graphs k–Feedback Edge Set is a lower ideal in the minor order.

Proof. We show that the three basic minor operations do not increase the number of edges required to remove all cycles of a graph. An isolated vertex deletion removes both a vertex and a component at the same time, so k is preserved in the formula $|E| = |V| - c + k$. For an edge deletion the number of components can increase by at most one, so with $|E|$ decreasing by one, the value of k does not increase. For an edge contraction, the number of vertices decreases by one, the number of edges decrease by at least one, and the number of components stays the same, so k does not increase. □

The above corollary allows us to characterize each k–Feedback Edge Set lower ideal in terms of obstruction sets. We abstractly characterize these below.

Theorem 129: A connected graph G is an obstruction for k–Feedback Edge Set if and only if $FES(G) = k + 1$ and every edge contraction of G removes at least two edges (i.e., the open neighborhoods of adjacent vertices overlap).

Proof. This follows from the fact that an edge contraction that does not remove at least two edges is the only basic minor operation that does not decrease the number of edges required to kill all cycles for a connected graph with every edge on some cycle. □

The above theorem gives us a precise means of testing for nonminimal t-parses (see step 1 of Section 4.3).

7.4.2 A complete FES testset

Somewhat surprisingly, a usable testset for each feedback edge set family has already been presented in Section 7.2.2. We now prove that those earlier feedback vertex set tests can also be used here.

Lemma 130: The testset T_t^k for the family k–FEEDBACK VERTEX SET is also a testset for k–FEEDBACK EDGE SET.

Proof. First observe that

$$\mathcal{F} = k\text{–FEEDBACK EDGE SET} \subseteq k\text{–FEEDBACK VERTEX SET}$$

so that the k–FEEDBACK VERTEX SET membership restriction for T_t^k graphs does not preclude any important tests (just includes some obsolete tests not in $\mathcal{F}$). Consider a fixed family $\mathcal{F}$ and boundary size t. It suffices to show that if $G \not\sim_{\mathcal{F}} H$ then there exists a test $T \in T_t^k$ that distinguishes G and H. Since G and H are not congruent there exists a t-boundaried graph Z such that, without loss of generality, $G \cdot Z \in \mathcal{F}$ and $H \cdot Z \notin \mathcal{F}$. We now show how to minimize Z into a $T \in T_t^k$. Let E be a witness edge set for $G \cdot Z \in \mathcal{F}$ and let $E_Z = E(Z) \setminus E$. The first transformation on Z is to set $Z' = (Z \setminus E_Z) \cup (|E_Z| \cdot K_3)$. Clearly Z' is also a distinguisher for G and H since (1) $G \cdot Z' \in \mathcal{F}$ by using the edges $E \setminus E_Z$ and one edge from each of the new K_3's as a witness set, and (2) $H \cdot Z' \notin \mathcal{F}$, for otherwise, $H \cdot Z$ would be in $\mathcal{F}$. Notice that Z' is a set of trees and isolated triangles. The final transformation on Z is to let Z'' be Z' with all non-boundary leaves deleted and non-boundary subdivided edges contracted to satisfy the conditions of a member of T_t^k. □

It is interesting to notice from the above proof that, in addition to the out-of-family tests, the isolated triangles in the tests for k–FEEDBACK EDGE SET do not contain any boundary vertices. Thus, the number of graphs in a testset for k–FEEDBACK EDGE SET is substantially smaller than the order of the testset for k–FEEDBACK VERTEX SET.

7.5 The FES Obstructions

Since the family k–FEEDBACK EDGE SET is contained in k–FEEDBACK VERTEX SET, the maximum treewidth of any obstruction for k–FEEDBACK EDGE SET is at most $k+2$. Thus, the same arguments given in Section 7.3 regarding pathwidth apply to k–FEEDBACK EDGE SET as well.

For the family 0–FEEDBACK EDGE SET, it is trivial to show that K_3 is the only obstruction. The connected obstructions for the graph families 1–FEEDBACK EDGE SET through 3–FEEDBACK EDGE SET are shown in Figures 7.2, 7.4 and 7.6. There are well over 100 connected obstructions for the 4–FEEDBACK EDGE SET family. Any disconnected obstruction for k–FEEDBACK EDGE SET is easily determined by combining connected obstructions for j–FEEDBACK EDGE SET, $j < k$, since $FES(G_1) + FES(G_2) = FES(G_1 \cup G_2)$.

An open problem is to determine a constructive method for finding all of the obstructions for k–FEEDBACK EDGE SET directly from $\mathcal{O}(j$–FEEDBACK EDGE SET), $j < k$. Some easily observed partial results are given next.

Observation 131: If G is a connected obstruction for k–FEEDBACK EDGE SET then the following are all connected obstructions for $(k+1)$–FEEDBACK EDGE SET.

1. G with an added subdivided edge attached to an edge of G.

2. G with an attached K_3 on one of the vertices of G.

3. G with an added edge (u, v) when there exists a path of length at least two between u and v in $G \setminus E$ for each feedback edge set E of $k+1$ vertices.

It is easy to see that if an obstruction has a vertex of degree two then it is predictable by observations 1–2. The first 2–FEEDBACK EDGE SET obstruction in Figure 7.4 (wheel W_3) and the second 3–FEEDBACK EDGE SET obstruction in Figure 7.6 (W_4) are two examples of graphs where observation 3 predicts the graph. Those 4–FEEDBACK EDGE SET and 5–FEEDBACK EDGE SET obstructions (pathwidth bound of 4) without degree two vertices and cut-vertices are shown in Figures 7.7 and 7.8. The third 4–FEEDBACK EDGE SET obstruction in Figure 7.7 is not predicatable from

the 3–Feedback Edge Set obstructions by using any of the above observations. Here deleting any edge from this obstruction leaves a contractable edge that does not remove any cycles, that is, all single edge deleted minors are "nonminimal" (see Theorem 129).

Figure 7.2: Connected obstructions for 1–Feedback Edge Set.

Figure 7.3: Connected obstructions for 1–Feedback Vertex Set.

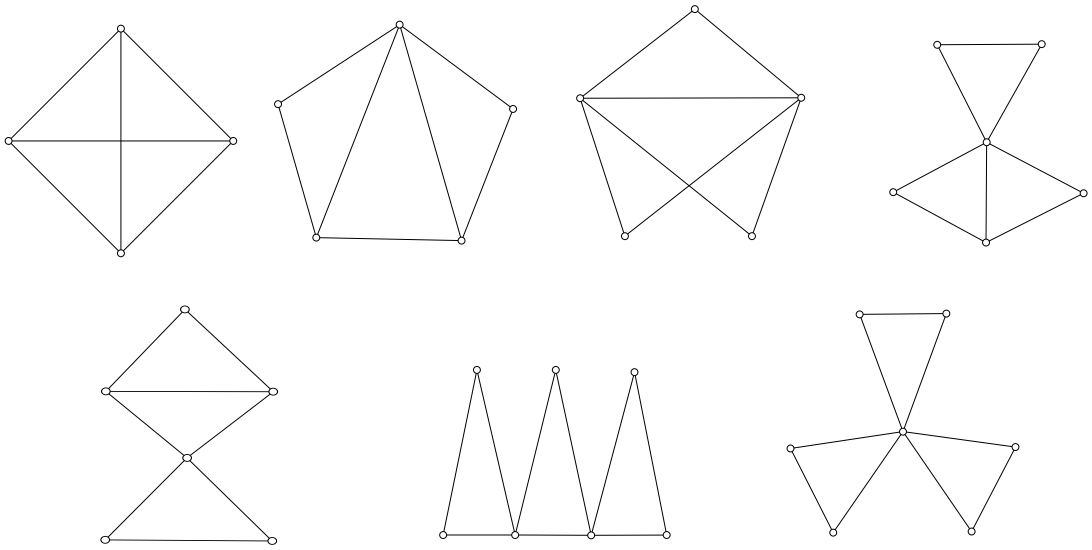


Figure 7.4: Connected obstructions for 2–Feedback Edge Set.

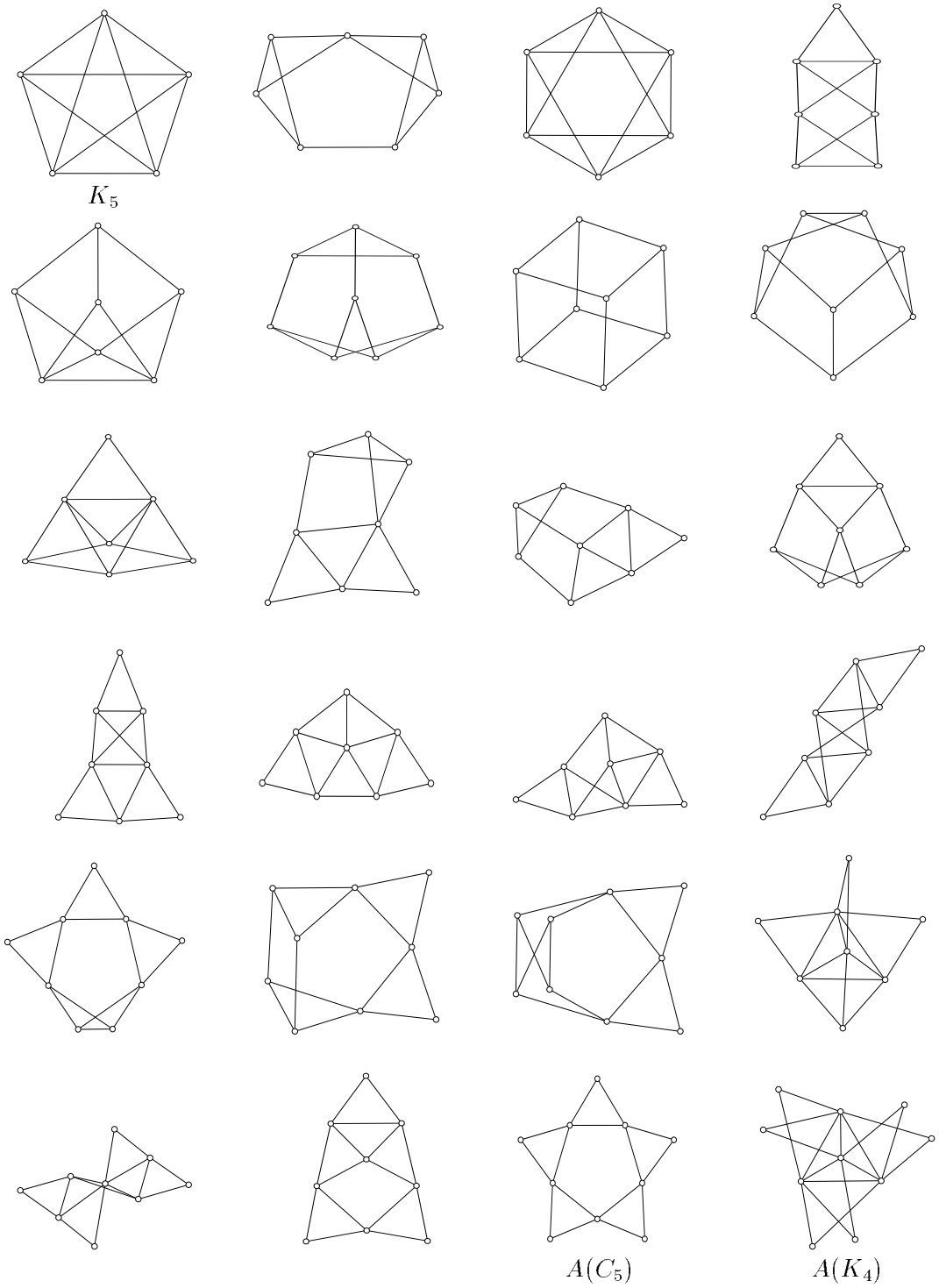


Figure 7.5: Connected obstructions for 2–Feedback Vertex Set, pathwidth ≤ 4.

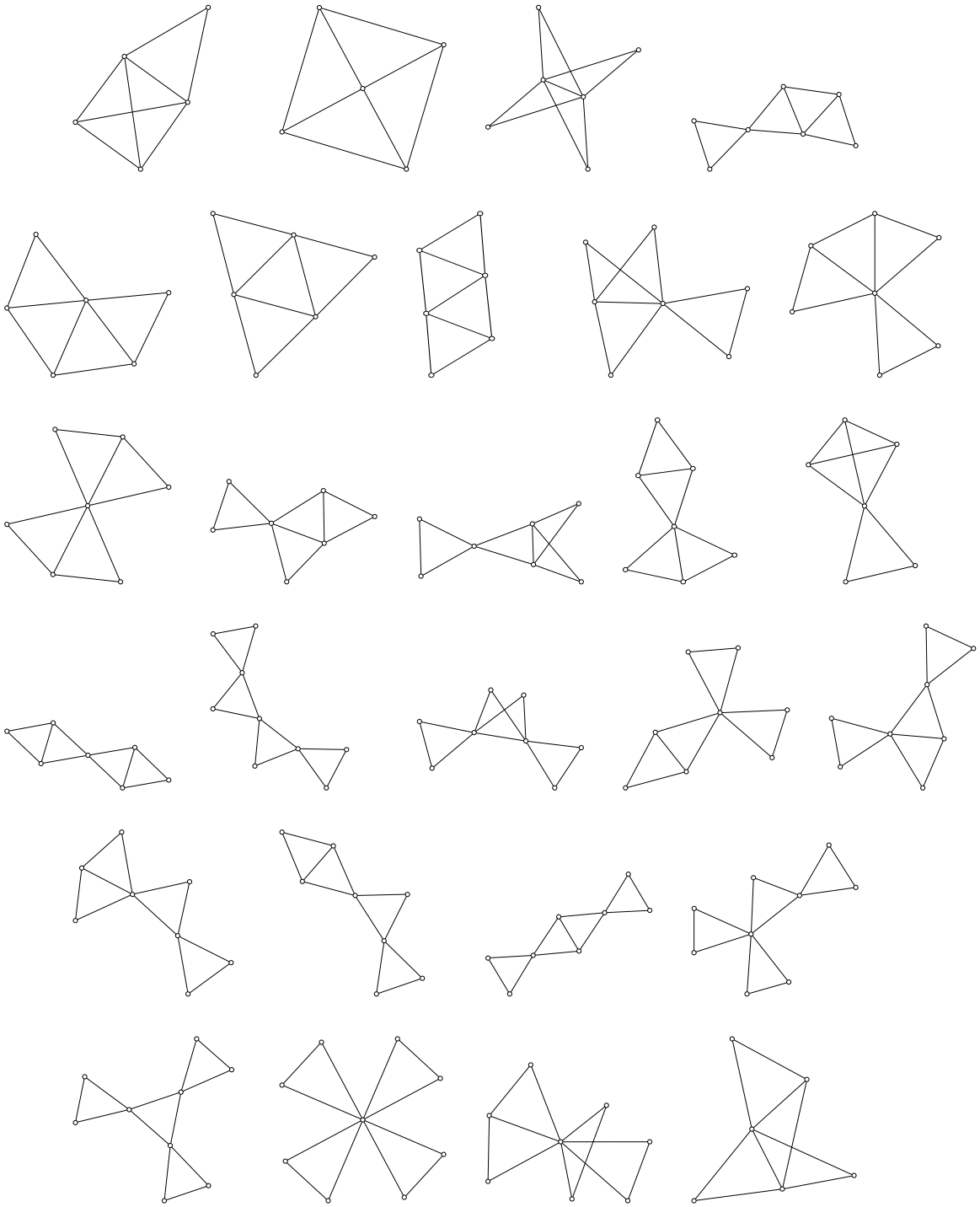


Figure 7.6: Known connected obstructions for 3–Feedback Edge Set.

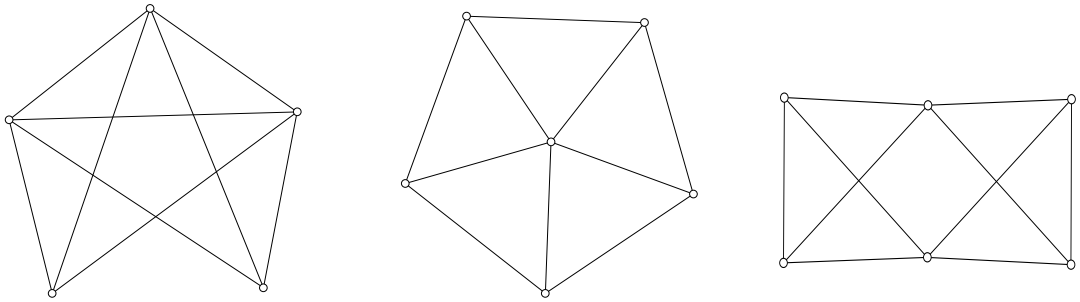


Figure 7.7: Biconnected 4–Feedback Edge Set obstructions without degree 2 vertices, pathwidth ≤ 4.

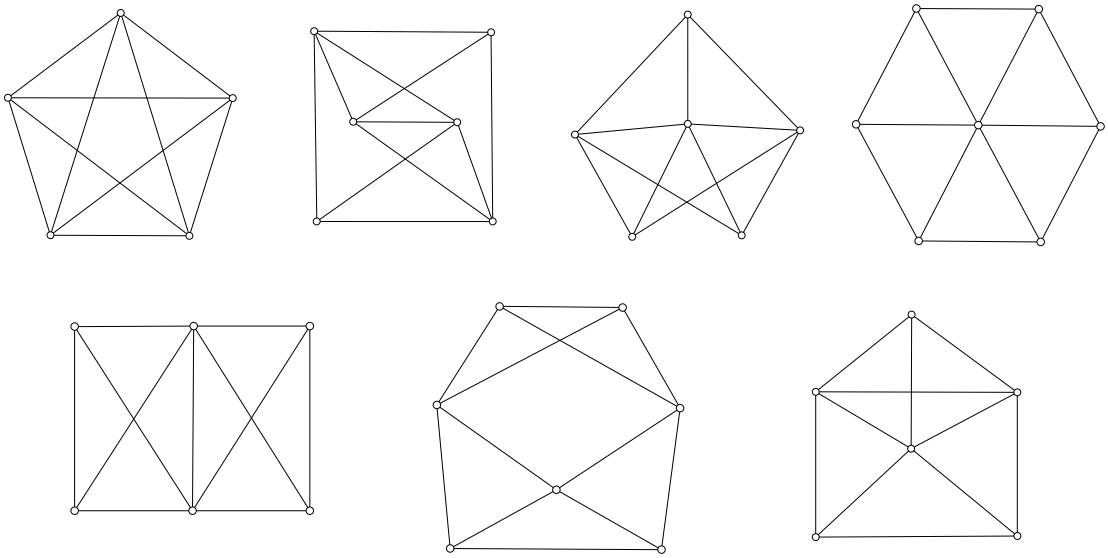


Figure 7.8: Biconnected 5–Feedback Edge Set obstructions without degree 2 vertices, pathwidth ≤ 4.

Chapter 8

Some Generalized VC and FVS Graph Families

This chapter develops problem-specific results leading to the characterization (in terms of obstruction sets) for several parameterized graph families. The previous two k–Vertex Cover and k–Feedback Vertex Set graph families are generalized into other graph-covering families that are lower ideals in the minor order. We also include efficient dynamic programs and computer-generated automata examples for various a path and cycle cover problems where the input has bounded combinatorial width.

8.1 Path Covers

We now generalize the k–Vertex Cover graph families (see Chapter 6). These path-cover families have applications in the design of minimal broadcast graphs via graph compounding (see [BFP95]).

Definition 132: For a graph G, let $MaxPath(G)$ be the length (number of edges) of the longest path. The base-level graph family is

$$0\text{–Path Cover}(p) = \{G \mid MaxPath(G) \leq p\}.$$

The graph family k–PATH COVER(p) consists of the graphs that are within k vertices of 0–PATH COVER(p), that is, a graph $G = (V, E)$ is in k–PATH COVER(p) if there exists a $V' \subseteq V$, $|V'| \leq k$, such that $MaxPath(G \setminus V') \leq p$.

Note that the parameterized family of graphs k–PATH COVER(0) is identical to the set of graphs in the k–VERTEX COVER family.

Lemma 133: The graph family k–PATH COVER(p) is a lower ideal in the minor order.

Proof. Let $G = (V, E)$ be a graph in k–PATH COVER(p) with $V' \subseteq V$ a covering. For any edge $e = (u, v) \in E$, let $G' = G \setminus \{e\}$. If either u or v is in V' then $G' \setminus V' = G \setminus V$ has a maximum path of length at most p. If not, $G' \setminus V' \subset G \setminus V$ and its maximum path length is less than or equal to p. Now consider an edge-contracted minor $G' = G/e$. If either u or v is in V' then $G' \setminus V' \subseteq G \setminus V$ has a maximum path of length at most p. If not, $G' \setminus V'$ is an edge-contracted minor of $G \setminus V$ and edge contractions do not increase the lengths of any paths. So $MaxPath(G' \setminus V') \leq p$. □

The following result follows from the treewidth result given in [FL89b] but is included and specialized here for completeness.

Theorem 134: The pathwidth of any graph is at most the height of its smallest depth-first-search (DFS) spanning tree.

Proof. Consider a DFS spanning tree T with root r of a graph G. Let $v_1, v_2, \ldots, v_m$ be the leaves of the tree T in visit order. Let V_i be the set of vertices from G on the path from r to v_i in T. We claim that $V_1, V_2, \ldots, V_m$ is a path decomposition of G. The width of this decomposition is the height of the tree T. Since a DFS spanning tree does not have any cross-edges, any edge $(u, v) \in G$ has both vertices in some V_i. Now consider $1 \leq i < k < j \leq m$. Suppose there is a vertex $v \in G$ such that $v \in (V_i \cap V_j)$. Vertex v is not a leaf of T since each leaf is in exactly one V_i. Also, for any non-leaf vertex v, $v \in V_i$ implies that v was visited before v_i in the DFS tree. Let $v_{i,j}$ be the nearest vertex to v_i in $V_i \cap V_j$ of T. Vertex v must be on the path from the root r to $v_{i,j}$ to be in both V_i and V_j. Note that $v_{i,k}$ comes after $v_{i,j}$ in the DFS visit order. This implies that the path $(r, \ldots, v_i, \ldots, v_{i,j})$ is a subpath of the path $(r, \ldots, v_{i,k})$. So $v \in V_k$ and $V_i \cap V_j \subseteq V_k$. □

Corollary 135: The maximum pathwidth of any graph G in k–PATH COVER(p) is $k + p$.

Proof. Let W be a set of witness vertices such that $MaxPath(G \setminus W) \leq p$. Every component C_i of $G \setminus W$ has a DFS spanning tree of height at most p. Thus, combining the vertex set W with the path decompositions of width at most p for each component C_i, shows that G has pathwidth at most $k + p$. □

Notice that the complete graph K_{k+p+1} has pathwidth $k + p$ and is a member of k–PATH COVER(p). However, we do not believe the next result is tight.

Corollary 136: If G is an obstruction for k–PATH COVER(p), then the pathwidth of G is at most $k + p + 1$

Proof. The minor $G' = G \setminus \{v\}$ for any vertex $v \in G$ is a member of k–PATH COVER(p). Thus G' has pathwidth at most $k + p$ and G has pathwidth at most $k + p + 1$. □

8.1.1 A path-cover congruence

We now consider the family 0–PATH COVER(p). A dynamic-programming congruence for t-parses is given below. Later we indicate how to handle the within k vertices cases.

Our finite-index congruence for 0–PATH COVER(p) is based on a general purpose $MaxPath()$ algorithm. The states of this dynamic program consists of "lengths" indexed by *path sequences* of the form:

$$(\ [I] \ b_1 \ (P|G) \ b_2 \ (P|G) \ b_3 \ \cdots \ b_{s-1} \ (P|G) \ b_s \ [I] \ )$$

where $b_1, b_2, \ldots, b_s$ are distinct boundary vertices, $0 \leq s \leq t + 1$.

We use square brackets to denote optional beginning and trailing I's. The notation $(P|G)$ means that either P or G is that letter of the sequence. The semantics of the variables I, P, and G are:

1. The variable I denotes a path to/from an interior vertex and the boundary.

2. The variable P denotes a path between two boundary vertices (this can be a boundary edge).

3. The variable G denotes a gap in the path (contributes 0 to the length).

Each length is set to -1 if no such path is possible, otherwise it is the maximum sum over all disjoint partial paths between an interior vertex and b_1, all b_i and b_{i+1}, and b_s and another interior vertex, where applicable. Note that the degenerate sequence $(I\ I)$ denotes, if positive, the length of a maximum path in the interior.

We also ignore path sequences that have the substring $(G\ b_i\ G)$ since replacing that substring with (G) represents the same type of "future" path (i.e., two gaps of length 0 add up to 0).

Lemma 137: For boundary size $t+1$ there are a finite number of path sequences.

Proof. A simple upper bound is obtained by counting the legal combinations of the variables. There are 4 ways that the variable I can appear (e.g., one or zero times at the front or end of the sequence). If there are $j > 0$ distinct boundary vertices, then there are 2^{j-1} possible ways the variables P and G are inserted between consecutive b_i and b_{i+1} (overcounting for consecutive G's). The remaining one boundary vertex case has $t+1$ possibilities. After combining these choices we have at most

$$4 \cdot \left(t+1+\sum_{j=1}^{t+1}(t+1)t(t-1)\cdots(t+2-j)\cdot 2^{j-1} \right) \leq 4\cdot(t+2)!\cdot 2^t$$

possible path sequences. □

We note that approximately one half of the path sequences counted in the previous lemma need to be kept since a sequence $(s_1\ s_2\ \cdots\ s_m)$ and its reverse $(s_m\ s_{m-1}\ \cdots\ s_1)$ represent the same type of path.

Example 138: The path sequence indices of interest (i.e., those path sequences

that determine a t-parse's state) for boundary size 2 are listed below:

$(I\ 0) = (I\ 0\ G\ 1)$	$(I\ 0\ I)$
$(I\ 1) = (I\ 1\ G\ 0)$	$(I\ 1\ I)$
$(I\ I)$	$(I\ 0\ P\ 1)$
$(0\ P\ 1)$	$(I\ 1\ P\ 0)$
$(0\ G\ 1)$ = '0'	$(I\ 0\ G\ 1\ I)$
	$(I\ 0\ P\ 1\ I)$

The above example illustrates that our bound given in Lemma 137 is not very tight (i.e., we need only 10 path sequences verses the 48 predicted by the lemma).

Theorem 139: There exists a linear time dynamic program that determines the maximum path length of a t-parse G_n.

Proof. It suffices to show how to correctly update the above path sequence state table (in constant time) for both vertex and edge operators. The maximum length over all non-'G' path sequence indices is the maximum path of the t-parse. Initially, for the empty prefix t-parse $G_0 = [\textcircled{0}, \textcircled{1}, \ldots, \textcircled{t}]$ all sequence lengths are set to -1 except that the following lengths have value 0:

$$(b_1\ G\ b_2) \text{ and } (b_1) \text{ where } b_1, b_2 \in \partial \text{ and } b_1 \neq b_2 \quad .$$

Case 1: edge operator $\boxed{i\ j}$

For this simple case, we just need to update only those indices for which adding an edge between boundary vertex i and j can create a longer path. That is, if the length m of a path sequence $(\cdots\ i\ G\ j\ \cdots)$ is not -1 then the length of $(\cdots\ i\ P\ j\ \cdots)$ is at least $m+1$. Here, we set the length for this sequence to the maximum of its previous value and $m+1$. By definition, no other path sequence needs to be increased. For example, the length for the trivial case $(i\ P\ j)$, where i and j are disconnected boundary vertices, would be set to 1 because $(i\ G\ j)$ always has length 0.

Case 2: vertex operator $\textcircled{i}$

For this case, we need to incorporate maximum-length paths that go through the boundary at vertex i into paths that go through the interior. We say that a

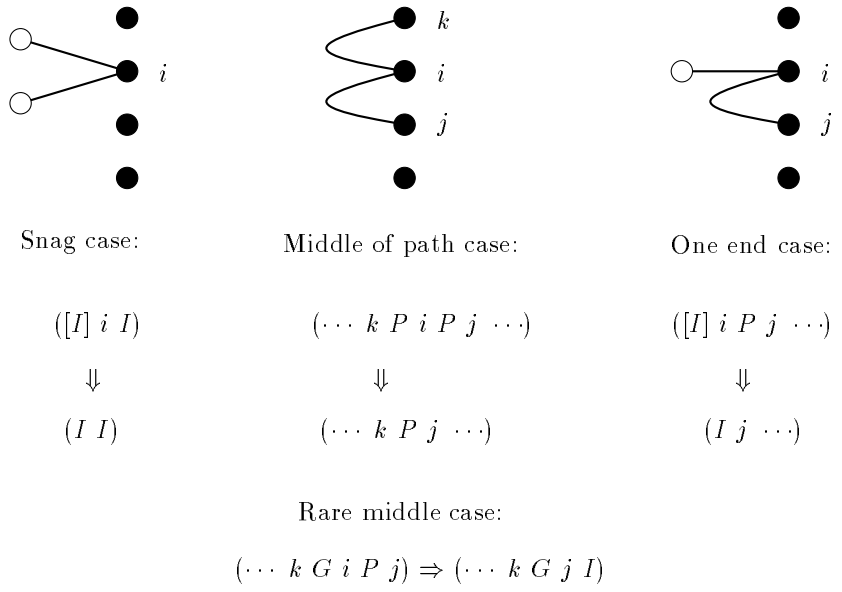


Figure 8.1: The vertex operator subcases for the proof of Theorem 139.

new boundary vertex i' replaces the old boundary vertex i. The possible cases to consider (the symmetric representative cases) are displayed in Figure 8.1. A "picture aid" of what path sequences to possibly alter is also given in the figure. Since none of the physical paths of the prefix t-parse G_m are increased in G_{m+1}, our dynamic program simply recategorizes some of the path sequence lengths. Again, like the edge operator case, the maximum of the previous sequence length and the length of the new candidate path is used to determine a path sequence's new length. The old path sequence containing boundary vertex i (on the path) may be assigned the value -1 since vertex i is now isolated. The path sequences with a gap G between i and another boundary vertex j need not be considered in this update process. For example during the move of boundary vertex i to the interior, the path sequence $(I\ i\ G\ j\ \cdots)$ represent a path for $(j\ \cdots)$, but $(j\ \cdots)$ already represents another of path with greater or equal length.

For both the above two operator cases the update process from G_m to G_{m+1} is done in time bounded by the constant number of path sequences. Since this happens only n times for the t-parse G_n, the overall running time is linear. □

Corollary 140: For a maximum path length p (i.e., for the family 0–Path Cover(p)),

our $MaxPath()$ algorithm can be used as a finite-index congruence.

Proof. This is accomplished by bounding the lengths of each path sequence to $p+1$. Since the maximum length over all of the path sequences never decreases, any extended t-parse stays out of the 0–PATH COVER(p) family. If x is the total number of path sequences, the number of equivalence classes of this congruence is bounded by $(p+3)^x$. □

We can convert the finite-state algorithm given in the proof of Corollary 140 to recognize graphs that are within k vertices of having p as the maximum path length (i.e., a k–PATH COVER(p) dynamic-programming congruence). This is accomplished by using the same technique that we used for our finite-state k–VERTEX COVER and k–FEEDBACK VERTEX SET algorithms. That is, we break the two operator cases $\textcircled{i}$ and $\boxed{i\ j}$ down into subcases that depend on conditions for each possible kill set $S \subseteq \partial$ (and keep witnesses for each of the possible killed vertices in the interior).

From the previous dynamic program, we also get the following simple corollary.

Corollary 141: For graphs of bounded pathwidth, (i.e., t-parses), the Hamiltonian path problem can be solved in linear time.

Proof. We just run the linear time $MaxPath()$ algorithm on a t-parse G and check if the answer equals $|G|$. □

8.2 Cycle Covers

We now generalize the k–FEEDBACK VERTEX SET graph families (see Chapter 7.)

Definition 142: For a graph G, let $MaxCycle(G)$ be the length of the longest cycle. The base-level graph family is

$$0\text{–CYCLE COVER}(l) = \{G \mid MaxCycle(G) \leq l\}.$$

The graph family k–CYCLE COVER(l) consists of the graphs that are within k vertices of 0–CYCLE COVER(l), that is, a graph $G = (V, E)$ is in k–CYCLE COVER(p) if there exists a $V' \subseteq V$, $|V'| \leq k$, such that $MaxCycle(G \setminus V') \leq l$.

The cycle cover family k–CYCLE COVER(2) is identical to the set of graphs in k–FEEDBACK VERTEX SET. The two degenerate families k–CYCLE COVER(0) and k–CYCLE COVER(1) are classified as follows: the first family

$$k\text{–CYCLE COVER}(0) = \{G \mid G \text{ has at most } k \text{ vertices}\}$$

has the single obstruction $(k+1) \cdot K_1$ consisting of isolated vertices, and the second family

$$k\text{–CYCLE COVER}(1) = k\text{–VERTEX COVER} = k\text{–PATH COVER}(0)$$

has already been seen.

Lemma 143: The graph family k–CYCLE COVER(l) is a lower ideal in the minor order.

Proof. This proof is similar to the proof given for Lemma 133 except that we observe that taking either subgraphs or edge contractions of a graph do not increase the length of any cycle. □

The next result relating treewidth and the maximum cycle length was first proved by Fellows and Langston [FL89b]. They show that if a graph has maximum cycle of length k, no back edge in any DFS tree can go back more than $k-1$ levels. One then reads off a tree decomposition of width $k-1$ from such a DFS tree. Our proof below is recursive and is based on separator sets and illustrates the fact that each biconnected component of a graph with a maximum cycle of length k can only have another disjoint cycle of length at most $k-4$.

Theorem 144: For any graph G, $treewidth(G) < MaxCycle(G)$. The degenerate cases are $MaxCycle(K_1) = 1$, and for any forest F with at least one edge, $MaxCycle(F) = 2$.

Proof. We first note that the theorem holds for the degenerate cases since clearly K_1 has treewidth 0 and any forest has treewidth at most 1. We prove the other cases by induction on the maximum cycle length. For a base case, consider a graph G

with $MaxCycle(G) = 3$. Since G does not contain any cycles of length 4, G does not contain either K_4 or $K_{2,3}$ as a minor. Thus, G must be outer-planar. From Lemma 156, we know that G has treewidth at most 2 and so the base case holds.

Now consider a graph G with $MaxCycle(G) = k$, $k > 3$. Without loss of generality, we can assume G is biconnected. (If G is not biconnected then a set of minimum tree decompositions $T_1, T_2, \ldots, T_m$ of the maximal biconnected subgraphs $S_1, S_2, \ldots, S_m$ of G can be pasted together to form a tree decomposition T of width equal to the maximum width of any T_i.) Let C be a maximum cycle in G and $C_1, C_2, \ldots, C_m$, $m \geq 1$ be the connected components of $G \setminus C$. If $m = 0$ then the treewidth/pathwidth of G is at most $|C| - 1$. Let C_i' be the vertex induced subgraph of G with vertices in the closed neighborhood of C_i, $N[V(C_i)]$. Also let $p_i \geq 1$ be the number of vertices in C adjacent to C_i, that is, $p_i = |V(C) \cap V(C_i')|$. We build a tree decomposition of G with root vertex $T_r = C$ combined with child tree decompositions from the components C_i' and show that the resulting tree decomposition has width at most $k - 1$.

We first take care of any component C_i with just one vertex v. The graph C_i', which consists of vertex v and its neighbors on C, can have at most $\lfloor k/2 \rfloor + 1$ vertices or otherwise C would not be a maximum cycle in G. Thus, for any such component C_i we can attach a vertex set consisting of $V(C_i')$ to T_r.

To make the rest of the proof more understandable, we first investigate the case where $p_i = 2$ and then later generalize for each $p_i > 2$. Let v_1 and v_2 denote the two vertices in $C \cap C_i'$. Consider a maximum cycle D, $|D| \geq 2$, in C_i. Since G is biconnected, there are two vertex disjoint paths $P_1 = v_1, \ldots, d_1$ and $P_2 = d_2, \ldots, v_2$ between C and D, $d_1 \neq d_2$ on D as shown in Figure 8.2. We claim that $|D| = k' \leq k - 4$. Let C_p be the longest path between v_2 and v_1 in C, D_p be the longest path between d_1 and d_2 in D, and E be the cycle (P_1, D_p, P_2, C_p). We have the following inequality

$$k \geq |E| = |P_1| + |P_2| + \left\lceil \frac{k}{2} \right\rceil + \left\lceil \frac{k'}{2} \right\rceil \geq 2 + \left\lceil \frac{k + k'}{2} \right\rceil = \left\lceil \frac{k + k' + 4}{2} \right\rceil \quad .$$

So the length k' of the maximum cycle D in C_i is at most $k-4$. By applying induction, we have a tree decomposition T_i of width at most $k - 5$ for C_i. If we add the $p_i = 2$ vertices v_1 and v_2 to each vertex set in T_i, we get a tree decomposition T_i' of width at

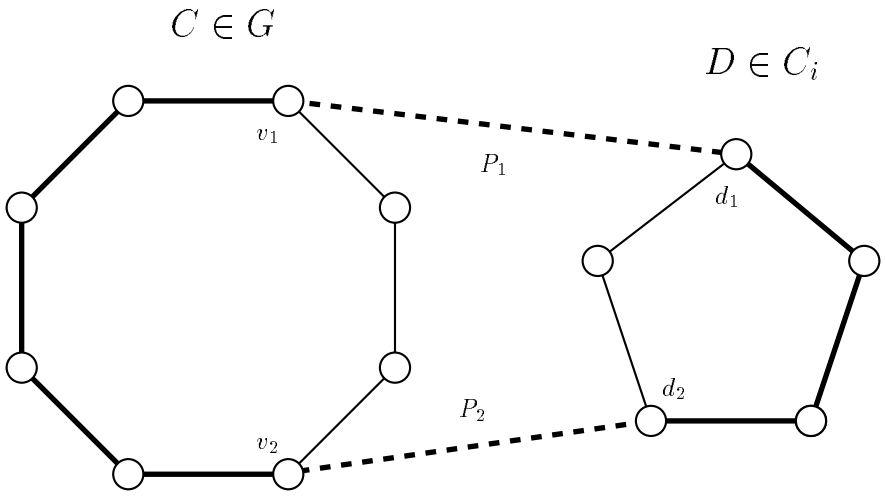


Figure 8.2: Illustrating the $p_i = 2$ case for the proof of Theorem 144.

most $k-3$ for C_i'. This subtree decomposition T_i' can be attached by any connecting edge to the separating vertex set T_r.

Finally we consider the generalized case, $2 < p_i \leq \lfloor k/2 \rfloor$, with $|C_i| > 1$. Let $v_1, v_2, \ldots, v_{p_i}$ be the vertices in $W = C \cap C_i'$. Consider a maximum cycle D, $|D| \geq 2$, in C_i. We partition the vertices in W into equivalence classes defined by $v_i \equiv v_j$ if and only if there does not exists disjoint paths $P_i = v_i, \ldots, d_i$ and $P_j = v_j, \ldots, d_j$ between C and D, $d_i \neq d_2$ on D. Since G is biconnected, there are at least two equivalence classes. We use the same construction as in the $p_i = 2$ case given above and show that $|D| = k' \leq k - p_i$. Assume v_i is a member of an equivalence class with $m < p_i$ members. Since any two vertices v_i and v_j in W can not be adjacent on C, the m vertices in v_i's equivalence class must preclude at least $2m+1$ positions on C for the locations of the remaining $p_i - m$ vertices in W. Likewise, the other equivalence classes collectively prevent at least $2p_i - 2m + 1$ positions on C. This means that there exists two non-equivalent vertices v_i and v_j that have distance at most $\lfloor (k-2(p_i-m-1)-2(m-1))/2 \rfloor$ on C. In other words, v_i and v_j are connected by a path of length at least $k - (k - 2p_i + 4)/2 = (k + 2p_i - 4)/2$ on C. Let C_p be this longest path between v_j and v_i in C, D_p be the longest path between d_i and d_j in D, and E be the cycle (P_i, D_p, P_j, C_p). We now have the following inequality

$$k \geq |E| \quad \geq \quad |P_1| + |P_2| + (k + 2p_i - 4)/2 + k'/2$$

$$\begin{aligned} &\geq 2 + (k + 2p_i - 4 + k')/2 \\ &= (k + k' + 2p_i)/2 \quad . \end{aligned}$$

So, as desired, the length k' of the maximum cycle D in C_i is at most $k - 2p_i$ (which is less than or equal to $k - p_i$). By applying induction, we have a tree decomposition T_i of width at most $k - 2p_i - 1$ for C_i. If we add the p_i vertices W to each vertex set in T_i, we get a tree decomposition T_i' of width at most $k - p_i - 1$ for C_i'. This sub-tree-decomposition T_i' can be attached, as we did for the $p_i = 2$ case, to the separating vertex set T_r. □

Corollary 145: The maximum treewidth of any graph G in k–Cycle Cover(l) is $k + l - 1$.

Proof. Let W be a set of witness vertices such that $MaxCycle(G \setminus W) \leq l$. From the above theorem, every component C_i of $G \setminus W$ has treewidth at most $l - 1$. Thus combining the vertex set W with the tree decompositions of each C_i and adding arbitrary edges in a tree-like fashion between the sub-decompositions shows that G has treewidth at most $k + l - 1$. □

Again, as we saw for the k–Path Cover(p) families, the complete graph K_{k+l} in k–Cycle Cover(l) has treewidth $k + l - 1$. We suspect that the next result can be improved.

Corollary 146: If G is an obstruction for k–Cycle Cover(l), then the treewidth of G is at most $k + l$

Proof. The minor $G' = G \setminus \{v\}$ for any vertex $v \in G$ is a member of the lower ideal k–Cycle Cover(l). Thus G' has treewidth at most $k + l - 1$ and G has treewidth at most $k + l$. □

Regarding membership algorithms, there is an efficient linear time algorithm for checking if a graph has a cycle of length at least 4 and for checking k–Cycle Cover(3) membership in time $O(n^{k+1})$. These algorithms are based on the following fact.

Lemma 147: A graph has a cycle of length at least 4 if and only if it has a biconnected component of at least 4 vertices.

Proof. If a graph has a 4-cycle then clearly an induced biconnected component containing the 4-cycle has at least 4 vertices. Let B be a biconnected component with at least 4 vertices. Since B is not a tree, let vertices a, b, and c be a triangle. Without loss of generality, let vertex $d \notin \{a, b, c\}$ be adjacent to a. Now, since a is not a cut-vertex, there exists a path from vertex d to either b or c avoiding vertex a. But this means, that the four vertices $\{a, b, c, d\}$ are in some common cycle of length at least 4. □

The above fact does not generalized for any cycle lengths greater than 4 (e.g., k–Cycle Cover(4)) since the graph with 5 vertices in Figure 8.3 is not Hamiltonian.

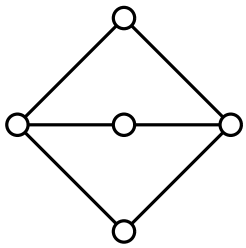


Figure 8.3: The smallest biconnected non-Hamiltonian graph.

With the recent powerful result of Courcelle (see [CM93, Cou93, Cou92b]) concerning 2nd-order monadic logic of graphs, we know for any fixed k there exists a linear time algorithm for 0–Cycle Cover(k), that is, this problem is fixed-parameter tractable (see [DF95]). Our application of this powerful result is sketch next. First, in the process of finding a depth-first spanning tree T of a graph G, if a back-edge (which is adjacent to a parent node) creates a cycle of length greater than k then we know the answer is 'no' and can stop. Otherwise, with this DFS tree T, we read off a tree decomposition of width at most $k-1$ for G [FL89b]. The 2nd-order monadic logic representation of this problem tells us that there is a linear time algorithm for graphs of bounded treewidth. For example, consider this simple logic for $k = 3$ on a graph $G = (V, E)$:

$$(\exists v_1 \in V) \wedge (\exists v_2 \in V) \wedge (\exists v_3 \in V) \wedge (v_1 \neq v_2) \wedge (v_2 \neq v_3)$$
$$\wedge((v_1, v_2) \in E) \wedge ((v_2, v_3) \in E) \wedge ((v_3, v_1) \in E) \quad .$$

8.3 Path/Cycle Cover Testsets

Recall that a testset for a canonical (family) congruence is a set T of boundaried graphs such that there exists a distinguisher Z in T for every two non-congruent graphs X and Y. That is, if $X \not\sim_{\mathcal{F}} Y$ then either $X \oplus Z \in \mathcal{F}$ while $Y \oplus Z \notin \mathcal{F}$ or vice versa. In this section we present testsets for various path and cycle cover families. With these testsets one can build practical membership automata for t-parses.

8.3.1 Some maximum path/cycle testsets

We first present testsets for the

$$\textsc{MaxCycle}(l) = 0\text{–}\textsc{Cycle Cover}(l) \quad \text{and}$$

$$\textsc{MaxPath}(p) = 0\text{–}\textsc{Path Cover}(p)$$

graph families for t-boundaried graphs. The $\textsc{MaxCycle}(l)$ tests resemble the Hamiltonian cycle tests presented in Section 4.3.4 (also see [Lu93]). Let $\mathcal{F}$ denote either $\textsc{MaxCycle}(l)$ or $\textsc{MaxPath}(p)$ where $l = p$. Assume X and Y are boundaried graphs that are not congruent. Without loss of generality, let Z be a test such that $X \oplus Z \in \mathcal{F}$ and $Y \oplus Z \notin \mathcal{F}$. Consider an out-of-family path/cycle P in $Y \oplus Z$ and let $Z' = \partial \cup (P \cap Z)$. Since $Z' \subseteq Z$, we have $X \oplus Z' \in \mathcal{F}$ and $Y \oplus Z' \notin \mathcal{F}$. The test Z' has components consisting of paths $P_1, P_2, \ldots, P_m$ where the boundary of Z' lies on the end-points of certain P_i. In the case of $\textsc{MaxPath}(p)$, at most two end-points of Z' are not boundary vertices. In the case of $\textsc{MaxCycle}(l)$, every end-point of Z' is a boundary vertex. The above discussion leads to the following result for any fixed boundary size.

Lemma 148: A finite testset for $\textsc{MaxCycle}(l)$ consists of tests of classes $T_\emptyset$ and T_a, and a testset for $\textsc{MaxPath}(p)$ consists of additional tests of classes T_b and T_c (see Figure 8.4).

For illustration, this testset (modulo boundary permutations) for $\textsc{MaxPath}(p)$ is shown in Figure 8.4 for boundary size 4. The testset for the $\textsc{MaxCycle}(l)$ family

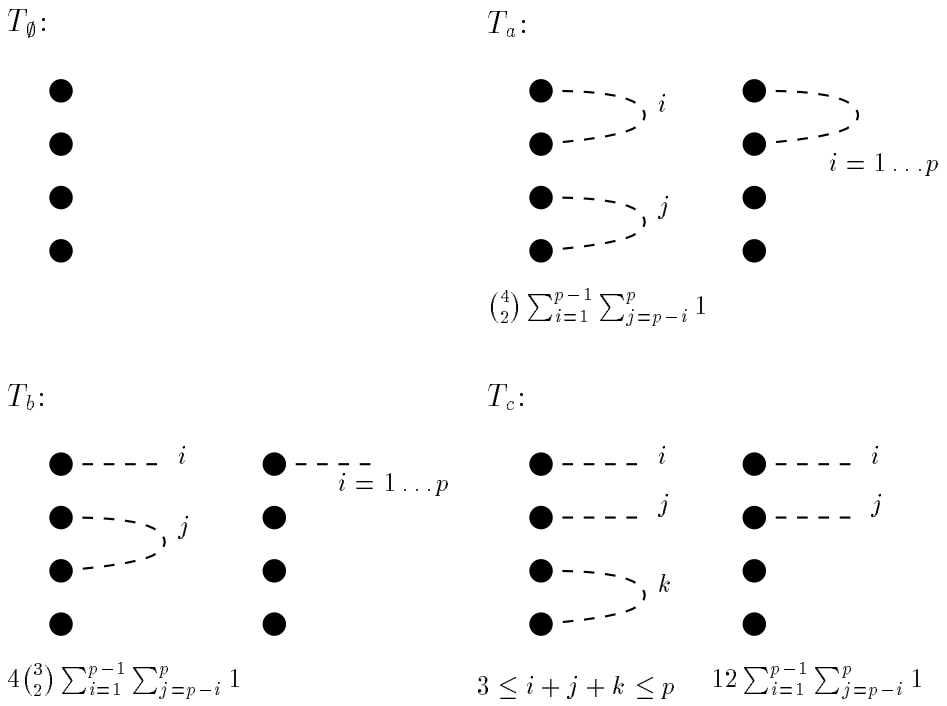


Figure 8.4: Four types of 4-boundaried tests for the MAXPATH(p) and MAXCYCLE(l) graph families, $l = p$.

consists of only tests of type $T_\emptyset$ and T_a. The variables i, j and k denote lengths of the paths and cycles of the tests. The summations (and inequalities) indicate how many tests in each class.

8.3.2 A maximum path automaton example

Using the MAXPATH(p) or MAXCYCLE(l) testsets given above, we can build fast membership automata that recognize t-parses in those graph families. A straightforward way of building automata from testsets is discussed in Section 11.3. Further information regarding building automata for bounded pathwidth and treewidth problems occur in the literature (e.g., see [AF93, APS91, KK94b]).

For a simple example, consider the family MAXPATH(4) over 1-parses. In Table 8.1 we give a transition diagram for the minimal finite state automaton that recognizes those graphs of pathwidth 1 (caterpillars) that have no paths of length 5

Table 8.1: The transition diagram for the MaxPath(4) automaton.

State	1-parse operators (0)	(1)	[0 1]	reference counts during $\mathcal{O}$-set search		
0	0	0	1	16	5	167
1	2	3	1	103	55	1
2	2	0	4	2	13	83
3	0	3	5	2	0	48
4	2	6	4	1	77	1
5	7	3	5	42	2	0
6	0	6	8	11	2	61
7	7	0	9	0	1	36
8	10	6	8	53	3	1
9	7	11	9	0	32	0
10	10	0	12	2	9	38
11	0	11	13	5	0	27
12	10	14	12	1	33	1
13	15	11	13	12	10	0
14	0	14	16	5	3	22
15	15	0	16	0	6	0
16	16	16	16	15	15	4

or greater. State 16 of this automaton is the out-of-family state. A pictorial view of this automaton is given in Figure 8.5.

Also, given in the right three columns of Table 8.1, for the MaxPath(4) automaton, is a reference count for each combination of state and alphabet symbol (transition) during the search for the single obstruction P_5 (path of length 5). See Section 11.3 for more details. The zero entries show that many branches where not taken during this type of obstruction set search. This indicates that computing obstruction sets via automata is not very efficient overall. In practice, we found that building an automaton takes over 95% of the computation time while searching for

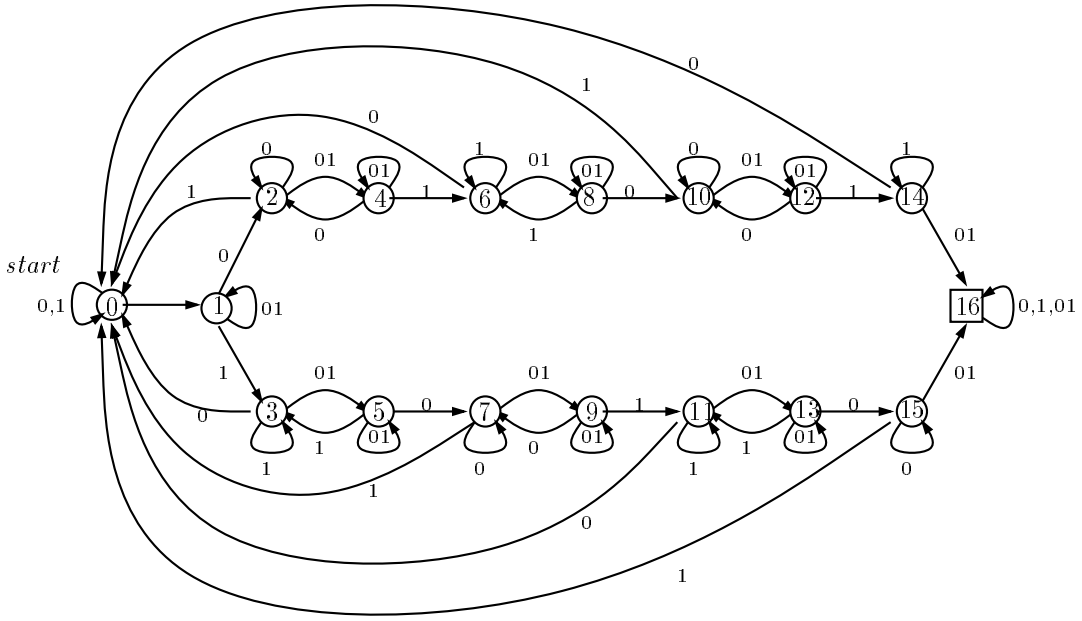


Figure 8.5: The automaton for MaxPath(4) corresponding to Table 8.1.

obstructions using a minimum automaton is very fast (by the search technique of Chapter 4). Our empirical results suggest that for obstruction set computations: if one has an automaton (or minimal congruence) available use it, otherwise consider another method.

8.3.3 Some generic cycle-cover testsets

We now describe how to create a testset for the graph family k–Cycle Cover(l), using boundary size t. The tests are a combination of the k–Feedback Vertex Set tests (see Section 7.2.2) and the maximum cycle tests given above. We illustrate the testset construction with the k–Cycle Cover(3) family.

Our k–Cycle Cover(3) testset Υ_t^k consists of all t-boundaried graphs that have the following properties:

1. Each graph is a member of k–Cycle Cover(3).
2. All degree zero and one vertices are boundary vertices.
3. There are no cycles of length greater than 4.

4. All 4 cycles, C_4, are isolated or attached to at most 1 boundary vertex.

5. Any degree two vertex is on a path of length at most 3 between some two boundary vertices.

6. Any edge e on an interior cycle C_3 is allowed if there exists a path between two boundary vertices a and b containing e that is larger than any other path between a and b.

Lemma 149: The set of t-boundaried graphs Υ_t^k is a testset for k–Cycle Cover(3).

Proof. Let $\mathcal{F} = k$–Cycle Cover(3) for some fixed constant k. The proof that Υ_t^k is a testset follows the same line of reasoning as our proof of correctness for the k–Feedback Vertex Set testset. For two graphs G and H such that $G \not\sim_{\mathcal{F}} H$ we need to show that Υ_t^k contains a distinguishing test. First, we can assume both G and H are members of $\mathcal{F}$ since the empty t-boundaried graph is a distinguisher. Without loss of generality, let T be any t-boundaried graph such that $G \oplus T \in \mathcal{F}$ and $H \oplus T \notin \mathcal{F}$. It is safe to assume that the graph T does not have any degree zero or one internal vertices because any such vertex does not contributed to any cycle created by the $\oplus$ operator.

If T contains any internal cycles of length four or greater then any kill witness set W (in either case) must contain one vertex on these cycles. So, another distinguishing test T' created by adding $|(T \cap W) \setminus \partial|$ isolated C_4's to a "pruned" $(T \cap W) \setminus \partial$, where W is a witness for $G \oplus T \in \mathcal{F}$. (A graph is considered to be pruned if it satisfies property 2 of Υ_t^k.) Now consider any remaining cycle C of length at least 4 in T'. This cycle must be covered by a boundary vertex of W. This "testing" boundary/internal cycle can be replaced with a test that satisfies property 4 of Υ_t^k. For the subset $S = \partial \cap W$ we construct a test T'' that is a "pruned" $T' \setminus S$ with $|S|$ four cycles added, each containing one of the removed boundary vertices in S. Thus, $G \oplus T'' \in \mathcal{F}$ while still $H \oplus T'' \notin \mathcal{F}$. If $H \oplus T'' \in \mathcal{F}$ then we can find a kill set W' containing $T \cap W$, where $|W'| \leq k$, that contradicts $H \oplus T \notin \mathcal{F}$.

Property 5 is trivially seen since contracting out any such degree two vertex (that does not satisfies this property) produces a smaller distinguisher. For the validity of the property 6 restriction, we first note that in order for an edge $e = (u, v)$ to occur

in a T''-type test (i.e., a test without cycles of length 4 or greater) the "longer paths" must all detour around a neighbor vertex z on a cycle $C_3 = \{x, y, z\}$. See test (a) in Figure 8.6. In this case, removing the edge does not allow for any smaller kill set for the out-of-family $H \oplus T''$. This is because a kill vertex z could have been replaced with x or y in a witness set. □

Note that there may be other test reductions besides the ones listed above for Υ_t^k, such as contracting edge e' of Figure 8.6. However, we have stated enough restrictions to guarantee a finite testset.

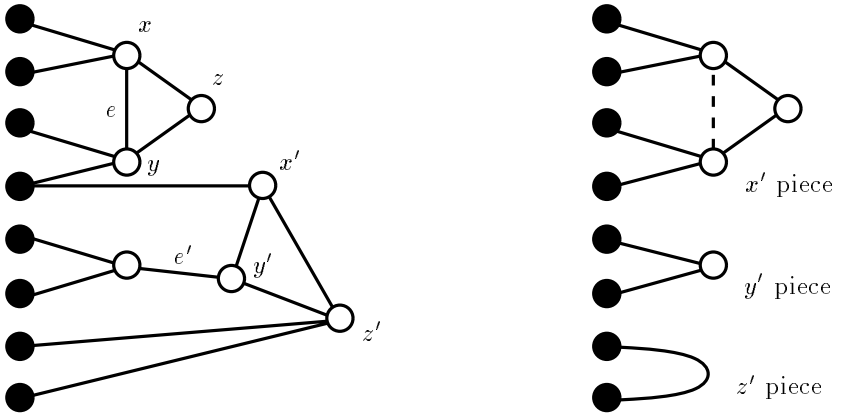
Lemma 150: The cycle-cover testset Υ_t^k has finite cardinality.

Proof. To show finiteness, it suffices to prove a bound $f(t)$ on the number of internal vertices excluding C_4's that a *connected* test can have for boundary size t. Since a test is a member of k–CYCLE COVER(3), at most $k-1$ isolated four cycles can be added for a general test. For our base cases, $f(1) = 0$ for the empty test and $f(2) = 2$ via a path of length 3.

We now consider some structural possibilities for a test. Notice that all vertices of any internal C_3 have degree greater than two such as the triangle $\{x', y', z'\}$ in Figure 8.6 because of property 6. If the test has a $C_3 = \{x', y', z'\}$ then removing the internal incident edges $\{(x', y'), (x', z'), (y', z')\}$ creates a 3 component test, not necessarily in Υ_t^k. If there were fewer than 3 components then we would contradict the assumption that there exists no cycles of length greater than 3. To satisfy the properties of Υ_t^k we can contract at most one edge incident to each of $\{x', y', z'\}$ as shown in Figure 8.6. Note that a contraction is not needed for a boundary vertex within the original C_3. Since each of these smaller components have at most $t-2$ boundary vertices, we get the following recurrence

$$f(t) \leq 3 \cdot f(t-2) + 3 \leq 3^t \quad .$$

Now consider the case that a test has no C_3's (in fact, no cycles). Following the argument of Lemma 120 modified for these tests, we see that if there is a degree two internal vertex then $f(t) \leq f(t-1) + 2$. Otherwise, we must have a tree of bounded height with all internal vertices of degree at least three. Again, the previous bound $f(t) \leq 3^t$ holds. □



(a) Edge e is not needed.

(b) Recursive pieces of previous test.

Figure 8.6: A t-boundaried test example (in Υ_t^k) for k–Cycle Cover(3).

8.4 Other VC/FVS Generalizations

The minimum *c-clique/hyperedge cover* (e.g., vertex covers correspond to K_2/binary-edge covers) is defined to be the minimum number of vertices needed to cover all cliques of size c of a graph. A parameterized graph family based on this invariant is

$$k\text{–Clique Cover}(c) = \{G \mid \text{there exists a } V' \subseteq V(G) \text{ with } |V'| \leq k \text{ such that } G \setminus V' \text{ has no cliques of order } c\} \quad .$$

Unfortunately, the graph family k–Clique Cover(c) is not a lower order in the minor order since graphs with a bounded *maximum clique size* is not a lower ideal; and hence, no obstruction set characterizations are possible (see Lemma 110).

The minimum *c-fixed cycle cover* (feedback vertex sets correspond to covering all cycles) is defined to be the minimum number of vertices needed to cover all cycles of length exactly c. The graph family with covers of size at most k is denoted by k–Fixed Cycle Cover(c). Like above, the graph family k–Fixed Cycle Cover(c) is not a lower ideal in the minor order for $c > 3$ since the graph consisting of $k+1$ disjoint C_{c+1} cycles is in k–Fixed Cycle Cover(c) but the minor consisting of one edge contracted from each C_{c+1} is not.

Note that the families k–Clique Cover(3) and k–Fixed Cycle Cover(3) are identical for any k since both require coverings of all K_3 subgraphs. For illustration

purposes, we now present a finite-state dynamic program for k–CLIQUE COVER(3) = k–FIXED CYCLE COVER(3) over t-parses. This algorithm, designed in the same spirit as our k–VERTEX COVER and k–FEEDBACK VERTEX SET algorithms, is based on a general-purpose linear time algorithm that computes the size of any minimum-sized K_3 cover.

For this K_3-cover dynamic program, the state of a t-parse $G_n = [g_1, g_2, \ldots, g_n]$ at prefix position $m \leq n$ consists of these three components:

1. Induced boundary ∂ subgraph of G_m.

2. For each $S \subseteq \partial$,

$$M_m[S] = \{C \mid C \text{ is the minimum } K_3 \text{ cover containing } S \subseteq \partial\} \quad .$$

3. For each $S \subseteq \partial$ and for each witness cover $C \supseteq S$ of G_m with $M_m[S]$ vertices,

$$P_m[C] = \left\{ (u,v) \;\middle|\; \begin{array}{c} u \in \partial \text{ and } v \in \partial \text{ and there exists a } w \in G_m \setminus (\partial \cup C) \\ \text{such that } (u,w) \text{ and } (v,w) \text{ are edges in } G_m \setminus C \end{array} \right\}$$

(each $P_m[C]$ has $\partial \setminus C$ as an available boundary attribute) and

$$W_m[S] = \{P_m[C] \mid C \supseteq S \text{ is a } K_3 \text{ cover for } G_m \text{ of size } M_m[S]\} \quad .$$

Note that it is not necessary to keep $P_m[C]$ in $W_m[S]$ if there exists another witness cover C' for S such that $C \cap \partial \subseteq C' \cap \partial$ and $P_m[C'] \subset P_m[C]$. This is because the following implication holds for any t-parse extension Z:

$$(G_m \cdot Z) \setminus C \in k\text{–CLIQUE COVER}(3) \Rightarrow (G_m \cdot Z) \setminus C' \in k\text{–CLIQUE COVER}(3) \quad .$$

The following algorithm updates these state components as m increases up to n. At the end of the algorithm, the minimum-sized K_3 cover of G_n is the value of $M_n[\emptyset]$. To convert this algorithm to a fixed-k version for k–CLIQUE COVER(3) we simply place an upper bound of $k+1$ on the values of the $M_m[S]$; this technique of "trimming" also yields a finite-state k–CLIQUE COVER(3) algorithm since, in this trimmed version, there are at most $k+2$ values of $M_m[S]$ and $\sum_{i=0}^{|\partial|} \binom{|\partial|}{i} \cdot 2^{\binom{i}{2}}$ different $P_m[C]$.

Algorithm 151: Finds minimum K_3 cover of a t-parse $G_n = [g_1, g_2, \ldots, g_n]$

Initial state for prefix $G_{t+1} = [\textcircled{0}, \ldots, \textcircled{t}]$:
For all $S \subseteq \partial$
    $M_{t+1}[S] = |S|$;
    $W_{t+1}[S] = \{P_{t+1}[S] = \emptyset\}$;
end
For $m = t+1, \ldots, n$ do for all $S \in 2^{\partial}$:
Case 1: $g_{m+1} = \textcircled{i}$ and $i \in S$

$M_{m+1}[S] \leftarrow M_m[S \setminus \{i\}] + 1$;
$W_{m+1}[S] \leftarrow \text{pulloff}(W_m[S \setminus \{i\}], i)$;

end case
Case 2: $g_{m+1} = \textcircled{i}$ and $i \notin S$

$M_{m+1}[S] \leftarrow M_m[S]$;
$W_{m+1}[S] \leftarrow \text{pulloff}(W_m[S], i)$;

end case
Case 3: $g_{m+1} = \boxed{i\ j}$ and $(i \in S$ or $j \in S)$

$M_{m+1}[S] \leftarrow M_m[S]$;
$W_{m+1}[S] \leftarrow W_m[S]$;

end case
Case 4: $g_{m+1} = \boxed{i\ j}$ and $i \notin S$ and $j \notin S$

Comment: $d(i,j)$ denotes the distance between vertex i and j
if $\exists\ P_m[C] \in W_m[C]$ such that $(i,j) \notin P_m[C]$ and $d(i,j) \neq 2$ for $\partial(G_m \setminus C)$
    $M_{m+1}[S] \leftarrow M_m[S]$;
    $W_{m+1}[S] \leftarrow \text{addEdgeOk}(W_m[S], i, j)$;
else
    $M_{m+1}[S] \leftarrow M_m[S] + 1$;
    $W_{m+1}[S] \leftarrow \text{dropBoundary}(W_m[S], i, j) \cup \text{otherPlus1}(W_m[S], i, j)$;
end

end case

Function pulloff($W_m[S]$, i)

$newW \leftarrow \emptyset$;
for each $P_m[C] \in W_m[S]$
    for every pair of boundary edges (i,j) and (i,k) of $\partial \setminus C$
        $P_m[C] \leftarrow P_m[C] \cup \{(j,k)\}$;
    end
    $newW \leftarrow newW \cup (P_m[C] \setminus \{(i,j) \mid i \neq j\})$;
end
return $newW$;

end function

Function addEdgeOk($W_m[S]$, i, j)

$newW \leftarrow \emptyset$;
for each $P_m[C] \in W_m[S]$
    if $(i,j) \notin P_m[C]$ and $d(i,j) \neq 2$ for $\partial(G_m \setminus C)$
        $newW \leftarrow newW \cup P_m[C]$;
    end
end
return $newW$;

end function

Function dropBoundary($W_m[S]$, i, j)

$newW \leftarrow \emptyset$;
for each $P_m[C] \in W_m[S]$
    $newW \leftarrow newW \cup P_m[C \setminus \{i\}] \cup P_m[C \setminus \{j\}]$;
end
return $newW$;

end function

Function otherPlus1($W_m[S]$, i, j)

$newW \leftarrow \emptyset$;
for each $k \in \partial \setminus S$
    if $M_m[S \cup \{k\}] = M_m[S] + 1$
        for each $P_m[C] \in S_m[S \cup \{k\}]$ with $(i,j) \notin P_m[C]$
            $newW \leftarrow newW \cup P_m[C]$;
        end
    end
end
return $newW$;

end function

return $M_n[\emptyset]$;

end algorithm

Theorem 152: Algorithm 151 correctly computes the size of the minimum-sized K_3 cover of a t-parse G_n in linear time.

Proof. The initial state of G_{t+1} is correct by definition. We now show that the state of the prefix G_{m+1} is correctly computed from the current operator g_{m+1} and the state of G_m.

Case 1: vertex operator $\textcircled{i}$, $i \in S$
By definition of the $M_m[S]$'s, $M_m[S \setminus \{i\}] \leq M_m[S]$. Let i' denote the new vertex in $V(G_{m+1}) \setminus V(G_m)$. We first show $M_{m+1}[S] \leq M_m[S \setminus \{i\}] + 1$. Let $C \supseteq S \setminus \{i\}$ be a minimum witness cover for G_m. We immediately see that $C' = C \cup \{i'\}$ is a witness cover for G_{m+1} with $M_m[S \setminus \{i\}]+1$ vertices. Now suppose $M_{m+1}[S] < M_m[S \setminus \{i\}]+1$ and let C' be a witness cover for $M_{m+1}[S]$. Since C' must contain i', we see that $C = C' \setminus \{i'\}$ is a witness cover for $M_m[S \setminus \{i\}]$ with fewer than $M_m[S \setminus \{i\}]$ vertices. This contradiction shows that $M_{m+1}[S] = \min(M_m[S], M_m[S \setminus \{i\}]) + 1$.

The `pulloff` function simply transforms a previous witness cover C representation for G_m to one for G_{m+1} by moving the old vertex i into the interior. Two things are done during this transformation: (1) any paths of length two between three boundary vertices with vertex i as the mid-point is now added to $P_m[C]$ and (2) any ordered pair with vertex i in $P_m[C]$ is removed.

Case 2: vertex operator $\textcircled{i}$, $i \notin S$
Since $M_m[S] \leq M_m[S \cup \{i\}]$ and $W_m[S \cup \{i\}] \subseteq W_m[S]$ if the $M_m[]$ equality holds, we just need to consider the previous state restricted to S. Clearly $M_{m+1}[S] = M_m[S]$ since the operator $\textcircled{i}$ does not add any edges and the new boundary vertex does not have to be part of a witness cover C.

Case 3: edge operator $\boxed{i\ j}$, $i \in S$ or $j \in S$
Since the new edge created by $\boxed{i\ j}$ is covered by any witness cover $C \supseteq S$ for $M_m[S]$, we see that $M_{m+1}[S] \leq M_m[S]$. Also, since G_m is a subgraph of G_{m+1} we also have $M_m[S] \leq M_{m+1}[S]$.

Case 4: edge operator $\boxed{i\ j}$, $i \notin S$ and $j \notin S$
Clearly $M_{m+1}[S] \geq M_m[S]$ and $M_{m+1}[S] \leq M_m[S] + 1$ since (1) any witness cover for G_{m+1} containing S is a witness cover for G_m and (2) at most one more vertex is needed to remove any new K_3's that the edge operator $\boxed{i\ j}$ creates.

If $M_{m+1}[S] = M_m[S]$ then there is a witness cover $C \supseteq S$ of G_m such that $G_m \setminus C$ does not have a path of length two between boundary vertices i and j. This can be detected by looking for a $P_m[C]$ such that $(i,j) \notin P_m[C]$ and that the remaining boundary $\partial \setminus C$ does not have a path of length two between i and j. (Note that if either i or j is in C then such a $P_m[C]$ satisfies these conditions.)

If no such witness cover C exists for G_m that can be used for G_{m+1}, then all "significant" witness covers with one more vertex can be found by two ways: (1) including either vertex i or vertex j in the various C covers for $M_m[S]$ (see the `dropBoundary` function), or (2) looking for another boundary vertex besides i and j that can be safely deleted (see the `otherPlus1` function). Note that we are not concerned with deleting a single internal vertex since they do not help in covering future K_3's created by the suffix $(g_{m+2}, \ldots, g_n)$ that is not provided by one of the above constructed covers. □

8.5 The Path and Cycle Cover Obstructions

Figures 8.7–8.10 present the known connected obstructions for the four graph families 1–Path Cover(1), 1–Path Cover(2), 1–Path Cover(3) and 2–Path Cover(1). We know of at least 43 connected obstructions for the 1–Path Cover(4) family and at least 64 connected obstructions for the 2–Path Cover(2) family. The observant reader may notice that 1–Path Cover(3) and 2–Path Cover(1) share a common obstruction.

Figure 8.11 presents the seven connected obstructions for the 1–Cycle Cover(3) graph family. There are at least 99 connected obstructions for the 2–Cycle Cover(3) graph family. Oddly enough, the 1–Cycle Cover(3) and 2–Path Cover(1) families have two identical obstructions.

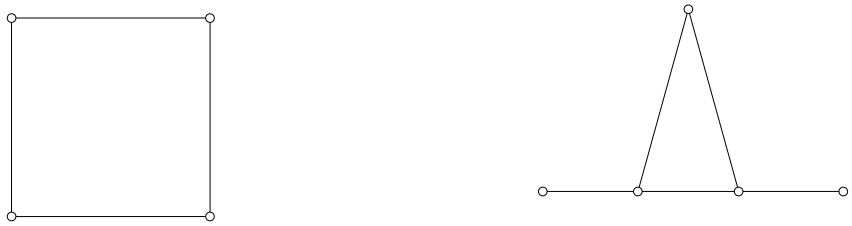


Figure 8.7: Connected obstructions for 1–Path Cover(1).

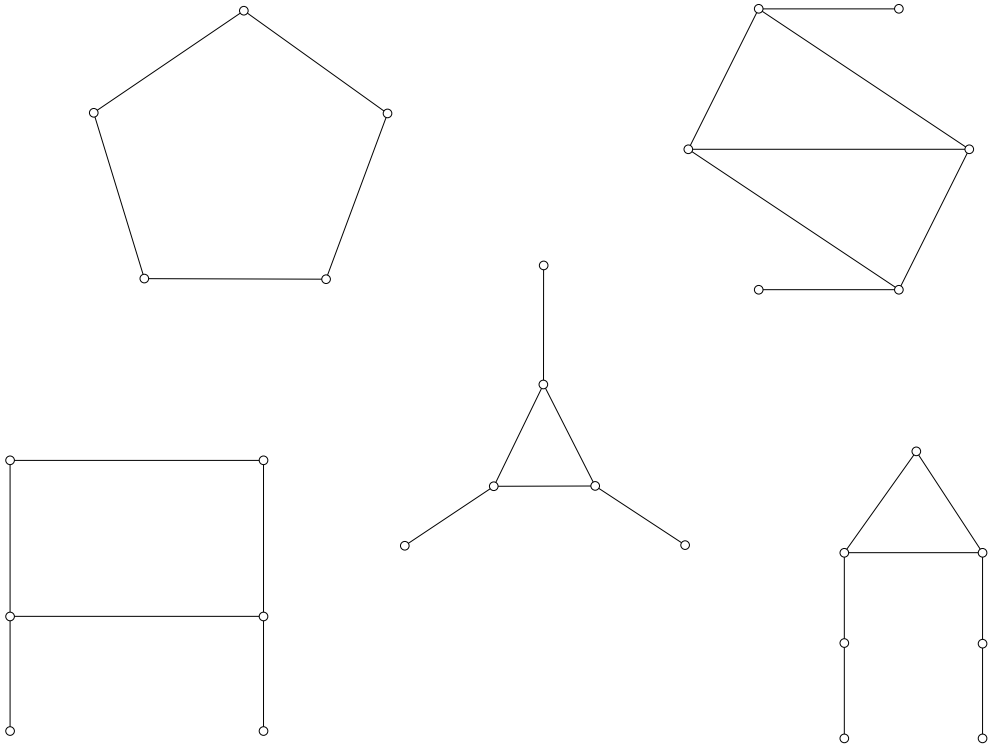


Figure 8.8: Connected obstructions for 1–Path Cover(2).

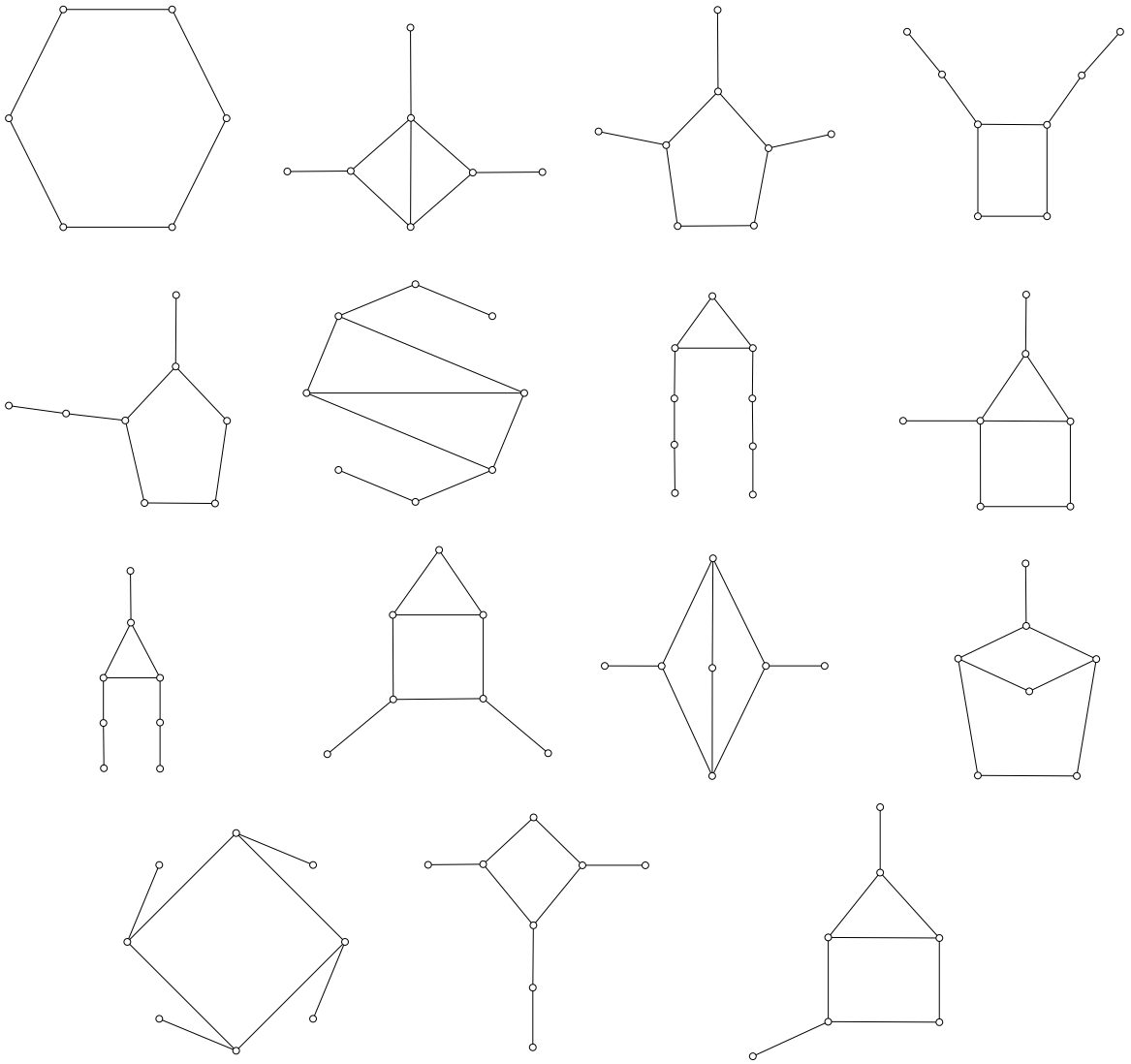


Figure 8.9: Known connected obstructions for 1–Path Cover(3).

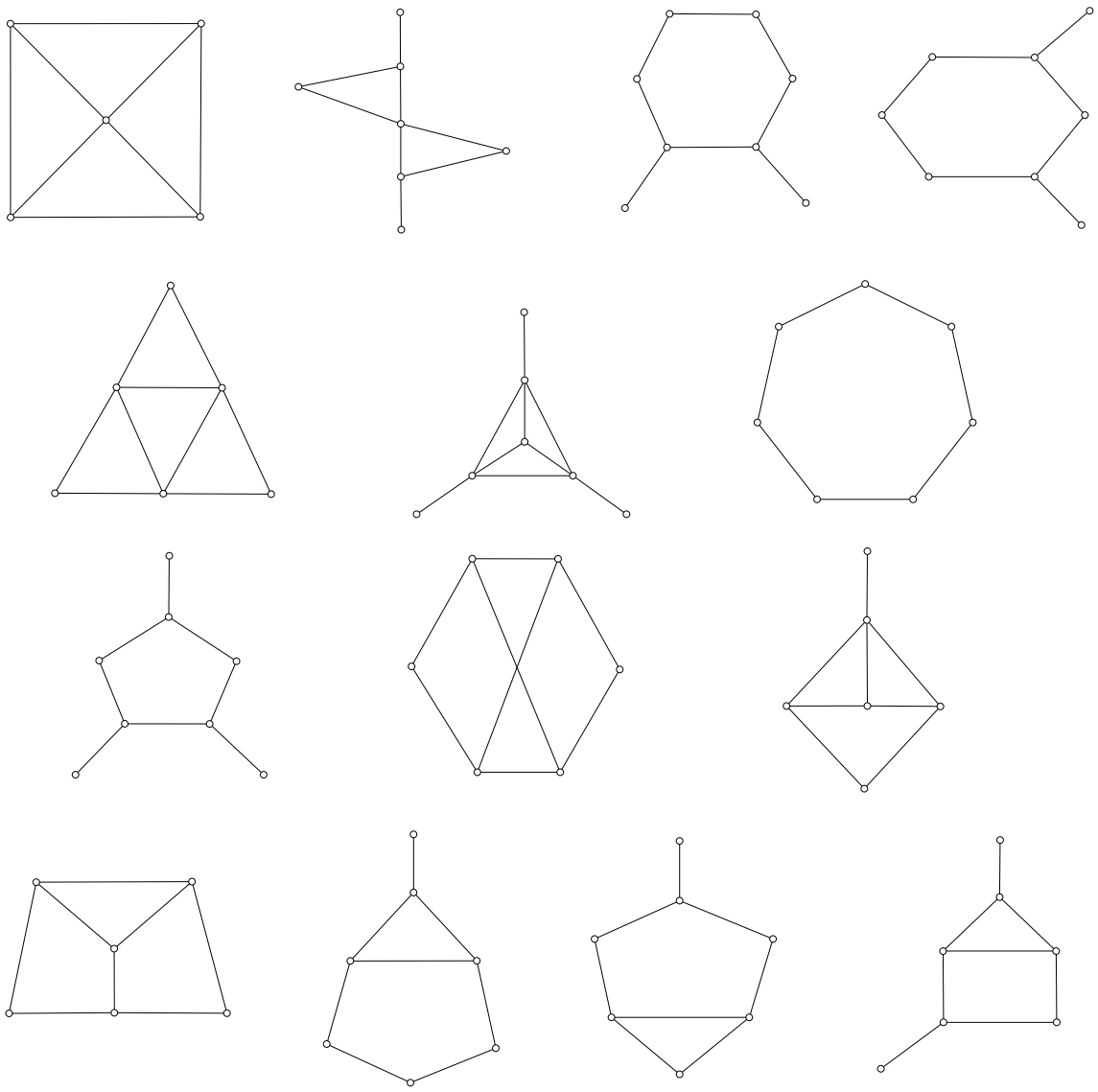


Figure 8.10: Known connected obstructions for 2–Path Cover(1).

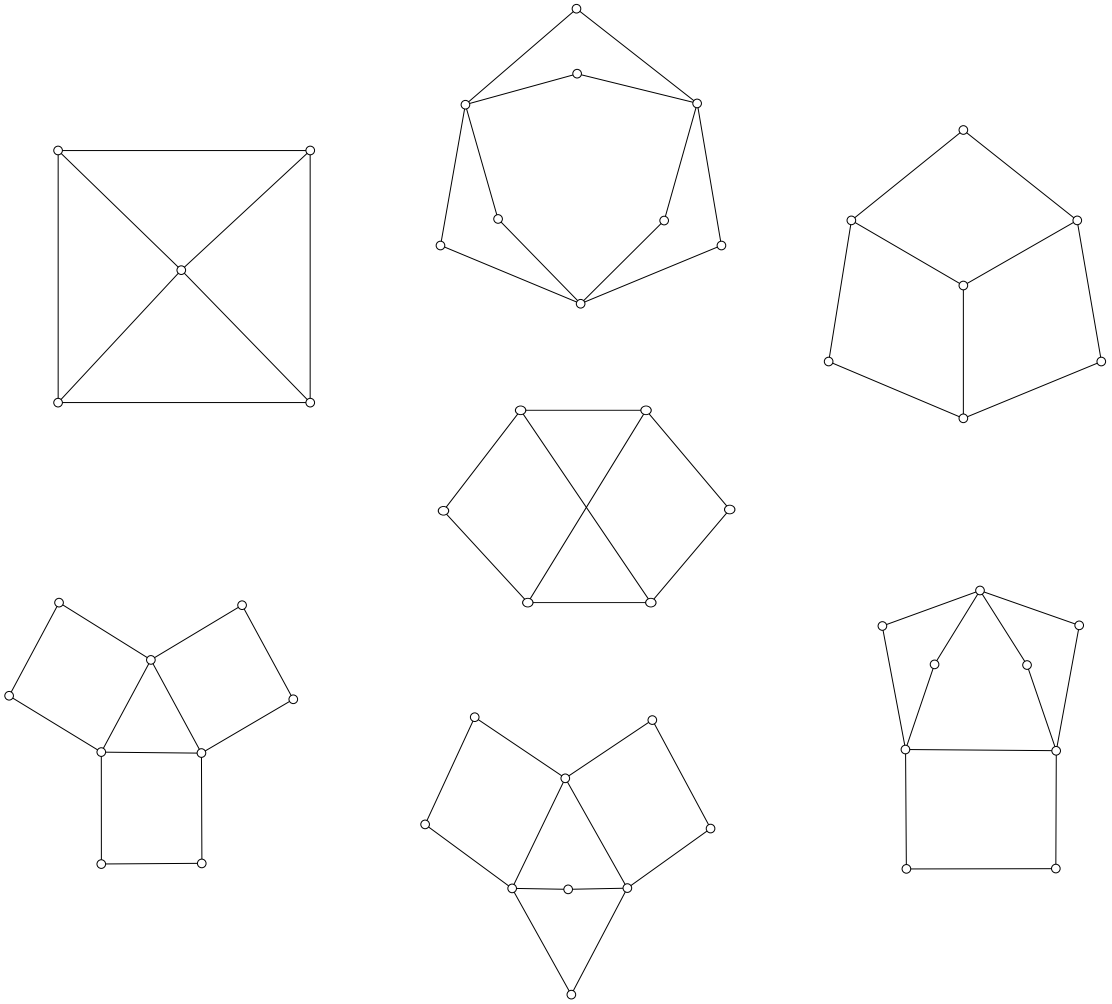


Figure 8.11: Connected obstructions for 1–Cycle Cover(3), pathwidth ≤ 4.

Chapter 9

Outer-Planar Graphs

Since several surface embedding families of graphs are closed under the minor order, the main purpose of this chapter is to develop our bounded combinatorial width technique for characterizing these types of lower ideals. This chapter uses an approximation search (via universal distinguishers) to find a set of forbidden minors for a simple generalization of the outer-planar graph family.

9.1 Introduction

Definition 153: A graph G is *outer-planar* if there exists a planar drawing of G with one face (region) containing all the vertices of G. The family of graphs that are within k vertices of outer-planar is defined as follows:

k–OUTER PLANAR =
$\{G \mid$ there exists a $V' \subseteq V(G)$ such that $G \setminus V'$ is outer-planar and $|V'| \leq k\}$

One reason that the class of outer-planar graphs is popular is that the art of drawing them is influenced by many computable criteria, most of which are based on geometric symmetries [Man90]. (It would be interesting to know whether the same methods apply to the k–OUTER PLANAR families.)

It is known that there are two obstructions to the family of outer-planar graphs, namely K_4 and $K_{2,3}$ (see for example [Tru93]). The computational complexity for

outer-planar membership problem is easily seen to be linear time by using one of the known linear time planarity algorithms along with the next result. Other linear time outer-planar algorithms are presented in [Mit79] and [SI79].

Observation 154: A graph G is outer-planar if and only if $G + K_1$ is planar.

Proof. Recall that $G + K_1$ denotes the graph created by adding a new vertex to G of degree $|G|$. If G is outer-planar then there exists a planar embedding of $G + K_1$ by placing K_1 in the outer face of an outer-planar embedding of G. If $G + K_1$ is planar then let $S = (v_1, v_2, \ldots, v_{|G|})$ be neighbors of K_1 in clockwise order in a planar embedding E of $G + K_1$. The face F, where S is a subsequence, that contains K_1 in $E \setminus K_1$ is an outer face of an outer-planar embedding of G. □

Another useful result states that every biconnected outer-planar graph with at least 3 vertices has a unique Hamiltonian cycle [CB81].

9.2 The 1–Outer Planar Computation

We now turn our attention to the family of graphs 1–Outer Planar (i.e., those graphs that are within one vertex of outer-planar). Using the above lemma we have a practical $O(n^2)$ membership algorithm. We now state some direct nonminimal (in the minor order) pretests.

Lemma 155: If a connected graph satisfies any of the following properties then it is *not* an obstruction to 1–Outer Planar:

1. Contains adjacent degree two vertices.
2. Contains a cut-edge (e.g., contains a degree one vertex).
3. Contains a cut-vertex (i.e., not biconnected).

Proof. Assume u and v are adjacent degree two vertices of an obstructions G. Let G' be the (u, v) edge contracted minor of G with the new vertex labeled w. Since

$G' \in$ 1–Outer Planar there exists a vertex x of G' such that $G'' = G \setminus \{x\}$ is outer-planar. If $x = w$ then $G \setminus \{u\}$ (or $G \setminus \{v\}$) would be outer-planar. Also x is not adjacent to w since removing x from G would also show that $G \in$ 1–Outer Planar (the pendent edge incident to vertex w can be subdivided). Likewise, if x is not adjacent to w (which means x is not adjacent to either u and v), then the graph $G' \setminus \{x\}$ can be embedded in the plane such that the degree two vertex w is on the outer face. Expanding w to its original (u, v) edge gives a contradiction to the fact that $G \notin$ 1–Outer Planar.

Now consider (u, v) to be a cut-edge of an obstruction G. Let C_u and C_v be the two components of $G \setminus \{(u, v)\}$. Since G is an obstruction, exactly one of these components must be in 1–Outer Planar (and not outer-planar) while the other is outer-planar. Without loss of generality, assume C_u is outer-planar. If $C_v \setminus \{v\}$ is outer-planar so would $G \setminus \{v\}$. Thus an interior vertex w ($w \neq v$) of C_v must be C_v's witness for being in 1–Outer Planar. However, the graph $C_v \setminus \{w\}$ has an embedding with v on the outer face and an edge added between (u, v) shows that $G \setminus \{w\}$ is outer-planar. Therefore, any obstruction to 1–Outer Planar can not have cut-edges.

At last consider a vertex v to be at least a three-way cut-vertex of an obstruction G. Let $C_1, C_2, \ldots, C_m$, $m \geq 3$, be the components of $G \setminus \{v\}$, and $C'_1, C'_2, \ldots, C'_m$ be the vertex induced components of G with vertices $\{v\} \cup V(C_i)$, $1 \leq i \leq m$. (We assume that every component C_i has at least one edge for otherwise we would have a cut-edge.) Since G is not within one vertex of outer-planar, at least one of the C_i, and hence one of the C'_i is not outer-planar. If any graph C'_i is outer-planar, let G' be the 1–Outer Planar graph created by deleting an edge e of C_i from G. Notice that if $G' \setminus \{v\}$ is outer-planar then so is $G \setminus \{v\}$. Let w ($w \neq v$) be a vertex of G' such that $G' \setminus \{w\}$ is outer-planar. Since the induced subgraph C'_i is already outer-planar, w must be in one of the C_j, $j \neq i$. Since we can put back the edge e in G', and have an outer-planar embedding of $G \setminus \{w\}$, this implies that each of the C'_i is not outer-planar. Let $G'' \in$ 1–Outer Planar be the graph G with an edge e' of C_1 deleted. Since v is the only shared vertex between the C'_i, v is the only possible witness to G'' being in 1–Outer Planar. Thus, $C'_2, C'_3, \ldots, C'_m$ must all be in 1–Outer Planar. If we repeat the above argument for $e' \in C_2$, then C'_1 is also

shown to be in 1–OUTER PLANAR. In these subcases vertex v is the witness, and thus v would be a witness for $G \in$ 1–OUTER PLANAR. However, since G is assumed to be an obstruction, we have a contradiction to the fact that C_3 exists. Therefore, G can not have three-way cut-vertices.

We now assume an obstruction G has a two-way cut-vertex v. Again consider C_1 and C_2, the components of $G \setminus \{v\}$, and C_1' and C_2' as described above. Let (u, v) be an edge in C_1'. The graph $G' = G \setminus \{(u, v)\}$ is in 1–OUTER PLANAR. There exists a witness vertex w of G', such that $G' \setminus \{w\}$ is outer-planar. Vertex w must be in C_2', for otherwise C_2' would be outer-planar. And $w \neq v$, for otherwise G would be a member of 1–OUTER PLANAR. So $w \in C_2$. If we repeat this process with an edge (u', v) in C_2', we get another witness vertex w' in C_1 for $G \setminus \{(u', v)\}$. Since $C_1 \subseteq C_1' \setminus \{(u, v)\}$ and $C_2 \subseteq C_2' \setminus \{(u', v)\}$, we see that $G \setminus \{v\}$ must be outer-planar. This contradicts the fact that G is an obstruction. So G can not have any cut-vertices. □

The following result (given with an alternate proof) is well-known, for example it is stated in Van Leeuwin's handbook without a proof [vL90].

Lemma 156: Any outer-planar graph has treewidth at most 2.

Proof. Clearly the complete graphs K_1 and K_2 have treewidth at most 2. Without loss of generality, assume that we have an outer-planar graph G that is triangulated (since removing edges from a graph only decreases its treewidth). The graph G has a degree two vertex v since it would otherwise contain the complete graph K_4 as a minor (see [Ram94]); we know that K_4 is an obstruction to outer-planar. Consider the outer-planar subgraph $G' = G \setminus \{v\}$. By induction, G' has a tree decomposition of vertex sets $\{V_i\}$, $i \in I$, of width 2. Let $e = (x, y)$ be the internal (non-boundary) edge of vertex v's triangle in G. Some V_i in the tree decomposition for G' contains both vertices x and y. We can create a tree decomposition of G of width 2 by adding a leaf vertex set $V_v = \{x, y, v\}$ and attaching it to V_i. □

Corollary 157: Any obstruction of 1–OUTER PLANAR has treewidth at most 4.

Proof. Here any outer-planar graph has treewidth at most 2. For such a tree decomposition, we can include two additional vertices in each set to account for any fixed

vertex in a deleted-vertex minor and one vertex for the witness "within one vertex." □

With the use of the above two results we can give a weak pathwidth bound for 1–OUTER PLANAR's obstruction set.

Lemma 158: If G is an obstruction to 1–OUTER PLANAR and has pathwidth greater than 5, then G has at least 14 vertices.

Proof. Let $G' = G \setminus \{v\}$ be a minor of G for any vertex $v \in G$. Observe G' is connected by Lemma 155. Since $G' \in$ 1–OUTER PLANAR there exists a vertex $w \in G'$ such that the graph $G'' = G' \setminus \{w\}$ is outer-planar. If G'' has pathwidth 3, then we can combine vertices v and w with all the vertex sets in a path decomposition of width 3 for G'' to get a path decomposition of width 5 for G.

Now consider the following two possibilities for G'' to have pathwidth at least 4.

If G'' contains a biconnected subgraph S of pathwidth 4, then consider the dual graph D of S without the outer face vertex (using any induced outer-planar embedding of G''). Without loss of generality, we can assume that S is triangulated. It is easy to see that the graph D must be a tree and have maximum degree three. The corresponding pathwidths of the S's for the 18 possible trees D (see Figure 9.1) of order 9 are all less than 4. Thus, since $|D| = |S| - 2$, we know that the smallest S with pathwidth 4 has 12 vertices (see Figure 9.2). So the obstruction G then has to have at least 14 vertices.

The other possibility is that G'' must have a cut-vertex u and that at least three of the outer-planar components $C_1, C_2, \ldots, C_m$ of $G'' \setminus \{u\}$, must have pathwidth 3 or greater. (We have taken care of the case that one of the C_i has pathwidth 4 by the argument above.) The smallest outer-planar graph of pathwidth 3 is the 6 vertex augmented graph $A(K_3)$; this is the second smallest pathwidth 2 obstruction (the smallest, K_4, is not outer-planar). Thus, in this case, G must have at least $3 \cdot 6 + 3 = 18$ vertices to have pathwidth greater than 5. □

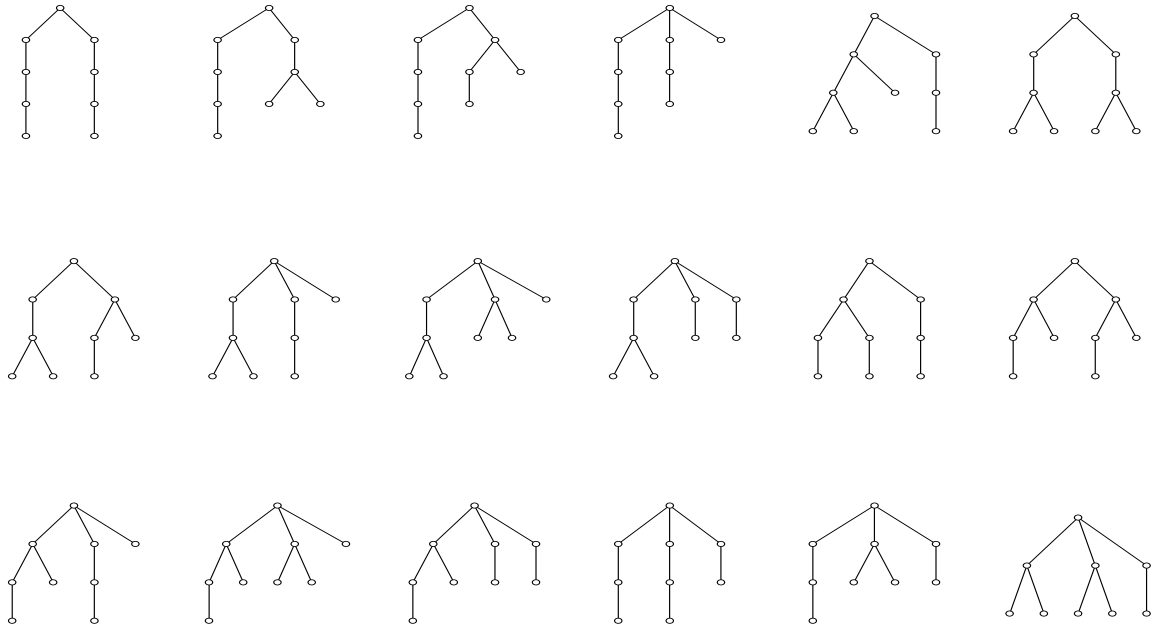


Figure 9.1: All 9 vertex outer-planar dual trees (duals minus outer face vertex).

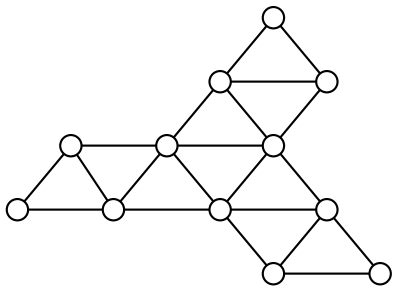


Figure 9.2: The smallest outer-planar graph with pathwidth 4.

9.2.1 An outer-planar congruence

Observe that a disconnected graph is outer-planar if and only if each component is independently outer-planar. Thus, without loss of generality, assume for each prefix G_m of a t-parse G_n at most one component has order greater than one.

We now define an outer-planarity state function f from t-parses to states. Our outer-planarity congruence on t-parses, $G \sim_o H$, is based on equality of states, $f(G) = f(H)$.

An *outer-planarity state* consists of a set of outer-planar embeddings and each *outer-planarity embedding* consists of these ingredients:

1. Clockwise circular sequence Seq of boundary vertices of the outer face with duplicates allowed but minimize in length (as indicated below).
2. A boundary indicator vector Gap. (Are there hidden interior vertices located between the gaps of the boundary sequence Seq?) Here $|Gap| = |Seq|$. The elements of a gap vector contain any of the following symbols.
   (a) 'E' – just an edge
   (b) 'P' – path of non-boundary vertices
   (c) 'C' – cycle/face between vertices
3. An optional add-edge list Add. (Which boundary edges can be added and still preserve the outer-planarity embedding?) Note $|Add| \leq \binom{t+1}{2}$.

To save on storage, one may want to keep only "canonical" Seq's and do pancake flips of the circular sequence (1/2 of the embeddings kept). This makes the congruence a bit more complicated and so we do not discuss this optimization further. Let $l_t(G)$ be the maximum length of Seq over all outer-planarity embeddings of a t-parse G. The following lemma shows that the range of this function is finite.

Lemma 159: For any t-parse G, $l_t(G) \leq \begin{cases} t+1 & \text{if } 0 \leq t \leq 1 \\ 2t & \text{if } t \geq 2 \end{cases}$.

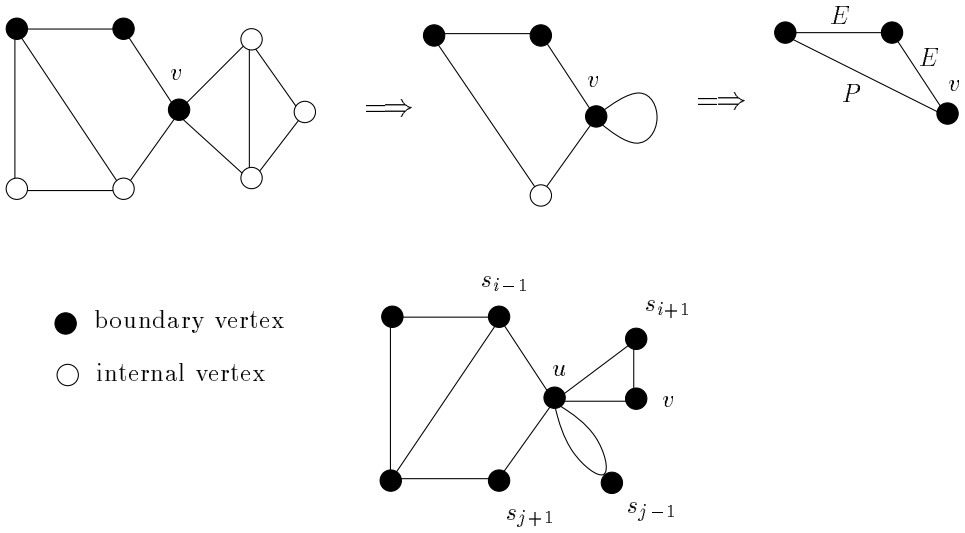


Figure 9.3: Illustrating the proof of Lemma 159.

Proof. Let $b = t + 1$ denote the number of boundary vertices. For $b = 1$ the component must be K_1, so it trivially follows that $l_0(G) = 1$. Likewise, for $b = 2$ the t-parse is acyclic so $l_1(G) = 2$.

For a fixed outer-planarity embedding, suppose that the same boundary vertex v appears on the outer face twice without another boundary vertex between them. The internal vertices between these consecutive locations in Seq form a loop with possible cords (see upper part of Figure 9.3). This means that these internal vertices can be embedded (merged) into any extended outer-planar embedding of the graph minus the "loop." Thus, consecutive identical boundary vertices in Seq are not needed. If $b > 2$ then accounting for boundary repetitions (e.g., vertex u in the lower part of Figure 9.3) we have the following bound.

$$l_t(G) \leq \max_{m \geq 1} \{l_{t-m}(G) + m + 1\}$$

The worst case is when $m = 1$. So after solving the recurrence relation for $l_t(G) \leq l_{t-1}(G) + 2$, we can conclude that $l_t(G) \leq 2t$ for $t \geq 2$. □

Observe that equality holds in the above lemma when G is a star. Furthermore, we conclude with the following corollary.

Corollary 160: There are only a finite number of outer-planarity embeddings for the family of t-parses. And hence, only a finite number of possible outer-planarity states with respect to $\sim_o$.

Proof. The number of combinations of the Seq and Gap vectors is bounded above by $\sum_{i=1}^{t+1} i^{2i} 3^{2i}$. The add-edge list Add is uniquely determined from Seq and Gap and does not add to the number of states. □

9.2.2 A finite-state algorithm

We now present a finite-state algorithm that recognizes those connected pathwidth t-parses that are outer-planar. This dynamic-programming algorithm finds the outer-planarity state of a t-parse G_n by finding, in order, the outer-planarity states of the prefix t-parses G_1, G_2, ..., G_{n-1}, where the subscript n denotes the number of operators scanned. Recall the outer-planar state function f from Section 9.2.1 for $\sim_o$. A t-parse G is outer-planar if and only if $f(G)$ is non-empty.

As mentioned in the previous section, we assume that only one component of any prefix t-parse has order greater than one. Any connected t-parse can be easily converted to this format.

For the base case consisting of an empty t-parse $G_0 = [\textcircled{0}, \ldots, \textcircled{t}]$, the initial state $F_0 = f(G_0)$ is the set of singleton outer-planarity embeddings $\{(Seq = [i], Gap = [E]) \mid 0 \leq i \leq t\}$. The gap 'E' acts as a loop which is dropped later.

For the inductive case, we now show how to find the outer-planarity state $F_{m+1} = f(G_{m+1})$ from the outer-planarity state $F_m = f(G_m)$. If g_{m+1} is a vertex operator at position $m+1$ of G_n we do Case 1 below.

Case 1: $g_{m+1} = \textcircled{i}$

$F_{m+1} \leftarrow \emptyset$;
For all embeddings $E \in F_m$ do
    For all vertices $v_j = \textcircled{i} \in Seq(E)$
        $Gap(E') \leftarrow MergeGaps(Left(Gap(E), j), v_j, Right(Gap(E), j))$;
    end
    $Seq(E') \leftarrow Seq(E) \setminus \{i\}$;

Table 9.1: The *MergeGaps* function for outer-planar **Case 1**.

Left Gap	*Right* Gap	Resulting Gap
E	E	P
E	P	P
E	C	C
P	E	P
P	P	P
P	C	C
C	E	C
C	P	C
C	C	C

$Add(E') \leftarrow \{(j,k) \mid (j,k) \in Add(E)$ and $i \neq j$ and $i \neq k\}$;
$F_{m+1} \leftarrow F_{m+1} \cup E'$;
end

end case

Note that if vertex j is located between two vertex i's in $Seq(E)$ then two j's are not adjacent in $Seq(E')$. Now consider an edge operator at position $m+1$.

Case 2: $g_{m+1} = \boxed{i\ j}$

$F_{m+1} \leftarrow \emptyset$;
For all embeddings $E \in F_m$ do
if vertices i and j are in the same component
if $(i,j) \in Add(E)$
$Seq(E') \leftarrow Seq(E)$;
if gap consists of $\{E|P\}^+$
$Gap(E') \leftarrow Path2Cycle(Gap(E), i, j)$;
else
$Gap(E') \leftarrow Gap(E)$;
end
$Add(E') \leftarrow Add(E) \setminus Crossings(Seq(E'), i, j)$;

$F_{m+1} \leftarrow F_{m+1} \cup E'$;

end

else if both i and j are isolated (i.e., the first edge)

$Seq(E') \leftarrow [i, j]$;

$Gap(E') \leftarrow [E, E]$;

$Add(E') \leftarrow \emptyset$;

$F_{m+1} \leftarrow F_{m+1} \cup E'$;

else (attaching isolated vertex, say j, to component)

For all vertices $v_k = \textcircled{i} \in Seq(E)$

$Seq(E') \leftarrow Left(Seq(E), 0 \ldots k) + j + Right(Seq(E), k \ldots |Seq(E)|)$;

$Gap(E') \leftarrow InsertEdgeGaps(Gap(E), k)$;

$Add(E') \leftarrow Add(E) \cup \{(j, v) \mid v \text{ is } \{E|P\}^+ \text{ away from } i\}$;

$F_{m+1} \leftarrow F_{m+1} \cup E'$;

end

end

end

end case

Note that we embed pendent vertices to the outside face. The gap vector is used in the *Crossings* routine to determine if future edges may be added between boundary vertices without hiding internal vertices that are currently on the outer face.

We now illustrate, with the above outer-planarity algorithm as our example, a standard proof technique for proving correctness of dynamic-programming algorithms (on t-parses).

Theorem 161: Our finite-state outer-planarity algorithm is correct.

Proof sketch. For any t-parse G_m with a set of outer-planar embeddings E_m, let $f(G_m) = E_m$. If G_m is not outer-planar then the set E_m is empty. For any fixed embedding $E \in E_m$ of G_m, let $g(E) = R$, where R is a reduced outer-planar embedding described above (represented by our finite-state algorithm). For a set of embeddings E_m, we also use the notation $g(E_m)$ to denote the image R_m of the map E_m onto

$\{g(E) \mid E \in E_m\}$. For any t-parse operator g_{m+1} in Σ_t, we define three meta-functions $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ that update t-parses, outer-planar embeddings, and reduced outer-planar embeddings, respectively, with the obvious actions for the different g_{m+1}. See Figure 9.4. The function $\mathcal{A}_1 : \Sigma_t^* \times \Sigma_t \to \Sigma_t^*$ is our usual t-parse building function. The function $\mathcal{A}_2$ takes each embedding E of E_m and an operator g_{m+1} and returns all possible extended embeddings. If g_{m+1} is a vertex operator then $\mathcal{A}_2(E, g_{m+1})$ returns exactly one embedding, while if g_{m+1} is a edge operator then $\mathcal{A}_2(E, g_{m+1})$ may return zero or more embeddings. The function $\mathcal{A}_3$ is the finite-state algorithm that we are proving correct.

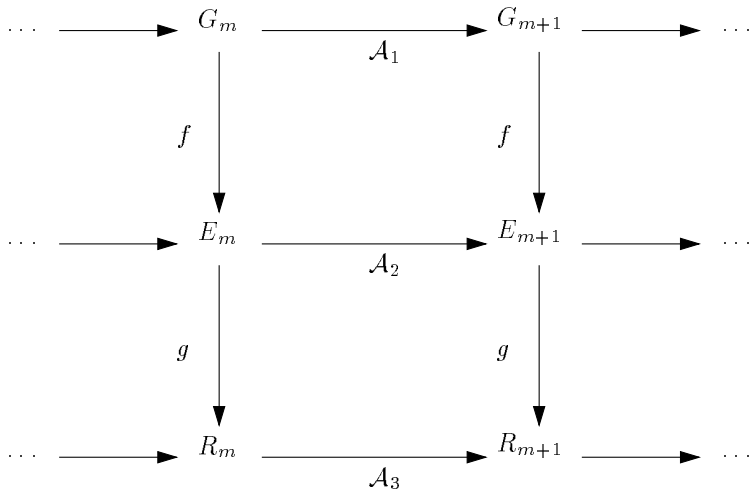


Figure 9.4: A commutative diagram for the proof of Theorem 161.

Notice that if a t-parse G_n has an outer-planar embedding, the set E_n is non-empty, and furthermore by the definition of the function g, the set R_n is non-empty. Otherwise, if G_n is not outer-planar, then both E_n and R_n are empty. Thus, in order to show that the outer-planar t-parse algorithm $\mathcal{A}_3$ is correct, all that we need to do is show that the diagram in Figure 9.4 commutes for each t-parse operator in Σ_t. We leave this messy case analysis to the reader. □

9.3 The 1–Outer Planar Obstructions

Using a universal distinguisher search (pathwidth 4), our list of obstructions for 1–Outer Planar stabilized to the ones listed in Figures 9.5 and 9.6. With consecutive randomized runs occurring without new obstructions appearing, we believe that this set is close to being complete. Each run takes about three weeks to complete using roughly 20 distributed worker processes to prove minimality or guess nonminimality of the t-parses.

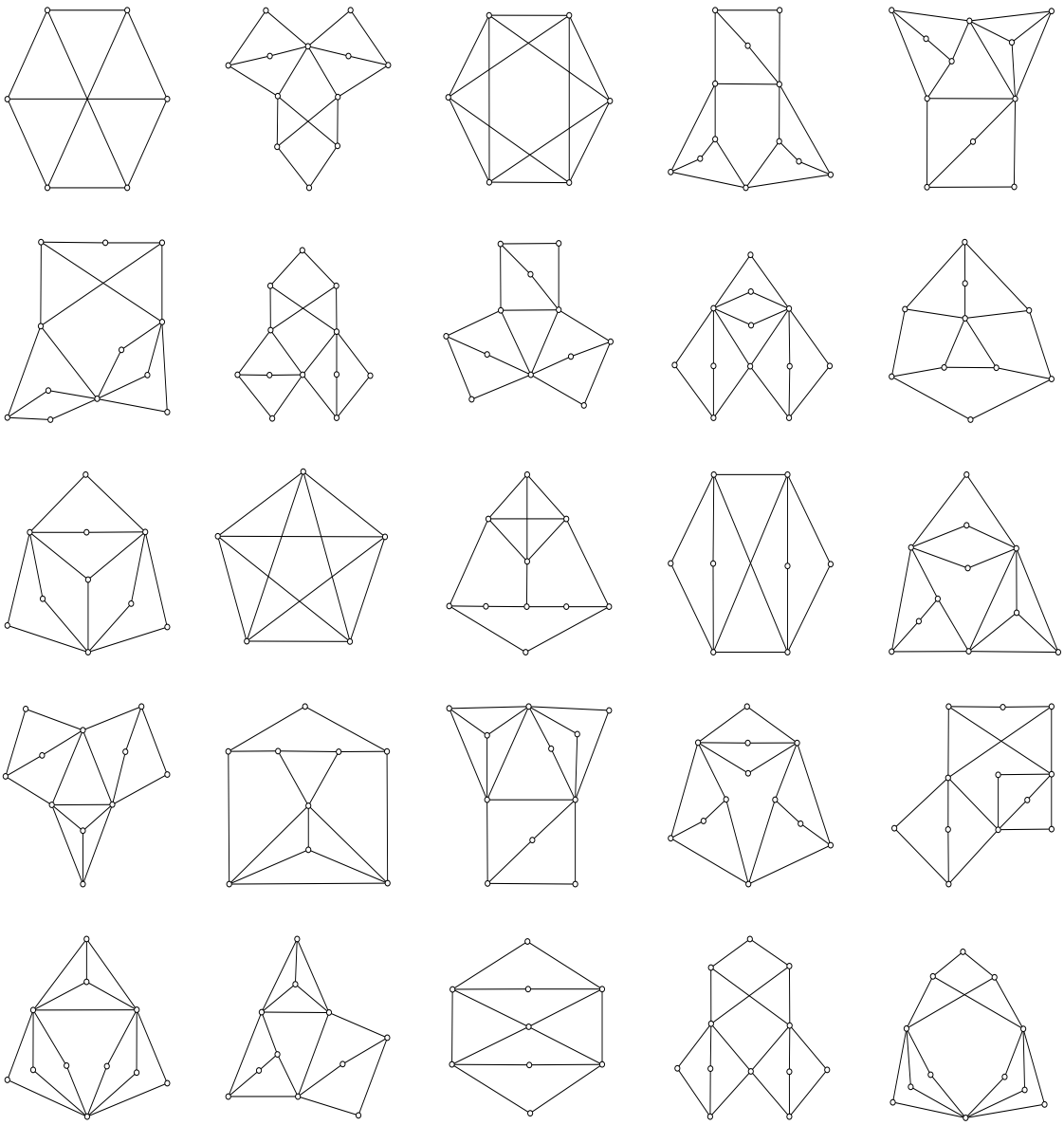


Figure 9.5: Known connected obstructions for 1–Outer Planar.

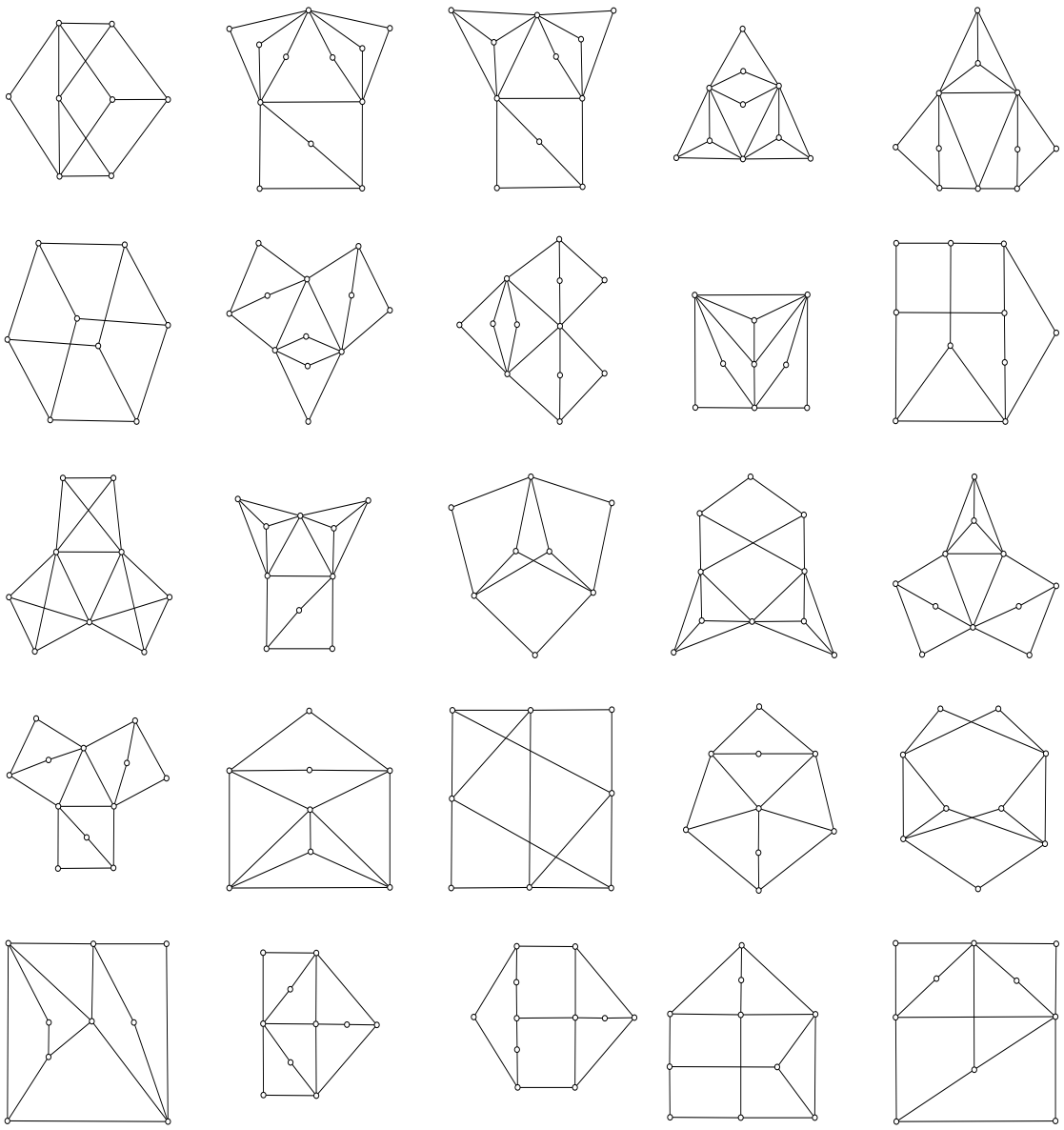


Figure 9.6: Continued: 1–Outer Planar obstructions.

9.3.1 Some other surface obstructions

Besides the previously known sets of 2 planar, 2 outer-planar, and 35 projective plane minor-order obstructions (and now our initial set of 50 1–Outer Planar obstructions), we have started to find the obstructions for the "within 1 vertex of planar" graph family, denoted 1–Planar. All of the connected forbidden minors with at most 10 vertices is displayed in Figures 9.7 and 9.8 on the following pages. Besides these 41 obstructions, one disconnected obstruction, namely $K_5 \cup K_5$ has 10 vertices. The other two disconnected obstructions, $K_5 \cup K_{3,3}$ and $K_{3,3} \cup K_{3,3}$, have 11 and 12 vertices, respectively. We expect that the complete set of obstructions will contain about 200 graphs. Thus, based on our feasibility conjecture, this may be the only "within k vertices of planar" family that can be computed during the foreseeable future. It is interesting to see that 5 of the 7 linkless embedding obstructions (recall Figure 1.8) are also obstructions for this almost planar graph family.

For the next grand-challenge surface to characterize, Neufeld has computed a partial list of toroidal obstructions (i.e., minor-order forbidden torus embeddable graphs) based on a brute force search [Neu93]. Within his set of just over two thousand obstructions, the densest one known has pathwidth 6. See Figure 9.9 for the drawing of this symmetric obstruction, which surprisingly, is the complement of the famous Petersen graph. There are exactly 3, 42 and 455 minor-order obstructions with 8, 9 and 10 vertices, respectively [Neu93]. The eight vertex obstructions are shown in Figure 9.10. Since it appears to be impractical to search through all the graphs on 11 or more vertices, our computational method may be the only feasible approach to classify this surface. However, there is a big gap between our pathwidth lower bound and the currently best known combinatorial-width upper bound (e.g., treewidth 1650 given by Seymour [Sey93]).

To tackle the torus, we have a finite-index congruence for any fixed genus k, based on Eulor's surface formula ($n - m + f = 2 - 2 \cdot k$, where f is the number of faces of a genus k embedding), but unfortunately it is not practical for the anticipated pathwidth bound. Also, we have been working on a t-boundaried graph testset but it also seems to be impractical (i.e., too large) at the moment.

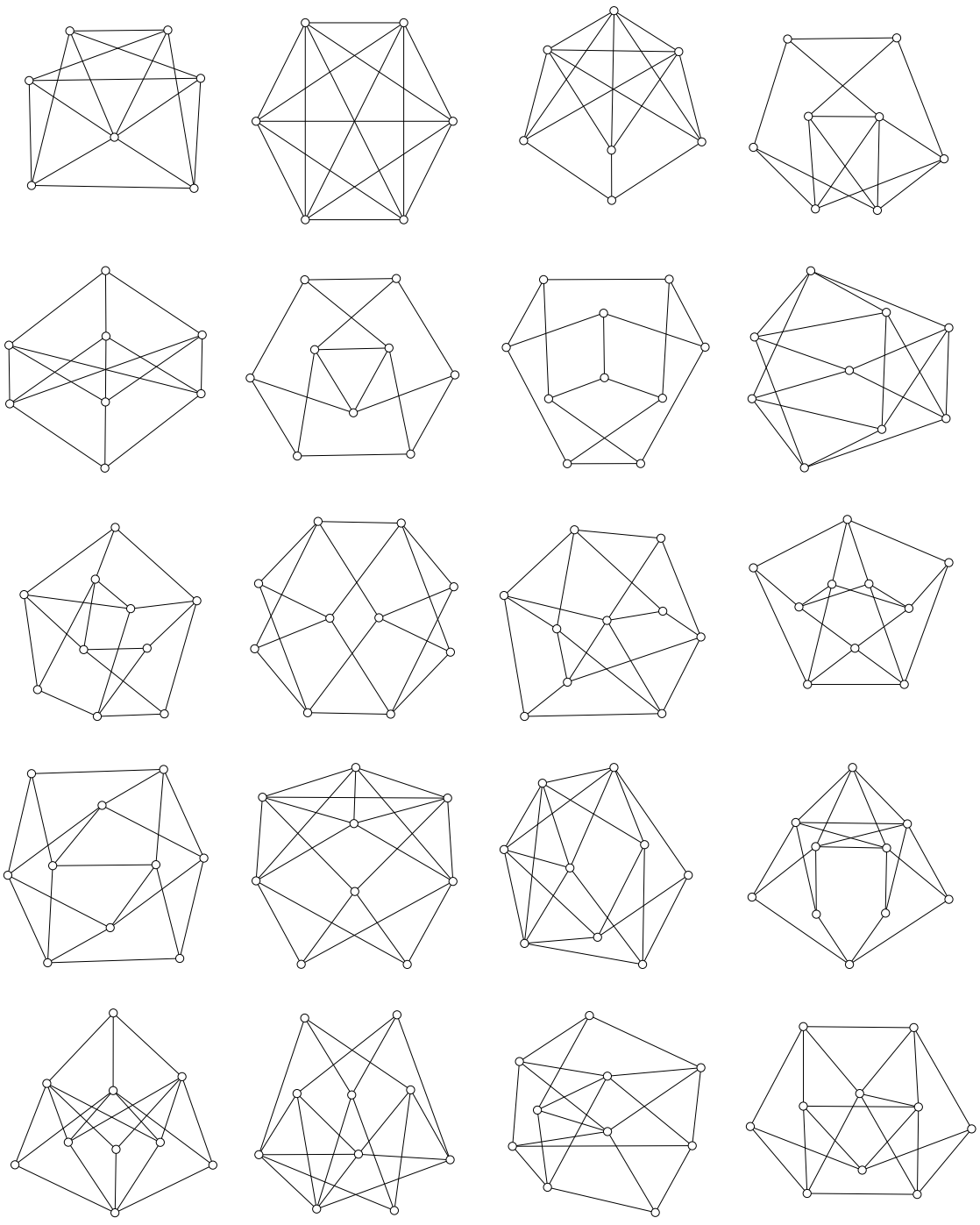


Figure 9.7: All connected obstructions with at most 10 vertices for 1–Planar.

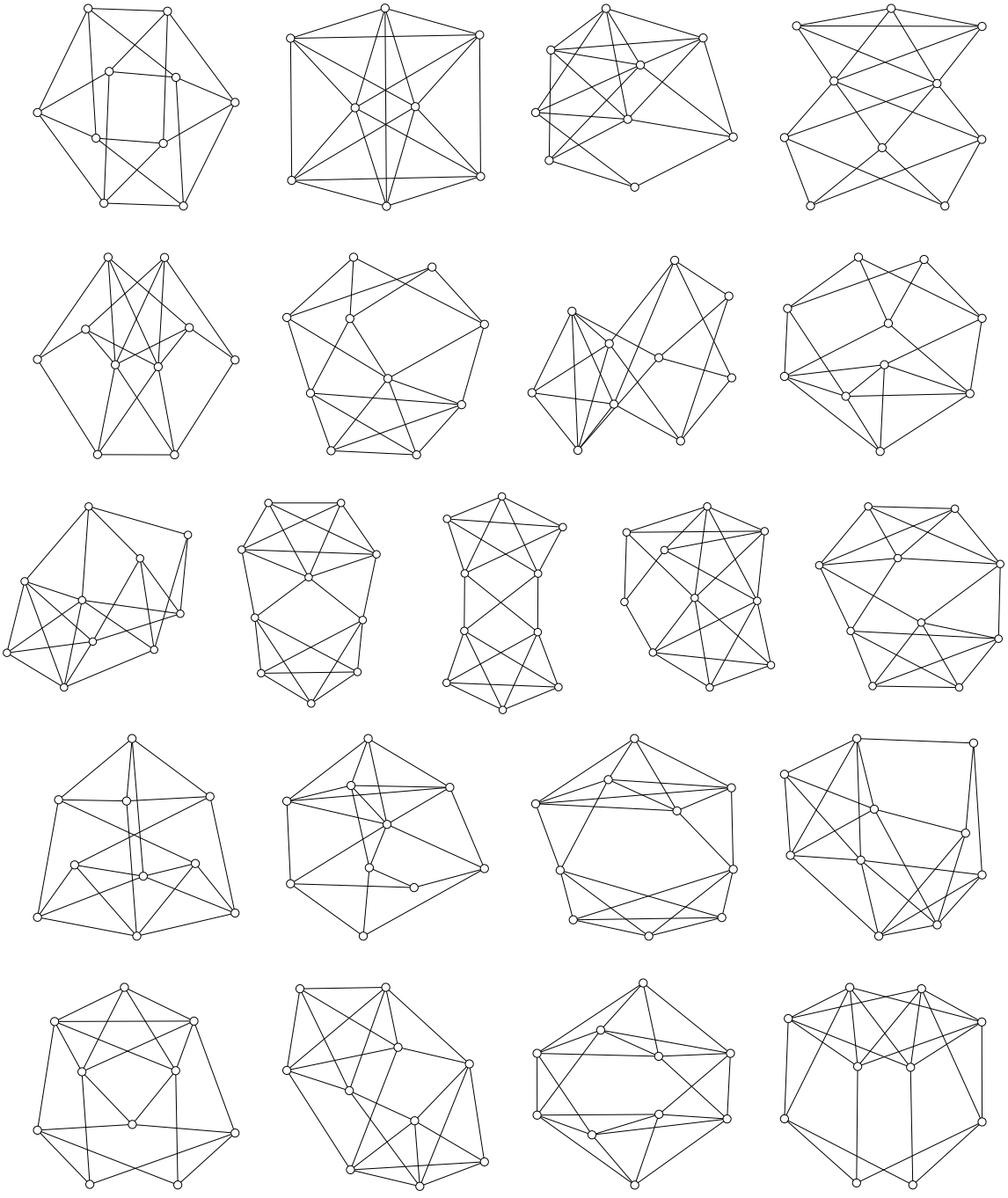


Figure 9.8: Continued: small obstructions for 1–Planar.

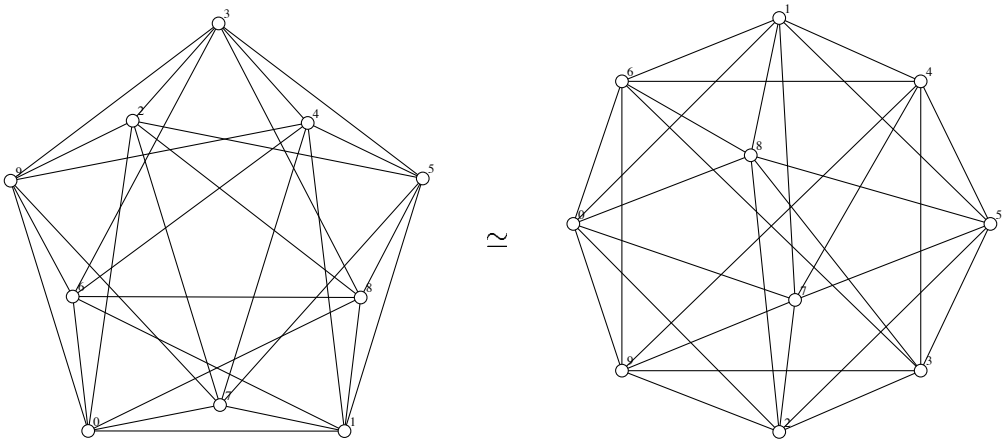
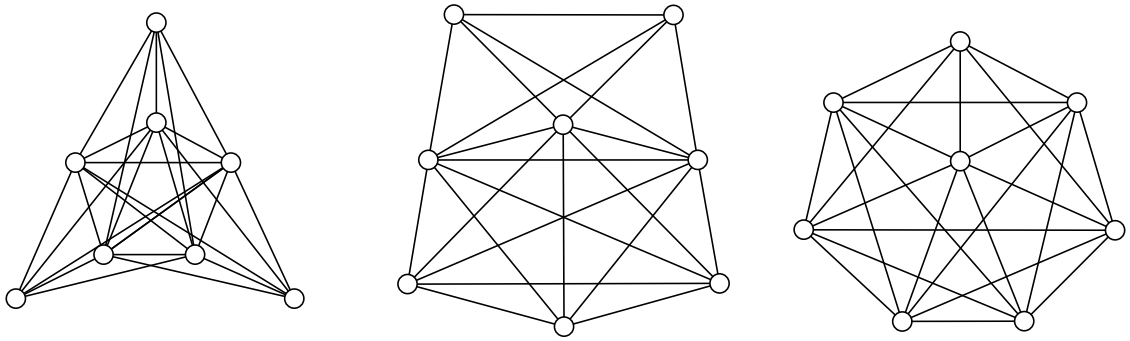


Figure 9.9: A dense symmetric toroidal obstruction with pathwidth 6.



$K_8 \setminus E(C_3)$ $K_8 \setminus (2 \cdot E(K_{1,3}))$ $K_8 \setminus (2 \cdot E(K_2) \cup E(P_2))$

Figure 9.10: The three smallest (8 vertices) toroidal obstructions.

Part III

Applied Connections

Chapter 10

Pathwidth and Biology

In this short chapter, we show that an important problem in computational biology is equivalent to a colored version of a well-known graph layout problem. In order to map the human genome, biologists use graph theory, particularly interval graphs, to model the overlaps of DNA clones (cut up segments of a genome) [Mir94]. For engineers, VLSI circuits must be laid out in order to minimize physical and cost constraints. The vertex separation (see below) of a graph layout is one such measurement of how good a layout is (i.e., the difference between the number of tracks required for the layout and the graph's pathwidth).

10.1 VLSI Layouts and DNA Physical Mappings

The $\mathcal{NP}$-complete combinatorial problem of *Intervalizing a Colored Graph* (ICG) first defined in [FHW93] (and independently given in [GKS93] as the *Graph Interval Sandwich* problem) is intended to be a limited, first-step model for finding DNA physical mappings. For this model, it is assumed that the biologist knows some of the overlaps—for instance, overlaps specified by some probability threshold based on the physical data. The question asked by the ICG problem is whether other edges can be properly added to differently colored vertices to form a colored interval graph.

The *Vertex Separation* (VS) graph problem is related to many diverse problems in computer science besides its importance to VLSI layouts. Lengauer showed that

solving the progressive black/white pebble game (important to compiler theory) and determining the vertex separation for a graph are polynomial time reducible to each other [Len81]. Node search number, a variant of search number [Par76], was shown equivalent to the vertex separation plus one by Kirousis and Papadimitriou [KP86]. From [EST94], the search number is informally defined in terms of pebbling to be the minimum number of searchers needed to capture a fugitive who is allowed to move with arbitrary speed about the edges of the graph. For node search number, a searcher blocks all neighboring nodes without the need to move along an incident edge. (The exact details of the above two pebbling games may be found in [Bie91].)

Kinnersley in [Kin92] has shown that the pathwidth of a graph is identical to the vertex separation of a graph. The concept of pathwidth has been popularized by the theories of Robertson and Seymour (see for example, [RS85b]). Thus, since the gate matrix layout cost, another well-studied VLSI layout problem [KL94, Möh90], equals the pathwidth plus one [FL89b], it also equals the vertex separation plus one.

Our result shows that the *Vertex Separation* problem is important to another area besides computer science, namely computational biology.

10.2 An Equivalence Proof

In this section, we formally define two parameterized problems k-ICG and k-CVS and then show that they are indeed equivalent.

Definition 162: A *layout* L of a graph $G = (V, E)$ is a one to one mapping $L : V \to \{1, 2, \ldots, |V|\}$.

If the order of a graph $G = (V, E)$ is n, we conveniently write a layout L as a permutation of the vertices $(v_1, v_2, \ldots, v_n)$. For any layout $L = (v_1, v_2, \ldots, v_n)$ of G let $V_i = \{v_j \mid j \leq i \text{ and } (v_j, v_k) \in E \text{ for some } k > i\}$ for each $1 \leq i \leq n$.

Definition 163: The *vertex separation* of a graph G with respect to a layout L is $vs(L, G) = \max_{1 \leq i \leq |G|}\{|V_i|\}$. The *vertex separation* of a graph G, denoted by $vs(G)$, is the minimum $vs(L, G)$ over all layouts L of G.

The k-coloring of a graph $G = (V, E)$ is a mapping $color : V \to \{1, 2, \ldots, k\}$. For any subset $V' \subseteq V$, let $Colors(V') = \{color(v) \mid v \in V'\}$.

Definition 164: A *colored layout* L of a k-colored graph $G = (V, E)$ is layout L such that for all $1 \leq i < n$, $color(v_{i+1}) \notin Colors(V_i)$.

Problem 165: Colored Vertex Separation (CVS)
Input: A k-colored graph G.
Parameter: k
Question: Is there a colored layout L of G where $vs(L, G) < k$?

Problem 166: Intervalizing a Colored Graph (ICG)
Input: A k-colored graph $G = (V, E)$.
Parameter: k
Question: Is there a properly colored supergraph $G' = (V', E')$ of G, $E \subseteq E'$, such that $V = V'$ and G' is an interval graph?

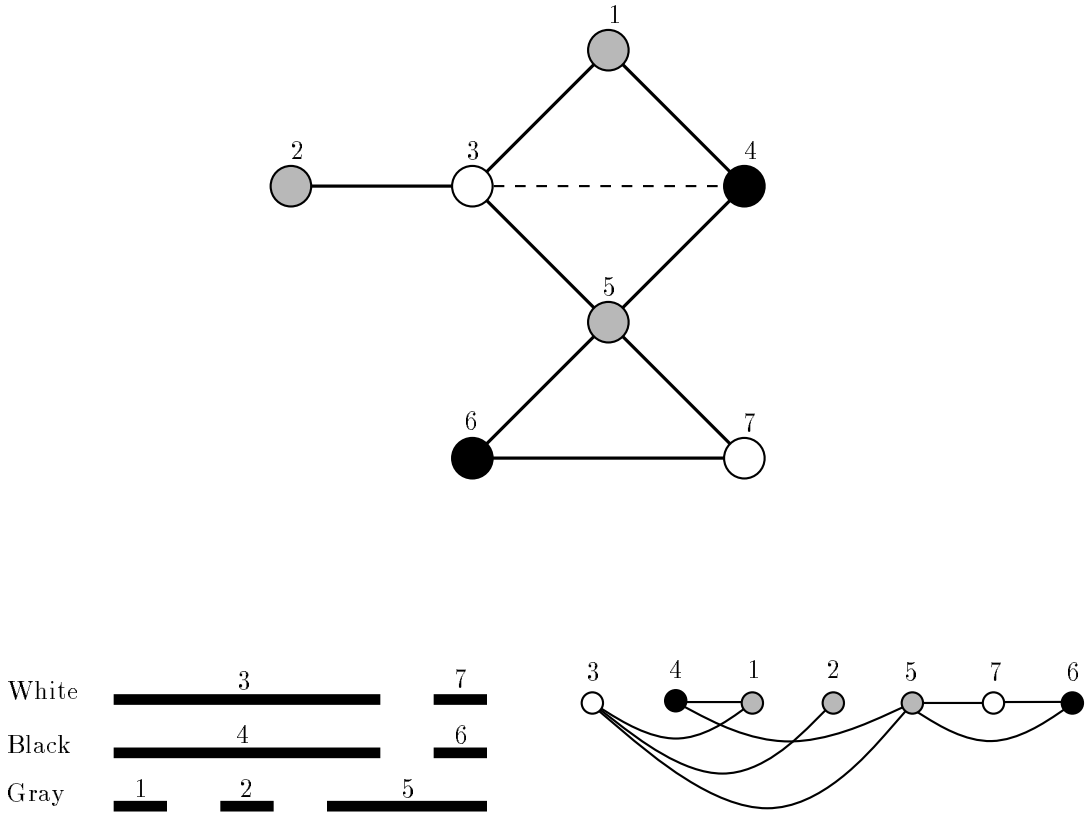
In Figure 10.1 we show a 3-colored graph with an interval supergraph represented on the left and a colored vertex separation layout given on the right. (The dashed edge between the different colored vertices is not an edge in the input graph.)

Theorem 167: For any fixed positive integer parameter k, both k-CVS and k-ICG are identical problems.

Proof. Let $L = (v_1, v_2, \ldots, v_n)$ be a colored layout of a k-colored graph $G = (V, E)$. We show how to construct a properly colored supergraph G' that is also an interval graph. For each vertex $v_i \in V$, define the interval:

$$I_{v_i} = [a_{v_i}, b_{v_i}] = [i, \max\{j \mid (v_i, v_j) \in E \text{ or } j = i\} + 0.5] \quad .$$

By definition, if edge $(u, v) \in E$ then $I_u \cap I_v \neq \emptyset$. Now define the supergraph $G' = (V, E')$ where an edge $(v_i, v_j) \in E'$ whenever $I_{v_i} \cap I_{v_j} \neq \emptyset$. It suffices to show that $color(v_i) \neq color(v_j)$ for each edge (v_i, v_j) in $E' \setminus E$. Without loss of generality, assume $i < j$ so that $b_{v_i} > a_{v_j}$. Again by the definition of I_{v_i}, there exists a vertex



(a) Interval supergraph

(b) Linear CVS layout

Figure 10.1: Illustrating the k-CVS and k-ICG problems.

v_k such that $j < k$ and $(v_i, v_k) \in E$. This implies that $v_i \in V_{j-1}$. (This also holds for $i = j - 1$.) Now L is a colored layout so $color(v_j) \notin Colors(V_{j-1})$. Thus, $color(v_i) \neq color(v_j)$. Therefore, G' is a properly-colored supergraph of G that is intervalizable.

For any k-colored graph $G = (V, E)$ that satisfies ICG, let $\{I_v \mid v \in V\}$ be an interval graph representation of a supergraph $G' = (V, E')$. Let $a_v < b_v$ be the endpoints of the interval $I_v = [a_v, b_v]$ for vertex v. Without loss of generality, assume that $a_u = a_v$ implies $u = v$. Let $L = (v_1, v_2, \ldots, v_n)$ be the unique layout such that $i < j$ if and only if $a_{v_i} < a_{v_j}$. We claim that L is a colored layout of G'. To prove this claim, we show that $color(v_{i+1}) \notin Colors(V_i)$, $1 \leq i < n$. If there exists a vertex $u \in V_i$ such that $color(u) = color(v_{i+1})$ then by definition of V_i vertex u must be

adjacent to a vertex v_j for some $j > i$. Further, $j > i+1$ since the edge (u, v_{i+1}) would not be a properly colored edge. Since $a_u < a_{v_j}$ and $(u, v_j) \in E'$, we must have $b_u > a_{v_j}$ in order to form an overlap. However, $b_u < a_{v_{i+1}} < a_{v_j}$. These inequalities hold since the intervals u and v_{i+1} with the same color do not overlap and $i+1 < j$. We have reached a contradiction. So $u \notin V_i$ if $color(u) = color(v_{i+1})$. Thus L is a colored layout.

Now suppose that for some $r < s$ there exist two vertices v_r and v_s in V_i with the same color. Since $v_r \in V_i$, there exists a vertex v_j with $j > i$ such that $(v_r, v_j) \in E'$. This implies $v_r \in V_{s-1}$. But this implication contradicts the fact $color(v_s) \notin Colors(V_{s-1})$. So $color(v_r) \neq color(v_s)$. Hence any set $V_i \cup \{v_{i+1}\}$ has at most one vertex of each color. Since there are k colors, each V_i must have $k-1$ or fewer vertices. Thus, $vs(L, G) \leq vs(L, G') < k$. □

10.3 Some Comments

Recently, the corresponding general problem of intervalizing a colored graph to a *unit* (proper) interval graph has been shown to be $\mathcal{NP}$-hard (and fixed-parameter hard for W[1]) by Kaplan and Shamir [KS93] (also see [GGKS93, KST94]). The good news from Kaplan and Shamir's paper is that for each parameter k (i.e., k colors) this unit interval problem has a polynomial time decision algorithm.

More recently, determining whether a polynomial time decision algorithm exists for k-ICG, or equivalently k-CVS, has been solved by Bodlaender and de Fluiter [BdF95]. Their main results are a quadratic-time algorithm to test if a 3-colored graph is a subgraph of a properly colored interval graph, and an $\mathcal{NP}$-completeness proof for this problem for 4 colors.

Chapter 11

Automata and Testsets

The main result of this chapter shows that finding a minimum sized testset for any minimal finite state automaton is $\mathcal{NP}$-hard. Also given is a simple graph algorithmic example which takes a boundaried graph testset and builds an automaton. At the end of this chapter, we show how these bounded-width membership automata for lower ideals can be efficiently used to find obstruction sets.

11.1 Introduction

The reader may have seen the following well-known result from classical automata theory (e.g., see [HU79]).

Theorem 168: (Myhill–Nerode) The following two statements are equivalent:

1. A set $L \subseteq \Sigma^*$ is accepted by some finite state automaton. (That is, the set L is a regular language.)
2. Let $\sim_L$ be an equivalence relation for x and y in Σ^* defined by: $x \sim_L y$ if and only if for all z in Σ^*, xz is in L exactly when yz is in L. The relation $\sim_L$ is of finite index.

For any regular language L over the alphabet Σ, two strings x and y are *distinguished* by a string z in Σ^* if ($xz \in L$ and $yz \notin L$) or ($yz \in L$ and $xz \notin L$). Otherwise,

x and y are said to be *congruent* with respect to the *canonical congruence* $\sim_L$. A set $T \subseteq \Sigma^*$ is a *testset* for the language L if every two strings x and y of Σ^* that are in different equivalence classes of $\sim_L$ are distinguished by some test z in T.

The proof of the Myhill–Nerode Theorem shows that if M is a *minimal* finite state automaton for a regular language L then there is a one-to-one correspondence between the states of M and the equivalence classes of $\sim_L$.

Since only one test is needed to distinguish any two states of an automaton M, clearly a testset T need not contain more than $\binom{|M|}{2}$ tests. Likewise, since a single test can only partition the states into two classes, a testset T needs to contain at least $\lceil \log_2(|M|) \rceil$ tests. An obvious problem arises on how to find the smallest such testset for an arbitrary regular language L. Regrettably, we present some negative results concerning this task in Section 11.2.

To illustrate the relationship between testsets and regular language decision problems, we give in Section 11.3 a testset for the graph connectivity decision problem for graphs of bounded pathwidth and treewidth. In fact, many graph decision problems for bounded pathwidth or treewidth can be solved in linear time by the use of automata (e.g., see [AF93]). There are *no* large hidden constants in the running times of these algorithms. A graph building alphabet (e.g., the alphabet Σ_t for t-parses) is used to represent the graphs of bounded combinatorial width. Each graph contains a special set of boundary vertices where tests can be appended. In this context, these bounded-width graph families are regular languages. There are many applications for these finite state automata such as (1) testing for VLSI layout properties (e.g., see [KK94b]) and (2) finding obstruction set characterizations (which is briefly discussed in Section 11.4).

11.2 Finding a Minimum Testset is $\mathcal{NP}$-hard

This section shows that for an arbitrary minimal finite state automaton determining the size of a minimum testset is at least as hard as any problem in the complexity class $\mathcal{NP}$. A formal definition of our testset problem now follows.

Problem 169: Automata Minimum Testset (AMT)
Input: A minimal deterministic finite automaton, DFA, $M = (Q, \Sigma, \delta : Q \times \Sigma \rightarrow Q, q_0 \in Q, F \subseteq Q)$, and a positive integer k.
Question: Does there exist a testset $T \subseteq \Sigma^*$ for M with $|T| \leq k$? That is, for a solution T, and all $q_i, q_j \in Q$, there exists a test $t \in T$ such that ($[q_i]t \in F$ and $[q_j]t \notin F$) or ($[q_i]t \notin F$ and $[q_j]t \in F$) where $[q]$ denotes any equivalence class representative for the state q.

If we can solve this AMT problem in polynomial time for each input k then we can determine in polynomial time the least k such that an automaton M has a testset of cardinality k (i.e., the optimization problem would also be in the complexity class $\mathcal{P}$). A related and classic problem is the following.

Problem 170: Minimum Test Collection for sets (MTC)
Input: A finite set S of "possible diagnoses", a collection C of subsets of S, representing binary "tests", and a positive integer $j \leq |C|$.
Question: Does there exist a subset C' of C, $|C'| \leq j$, such that for every pair $x, y \in S$, there is some test c in C' for which $|\{x, y\} \cap c| = 1$ (here, a test c eliminates one of the two diagnoses x and y)?

Our main result of this chapter is now presented.

Theorem 171: The decision problem AMT is $\mathcal{NP}$-hard.

Proof. Gary and Johnson have shown that MTC is $\mathcal{NP}$-complete [GJ79]. We show that AMT can be restricted to MTC. This then shows that AMT is $\mathcal{NP}$-hard.

Given an instance (S, C) of MTC with $j \leq |C|$, we build an automaton $M = (Q, \Sigma, \delta, q_0, F)$ as displayed in Figure 11.1. For convenience, we assume (relabel if necessary) that the finite set S equals $\{1, 2, \ldots, n\}$. The state $q_i \in Q$ corresponds to a particular $i \in S$ for $1 \leq i \leq n$.

We first show that the automaton M is minimal. Let L be the language accepted by M. Our automaton is nonminimal if and only if there exists two states q and q' such that $[q]z \in L \Leftrightarrow [q']z \in L$ for all $z \in \Sigma^*$. The empty test distinguishes F from

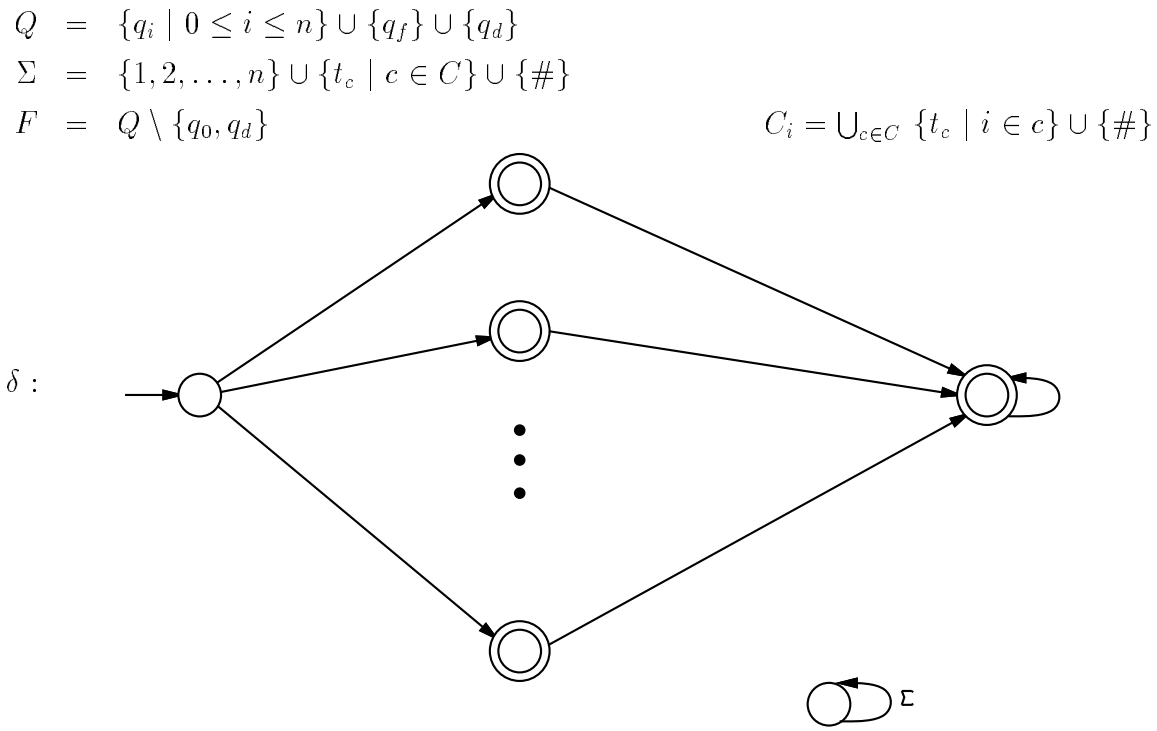


Figure 11.1: The AMT automaton M built from an MTC instance.

$\{q_0, q_d\}$. The test # distinguishes state q_f from the states $q_1, q_2, \ldots, q_n$ and between states q_0 and q_d. Thus, the only way that the automaton M is nonminimal is if there are two diagnoses in S that share identical tests in C. In this case, which is easily checked in polynomial time, there does not exist any $C' \subseteq C$ to solve the MTC problem. Hence, hereafter M is assumed to be minimal.

To complete the reduction, we set k, the inputed positive integer for AMT, to be $j + 2$ where j was the queried positive integer for MTC. We claim the MTC problem has a collection $C' \subseteq C$ such that $|C'| \le j$ if and only if the AMT problem has a testset T such that $|T| \le k$.

Assume that the MTC instance has a subset C' of C such that $|C'| \le j$. We claim that the set $T = \{t_c \mid c \in C'\} \cup \{1, \#\}$ is a testset for M. The tests 1 and # are sufficient to distinguish states q_0, q_d, and q_f from the rest of Q.

1. Test # distinguishes q_d from $Q \setminus \{q_f\}$ and test 1 distinguishes q_d and q_f.

2. Test 1 distinguishes q_f from $Q \setminus \{q_0\}$ and test # distinguishes q_f and q_0.

3. Test 1 distinguishes q_0 from $Q \setminus \{q_f\}$ and test # distinguishes q_0 and q_f.

States q_a and q_b, $1 \leq a < b \leq n$, are distinguished by some t_c since $|\{a,b\} \cap c| = 1$ for some c in C'. If $\{a,b\} \cap c = \{a\}$ then by the definition of $C_i = \bigcup_{c \in C} \{t_c \mid i \in c\} \cup \{\#\}$ we have $[q_i]t_c \in L$ and $[q_j]t_c \notin L$. Since $|T| \leq k$, the corresponding AMT instance is also true.

We now show that if the AMT instance is true then the MTC instance must also be true. First note that any test prefixed with a symbol t_c does not help in distinguishing the three states $\{q_0, q_f, q_d\}$. Similarly, any test prefixed with $\{\#, 1, 2, \ldots, |S|\}$ does not help in distinguishing $\{q_i \mid i \in S\}$. Since all of the possible tests in Σ^* have been exhausted, we know that it requires independent tests $T = T_1 \cup T_2$ to distinguish all of the states of Q, that is, a set T_1 for $\{q_0, q_f, q_d\}$ and a set T_2 for $\{q_i \mid i \in S\}$.

At least two tests are needed to show that the three states q_0, q_f, and q_d are not equivalent since $\lceil \log_2(3) \rceil = 2$. So, without loss of generality, we can assume $T_1 = \{0, \#\}$ distinguishes states q_0, q_d, and q_f from the rest of Q. This implies that j or fewer tests is sufficient for a complete T_2. Any significant test in T_2 must be of length 1, for otherwise, it does not distinguish anything in $\{q_i \mid i \in S\}$. Those $j' \leq j$ tests in T_2 that distinguish between all the states of S correspond to j' subset collections of C. Thus, the MTC instance must be true. □

The above construction builds a finite automaton that has a minimum testset with single character tests. It would be interesting to find a construction that bounds the alphabet size $|\Sigma|$ while allowing longer test lengths, since more natural problems seem to be of this type.

It is still an open problem whether AMT is $\mathcal{NP}$-complete. Two standard means of attack for the general problem are (1) to show AMT is in $\mathcal{NP}$, and (2) to find a reduction from AMT to a known $\mathcal{NP}$-complete problem. For method (1) we have the following result:

Lemma 172: Given an automaton $M = (Q, \Sigma, \delta, q_0, F)$ there exists a test of length at most $|Q| \cdot (|Q| - 1)$ to distinguish any two states of M.

Proof. For any two non-equivalent states q and q' let $t = a_1a_2 \ldots a_i \ldots a_j \ldots a_n$ be a distinguisher, where each $a_i \in \Sigma$. For any prefix $t_i = a_1a_2 \ldots a_i$ of t, the state $[q]t_i$ must be different than the state $[q']t_i$ since the automaton M is deterministic and this would force $[q]t = [q']t$. If the state pair $([q]t_i, [q']t_i)$ equals $([q]t_j, [q']t_j)$ for some j, $1 \leq i < j \leq n$, then $t' = a_1a_2 \ldots a_ia_{j+1} \ldots a_n$ would be a shorter test. Thus the number of distinct ordered state pairs, $|Q| \cdot (|Q| - 1)$, is an upper bound on the needed test length for two non-equivalent states. □

If we are interested in a restricted version of the AMT problem where the tests have a bounded (polynomial) length then this problem is $\mathcal{NP}$-complete. In light of the above fact, we define the following problem.

Problem 173: Bounded Automata Minimum Test Set (BAMT)
Input: A minimal deterministic finite automaton, DFA, $M = (Q, \Sigma, \delta, q_0, F)$, and a positive integer k.
Question: For the automaton M, does there exist a testset $T \subset \Sigma^*$ with $|t| \leq |Q| \cdot (|Q| - 1)$ for all $t \in T$ and $|T| \leq k$?

Corollary 174: The decision problem BAMT is $\mathcal{NP}$-complete.

Proof. First observe that the length of any useful test $t \in T$ given in the proof of Theorem 171 has a bounded length of 1. So using the previous restriction to MTC instances, we see that BAMT is $\mathcal{NP}$-hard. All that remains is to show that in polynomial time one can verify a solution to BAMT. For each test $t \in T$ one needs to check at most $\binom{|M|}{2}$ states of the automaton M to verify that each pair of states has a distinguisher. It takes at most $|M| \cdot (|M| - 1)$ steps to run each test through the automaton since that is the maximum length of any test $t \in T$. Thus, the verification takes polynomial time. □

11.3 A Testset Example: Building Membership Automata

We now present a testset application for input graphs of bounded combinatorial width. Many different sets of graph building operators can be used to construct graphs, as we saw in Section 2.2.3. We illustrate the following application with our quadratic-sized operator alphabet $\Sigma_t = V_t \cup E_t$ for pathwidth t-parses (see Chapter 2). Later in Section 11.3.1 we develop a setting for the bounded treewidth case.

Assume that we have a graph family $\mathcal{F}$ with a finite-index canonical congruence $\sim_{\mathcal{F}}$. This implies that the bounded-width canonical congruence over the pathwidth operator set Σ_t is finite-index (i.e., this family $\mathcal{F} \subseteq \Sigma_t^*$ is a regular language). Recall from Section 4.1 that there is a slight distinction between these two canonical congruences. In the bounded-width case, a language $\mathcal{F}$ is a subset of the graphs (t-parses) of pathwidth at most t. One can view a test Z for $\sim_{\mathcal{F}}$ as any $(t+1)$-boundaried graph, preferably representable by a string of operators. Recall that a t-parse G and a test Z are combined to form a new graph $G \oplus Z$ by taking the union of the underlying boundaried graphs G and Z, except that the boundary vertices are coalesced. We say that G *passes* the test Z if $G \oplus Z$ is in $\mathcal{F}$. (Note that we could just as easily use the definition "G passes the test $Z \in \Sigma_t^*$ if $G \cdot Z$ is in $\mathcal{F}$," where we use the concatenation operator for t-parses instead of the circle plus operator.)

We now present a simple example. Consider the family $\mathcal{F}$ of connected graphs of pathwidth at most t and order at least $t+1$. For each member G of $\mathcal{F}$ assign a vertex boundary of size $t+1$ that corresponds to a boundary of a t-parse representation of G. We see that the number of equivalence classes for $\sim_{\mathcal{F}}$ equals the number of set partitions of $\{0, 1, \ldots, t\}$ plus one. The equivalence class for an arbitrary $(t+1)$-boundaried graph G is determined as follows:

1. If the graph G has a component without any boundary vertices then it is in a "dead state" equivalence class.

2. Otherwise, the components of G partition the set of $t+1$ boundary vertices (i.e., boundary vertices u and v are contained in the same set of the partition if there

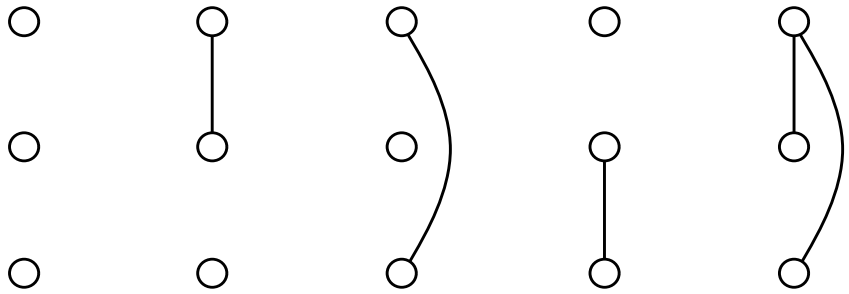


Figure 11.2: A graph connectivity testset for 3-boundaried graphs.

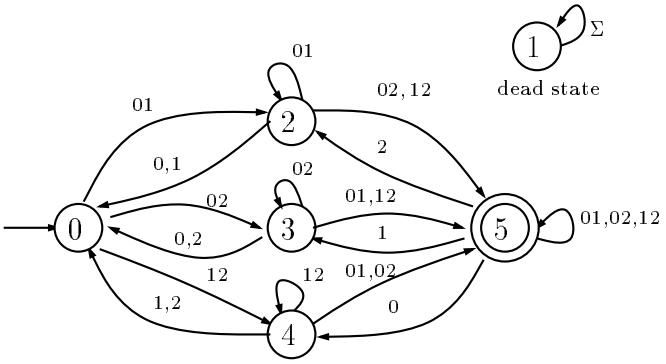
exists a path between u and v in G). Take this partition as the equivalence class index for G.

It is easy to see that if two boundaried graphs G and H fall into different equivalence classes then there exists a test Z, consisting of only edges, such that exactly one of the two graphs $G \oplus Z$ and $H \oplus Z$ is connected. Since these small tests can be represented as t-parses, the bounded-width canonical congruence is equivalent to $\sim_{\mathcal{F}}$. We use this observation to build a finite testset for the graph connectivity problem for any set of k-boundaried graphs.

Example 175: The set of five 3-boundaried graphs, displayed in Figure 11.2, is a testset for graphs within Σ_2^*. A 2-parse is connected if and only if is in the same equivalence class that is indexed by passing all of the tests (in particular, passing the first test implies passing the other tests).

Generally speaking, if we know (1) a testset for the family $\mathcal{F}$ and (2) a membership algorithm for $\mathcal{F}$, then we can compute a minimal finite state automaton that recognizes any graph of pathwidth t within $\mathcal{F}$. This process is mentioned shortly. The next example presents the constructed automaton obtained from the graph connectivity testset of Example 175.

Example 176: Consider the alphabet $\Sigma = \Sigma_2$, the alphabet for 2-parses. A pictorial display of the automaton generated from the connectivity testset (given in



State \ Symbol	(0)	(1)	(2)	[0 1]	[0 2]	[1 2]
0 – initial state	1	1	1	2	3	4
1 – dead state	1	1	1	1	1	1
2 – another state	0	0	1	2	5	5
3 – another state	0	1	0	5	3	5
4 – another state	1	0	0	5	5	4
5 – accept state	4	3	2	5	5	5

Figure 11.3: A graph connectivity membership automaton for 2-parses.

Figure 11.2) is shown above in Figure 11.3. Note that the missing arcs go to the dead state. Listed below the automaton is the transition diagram $\delta : Q \times \Sigma \to Q$.

We construct membership automata, such as the one just presented, by the following simple rules. The two ingredients in building an automaton M for accepting t-parses with respect to a finite-state graph family $\mathcal{F}$ are:

1. A testset T of $(t+1)$-boundaried graphs for the canonical congruence $\sim_{\mathcal{F}}$.

2. Any membership algorithm $\mathcal{A}$ for recognizing graphs in $\mathcal{F}$.

During the building process of M we keep an equivalence class representative G_i (t-parse) for each state q_i of M as encountered. Initially, for the start state q_0 the smallest t-parse $G_0 = [\,(0), \ldots, (t)\,]$ is our only representative. Next, for each operator $\sigma \in \Sigma_t$ we check if $G_0 \cdot [\sigma]$ is congruent (via the testset) to any state representative kept so far. Recall that we can easily check whether two t-parses G_1 and G_2 are

congruent, $G_1 \sim_{\mathcal{F}} G_2$, by using T and $\mathcal{A}$:

$$(\text{for every } T_i \in T,\ G_1 \oplus T_i \in \mathcal{F} \iff G_2 \oplus T_i \in \mathcal{F}) \iff G_1 \sim_{\mathcal{F}} G_2 \quad .$$

If it happens that $G_0 \cdot [\sigma]$ is congruent to some state representative G_i then a transition entry from (q_0, σ) to q_i is added to the transition function δ. Otherwise, $G_0 \cdot [\sigma]$ must be a representative for a new state q_i. We repeat the process of appending each single operator $\sigma \in \Sigma_t$ to each new state representative G_i, as they are discovered, and do congruence checks for $G_i \cdot [\sigma]$ against all previously encountered state representatives. The automaton is built when all the possible operator transitions from the representatives for each G_i lead to previous states (i.e., when the transition function δ is not a partial function.) It is safe to terminate the construction at this point since every state of $\sim_{\mathcal{F}}$ is reached from the start state by some string of operators.

11.3.1 Using tree automata for bounded treewidth

We can easily extend the previous testset application for treewidth t-parses. Here we build a tree automaton instead of a linear automaton from a testset for the canonical congruence $\sim_{\mathcal{F}}$. General information about tree automata may be found in the survey paper [Tha73]. We are interested in (rooted) tree automata that evaluate their input arguments (parse trees) in a leaf to root order. For treewidth t-parses we have the following customized definition.

Definition 177: A *tree automaton* $M = (Q, \Sigma = \Sigma_t \cup \{\oplus\}, \delta, \delta_2, q_0, F)$ *for the treewidth* t*-parse alphabet* is an extended linear (pathwidth) automaton $(Q, \Sigma_t, \delta, q_0, F)$ where

1. Q is the set of states.
2. Σ is the alphabet for the input trees.
3. δ is a function from $Q \times (V_t \cup E_t)$ to Q (i.e., δ is a transition function for the unary operators Σ_t).
4. δ_2 is a function from $Q \times Q$ to Q (i.e., δ_2 is a transition function for the binary operator $\oplus$).

5. $q_0 \in Q$ is the start state.

6. $F \subseteq Q$ is the accepting (membership) states.

Just as a linear automaton determines a state for every prefix of an input string, a tree automaton determines a state for every subtree of an input tree of symbols (while processing the input). A t-parse G is accepted by a tree automaton M if the state assigned to the root symbol (operator) is in the set of accepting states F. The rules for evaluating a treewidth t-parse G with respect to M are defined recursively as follows, where $eval_M$ denotes the evaluation map.

1. If $G = [\textcircled{0}, \textcircled{1}, \ldots, \textcircled{t}]$ then $eval_M(G) = q_0$.

2. If $G = G_1 \oplus G_2$ then $eval_M(G) = \delta_2(eval_M(G_1), eval_M(G_2))$.

3. If $G = G_1 \cdot [g_n]$, $g_n \in \Sigma_t = V_t \cup E_t$, then $eval_M(G) = \delta(eval_M(G_1), g_n)$.

An iterative version (for implementation reasons) of this evaluation map $eval_M$ is easy to construct. Here we first run the pathwidth automaton starting at each leaf operator until a $\oplus$ operator is reached. Then the δ_2 function is used to merge together two pathwidth branches (or treewidth subtrees). The procedure then continues up the input tree assigning states to operators using both the δ and δ_2 transition functions.

For an example of a tree automata for t-parses, we extend our graph-connectivity linear automaton of Example 176.

Example 178: A tree automaton that accepts treewidth 2-parses is obtained by adding the following function δ_2 to the linear automaton presented in Figure 11.3.

δ_2	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	1	1	1	1	1
2	2	1	2	5	5	5
3	3	1	5	3	5	5
4	4	1	5	5	4	5
5	5	1	5	5	5	5

Notice that since the binary operator $\oplus$ is commutative the δ_2 function should be symmetric for $\sim_{\mathcal{F}}$ (i.e., for all $x, y \in Q$, $\delta_2(x, y) = \delta_2(y, x)$).

Before ending this section, we need to justify that our definition of tree automata for treewidth t-parses is sound. The next previously known result shows that it is sufficient to add a transition function δ_2 to a linear automaton for pathwidth t-parses.

Lemma 179: For any canonical congruence for a regular language family $\mathcal{F}$, if $X_1 \sim_{\mathcal{F}} X_2$ and $Y_1 \sim_{\mathcal{F}} Y_2$ then $X_1 \oplus Y_1 \sim_{\mathcal{F}} X_2 \oplus Y_2$.

Proof. Since $X_1 \sim_{\mathcal{F}} X_2$, we have for each boundaried graph Z,

$$X_1 \oplus Z \in \mathcal{F} \iff X_2 \oplus Z \in \mathcal{F}$$

which implies

$$X_1 \oplus (Y_1 \oplus Z) \in \mathcal{F} \iff X_2 \oplus (Y_1 \oplus Z) \in \mathcal{F} \quad .$$

By associativity of $\oplus$, we see that $X_1 \oplus Y_1 \sim_{\mathcal{F}} X_2 \oplus Y_1$. Likewise, since $Y_1 \sim_{\mathcal{F}} Y_2$ we get

$$Y_1 \oplus (X_2 \oplus Z) \in \mathcal{F} \iff Y_2 \oplus (X_2 \oplus Z) \in \mathcal{F} \quad .$$

Or, $Y_1 \oplus X_2 \sim_{\mathcal{F}} Y_2 \oplus X_2$. Now since $\sim_{\mathcal{F}}$ is both a symmetric and a transitive relation we see that $X_1 \oplus Y_1 \sim_{\mathcal{F}} X_2 \oplus Y_2$. □

From the above result we can always first build a membership automaton for pathwidth t-parses and then add a transition table δ_2 (for the $\oplus$ operator case) to get a membership tree automaton. Here, the testset method of the previous section is first applied to produce the $m = |Q|$ equivalence classes of $\sim_{\mathcal{F}}$. Then for each unordered pair of state representatives G_i and G_j, $0 \leq i \leq j < m$, we use the testset to find the equivalence class of $G_i \oplus G_j$. Recall that there are at least three flavors of the canonical congruence $\sim_{\mathcal{F}}$:
'$(t+1)$-boundaried graph $\sim_{\mathcal{F}}$' $\Rightarrow$ 'treewidth t-parse $\sim_{\mathcal{F}}$' $\Rightarrow$ 'pathwidth t-parse $\sim_{\mathcal{F}}$' .
We extend a pathwidth t-parse automaton to get a treewidth t-parse automaton if a testset for one of the first two canonical congruences is available.

11.4 Quickly Finding Obstructions using Automata

Suppose we have a minimal finite state automaton M that accepts t-parses in some minor-order lower ideal $\mathcal{F}$. It is straightforward, as seen below, to compute the minor-order obstructions of width at most t for $\mathcal{F}$ using M. This method can use either linear or tree automata.

The states of a minimal automaton M for $\mathcal{F}$ represent the equivalence classes of the canonical congruence $\sim_{\mathcal{F}}$ (by the Myhill–Nerode Theorem). In a breadth-first or depth-first order we start enumerating all t-parses, possibly only the free-boundary isomorphic t-parses (as described in Section 2.3). The search process is pruned whenever a t-parse G has a boundaried minor that falls in the same state as G (i.e., a nonminimal t-parse is found). We can also restrict ourselves to one-step ∂-minors as was justified in Chapter 4. The minor-order obstructions can then be extracted from the minimal t-parses that fall in the non-accept (out-of-family) state of the automaton. Since $\mathcal{F}$ is a lower ideal there are no transitions out of a non-accept state of M. (It follows from the definition of the canonical congruence $\sim_{\mathcal{F}}$ that there is only one of these out-of-family states.)

Notice that a depth-first enumeration order is preferred. We keep in memory each t-parse G only as long as there is an active t-parse G' that has G as a prefix parse. A t-parse in a search tree is called *active* if it has is not been proven minimal or nonminimal.

Example 180: Using the finite state automaton for MaxPath(4) over Σ_1 (see Table 8.1 on page 172), we can compute the single obstruction P_5, a path of length 5, for the family of graphs that have all paths of length 4 or less. A t-parse representation for this obstruction is given below.

$$P_5 = [\textcircled{0}, \textcircled{1}, \boxed{0\ 1}, \textcircled{0}, \boxed{0\ 1}, \textcircled{1}, \boxed{0\ 1}, \textcircled{0}, \boxed{0\ 1}, \textcircled{1}, \boxed{0\ 1}]$$

```
Prefix states:  -,  0,  1,  2,  4,  6,  8, 10,  12, 14,  16
```

To help follow the t-parse P_5 through the MaxPath(4) automaton, we listed the states for each prefix t-parse immediately below the last operator.

Besides these prefix t-parses the smallest search tree contains just two additional t-parses (both nonminimal), namely

$$[\textcircled{0}, \textcircled{1}, \boxed{0\ 1}, \textcircled{0}, \boxed{0\ 1}, \textcircled{0}] \quad \text{and}$$
$$[\textcircled{0}, \textcircled{1}, \boxed{0\ 1}, \textcircled{0}, \boxed{0\ 1}, \textcircled{1}, \boxed{0\ 1}, \textcircled{1}] \quad .$$

If we have an automaton for a lower ideal then we may not want to restrict ourselves to the smallest possible search tree since minimality/nonminimality proofs are easy to obtain. Thus, we can avoid the t-parse canonic checks that were mentioned in Part I of this dissertation. Here we do more isomorphism checks at the end of the obstruction set computation since there are more boundaried obstructions.

The reference counts in Table 8.1 indicate how many times each transition was used during that particular obstruction search. Besides every t-parse in the search tree, the membership automaton is invoked for every one-step minor. The counts indicate that many transitions were not used. We believe that some unnecessary work was used during the creation of the automaton. If the goal is only to compute obstruction sets for a lower ideal, and an automaton is not available, we recommend using the techniques of Chapter 4.

Chapter 12

Computing Pathwidth by Pebbling

After several rounds of improvement [RS95b, Lag90, Ree92] the best known algorithm for finding tree decompositions is due to Bodlaender [Bod93c]. For each fixed k, this algorithm running in time $O(2^{k^2}n)$ determines that either the treewidth is greater than k, or produces a tree decomposition of width at most k. By first running this algorithm and then applying the algorithm of [BK91, Klo93a], a similar result holds for pathwidth. Both of the algorithms involved are quite complicated.

We describe in this chapter a very simple algorithm based on "pebbling" the graph using a pool of $O(4^k)$ pebbles, that in linear time (for fixed k), either determines that the pathwidth of a graph is more than k, or finds a path decomposition of width at most the number of pebbles actually used. The main advantages of this algorithm over previous results are: (1) the simplicity of the algorithm and (2) the improvement of the hidden constant for a determination that the pathwidth is greater than k. The main disadvantage is in the width of the resulting "approximate" decomposition when the width is less than or equal to k.

12.1 Preliminaries

An *(homeomorphic) embedding* of a graph $G_1 = (V_1, E_1)$ in a graph $G_2 = (V_2, E_2)$ is an injection from vertices V_1 to V_2 with the property that the edges E_1 are mapped to disjoint paths of G_2. (These disjoint paths in G_2 represent possible *subdivisions*

of the edges of G_1.) The set of homeomorphic embeddings between graphs is the topological partial order that was mentioned in Chapter 3.

Recall that the pathwidth of a graph G is the minimum pathwidth over all path decompositions of G. As mentioned in Chapter 10, determining pathwidth is equivalent to several VLSI layout problems such as gate matrix layout and vertex separation [Möh90, EST87].

We have shown in Chapters 2 and 3 that the family of graphs of pathwidth at most t, t–PATHWIDTH, is a lower ideal in the minor (and hence, topological) order and those graphs with order n have at most $nt - (t^2 + t)/2$ edges.

Let B_h denote the complete binary tree of height h and order $2^h - 1$. Let $h(t)$ be the least value of h such that $B_{h(t)}$ has pathwidth greater than t, and let $f(t)$ be the number of vertices of $B_{h(t)}$. To find a bound for $f(t)$, $B_{h(t)}$ needs to contain (above in the topological order) at least one obstruction of pathwidth t. In [EST87] it is shown that all topological tree obstructions of pathwidth t can be recursively generated by the following rules.

1. The single edge tree K_2 is the only obstruction of pathwidth 0.
2. If T_1, T_2 and T_3 are any 3 tree obstructions for pathwidth t then the tree T consisting of a new degree three vertex attached to any vertex of T_1, T_2 and T_3 is a tree obstruction for pathwidth $t + 1$.

From this characterization we see that the orders of the tree obstructions of pathwidth t are precisely $(5 \cdot 3^t - 1)/2$, (e.g., orders $2, 7, 22$ and 57 for pathwidth $t = 0, 1, 2$ and 3). We can easily embed at least one of the tree obstructions for pathwidth t, as shown in Figure 12.1, in the complete binary tree of height $2t + 2$. Thus, the pathwidth of the complete binary tree of order $f(t) = 2^{2t+2} - 1 = O(4^t)$ is greater than t.

12.2 A Simple Linear-Time Pathwidth Algorithm

We say that a boundary size k *factorization* of a graph G is two k-boundaried graphs A and B such that $G = A \oplus B$. Using the $f(t)$ bound given in the previous section,

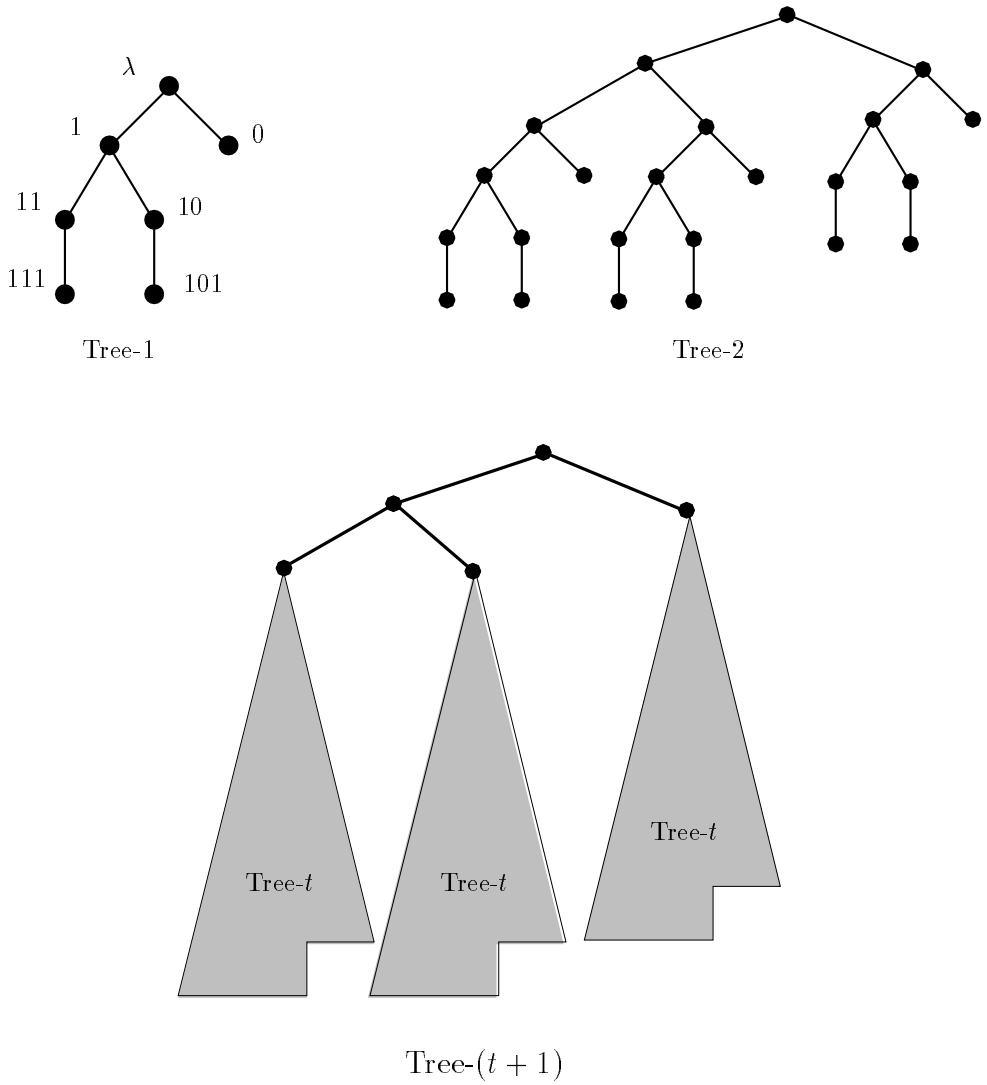


Figure 12.1: Embedding pathwidth tree obstructions Tree-t in binary trees.

the main result of this chapter now follows:

Theorem 181: Let H be an arbitrary undirected graph, and let t be a positive integer. One of the following two statements must hold:
(a) The pathwidth of H is at most $f(t) - 1$.
(b) The graph H can be factored: $H = A \oplus B$, where A and B are boundaried graphs with boundary size $f(t)$, the pathwidth of A is greater than t and less than $f(t)$.

Proof. We describe a algorithm that terminates either with a path decomposition of H of width at most $f(t) - 1$, or with a path decomposition of a suitable factor A with the last vertex set of the decomposition consisting of the boundary vertices.

If we find an homeomorphic embedding of the *guest tree* $B_{h(t)}$ in the *host graph* H then we know that the pathwidth of H is greater than t. During the search for such an embedding, we work with a partial embedding. We refer to the vertices of $B_{h(t)}$ as *tokens*, and call tokens *placed* or *unplaced* according to whether or not they are mapped to vertices of H in the current partial embedding. A vertex v of H is *tokened* if a token maps to v. At most one token can be placed on a vertex of H at any given time. We recursively label the tokens by the following standard rules:

1. The root token of $B_{H(t)}$ is labeled by the empty string λ.
2. The left child token and right child token of a height h parent token $P = b_1 b_2 \cdots b_h$ are labeled $P \cdot 1$ and $P \cdot 0$, respectively.

Let $P[i]$ denote the set of vertices of H that are tokened at time step i. The sequence $P[0], P[1], \ldots, P[s]$ describes a path decomposition either of the entirety of H or of a factor A fulfilling the conditions of Theorem 181. In the case of outcome (b) the boundary of the factor A is indicated by $P[s]$.

The placement algorithm is described as follows. Initially consider that every vertex of H is colored *blue*. In the course of the algorithm a vertex of H has its color changed to *red* when a token is placed on it, and stays red if the token is removed. Only blue vertices can be tokened, and so a vertex can only be tokened once.

Algorithm 182: A linear time path decomposition algorithm

function GrowTokenTree
1 **if** root token λ is not placed on H **then**
arbitrarily place λ on a blue vertex of H
endif
2 **while** there is a vertex $u \in H$ with token T and blue neighbor v,
and token T has an unplaced child $T \cdot b$ **do**
2.1 place token $T \cdot b$ on v
endwhile
3 **return** {tokened vertices of H}

program PathDecompositionOrSmallFatFactor
1 $i \leftarrow 0$
2 $P[i] \leftarrow$ **call** GrowTokenTree
3 **until** $|P[i]| = f(t)$ **or** H has no blue vertices **repeat**
3.1 pick a token T with an unplaced child token
3.2 remove T from H
3.3 **if** T had one tokened child **then**
replace all tokens $T \cdot b \cdot S$ with $T \cdot S$
endif
3.4 $i \leftarrow i + 1$
3.5 $P[i] \leftarrow$ **call** GrowTokenTree
enduntil
done

end algorithm

Before we prove the correctness of the algorithm, we note some properties: (1) the root token needs to be placed (step 1 of the GrowTokenTree) at most once for each component of H; (2) the GrowTokenTree function only returns when $B_{h(t)}$ has been embedded in H or all parent tokens with unplaced children have no blue neighbors in the underlying host H; (3) the algorithm terminates since during each iteration

of step 3.2 a tokened red vertex becomes untokened, and this can happen at most n times, where n denotes the order of the host H.

Since tokens are placed only on blue vertices and are removed only from red vertices, it follows that the interpolation property of a path decomposition is satisfied. Suppose the algorithm terminates at time s with all of the vertices colored red. To see that the sequence of vertex sets $P[0], \ldots, P[s]$ represents a path decomposition of H, it remains only to verify that for each edge (u, v) of H there is a time i with both vertices u and v in $P[i]$. Suppose vertex u is tokened first and untokened before v is tokened. But vertex u can be untokened only if all neighbors, including vertex v, are colored red (see step 3.1 and comment (2) above).

Suppose the algorithm terminates with all tokens placed. The argument above establishes that the subgraph A of H induced by the red vertices, with boundary set $P[s]$ has pathwidth at most $f(t)$. To complete the proof we argue that in this case the sequence of token placements establishes that A contains a subdivision of $B_{h(t)}$, and hence must have pathwidth greater than t. Since the GrowTokenTree function only attaches pendent tokens to parent tokens we need to only to observe that the operation in step 3.3 subdivides the edge between T and its parent. □

Corollary 183: Given a graph H of order n and an integer t, there exists an linear time algorithm that gives evidence that the pathwidth of H is greater than t or finds a path decomposition of width at most $f(t) - 1 = O(4^t)$.

Proof. We show that program PathDecompositionOrSmallFatFactor runs in linear time. First, if H has more than $t \cdot n$ edges, then the pathwidth of H is greater than t. By the proof of Theorem 181, the program terminates with either the embedded binary tree as evidence, or a path decomposition of width at most $f(t)$.

Note that the guest tree $B_{h(t)}$ has constant order $f(t)$, and so token operations that do not involve scanning H are constant time. In function GrowTokenTree, the only non-constant time operation is the check for blue neighbors in step 2. While scanning the adjacent edges of vertex u any edge to a red vertex can be removed, in constant time. Edge (u, v) is also removed when step 2.1 is executed. Therefore, across all calls to GrowTokenTree, each edge of H needs to be considered at most once, for a total of $O(n)$ steps. In program PathDecompositionOrSmallFatFactor, all steps

except for GrowTokenTree are constant time. The total number of iterations through the loop is bounded by n, by the termination argument following the program. □

The next result shows that we can improve the pathwidth algorithm by restricting the guest tree. This allows us to use the subdivided tree obstructions given in Figure 12.1 as guests.

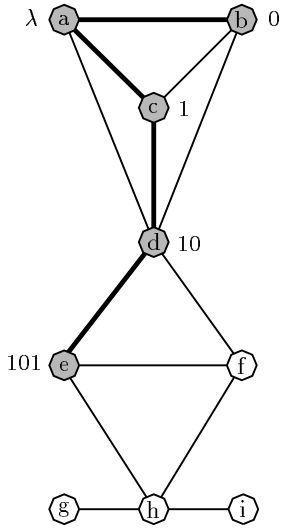
Corollary 184: Any subtree of the binary tree $B_{h(t)}$ that has pathwidth greater than t may be used in the algorithm for Theorem 181.

Proof. The following simple modifications allow the algorithm to operate with a subtree. The subtree is specified by a set of *flagged* tokens in $B_{h(t)}$. At worst, the algorithm can potentially embed all of $B_{h(t)}$.

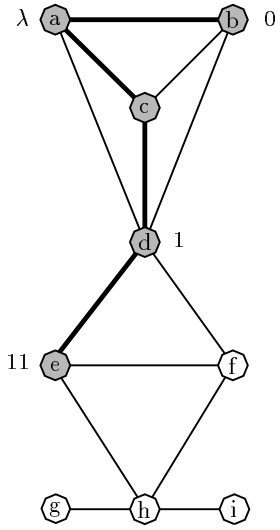
In step 2 of GrowTokenTree, the algorithm only looks for a flagged untokened child $T \cdot b$ to place, since unflagged tokens need not be placed. The stopping condition in step 3 of PathDecompositionOrSmallFatFactor is changed to "all flagged tokens of $B_{h(t)}$ are placed **or** ...," so that termination occurs as soon as the subtree has been embedded. The relabeling in step 3.3 can place unflagged tokens of $B_h(t)$ on vertices of H since all the rooted subtrees of a fixed height are not isomorphic. If that happens, we expand our guest tree (by adding flags) with those new tokens. It is easy to see that the new guest is still a tree. These newly flagged tokens may be relabeled by future edge subdivisions that occur above the token in the host tree. Duplication of token labels does not happen if unflagging is not allowed. Thus, the $f(t)$ width bound is preserved. □

Example 185: Using Tree-1 from Figure 12.1 (subdivided $K_{1,3}$) as the guest tree of pathwidth 2 the program trace given in Figure 12.2 terminates with all vertices colored red (gray) yielding a path decomposition of width 5.

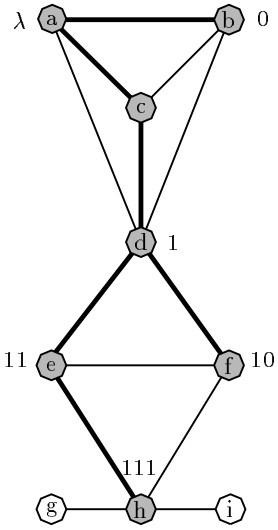
In the proof of Corollary 184 one may wish to not expand the guest tree by flagging new tokens. This can be done and, in fact, is what we would do in practice. Without loss of generality, suppose token $T \cdot 1$ is on vertex $u \in H$ and has children that can not be placed on H, and $T \cdot 1$ has one unflagged sibling token $T \cdot 0$ on $v \in H$.



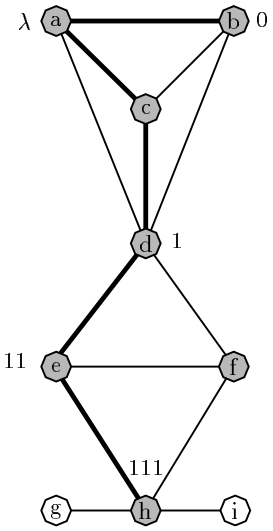
After step 2,
$P[0] = \{a, b, c, d, e\}$



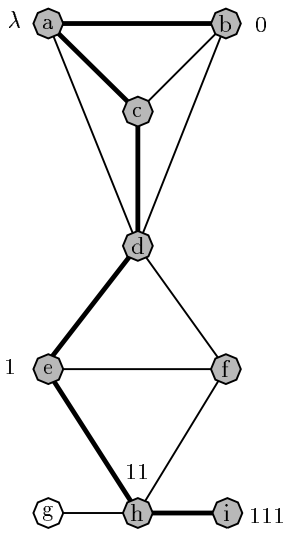
After 3.3, $T = 1$



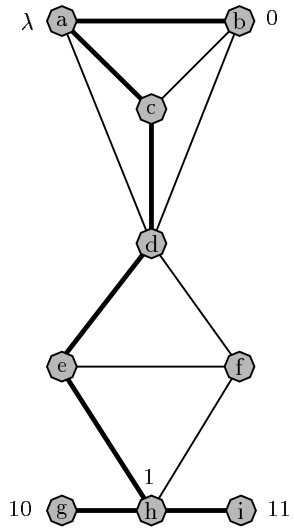
After 3.5
$P[1] = \{a, b, d, e, f, h\}$



$T = 10$, after 3.5
$P[2] = \{a, b, d, e, h\}$



$T = 1$, after 3.5
$P[3] = \{a, b, e, h, i\}$



$T = 1$, after 3.5
$P[4] = \{a, b, g, h, i\}$

Figure 12.2: Illustrating our linear time pathwidth algorithm.

If we ignore the flagging of new vertices in the current algorithm, the token $T \cdot 1$ would be removed and the parent T (which has only one legitimate child) would be placed on vertex v. What happens to any blue vertices that are adjacent to only vertex u (or its unflagged subtree)? The answer is that they are lost and the algorithm would not terminate unless it could embed the guest tree in the remaining portion of H. We can fix this problem by checking for unflagged siblings before step 3.3 and to shift the token $T \cdot 1$ from u to v. See step 3.3′ below.

3.3′ **if** T had one tokened child **then**
replace all tokens $T \cdot b \cdot S$ with $T \cdot S$
else if $T = P \cdot b$ had an unflagged sibling **then**
replace all tokens $P \cdot b \cdot S$ with $P \cdot \text{not}(b) \cdot S$
endif

Observe that our pathwidth algorithm provides an easy proof of basically the main result of [BRST91] (or its earlier variant [RS83]) that for any forest F, there is a constant c, such that any graph not containing F as a minor has pathwidth at most c.

Corollary 186: Every graph with no minor isomorphic to forest F, where F is a minor of a complete binary tree B, has pathwidth at most $c = |B| - 2$.
Proof. Without loss of generality, we can run our pathwidth algorithm using (as the guest) any subtree T of B that contains F as a minor. Since at most $|B| - 1$ pebbles are used when we do *not* find an embedding of T (in any host graph), the resulting path decomposition has width at most $|B| - 2$. □

Our constant c is identical to the one given in [BRST91] when $F = B$. They point out that their constant $c = |F| - 2$ is the best possible since the complete graph K_{c-1}, with pathwidth $c - 2$, does not contain any forest with c vertices.

12.3 Further Directions

In the case that the pathwidth of an input graph G is at most k, our algorithm yields a path decomposition that can have a width exponential in k, but that is equal in

any case to the maximum number of tokens placed on the graph at any given time, minus 1. It would be interesting to know if this exponential bad behavior is "normal" or whether the algorithm tends to use a smaller number of tokens in practice. Since the pebbling proceeds according to a greedy strategy with much flexibility, there may be placement heuristics that can improve its performance on "typical" instances.

Chapter 13

Conclusion

This chapter summarizes the results of this dissertation and then concludes with two important applications that concern forbidden substructure characterization of graphs.

13.1 A Summary of the Main Results

Our main computational achievements, given in Part II of this dissertation, show that computing obstructions for "simple" minor-order lower ideals is feasible for small pathwidth. Here we developed a computational theory, given in Part I, that extends the original Fellows–Langston approach for computing obstruction sets. For a wide range of targeted graph families (in fact, all being parameterized lower ideals), Table 13.1 summarizes the largest pathwidth bounds that our software can currently explore. Our experimental research has shown that these pathwidth bounds may increase by one if universal distinguisher searching is available (see Sections 4.4.2 and 5.3.1). Furthermore, with new theory being developed, there is ample opportunity to characterize additional lower ideals with our basic method. We expect that for any lower ideal the corresponding treewidth bounds (regarding achievable obstruction set computations) will be equivalent to the largest feasible pathwidth bounds. That is, we have no evidence to suggest that our treewidth enumeration methods become impractical at smaller widths.

Table 13.1: Tractability bounds for various parameterized lower ideals.

Minor Order Lower Ideal	Largest Pathwidth?
k–Edge Bounded IndSet	6
5–Vertex Cover	6
4–Feedback Edge Set	5
1–Path Cover(3)	5
2–Path Cover(1)	4
1–Cycle Cover(3)	4
2–Feedback Vertex Set	4
1–Outer Planar	4
1–Planar	4
1–Genus (torus)	4
2–Pathwidth	3

We now list a few noteworthy theory results that were presented in Part I of this dissertation.

- Introduced practical t-parse enumeration schemes based on canonic representations for graphs of bounded pathwidth and treewidth.
- Presented a small search-space technique (of bounded combinatorial width) for computing minor-order obstructions. Our Prefix Lemma establishes a search tree of minimal t-parses (as growable nodes), guaranteeing a search that terminates.
- Developed sufficient rules for computing disconnected minor-order obstructions for certain parameterized lower ideals. The expected growth rate on the number of obstructions for parameterized lower ideals was also studied.
- Utilized a distributive programming approach, across many hardware platforms, for obstruction set computations.

Besides characterizing many flavors of graph families by obstruction sets, in Part II of this dissertation we provided several other useful results such as the following family-specific items for input graphs of bounded pathwidth (t-parses).

- An optimal finite-state algorithm for k–VERTEX COVER (including a linear time vertex cover algorithm).
- A finite-state dynamic program for k–FEEDBACK VERTEX SET (including a linear time feedback vertex set algorithm).
- A testset for both the k–FEEDBACK VERTEX SET and k–FEEDBACK EDGE SET graph families (and a nonminimal pretest for k–FEEDBACK EDGE SET).
- A linear time dynamic program to determine the maximum path length and a congruence for k–PATH COVER(p).
- A testset for k–CYCLE COVER(3) and some testsets for various families of graphs that have bounded path or cycle lengths.
- A linear time dynamic program for 0–OUTER PLANAR and the minimum K_3-cover.

Lastly, in Part III of this dissertation we contributed the following applied results.

- Equated a VLSI layout problem (k-CVS) with a computational biology problem (k-ICG).
- Showed that finding the minimum size testset for a minimal automaton (or canonical congruence) is $\mathcal{NP}$-hard.
- Developed a simple linear time algorithm that (for fixed k) determines if a graph has pathwidth greater than k or finds a path decomposition of width at most $O(4^k)$.

13.2 Two Key Applications

We conclude this dissertation by mentioning two areas of research that are important consequences of our work. These two applications are of interest to both the theoretical and industrial computer science communities.

First, we have empirically shown that mathematical theorem proving, specifically for the case of finding substructure characterizations of minor-order lower ideals, can be automated. We have presented a natural and easy way to find obstruction sets for those characterizable families whose obstructions are bounded by some constant combinatorial width. (The Graph Minor Theorem guarantees a finite number of obstructions and hence a bound on both the pathwidth and treewidth.) Our method requires two additional ingredients consisting of a membership algorithm and a refinement of the canonical congruence for the lower ideal. This dissertation has shown that these required congruences are easy to produce. We demonstrated with several examples two design techniques for obtaining these family-specific congruences, namely by testsets of t-boundaried graphs and finite-state dynamic programs for t-parses.

For an application with real-world potential, we have developed a tool that provides graph-algorithm designers with the following opportunity. As mentioned earlier in Section 5.3.2, one can take a partial set of obstructions for any minor-order lower ideal $\mathcal{F}$ and build an approximating automaton for recognizing those graphs of bounded width within $\mathcal{F}$. We can do this for any desired pathwidth or treewidth. A randomized feature of our software allows one to generate partial sets of obstructions without the need for a finite-index family congruence for $\mathcal{F}$, thereby allowing the non-specialist to take advantage of our system. That is, we can now implement a useful graph-algorithm compiler.

Annotated Bibliography

[ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8:277–284, April 1987.
[Contains the first polynomial time algorithm for determining the treewidth of a graph for fixed k. Also, the partial k-tree problem is shown to be $\mathcal{NP}$-complete. Provides a listing of the treewidth three obstructions.]

[ACPS91] S. Arnborg, D. G. Corneil, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. In *Proceedings of the Fourth Workshop on Graph Grammars and Their Applications to Computer Science*, volume 532 of *Lecture Notes on Computer Science*, pages 70–83. Springer-Verlag, 1991. To appear *Journal of the Association Computing Machinery.*
[Shows how membership in classes of graphs definable in monadic second order logic and of bounded treewidth can be decided by finite sets of terminating reduction rules.]

[AF93] K. Abrahamson and M. Fellows. Finite automata, bounded treewidth and well-quasiordering. In N. Robertson and P. D. Seymour, editors, *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 539–564, 1993. Formally known as "Cutset regularity beats well-quasi-ordering for bounded treewidth".
[Contains a necessary and sufficient condition (cutset regularity) for a family of graphs to be recognizable from structural parse trees by finite-state tree automata.]

[AH85] D. S. Archdeacon and P. Huneke. On cubic graphs which are irreducible for nonorientable surfaces. *Journal of Combinatorial Theory, Series B*, 39:233–264, 1985.

[ALS91] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991. Also see "extended abstract" in volume 317 of *Lecture Notes on Computer Science* (1988) 38–51.
[Monodic second order logic is presented to show that a large set of graph families is finite-state. This is an alternate proof of Courcelle's result.]

[AP86] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM Journal on Algebraic Discrete Methods*, 7(2):305–314, April 1986.
[A set of confluent graph reductions is given such that a graph can be reduced to the empty graph if and only if it is a subgraph of a 3-tree.]

[AP89] S. Arnborg and A. Proskurowski. Linear algorithms for $\mathcal{NP}$-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, pages 11–24, 1989.

[AP92] S. Arnborg and A. Proskurowski. Canonical representations of partial 2- and 3- trees. *BIT*, 32:197–214, 1992.

[APC87] S. Arnborg, A. Proskurowski, and D. Corneil. Minimal forbidden minor characterization of a class of graphs. *Colloquia Mathematica Societatis János Bolyai*, 52:49–62, 1987.
[A graph is a partial 3-tree if and only if it does not have a minor isomorphic to any of four obstructions.]

[APS90] S. Arnborg, A. Proskurowski, and D. Seese. Forbidden minors characterization of partial 3-trees. *Discrete Mathematics*, 80:1–19, 1990.

[APS91] S. Arnborg, A. Proskurowski, and D. Seese. Monadic second order logic, tree automata and forbidden minors. In *Proceedings of the 4th Workshop on Computer Science Logic, CSL'90*, volume 533 of *Lecture Notes on Computer Science*, pages 1–16. Springer-Verlag, 1991.

[Arc80] D. Archdeacon. *A Kuratowski Theorem for the projective plane*. Ph.D. dissertation, The Ohio State University, 1980. Also see *Journal of Graph Theory* 5:243–246 (1981).
[The author sketches a proof that the list of 103 irreducible graphs for the projective plane is complete. Also it is pointed that there are at least 4000 irreducible graphs for the Klein bottle.]

[Arc90] D. Archdeacon. The complexity of the graph embedding problem. In R. Bodendiek and R. Henn, editors, *Topics in Combinatorics and Graph Theory*, pages 59–64. Physica-Verlag, 1990.

[Arc95] D. Archdeacon. private communication, 1995. Dept. of Mathematics and Statistics, University of Vermont.

[Arn78] S. Arnborg. Reduced state enumeration—another algorithm for reliability evaluation. *IEEE Transactions on Reliability*, 27:101–105, 1978.

[Arn85] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT*, 25:2–23, 1985. Invited paper.
[This is a good survey paper on table-based reduction methods for graphs with bounded treewidth.]

[Arn91] S. Arnborg. Graph decompositions and tree automata in reasoning with uncertainty. *Journal of Experimental and Theoretical AI*, 1991. to appear.

[AST90] N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings of the ACM Symposium on the Theory of Computing*, volume 22, pages 293–299, 1990. (Baltimore, Maryland).
[Supplies extensions of the many known applications of the Lipton-Tarjan separator theorem for the class of planar graphs to any class of graphs with an excluded minor.]

[BC87] M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical System Theory*, 20:83–127, 1987.

[The notion of a context-free graph grammar is introduced. The notion of an equational set of graphs follows from a defined algebraic structure. A notion of graph rewriting is also given based on a categorical approach.]

[BD88] D. Bienstock and N. Dean. Some results on minimum face covers in planar graphs, February 1988. Bell Communications Research, Morristown, New Jersey.
[This paper discusses some obstructions to the existence of small face covers.]

[BdF95] H. L. Bodlaender and B. L. E. de Fluiter. Intervalizing k-colored graphs. Technical report UU-CS-1995-15, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, 1995. Extended abstract to appear in proceedings ICALP'95.
[Solves the complexity issues regarding k-ICG problem.]

[BFH94] H. L. Bodlaender, M. R. Fellows, and M. T. Hallett. Beyond $\mathcal{NP}$-completeness for problems of bounded width: Hardness for the W-hierarchy (extended abstract). In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 449–458, 1994.

[BFL89] D. J. Brown, M. R. Fellows, and M. A. Langston. Polynomial-time self-reducibility: Theoretical motivations and practical results. *International Journal of Computer Mathematics*, 31:1–9, 1989.
[The notion of self-reducibility is explained for tackling the 'promised' polynomial-time decision algorithms which are only known via nonconstructive means.]

[BFP95] J.-C. Bermond, P. Fraigniaud, and J. G. Peters. Antepenultimate broadcasting. *Netorks*, 26:125–137, 1995.
[This; paper uses path-covers in the design of minimum broadcast graphs.]

[BGHK91] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, and minimum elimination tree height. Technical Report RUU-CS-91-1, Dept. of Computer Science, University of Utrecht, P.O. Box 80.012, 3508 TA Utrecht, the Netherlands, January 1991. To appear *Journal of Algorithms*.
[Vertex separators are used to approximate width problems.]

[BH80] T. Beyer and S. M. Hedetniemi. Constant time generation of rooted trees. *SIAM Journal on Computing*, 9:706–712, 1980.

[BHKY62] J. Battle, F. Harary, Y. Kadama, and J. Youngs. Additivity of the genus of a graph. *Bulletin of the American Mathematical Society*, 68:565–568, 1962.
[Shows that the genus of a graph equals the sum of the genus of the individual blocks.]

[Bie91] D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). In *Reliability of Computer and Communication Networks*, volume 5 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 33–49. Association for Computing Machinery, 1991.

[BJM79] T. Beyer, W. Jones, and S. L. Mitchell. Linear algorithms for isomorphism of maximal outerplanar graphs. *Journal of the Association Computing Machinery*, 26(4):603–610, 1979.

[BK91] H. L. Bodlaender and T. Kloks. Better algorithms for pathwidth and treewidth of graphs. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, volume 510 of *Lecture Notes on Computer Science*, pages 544–555. Springer-Verlag, 1991. 18th ICALP.
[For all constants k, an explicit $O(n \log^2 n)$ algorithm is given to decide whether the treewidth (pathwidth) of a graph is at most k, and if so, finds the decompositions.]

[BK92] H. L. Bodlaender and T. Kloks. Approximating treewidth and pathwidth of some classes of perfect graphs. In *Proceedings of the 3rd International Symposium on Algorithms and Computation, ISAAC'92*, volume 650 of *Lecture Notes on Computer Science*, pages 116–125. Springer-Verlag, 1992.

[BK93] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs, 1993. preprint.

[BL] D. Bienstock and M. A. Langston. Algorithmic implications of the graph minor theorem. to appear *Handbook of Operations Research and Management Science: Volume on Networks and Distribution.*

[BLW85] M. W. Bern, E. L. Lawler, and A. L. Wong. Why certain subgraph computations require only line time. In *IEEE Symposium on Foundations of Computer Science Proceedings*, pages 117–125, 1985. 26th FOCS, Portland, OR.

[BLW87] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8:216–235, 1987. Full version of FOCS'95 paper.
[For the class of graphs with bounded combinatorial width, a dynamic programming approach is used for general subgraph problems.]

[BM76] J. Bondy and U. Murty. *Graph Theory with Applications.* MacMillan, 1976.

[BM88] D. Bienstock and C. L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, 17:53–76, 1988.
[An algorithm is give which either determines if a graph is not k-planar or generates an appropriate embedding and associated minimum cover in $O(c^k n)$ time, where c is a constant.]

[BM93] H. Bodlaender and R. Möhring. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 6:181–188, 1993.
[Shows that the pathwidth equals the treewidth for cographs and gives a linear-time algorithm for finding a tree-decomposition.]

[Bod86a] H. L. Bodlaender. Classes of graphs with bounded tree-width. Technical Report RUU-CS-86-22, Dept. of Computer Science, University of Utrecht, P.O. Box 80.012, 3508 TA Utrecht, the Netherlands, December 1986. (Also in *Bulletin of the EATCS* 36 (1988), 116–126).
[A number of classes of graphs are shown to be subclasses of the graphs with tree-width, bounded by some constant k.]

[Bod86b] H. L. Bodlaender. Planar graphs with bounded tree-width. Technical report, Dept. of Computer Science, University of Utrecht, P.O. Box 80.012, 3508 TA Utrecht, the Netherlands, 1986.
[Discusses k-outerplanar graphs, planar graphs with bounded radius, and Halin graphs.]

[Bod88b] H. L. Bodlaender. Some classes of graphs with bounded treewidth. *Bulletin of the EATCS*, 36:116–126, 1988.

[Bod88a] H. L. Bodlaender. Dynamic programming algorithms on graphs with bounded treewidth. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, volume 317 of *Lecture Notes on Computer Science*, pages 105–119. Springer-Verlag, 1988. 15th ICALP.
[Gives an $O(n)$ time algorithm for the k disjoint cycles problem.]

[Bod89] H. L. Bodlaender. Improved self-reduction algorithms for graphs with bounded treewidth. In *Graph-Theoretic Concepts in Computer Science*, volume 411 of *Lecture Notes on Computer Science*, pages 232–244. Springer-Verlag, 1989. Also in *Discrete Applied Mathematics* 54.
[Self-reduction is used to construct solutions to many of the Robertson–Seymour nonconstructive polynomial-time families.]

[Bod90] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11:631–644, 1990.
[Two different approaches are used to show that the Chromatic Index and Graph Isomorphism problems are solveable in polynomial time when restricted to the class of graphs with treewidth $\leq k$.]

[Bod92] H. L. Bodlaender. On disjoint cycles. In *Proceedings of the 17th International Workshop on Graph-Theoretic Concepts in Computer Science WG'91*, volume 570 of *Lecture Notes on Computer Science*, pages 230–239. Springer-Verlag, 1992. Also University of Utrecht technical report RUU-CS-90-29.
[Gives an $O(n)$ k-FVS algorithm.]

[Bod93c] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993.
[A short overview of algorithmic graph theory that deal with the notions of treewidth and pathwidth.]

[Bod93b] H. L. Bodlaender. On linear time minor tests and depth first search. *Journal of Algorithms*, 14:1–23, 1993. Also in volume 382 of *Lecture Notes on Computer Science* (1989) 577–590.
[Presents efficient minor tests when the set of graphs has a $2 \times k$ grid as a minor and a circus graph as a minor. Also gives an $O(k!2^k n)$ time algorithm to determine whether a graph has a cycle (or path) or length $\geq$ k if it exists.]

[Bod93a] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the ACM Symposium on the Theory of Computing*, volume 25, 1993.
[This is the end-of-the-story (except, perhaps, finding a practical algorithm) regarding the k parameterized treewidth problem. As a consequence, every minor-closed class of graphs that does not contain all planar graphs has a linear-time recognition algorithm.]

[Bod95] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. Preliminary manuscript, Dept. of Computer Science, University of Utrecht, P.O. Box 80.012, 3508 TA Utrecht, the Netherlands, October 1995.

[Bon75] J. A. Bondy. *Pancyclic graphs: recent results*, pages 181–187. North Holland, Amsterdam, 1975. Colloquia Mathematica Societatis János Bolyai, volume 10.
[Halin graphs are introduced.]

[Bor88] R. B. Borie. *Recursively Constructed Graph Families: Membership and Linear Algorithms*. Ph.D. thesis, Georgia Institute of Technology, School of Information and Computer Science, 1988.
[This thesis uses t-terminal graphs (boundaried graphs) with recursive rules for graph families (e.g., partial k-trees) and an automated technique for generating linear algorithms from a predicate calculus.]

[BP71] L. Beineke and R. Pippert. Properties and characterizations of k-trees. *Mathematika*, 18:141–151, 1971.

[BPT88] R. Borie, R. Parker, and C. Tovey. The regular forbidden minors of partial 3-trees. Technical report, School of ISYE, Georgia Institute of Technology, 1988. Tech. Report No. J-88-14.

[BPT91] R. B. Borie, R. G. Parker, and C. A. Tovey. Deterministic decomposition of recursive graph classes. *SIAM Journal on Discrete Mathematics*, 4(4):481–501, 1991.
[Examines how the series parallel graphs can be generalized (i.e., with recursive rules).]

[BRST91] D. Bienstock, N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a forest. *Journal of Combinatorial Theory, Series B*, 52:274–283, 1991.

[CB81] C. Colbourn and K. Booth. Linear time automorphism algorithms for trees, interval graphs and planar graphs. *SIAM Journal on Computing*, 10(1):203–225, February 1981.

[CD94] K. Cattell and M. J. Dinneen. A characterization of graphs with vertex cover up to five. In V. Bouchitte and M. Morvan, editors, *Orders, Algorithms and Applications, ORDAL'94*, volume 831 of *Lecture Notes on Computer Science*, pages 86–99. Springer-Verlag, July 1994.
[Contains a previous version of Chapter 6.]

[CDDF95] K. Cattell, M. J. Dinneen, R. G. Downey, and M. R. Fellows. Computational aspects of the graph minor theorem: Obstructions for unions and intertwines, 1995. University of Victoria working manuscript (1995).

[CDF95a] K. Cattell, M. J. Dinneen, and M. R. Fellows. Obstructions to within a few vertices or edges of acyclic. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures, WADS'95*, volume 955 of *Lecture Notes on Computer Science*, pages 415–427. Springer-Verlag, August 1995.
[This paper is an extended abstract version of Chapter 7.]

[CDF95b] K. Cattell, M. J. Dinneen, and M. R. Fellows. A simple linear-time algorithm for finding path-decompositions of small width. *Information Processing Letters*, 1995. to appear.
[This paper is almost identical to Chapter 12.]

[CG83] J. H. Conway and C. Gordon. Knots and links in spatial graphs. *Journal of Graph Theory*, 7:445–453, 1983.

[CG93] J. Chen and J. L. Gross. Kuratowski-type theorems for average genus. *Journal of Combinatorial Theory, Series B*, 57:100–121, 1993.
[The authors define average genus of a graph and provide obstructions for small fractional averages up to $c = 1$. They ask if there is a way to implement this method systematically for fractions up to $c = 2$, or 3.]

[CKK72] V. Chvatal, D. A. Klarner, and D. E. Knuth. Selected combinatorial research problems. Technical report, Dept. of Computer Science, Stanford University, June 1972.
[The paper characterizes unit distance graphs by two forbidden subgraphs K_4 and $K_{2,3}$.]

[CL86] G. Chartrand and L. Lesniak. *Graphs and Digraphs*. Wadsworth Inc., 1986.

[CM93] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109:49–82, 1993.
[Presents graphs as logical structures (almost like our t-parse operator sets).]

[Cou90a] B. Courcelle. The monadic second order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[Cou89] B. Courcelle. The monadic second order logic of graphs II: Definable sets of infinite graphs. *Mathematical Systems Theory*, 21:187–221, 1989.

[Cou93] B. Courcelle. The monadic second-order logic of graphs III: treewidth, forbidden minors and complexity issues. *Informatique Théorique*, 26:257–286, 1993.

[Cou90b] B. Courcelle. The monadic second order logic of graphs IV: Every equational graph is definable. *Annals of Pure and Applied Logic*, 49:193–255, 1990.

[Cou91] B. Courcelle. The monadic second order logic of graphs V: On closing the gap between definability and recognizability. *Theoretical Computer Science*, 80:153–202, 1991.

[Cou94] B. Courcelle. The monadic second order logic of graphs VI: On several representations of graphs by related structures. *Discrete Applied Mathematics*, 54:117–150, 1994.

[Cou92b] B. Courcelle. The monadic second order logic of graphs VII: Graphs as relational structures. *Theoretical Computer Science*, 101:3–33, 1992.

[COS95] D. G. Corneil, S. Olariu, and L. Stewart. Computing a dominating pair in an asteroidal triple-free graph in linear time. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures, WADS'95*, volume 955 of *Lecture Notes on Computer Science*, pages 358–368. Springer-Verlag, August 1995.

[Cou87] B. Courcelle. A representation of graph by algebraic expressions and its use for graph rewriting systems. In *Proceedings of the 3rd International Workshop on Graph Grammers*, volume 291 of *Lecture Notes on Computer Science*, pages 112–132. Springer-Verlag, 1987.

[Cou92a] B. Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. *Bulletin of the EATCS*, 46:193–226, 1992. Also in *Contemporary Mathematics* 147 (1993) 565–590.

[This is a survey of the relationships between the descriptions of sets of graphs by formulas of monadic second-order logic, by context-free hyperedge and vertex replacement graph grammars and by forbidden minors.]

[CS89] F. R. K. Chung and P. D. Seymour. Graphs with small band-width and cut-width. *Discrete Mathematics*, 75:113–119, 1989.

[CSTV92] R. Cohen, S. Sairam, R. Tamassia, and J. S. Vitter. Dynamic algorithms for bounded tree-width graphs. Technical report, Dept. of Computer Science, Brown University, 1992. Tech. Report CS-92-19.

[CV84] D. Coppersmith and U. Vishkin. Solving $\mathcal{NP}$-hard problems in 'almost trees': Vertex cover. *Discrete Applied Mathematics*, 10:27–45, 1984.
[The authors present a linear-time algorithm for determining the vertex cover for graphs that are parameterized by two parameters: a the number of edges more than a tree and k the maximum a over all biconnected components of the graph.]

[Dec78] R. W. Decker. *The Genus of Certain Graphs.* Ph.D. dissertation, The Ohio State University, 1978.
[The concept of irreducibile graphs for the torus is investigated (partial list is given?).]

[Deo74] N. Deo. *Graph Theory with Applications to Engineering and Computer Science.* Prentice-Hall, 1974.

[DF95] R. Downey and M. R. Fellows. Parameterized computational feasibility. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 219–244. Birkhauser, 1995. Proceedings of the 2nd Cornell Workshop on Feasible Mathematics.

[Dis94] *Special issue: Efficient Algorithms and partial k-trees*, volume 54 of *Discrete Applied Mathematics*, October 1994. numbers 2–3.

[DR83] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[DR91] H. Didjev and J. Reif. An efficient algorithm for the genus problem with explicit construction of forbidden subgraphs. In *IEEE Symposium on Foundations of Computer Science Proceedings*, volume 31, pages 337–347, 1991.
[Gives some order bounds for genus k obstructions.]

[DS90] W. W.-M. Dai and M. Sato. Minimal forbidden minor characterization of planar partial 3-trees and applications to circuit layout. *IEEE International Symposium on Circuits and Systems*, pages 2677–2681, 1990.

[Duf65] R. J. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10:303–318, 1965.
[Good motivation of these network graphs with regards to the resistance rules of Ohm. Defines confluent graphs and shows they are precisely series-parallel graphs.]

[EMC88] E. El-Mallah and C. Colbourn. Partial k-tree algorithms. *Congressus Numerantium*, 64:105–119, 1988.

[EMC90] E. El-Mallah and C. Colbourn. On two dual classes of planar graphs. *Discrete Mathematics*, 80:21–40, 1990.
[Gives a characterization of planar partial 3-trees in terms of two obstructions.]

[Epp92] D. Eppstein. Parallel recognition of series-parallel graphs. *Information and Computing*, 98(1):41–55, May 1992.

[EST87] J. A. Ellis, I. H. Sudborough, and J. Turner. Graph separation and search number. Report DCS-66-IR, Dept. of Computer Science, University of Victoria, P.O. Box 3055, Victoria, B.C. Canada V8W 3P6, August 1987.
[Contains the first polynomial-time algorithm for determining if a graph has pathwidth k or less and a fast linear-time algorithm for determining the pathwidth of trees.]

[EST94] J. Ellis, I. H. Sudborough, and J. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.

[FBS89] D. Fernández-Baca and G. Slutzki. Solving parametric problems on trees. *Journal of Algorithms*, 10:381–402, 1989.

[FBS92] D. Fernández-Baca and G. Slutzki. Parametric problems on graphs of bounded treewidth. In *Proceedings of the 3rd Scandinavian Workshop on Algorithm Theory*, volume 621 of *Lecture Notes on Computer Science*, pages 304–316. Springer-Verlag, 1992.

[FBS94] D. Fernández-Baca and G. Slutzki. Optimal parametric problems on graphs of bounded treewidth. In *Proceedings of the 4rd Scandinavian Workshop on Algorithm Theory*, volume 824 of *Lecture Notes on Computer Science*, pages 155–166. Springer-Verlag, 1994.

[Fel] M. R. Fellows. private communication. Dept. of Computer Science, University of Victoria.

[Fel89] M. R. Fellows. The Robertson-Seymour theorems: A survey of applications. *Contemporary Mathematics*, 89:1–17, 1989.
[Applications of the Robertson–Seymour theorems to a variety of problems in concrete computational complexity are surveyed.]

[FG93] P. A. Firby and C. F. Gardiner. *Surface Topology*, volume 72 of *Mathematics and its Applications*. Ellis Horwood, second edition, 1993.

[FHW93] M. R. Fellows, M. T. Hallett, and H. T. Wareham. DNA physical mapping: Three ways difficult. In T. Lengauer, editor, *Proceedings of European Symposium on Algorithms (ESA'93)*, volume 726 of *Lecture Notes in Computer Science*, pages 157–168. Springer-Verlag, Berlin, 1993.
[The combinatorial problem of Intervalizing Colored Graphs (or ICG) is shown to be intractable in three different ways: (1) it is $\mathcal{NP}$-complete, (2) it is hard for the parameterized complexity class W[1], and (3) it is not finite-state for bounded treewidth or pathwidth.]

[Fis85] P. C. Fishburn. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*. John Wiley, New York, 1985.

[FKL88] M. R. Fellows, N. G. Kinnersley, and M. A. Langston. Finite-basis theorems and a computation-integrated approach to obstruction set isolation. Extended abstract, Dept. of Computer Science, Washington State University, Pullman, WA 99164-1210, November 1988. Also in Proceedings of Computers and Mathematics Conference (1989).

[FKMP95] M. Fellows, J. Kratochví, M. Middendorf, and F. Pfeiffer. Induced minors and related problems. *Algorithmica*, 13:266–282, 1995.
[Explores the similarities and differences of minors and induced minors. Combinatorial problems with restricted input to planar graphs are also considered.]

[FL87] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Information Processing Letters*, 26:157–162, October 1987.
[The connection between the Robertson–Seymour posets and the Gate Matrix Layout problem is given.]

[FL88a] M. R. Fellows and M. A. Langston. Layout permutation problems and well-partially-ordered sets. In *Proceedings of the 5th MIT Conference on Advanced Research in VLSI (Cambridge Mass., Mar. 28–30)*, pages 315–327. MIT press, 1988.
[Low-degree polynomial-time algorithms for a number of well-studied VLSI layout problems (along with self-reducibility results) are given.]

[FL88b] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the Association Computing Machinery*, 35:727–739, 1988.
[Defines what Robertson–Seymour posets are and shows how to use these tools for determining the complexity of difficult combinatorial decision problems.]

[FL89a] M. R. Fellows and M. A. Langston. An analogue of the Myhill-Nerode Theorem and its use in computing finite-basis characterizations. In *IEEE Symposium on Foundations of Computer Science Proceedings*, volume 30, pages 520–525, 1989.
[Using a graph-theoretic analogue of the Myhill-Nerode characterization of regular languages, this paper establishes that, for many applications, obstruction sets are computable by known algorithms.]

[FL89b] M. R. Fellows and M. A. Langston. On search, descision and the efficiency of polynomial-time algorithms. In *Proceedings of the ACM Symposium on the Theory of Computing*, volume 21, pages 501–512, 1989. Also in *Journal of Computer and System Sciences* (1995).
[Discusses the problem of equating tractability with polynomial-time decidability in light of the recent advances in well-partial-order theory, especially the seminal contributions of Robertson and Seymour.]

[FL92] M. R. Fellows and M. A. Langston. On well-partial-order theory and its application to combinatorial problems of VLSI design. *SIAM Journal on Discrete Mathematics*, 5:117–126, February 1992.
[Low degree polynomial-time algorithms are nonconstructively shown for the first time for many well-studied graph layout, placement, and routing problems.]

[FMR79] I. S. Filotti, G. L. Miller, and J. Reif. On determining the genus of a graph in $O(v^{O(g)})$ steps. In *Proceedings of the ACM Symposium on the Theory of Computing*, volume 11, pages 27–37, 1979. Atlanta, Georgia (Preliminary Report).
[First polynomial-time algorithms are presented for testing for fixed genus g.]

[FRS87] H. Friedman, N. Robertson, and P. D. Seymour. The meta-mathematics of the graph minor theorem. In *Applications of Logic to Combinatorics*, volume 65 of *Contemporary Mathematics*, pages 229–261. American Mathematical Society, 1987.

[Gav74] F. Gavril. The intersection of graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory*, 16:47–56, 1974.

[GGJK78] M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34(3):477–494, May 1978.
[Bandwidth is $\mathcal{NP}$-complete even when restricted to free trees with maximum degree three.]

[GGKS93] P. W. Goldberg, M. C. Golumbic, H. Kaplan, and R. Shamir. Four strikes against physical mapping of DNA. Technical report 287/93, The Moise and Frida Eskenasy Institute of Computer Sciences, Tel Aviv University, December 1993.
[Four simplified models of the DNA fragment reconstruction problem lead to $\mathcal{NP}$-complete decision problems. In particular, a simple $\mathcal{NP}$-completeness proof is given regarding the pathwidth of a bipartite graph.]

[GH79] H. H. Glover and J. P. Huneke. There are finitely many Kuratowski graphs for the projective plane. In *Graph Theory and Related Topics*, pages 201–206. Academic Press, Inc., 1979.
[The paper shows that a graph embeds in the projective plane if and only if it does not contain a subgraph homeomorphic to one of a finite list of graphs. This is done independent of any detailed list of graphs.]

[GHW79a] H. H. Glover, J. P. Huneke, and C. S. Wang. 103 graphs which are irreducible for the projective plane. *Journal of Combinatorial Theory, Series B*, 27:332–370, 1979.

[GHW79b] H. H. Glover, J. P. Huneke, and C. S. Wang. On a Kuratowski theorem for the projective plane. In *Graph Theory and Related Topics*, pages 207–218. Academic Press, Inc., 1979.
[The projective plane can be systematically characterized from thirty-five particular irreducible graphs. An appendix contains drawings of all such obstructions.]

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-completeness*. W. H. Freeman and Company, 1979.
[The best introduction to the theory of $\mathcal{NP}$-completeness and a catalog of classical $\mathcal{NP}$-complete problems.]

[GKS92] M. C. Golumbic, H. Kaplan, and R. Shamir. Graph sandwich problems. Technical report 270/92, The Moise and Frida Eskenasy Institute of Computer Sciences, Tel Aviv University, December 1992.

[The graph sandwich problem for property X is defined. Many $\mathcal{NP}$-completeness proofs are given for comparability graphs, permutation graphs and several other families.]

[GKS93] M. C. Golumbic, H. Kaplan, and R. Shamir. On the complexity of DNA physical mapping. Technical report 271/93, The Moise and Frida Eskenasy Institute of Computer Sciences, Tel Aviv University, January 1993. to appear *Advances in Applied Mathematics*.
[They independently defined the k-ICG problem under the name Colored Interval Sandwich Problem and show it to be $\mathcal{NP}$-complete by a reduction that does not imply W-hardness.]

[GLR94a] R. Govindan, M. A. Langston, and S. Ramachandramurthi. Cutwidth approximation in linear time. Technical report, Dept. of Computer Science, University of Tennessee, Knoxville, TN 37996–1301, 1994.
[An algorithm to determine whether K_4 is immersed in a graph is given.]

[GLR94b] R. Govindan, M. A. Langston, and S. Ramachandramurthi. A practical approach to layout optimization. Technical report, Dept. of Computer Science, University of Tennessee, Knoxville, TN 37996–1301, 1994.
[Presents fast layout algorithms generated by using partial obstruction sets.]

[GN94] A. Gupta and N. Nishimura. Sequential and parallel algorithms for embedding problems on classes of partial k-trees. In *Proceedings of the 4rd Scandinavian Workshop on Algorithm Theory*, volume 824 of *Lecture Notes on Computer Science*, pages 172–182, July 1994.
[Includes subgraph isomorphism and topological embedding algorithms, known to be $\mathcal{NP}$-complete for general partial k-trees.]

[GSK94] D. Granot and D. Skorin-Kapov. On some optimization problems on k-trees and partial k-trees. *Discrete Applied Mathematics*, 48:129–145, 1994.

[GS84] E. M. Gurari and I. H. Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the MinCut linear arrangement problem. *Journal of Algorithms*, 5:531–564, 1984.
[Bandwidth and cutwidth for fixed k can be solved in $O(n^k)$.]

[GS94] D. Grinstead and P. Slater. A recurrence template for several parameters in series-parallel graphs. *Discrete Applied Mathematics*, 54:151–168, 1994.

[GSV84] Y. Gurevich, L. Stockmeyer, and U. Vishkin. Soving $\mathcal{NP}$-hard problems on graphs that are almost trees and an application to facility location problems. *Journal of the Association Computing Machinery*, 31:459–473, 1984.

[GT78] M. R. Garey and R. E. Tarjan. A linear-time algorithm for finding all feedback vertices. *Information Processing Letters*, 7(6):274–276, October 1978.
[The authors present an $O(m)$-time algorithm which uses depth-first search to find all feedback vertices in an arbitrary strongly connected graph. And generalized for fixed k, the algorithm finds feedback sets of size k in time $O(n^{k-1}m)$.]

[Gup90] A. Gupta. *Constructive Issues in Tree Minors*. Ph.D. thesis, University of Toronto, 1990.

[Gives a constructive proof that trees are well-quasi-ordered under the minor ordering and shows that finite automata are sufficient for recognizing minor closed tree families.]

[Har69] F. Harary. *Graph Theory*. Addison-Wesley, Mass., 1969.

[He91] X. He. Efficient parallel algorithms for series-parallel graphs. *Journal of Algorithms*, 12(3):409–430, 1991.
[Contains algorithms for 3-coloring and depth first & breadth first - spanning trees.]

[HHL+87] E. Hare, S. Hedetniemi, R. Laskar, K. Peters, and T. Wimer. Linear-time computability of combinatorial problems on generalized-series-parallel graphs. In D. S. Johnson, T. Nishizeki, A. Nozaki, and H. S. Wilf, editors, *Perspectives in Computing*, volume 15, pages 437–457. Academic Press, Inc., 1987. Proceedings of the Japan-U.S. Joint Seminar, June 4–6, 1986, Kyoto, Japan.
[Illustrates an emerging theory/methodology for constructing linear-time graph algorithms by providing such algorithms for finding the maximum-cut and the maximum cardinality of a minimal dominating set for a generalized series-parallel graph.]

[HM87] J. P. Hutchinson and G. L. Miller. On deleting vertices to make a graph of positive genus planar. *Prospectives in Computing*, 15:81–98, 1987.

[HR90] N. Hartsfield and G. Ringel. *Pearls in Graph Theory: A Comprehensive Introduction*. Academic Press, Inc., 1990.

[HT74] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of Algorithms*, 21:549–568, 1974.
[The classic first-known linear-time algorithm for testing planarity is presented here.]

[HT86] R. Hassin and A. Tamir. Efficient algorithms for optimization and selection on series-parallel graphs. *SIAM Journal on Algebraic Discrete Methods*, 7:379–389, 1986.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory*. Addison-Wesley, Mass., 1979. Formerly titled *Formal Languages and their Relation to Automata* (1969).

[HY87] X. He and Y. Yesha. Binary tree algebraic computation and parallel algorithms for simple graphs. *Journal of Algorithms*, 9:92–113, 1987.
[Presents dynamic programming techniques for series parallel graph and other related families.]

[IR77] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 1–10, 1977. 9th STOC.
[Gives an $O(n^2)$ algorithm which finds either the shortest cycle in a graph, or shortest plus one.]

[Joh85] D. S. Johnson. The $\mathcal{NP}$-completeness column: An ongoing guide. *Journal of Algorithms*, 6:434–451, 1985. 16th edition; reduced graph family instances.

[Joh87] D. S. Johnson. The $\mathcal{NP}$-completeness column: An ongoing guide. *Journal of Algorithms*, 8:285–303, 1987. 19th edition; The many faces of polynomial time. [Discusses Robertson–Seymour subgraph homeomorphism and minor containment problems as well as nonconstructive proofs.]

[Kar90] A. Karabeg. Classification and detection of obstructions to planarity. *Linear and Multilinear Algebra*, 26:15–38, 1990.

[KGS95] D. Kaller, A. Gupta, and T. Shermer. Regular-factors in the complements of partial k-trees. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures, WADS'95*, volume 955 of *Lecture Notes on Computer Science*, pages 403–414. Springer-Verlag, August 1995.

[Kin92] N. G. Kinnersley. The vertex separation number of a graph equals its pathwidth. *Information Processing Letters*, 42:345–350, 1992.
[Contains a simple proof showing the equivalence between the pathwidth and a VLSI linear layout problem.]

[Kin89] N. G. Kinnersley. *Obstruction set isolation for layout permutation problems.* Ph.D. Thesis, Dept. of Computer Science, Washington State University, Pullman, WA 99164, 1989.

[Kin94] N. G. Kinnersley. Constructive obstruction set isolation for the Min Cut Linear Arrangement. Technical Report TR-94-1, Dept. of Computer Science, University of Kansas, January 1994.

[KIU86] Y. Kajitani, A. Ishizuka, and S. Ueno. Characterization of partial 3-trees in terms of 3 structures. *Graphs and Combinatorics*, 2:233–246, 1986.

[KK94a] N. G. Kinnersley and W. M. Kinnersley. An efficient polynomial-time algorithm for three-track gate matrix layout. *The Computer Journal*, 37(5):449–462, 1994.

[KK94b] N. G. Kinnersley and W. M. Kinnersley. Tree automata for cutwidth recognition. Technical Report TR-94-2, Dept. of Computer Science, University of Kansas, January 1994.

[KL94] N. G. Kinnersley and M. A. Langston. Obstruction set isolation for the Gate Matrix Layout problem. *Discrete Applied Mathematics*, 54:169–213, 1994.

[Klo93a] T. Kloks. *Treewidth.* Ph.D. dissertation, Dept. of Computer Science, Utrecht University, P.O. Box 80.012, 3508 TA Utrecht, the Netherlands, 1993.
[Among many results, this dissertation gives an algorithm to determine if a graph is pathwidth (or treewidth) $\leq k$ when given a decomposition of any width greater than k.]

[Klo93b] T. Kloks. In *Treewidth: computations and approximations*, volume 842 of *Lecture Notes on Computer Science*. Springer-Verlag, 1993.
[This is a more available source than Kloks' Ph.D. dissertation.]

[KM92] A. Kézdy and P. McGuinnes. Sequential and parallel algorithms to find a K_5 minor. *Discrete Applied Mathematics*, 36:87–92, 1992.
[An $O(n^2)$ sequential algorithm is presented that, given a graph, either returns a K_5 minor or reports that no such minor exists.]

[Koz92] D. Kozen. On the Myhill-Nerode Theorem for trees. *Bulletin of the EATCS*, 47:170–173, 1992.

[KP86] L. M. Kirousis and C. H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47:205–216, 1986.

[KPS93] M. Khalifat, T. Politof, and A. Satyanarayana. On minors of graphs with at least $3n - 4$ edges. *Journal of Graph Theory*, 17(4):523–529, 1993.

[Kra87] D. Kratsch. Finding the minimum bandwidth of an interval graph. *Information and Computation*, 74(2):140–158, 1987.
[Gives an $O(n^2)$ algorithm for determining the bandwidth of interval graphs although it is known to be $\mathcal{NP}$-complete for catepillers with hairs of length at most 3 and for binary trees.]

[Kru60] J. B. Kruskal. Well-quasi-ordering, the tree theorem, and Vaszsonyi's conjecture. *Transactions of American Mathematical Society*, 95:210–225, 1960.

[Kru72] J. B. Kruskal. The theory of well-quasi-ordering: a frequently rediscovered concept. *Journal of Combinatorial Theory, Series A*, 13:297–305, 1972.

[KS93] H. Kaplan and R. Shamir. Pathwidth, bandwidth and completion problems to proper interval graphs. Technical report 285/93, The Moise and Frida Eskenasy Institute of Computer Sciences, Tel Aviv University, November 1993.
[Two problems motivated by molecular biology are studied. Both are shown to be polynomial for fixed k but, in general, one is $\mathcal{NP}$-hard while the other is hard for W[1].]

[KST94] H. Kaplan, R. Shamir, and R. E. Targan. Tractability of parameterized completion problems on chordal and interval graphs: minimum fill-in and physical mapping. In *Proceedings of the 35th Annual IEEE Conference on the Foundations of Computer Science*, pages 780–791, 1994.

[KT92] A. Kornai and Z. Tuza. Narrowness, pathwidth, and their application in natural language processing. *Discrete Applied Mathematics*, 36:87–92, 1992.
[Points out the importance of graphs with pathwidth ≤ 6 in connection with natural language processing.]

[Kur30] K. Kuratowski. Sur le problè des courbes gauches en topologie. *Fundamental Mathematics*, 15:271–283, 1930.

[KyK83] T. Kikuno, N. yoshida, and Y. Kakuda. A linear algorithm for the domination number of series-parallel graphs. *Discrete Applied Mathematics*, 37:299–311, 1983.

[LA91] J. Lagergren and S. Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, volume 510 of *Lecture Notes on Computer Science*, pages 533–543. Springer-Verlag, 1991. 18th ICALP.
[Using an algebra of i-sourced graphs, an effective way to compute the minimal forbidden minors of graphs of bounded treeidth is derived from finite congruences that recognizes minor-closed families. Also, an algorithm that recognizes graphs of treewidth at most k that runs in linear time is presented.]

[Lag90] J. Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *IEEE Symposium on Foundations of Computer Science Proceedings*, pages 173–182, 1990. 31th FOCS.

[Lag91] J. Lagergren. *Algorithms and Minimal Forbidden Minors for Tree-decomposible Graphs*. Ph.D. dissertation, Royal Institute of Technology, Stockholm, Sweden, March 1991. Dept. of Numerical Analysis and Computing Sciences.

[Lag93] J. Lagergren. An upper bound on the size of an obstruction. In N. Robertson and P. D. Seymour, editors, *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 601–621, 1993.
[The author proves constructively that every minor closed family which is recognized by a finite congruence and has an obstruction set that is of bounded treewidth has a finite number of obstructions. A bound $O(2^{c^{w+1}w!})$ on the maximum number of edges of an obstruction is given in terms of the treewidth bound w and the index c of a finite congruence.]

[Lag94] J. Lagergren. The nonexistence of reduction rules giving an embedding into a k-tree. *Discrete Applied Mathematics*, 54:219–224, 1994.

[Lag95] J. Lagergren. Upper bounds on the size of obstructions and intertwines, 1995. Dept. of Numerical Analysis and Computing Science Manuscript, The Royal Institute of Technology, Stockholm, Sweden.
[The author proves constructively that if a minor closed family L has obstructions of treewidth at most k and there is a pseudo minor order with finite height for L, then L has a finite number of obstructions. For another pseudo minor order, an upper bound on the size of an intertwine of two given planar graphs H and H' is also obtained.]

[Lan92] M. A. Langston. Current progress on obstruction-based layout optimization. Technical report, Dept. of Computer Science, University of Tennessee, Knoxville, TN 37996–1301, 1992.

[Lau88] C. Lautemann. Decomposition trees: Structured graph representation and efficient algorithms. In *Proceedings of the 13th Colloquium on Trees in Algebra and Programming CTAP'88*, volume 299 of *Lecture Notes on Computer Science*, pages 28–39. Springer-Verlag, 1988.

[LB62] C. G. Lekkerkerker and J. C. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51:45–64, 1962.

[Len81] T. Lengauer. Black-white pebbles and graph separation. *Acta Informatica*, 16:465–475, 1981.

[Lei81] F. Leighton. New lower bound techniques for VLSI. In *IEEE Symposium on Foundations of Computer Science Proceedings*, volume 22, pages 1–22, 1981.

[LG80] P. Liu and R. Geldmacher. An $O(\max(m,n))$ algorithm for finding a subgraph homeomorphic to K_4. *Congressus Numerantium*, 29:597–609, 1980. (Also in the Proceedings of the Eleventh Southeastern Conference on Combinatorics, Graph Theory, and Computing, 1980, 597–609.).

[Lin89] A. Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63:295–302, 1989.

[LPS95] G. T. Linger, T. Politof, and A. Satyanarayana. A forbidden minor characterization and reliability of a class of partial 4-trees. *Networks*, 25:139–146, 1995.
[Gives seven minor-order obstructions to a family of graphs that is contained in the set of partial 4-trees.]

[LR92] M. A. Langston and S. Ramachandramurthi. Dense layouts for series-parallel circuits. Technical report, Dept. of Computer Science, University of Tennessee, Knoxville, TN 37996–1301, 1992.
[Uses K_4 as the only test for the Gate Matrix Layout problem with three tracks.]

[LT79] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[Lu93] X. Lu. Finite state properties of bounded pathwidth graphs. Master's project report, Dept. of Computer Science, University of Victoria, P.O. Box 3055, Victoria, B.C., Canada V8W 3P6, 1993.

[Mah80] N. V. R. Mahadar. Master's thesis, University of Waterloo, 1980.
[Found (checked?) that there were 35 minor-order minimal graphs within the previously known topological projective plane obstructions.]

[Man90] J. B. Manning. *Geometric Symmetry in Graphs*. Ph.D. thesis, Purdue University, December 1990.

[MCH79] S. L. Mitchell, E. J. Cockayne, and S. T. Hedetniemi. Linear algorithms on recursive representations of trees. *Journal of Computer and System Sciences*, 18(1):76–85, 1979.
[The authors claim that their algorithm techniques generalize to k-trees and chordal graphs.]

[Mil85] E. C. Milner. Basic WQO and BQO theory. In I. Rival, editor, *Graphs and Orders*, volume 147 of *Mathematical and Physical Sciences*, pages 487–502. D. Reidel Publishing Company, Amsterdam, 1985.
[A survey of the achievements in the theory of well-quasi-(wqo) and better-quasi-ordering (bqo) since the early 1950's.]

[Mir94] B. G. Mirkin. *Graphs and Genes*. Springer-Verlag, 1994. (translated from Russian by H.L. Beus).

[Mit79] S. L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters*, 9(5):229–232, December 1979.

[MK90] P. J. McGuinnes and A. E. Kézdy. An algorithm to find a K_5 minor, 1990.
[An $O(n^2)$ algorithm is presented that, given a graph, either returns a K_5 minor or reports that no such minor exists.]

[MM91] E. Mata-Montero. Resilience of partial k-tree networks with edge and node failures. *Networks*, 21:321–344, 1991.

[Möh90] R. H. Möhring. Graph problems releted to gate matrix layout and PLA folding. In G. Tinhofer, E. Mayr, H. Noltemeier, and M. S. (in cooperation with R. Albrecht), editors, *Computational Graph Theory, Computing Supplementum 7*, pages 17–51. Springer-Verlag, 1990.
[Graph problems, occurring in linear VLSI layout architectures such as gate matrix layout (pathwidth), folding of programmable logic arrays, and Weinberger arrays are surveyed.]

[Moh93] B. Mohar. Projective planarity in linear time. *Journal of Algorithms*, 15:482–502, 1993.
[First such linear-time algorithm for testing projective planarity based on the embeddings of K_5 and $K_{3,3}$.]

[Moo69] J. Moon. The number of labeled k-trees. *Journal of Combinatorial Theory, Series B*, 6:196–199, 1969.

[MPS85] F. Makedon, C. H. Papadimitriou, and I. H. Sudborough. Topological bandwidth. *SIAM Journal on Algebraic Discrete Methods*, 6(3):418–444, 1985.

[MRS88] R. Motwani, A. Raghunathan, and H. Saran. Constructive results from graph minors: Linkless embeddings. In *IEEE Symposium on Foundations of Computer Science Proceedings*, pages 398–407, 1988. 29th FOCS, White Plains, N.Y.
[The authors prove that there are seven linkless embeddable minor-minimal obstructions.]

[MS83] F. Makedon and I. H. Sudborough. Minimizing width in linear layouts. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, volume 154 of *Lecture Notes on Computer Science*, pages 478–490. Springer-Verlag, 1983. Also see *Discrete Applied Mathematics* 23 (1989), 243–265.
[Contains forbidden subgraphs for cutwidth 2 and also characterizes cutwidth 3 family. A dynamic programming algorithm is improved to show that, for any $k \geq 2$, the problem of deciding if a given graph has cutwidth at most k can be done in $O(n^{k-1})$.]

[MS88] B. Monien and I. H. Sudborough. Min cut is $\mathcal{NP}$-complete for edge weighted trees. *Theoretical Computer Science*, 58:209–229, 1988. Also in Lecture Notes in Computer Science 226 (1986), pp. 265–274.
[The Min Cut Linear Arrangement Problem (cutwidth) is $\mathcal{NP}$-complete for planar graphs with maximum vertex degree three. This is used to show that the Search Number, Vertex Separation (pathwidth), Progressive Black/White Pebble Demand, and Topological Bandwidth is $\mathcal{NP}$-complete for planar graphs with maximum vertex degree three.]

[MT91] J. Matoušek and R. Thomas. Algorithms finding tree-decompositions of graphs. *Journal of Algorithms*, 12:1–22, 1991.
[A probalistic algorithm (maybe practical!) is given with execution time $O(n \log^2 n + n \log n |\log p|)$ that finds either a tree-decomposition of width $\leq 6k$ or answers that the tree-width is $> k$.]

[MT92] J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108:343–364, 1992.

[Nar89] C. Narayanan. *Fast parallel algorithms and enumeration techniques for partial k-trees.* Ph.D. dissertation, Dept. of Computer Science, Clemson University, August 1989.
[Gives an overview of parallel algorithms for partial k-trees.]

[Nar90] C. Narayanan. Isomorphism testing of k-trees is in $\mathcal{NC}$, for fixed k. *Information Processing Letters*, 34:283–287, 1990.

[Ner58] A. Nerode. Linear automata transformations. *Proceedings of the American Mathematical Society*, 9:541–544, 1958.

[Neu93] E. T. Neufeld. Practical toroidality testing. Master's thesis, Dept. of Computer Science, University of Victoria, P.O. Box 3055, Victoria, B.C., Canada V8W 3P6, 1993.

[NW63] C. S. J. A. Nash-Williams. On well-quasi-ordering finite trees. *Proceedings of the Cambridge Philosophical Society*, 59:833–835, 1963.
[Presents a simple proof that any sequence of finite trees $T_1, T_2, \ldots$, there exists $i < j$ such that T_i is homeomorphic to a subtree of T_j.]

[OvW79] R. Otten and J. van Wijk. Graph representations in interactive layout design. In *Proceedings of the IEEE Symposium on Circuits and Systems*, pages 914–918, 1979.

[Par76] T. D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and applications of graphs*, pages 426–441. Springer-Verlag, 1976.

[PLH84] J. Pfaff, R. Laskar, and S. T. Hedetniemi. Linear algorithms for independent domination and total domination in series-parallel graphs. *Congressus Numerantium*, 45:71–82, 1984.

[Pou85] M. Pouzet. Applications of well quasi-ordering and better quasi-ordering. In I. Rival, editor, *Graphs and Orders*, volume 147 of *Mathematical and Physical Sciences*, pages 503–519. D. Reidel Publishing Co., 1985.

[Pro79] A. Proskurowski. Minimum dominating cycles in 2-trees. *International Journal of Computation and Information Science*, 8:405–417, 1979.

[Pro84] A. Proskurowski. Separating subgraphs in k-trees: Cables and caterpillars. *Discrete Mathematics*, 49:275–285, 1984.

[Pro93] A. Proskurowski. Graph reductions, and techniques for finding minimal forbidden minors. In N. Robertson and P. D. Seymour, editors, *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 591–600. American Mathematical Society, 1993.
[A brief historical account of some techniques used to find small sets of minimal forbidden minors for a few classes of graphs with treewidth at most 3 is given.]

[PRS92] A. Proskurowski, F. Ruskey, and M. Smith. Listing k-paths. Technical Report DCS-178-IR, Dept. of Computer Science, University of Victoria, P.O. Box 3055, Victoria, B.C. Canada V8W 3P6, June 1992.
[A simple and efficient algorithm for enumerating k-paths represented as k-ary strings is given.]

[PS81] A. Proskurowski and M. M. Syslo. Minimum dominating cycles in outerplanar graphs. *International Journal of Computation and Information Science*, 10:127–139, 1981.

[PS90] A. Proskurowski and M. M. Syslo. Efficient computations in tree-like graphs. In G. Tinhofer, E. Mayr, H. Noltemeier, and M. S. (in cooperation with R. Albrecht), editors, *Computational Graph Theory, Computing Supplementum 7*, pages 1–15. Springer-Verlag, 1990.
[This is a good survey paper on efficient algorithms for partial k-trees.]

[PS94] T. Politof and A. Satyanarayana. Minors of quasi 4-connected graphs. *Discrete Mathematics*, 126:245–256, 1994.

[Ram94] S. Ramachandramurthi. *Algorithms for VLSI Layout Based on Graph Width Metrics*. Ph.D. dissertation, Dept. of Computer Science, University of Tennessee, Knoxville, TN 37996–1301, August 1994.

[Ree92] B. A. Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of the ACM Symposium on the Theory of Computing*, volume 24, pages 221–228, 1992.
[Gives an approximate separator algorithm that yields an $O(n \log n)$ algorithm that finds a width k (for fixed k) tree decomposition if it exists.]

[Ric85] M. B. Richey. *Combinatorial optimization on series-parallel graphs: algorithms and complexity*. Ph.D. dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1985.

[Ros73] D. J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7:317–322, 1973.
[Contains four simple characterizations of k-trees involving cliques, paths, and separators.]

[RS83] N. Robertson and P. D. Seymour. Graph Minors. I. Excluding a Forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[RS86a] N. Robertson and P. D. Seymour. Graph Minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[RS84c] N. Robertson and P. D. Seymour. Graph Minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36:49–64, 1984.

[RS90a] N. Robertson and P. D. Seymour. Graph Minors. IV. Tree-width and well-quasi-ordering. *Journal of Combinatorial Theory, Series B*, 48:227–254, 1990.

[RS84d] N. Robertson and P. D. Seymour. Graph Minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41:92–114, 1984.

[RS86b] N. Robertson and P. D. Seymour. Graph Minors. VI. Disjoint paths across a disc. *Journal of Combinatorial Theory, Series B*, 41:115–138, 1986.

[RS88] N. Robertson and P. D. Seymour. Graph Minors. VII. Disjoint paths on a surface. *Journal of Combinatorial Theory, Series B*, 45:212–254, 1988.

[RS90c] N. Robertson and P. D. Seymour. Graph Minors. VIII. A Kuratowski theorem for general surfaces. *Journal of Combinatorial Theory, Series B*, 48:255–288, 1990.

[RS90b] N. Robertson and P. D. Seymour. Graph Minors. IX. Disjoint crossed paths. *Journal of Combinatorial Theory, Series B*, 49:40–77, 1990.

[RS91a] N. Robertson and P. D. Seymour. Graph Minors. X. Obstructions to tree-decompositions. *Journal of Combinatorial Theory, Series B*, 52:153–190, 1991.

[RS94] N. Robertson and P. D. Seymour. Graph Minors. XI. Circuits on a surface. *Journal of Combinatorial Theory, Series B*, 60:72–106, 1994.

[RS95a] N. Robertson and P. D. Seymour. Graph Minors. XII. Distance on a surface. *Journal of Combinatorial Theory, Series B*, 64(2):240–, July 1995.

[RS95b] N. Robertson and P. D. Seymour. Graph Minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

[RS91b] N. Robertson and P. D. Seymour. Graph Minors. XIV. Extending an embedding, 1991. submitted.

[RSa] N. Robertson and P. D. Seymour. Graph Minors. XV. Giant steps. submitted.

[RSb] N. Robertson and P. D. Seymour. Graph Minors. XVI. Excluding a non-planar graph. submitted.

[RS87] N. Robertson and P. D. Seymour. Graph Minors. XVII. Taming a vortex, May 1987. submitted.

[RSh] N. Robertson and P. D. Seymour. Graph Minors. XVIII. Tree-decompositions and well-quasi-ordering. in progress.

[RSg] N. Robertson and P. D. Seymour. Graph Minors. XIX. Well-quasi-ordering on a surface. in progress.

[RSc] N. Robertson and P. D. Seymour. Graph Minors. XX. Wagner's conjecture. in progress.

[RSd] N. Robertson and P. D. Seymour. Graph Minors. XXI. Graphs with unique linkages. submitted.

[RSe] N. Robertson and P. D. Seymour. Graph Minors. XXII. Irrelevant vertices in linkage problems. submitted.

[RSf] N. Robertson and P. D. Seymour. Graph Minors. XXIII. Nash-Williams' immersions conjecture. in progress.

[RS42] J. Riordan and C. E. Shannon. The number of two-terminal series-parallel networks. *Journal of Mathematical Physics*, 21:83–93, 1942.

[RS84a] N. Robertson and P. D. Seymour. Graph width and well-quasi-ordering: a survey. In J. Bondy and U. Murty, editors, *Progress in Graph Theory*, pages 399–406. Academic Press, Inc., Toronto, 1984.

[RS84b] N. Robertson and P. D. Seymour. Generalizing Kuratowski's theorem. *Congressus Numerantium*, 45:129–138, 1984.

[RS84e] N. Robertson and P. D. Seymour. Some new results on the well quasi ordering of graphs. In M. Pouzet and D. Richard, editors, *Orders: Description and Roles*, volume 23 of *Annals of Discrete Mathematics*, pages 343–354. Elsevier Science, Amsterdam, 1984. Proceedings of the Conference on Ordered Sets and Their Applications, Château de la Tourette, l'Arbresle, July 5-11, 1982.

[RS85a] N. Robertson and P. D. Seymour. Disjoint paths – A survey. *SIAM Journal on Discrete Mathematics*, 6:300–305, 1985.

[RS85b] N. Robertson and P. D. Seymour. Graph Minors – A survey. In *Surveys in Combinatorics*, volume 103, pages 153–171. Cambridge University Press, 1985. [Presents a survey of results concerning Wagner's conjecture and the Disjoint Connecting Paths problem.]

[RS90d] N. Robertson and P. D. Seymour. *An Outline of a Disjoint Paths Algorithm*, pages 267–292. 1990. *Algorithms and Combinatorics*, Volume 9.

[RS93a] N. Robertson and P. D. Seymour. Excluding a graph with one crossing. In N. Robertson and P. D. Seymour, editors, *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 669–675, 1993.

[RS93b] N. Robertson and P. D. Seymour, editors. *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*. American Mathematical Society, 1993. Proceedings of a Joint Summer Research Conference on Graph Minors held June 22 to July 5, 1991, at the University of Washingtion, Seattle.

[RST93c] N. Robertson, P. D. Seymour, and R. Thomas. A survey of linkless embeddings. In N. Robertson and P. D. Seymour, editors, *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 125–136. AMS, 1993.
[It is shown that a graph has a flat (linkless) embedding if and only if it has no minor isomorphic to one of seven Peterson family obstructions These graphs are obtained from K_6 by means of $Y\Delta$- and ΔY-exchanges.]

[RST93b] N. Robertson, P. D. Seymour, and R. Thomas. Structural descriptions of lower ideals of trees. In N. Robertson and P. D. Seymour, editors, *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 525–538, 1993.

[RST93a] N. Robertson, P. D. Seymour, and R. Thomas. Linkless embeddings of graphs in 3-space. *Bulletin of the American Mathematical Society*, 28:84–89, 1993.

[RST94] N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62:323–, 1994.

[RST95a] N. Robertson, P. D. Seymour, and R. Thomas. Kuratowski chains. *Journal of Combinatorial Theory, Series B*, 64(2):127–154, 1995.

[RST95b] N. Robertson, P. D. Seymour, and R. Thomas. Petersen family minors. *Journal of Combinatorial Theory, Series B*, 64(2):155–184, 1995.

[RST95c] N. Robertson, P. D. Seymour, and R. Thomas. Linkless embedding conjecture. *Journal of Combinatorial Theory, Series B*, 64(2):185–, 1995.

[RTL76] D. J. Rose, R. E. Tarjan, and G. S. Leuker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.
[Relates graph elimination to performing Gaussian elimination on sparse symmetric positive definite matrices.]

[San92] D. P. Sanders. On linear recognition of tree-width at most four, 1992. Georgia Tech manuscript (submitted).

[Sar89] H. Saran. *Constructive Results from Graph Minors: Linkless Embeddings.* Ph.D. Thesis, University of California, Berkeley, 1989.

[Sch89] P. Scheffler. An $O(n \log(n))$ algorithm for the pathwidth of trees. In *Fundamentals of Computation Theory (FCT'89)*, 1989. Karl-WeierstraB-Institut Für Mathematik, Akademie der Wissenshaften der DDR, Mohrenstr. 39. Berlin, DDR-1086.

[Sch90] P. Scheffler. A linear algorithm for the pathwidth of trees. In R. Bodendiek and R. Henn, editors, *Topics in combinatorics and graph theory*, pages 613–620. Physica-Verlag, 1990.

[SD76] D. C. Schmidt and L. E. Druffel. A fast backtracking algorithm. *Journal of the Association Computing Machinery*, 23(3):?–445, July 1976.
[Contains a nice graph isomorphism algorithm which is very practical. (We use this.)]

[Sey76a] P. D. Seymour. A forbidden minor characterization of matroid ports. *Quarterly Journal of Mathematics, Oxford*, 27(2):407–413, 1976.

[Sey76b] P. D. Seymour. The forbidden minors of binary clutters. *Journal of London Mathematical Society*, 12(2):356–360, 1976.

[Sey90] P. D. Seymour. Colouring series-parallel graphs. *Combinatorica*, 10:379–392, 1990.

[Sey93] P. D. Seymour. A bound on the excluded minors for a surface, February 1993. Bell Communications Research, Morristown, New Jersey.
[For every surface of complexity g (e.g. twice the genus), every excluded minor has at most 2^{2^k} vertices, where $k = (3g+9)^9$. The treewidth is also bounded by $14(g+3)^3$.]

[SI79] M. Syslo and M. Iri. Efficient outerplanarity testing. *Fundamenta Informaticae*, 2:261–275, 1979.

[ST90] A. Satyanarayana and L. Tung. A characterization of partial 3-trees. *Networks*, 20:299–322, 1990.

[ST93] P. D. Seymour and R. Thomas. Graph searching, and a min-max theorem for treewidth. *Journal of Combinatorial Theory, Series B*, 58:22–33, 1993.

[Ste92] M. Steinby. On generalizations of the Myhill-Nerode Theorems. *Bulletin of the EATCS*, 48:191–196, 1992.

[Sti93] J. Stillwell. *Classical Topology and Combinatorial Group Theory*, volume 72 of *Graduate Texts in Mathematics*. Springer-Verlag, second edition, 1993.

[SW75] G. Smith and R. Walford. The identification of a minimal feedback vertex of a directed graph. *IEEE Transactions on Circuits and Systems*, CAS-22:146–160, 1975.

[Sys82] M. M. Syslo. The subgraph isomorphism problem for outerplanar graphs. *Theoretical Computer Science*, 17:91–97, 1982.

[Sys83] M. M. Syslo. $\mathcal{NP}$-complete problems on some tree-structured graphs: a review. In *Proceedings of the Workshop on Graph Theoretic Concepts in Computer Science (WG'93)*, pages 342–353. Trauner Verlag, Linz (1984), June 1983.

[TGS95] P. J. Tanenbaum, M. T. Goodrich, and E. R. Scheinerman. Characterization and recognition of point-halfspace and related orders. In *Graph Drawing: DIMACS International Workshop GD'94 Princeton, N.J., October 1994 Proceedings*, volume 894 of *Lecture Notes on Computer Science*, pages 234–245. Springer-Verlag, 1995.
[The authors characterize four classes of geometric membership and containment orders structurally in terms of forbidden subposets.]

[Tha73] J. W. Thatcher. Tree automata: an informal survey. In A. V. Aho, editor, *Currents in the Theory of Computing*, pages 143–172. Prentice-Hall, 1973.
[This is a good introduction to tree automata.]

[Tho] C. Thomassen. *Handbook of Combinatorics*, chapter Embeddings and minors. North Holland. (to appear).

[Tho90] R. Thomas. A Menger-like property of tree-width. the finite case. *Journal of Combinatorial Theory, Series B*, 48:67–76, 1990.
[Shows the existence of linked (vertex disjoint paths between bags) tree-decompositions.]

[Tho95] R. Thomas. private communication, 1995. Dept. of Mathematics, Georgia Institute of Technology.

[TM76] W. T. Trotter and J. I. Moore. Characterization problems for graphs, partially ordered sets, lattices and families of sets. *Discrete Mathematics*, 4:361–368, 1976.

[TNS82] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *Journal of the Association Computing Machinery*, 29:623–641, 1982.
[Shows that $\mathcal{NP}$-complete problems characterizable in terms of a finite number of forbidden subgraphs admit linear-time algorithms if their instances are restricted to series-parallel graphs.]

[Tro92] W. T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory*. Johns Hopkins Univ. Press, Baltimore, 1992. Johns Hopkins Series in Mathematical Sciences.

[Tru93] R. J. Trudeau. *Introduction to Graph Theory*. Dover, 1993. Revised edition of *Dots and Lines* (1976).

[TUK91] A. Takahashi, S. Ueno, and Y. Kajitani. Minimal acyclic forbidden minors for the family of graphs with bounded path-width. In *SIGAL 91-19-3, IPSJ*, 1991. To appear in: *Annals of Discrete Mathematics* (Proceedings of 2nd Japan conference on graph theory and combinatorics, 1990).
[Shows that there are at least $k!^2$ tree obstructions for the family of graphs with pathwidth at most k.]

[TW65] J. W. Thatcher and J. B. Wright. Generalized finite automata. *Notices of the American Mathematical Society*, 12:820, 1965.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–81, 1968.
[Many of the important concepts and results of conventional finite automata theory are developed for a generalization in which finite algebras take the place of finite automata.]

[vL90] J. van Leeuwen. *Handbook of Theoretical Computer Science: A: Algorithms and Complexity Theory.* MIT Press / North Holland Publishing Company, Amsterdam, 1990. See "Graph Algorithms" chapter in pages 527–631.
[This handbook contains several sections regarding combinatorial bounded width and forbidden minors.]

[Vol86] W. Vollmerhaus. On computing all minimal graphs that are not embeddable in the projective plane, parts I and II., 1986. (cited manuscript).
[The author independently verified the 103 projective plane irreducible graphs and may have also verified the 35 minor-minimal graphs.]

[VTL82] J. Valdes, R. Tarjan, and E. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11:298–313, 1982. (Also see STOC'79, pages 1–12).
[Presents a 4-vertex forbidden subgraph for vertex series parallel DAGs (in the transitive closure).]

[Wag37] K. Wagner. Uber einer eigenschaft der ebener complexe. *Mathematics Annals*, 14:570–590, 1937.

[WC83] J. A. Wald and C. J. Colbourn. Steiner trees, partial 2-trees and minimum IFI networks. *Networks*, 13:159–167, 1983.
[Studying isolated failure immune networks, the authors find a linear time Steiner tree algorithm for 2-trees.]

[Weg67] G. Wegner. *Eigenshaften der nerven homologische eihfacter familien in R^n.* Ph.d. thesis, Götingen, 1967.
[Gives three induced subgraph obstructions (asteroidal triples) for chordal graphs which are not proper interval graphs.]

[WHL85] T. V. Wimer, S. T. Hedetniemi, and R. Laskar. A methodology for constructing linear time graph algorithms. *Congressus Numerantium*, 50:43–60, 1985.

[Wil79] R. J. Wilson. *Introduction to Graph Theory.* Academic Press, Inc., 1979.

[Wil87] H. S. Wilf. Finite list of obstructions (the editor's corner). *American Mathematics Monthly*, pages 267–271, March 1987.
[An introduction to the notion of obstructions and introduces the work of Robertson and Seymour.]

[Wil89] H. S. Wilf. *Combinatorial Algorithms: An Update*, volume 55 of *CBMS-NSF Regional Conference Series in Applied Mathematics*, chapter Listing Free Trees, pages 31–36. SIAM, 1989.

[Wim87a] T. V. Wimer. Linear algorithms for the dominating cycle problems in series-parallel graphs, 2-trees and Halin graphs. *Congressus Numerantium*, 56, 1987.

[Wim87b] T. V. Wimer. *Linear algorithms on k-terminal graphs.* Ph.D. dissertation, Dept. of Computer Science, Clemson University, August 1987. Report No. URI-030.

[Win86] P. Winter. Generalized Steiner problem in series-parallel networks. *Journal of Algorithms*, 7:549–566, 1986.
[A linear time algorithm is given for a generalized Steiner problem.]

[WROM86] R. A. Wright, B. Richard, A. Odlyzko, and B. D. McKay. Constant time generation of free trees. *SIAM Journal on Computing*, 15:540–548, 1986.

[WWY95] J. Wang, D. B. West, and B. Yao. Maximum bandwidth under edge addition. *Journal of Graph Theory*, 20(1):87–90, 1995.

[Yap83] C. K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26:287–300, 1983.
[The main result shows that if the 'non-uniform polynomial-time hierarchy' collapses at level $i > 0$, i.e., $\Sigma_i/\text{poly} = \Pi_i/\text{poly}$, then the Meyer–Stockmeyer hierarchy collapses at level $i + 2$, i.e., $\sum_{i+2} = \Pi_{i+2}$.]

[ZiNN94] X. Zhou, S. ichi Nakano, and T. Nishizeki. A parallel algorithm for edge-coloring partial k-trees. In *Proceedings of the 4rd Scandinavian Workshop on Algorithm Theory*, volume 824 of *Lecture Notes on Computer Science*, pages 359–369. Springer-Verlag, 1994.

Index

Vita

Surname: Dinneen **Given Names:** Michael John

Michael was born in Idaho Falls, Idaho (U.S.A.) and lived there with his parents, two brothers and a sister until he left for undergraduate school. After receiving two bachelor degrees, he enrolled in graduate school in 1989. Michael's first "real job" was a computer programming job at Idaho Falls School District #91 for just over four years during the school breaks of his undergraduate career. Recently, Michael has been working at Los Alamos National Laboratory (New Mexico) as a Graduate Research Assistant (for the Computer Research and Applications Group) for an average of six months a year since 1989.

Educational Institutions Attended:

University of Victoria, Victoria B.C.	1990–1995
Washington State University, Pullman WA	1989–1990
University of Idaho, Moscow ID	1985–1989

Degrees Awarded:

M.S. Computer Science, University of Victoria, May 1992
Thesis – "Algebraic Methods for Efficient Network Constructions"
B.S. Computer Science (Magna Cum Laude), University of Idaho, May 1989
B.S. Mathematics: Applied (Cum Laude), University of Idaho, May 1989

Awards and Scholarships:

University of Victoria Teaching Fellowship (1995)
University of Victoria Fellowship (1990–1994)
Washington State University Teaching Assistantship (1989–1990)
Chevron Oil's Science and Engineering Scholarship (1987–1989)
Taylor, Eugene and Osa Scholarship (1987–1989)
J. Lawrence Botsford Scholarship (1986–1987)
General University of Idaho Scholarship (1985–1986)
University of Idaho Dean's List: College of Engineering (1985–1989)
Faculty of Letters and Science (1987–1989)

Selected Papers:

1. Algebraic constructions of efficient broadcast networks. In H. F. Mattson, T. Mora, and T. R. N. Rao, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 539 of *Lecture Notes in Computer Science*, pages 150–158. Springer-Verlag, October 1991. Proceedings of the 9th International Symposium, AAECC-9. with Vance Faber and Michael R. Fellows.

2. Small diameter symmetric networks from linear groups. *IEEE Transactions on Computers*, 41:218–220, 1992. with Lowell Campbell, Gunnar E. Carlsson, Vance Faber, Michael R. Fellows, Michael A. Langston, James W. Moore, Andrew P. Mullhaupt, and Harlan B. Sexton.

3. Recent examples in the theory of partition graphs. *Discrete Mathematics*, 113:255–258, 1993. with Duane W. DeTemple, Kevin L. McAvaney, and Jack M. Robertson.

4. A characterization of graphs with vertex cover up to five. In V. Bouchitte and M. Morvan, editors, *Orders, Algorithms and Applications, ORDAL'94*, volume 831 of *Lecture Notes in Computer Science*, pages 86–99. Springer-Verlag, July 1994. with Kevin Cattell.

5. New results for the degree/diameter problem. *Networks*, 24:359–367, October 1994. with Paul R. Hafner.

6. Obstructions to within a few vertices or edges of acyclic. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures, WADS'95*, volume 995 of *Lecture Notes in Computer Science*, pages 415–427. Springer-Verlag, August 1995. with Kevin Cattell and Michael R. Fellows.

7. A computational attack on the conjectures of Graffiti: New counterexamples and proofs. *Discrete Mathematics*, In press (accepted 1994). with Tony L. Brewster and Vance Faber.

8. A simple linear-time algorithm for finding path-decompositions of small width. Accepted *Information Processing Letters.* with Kevin Cattell and Michael R. Fellows.