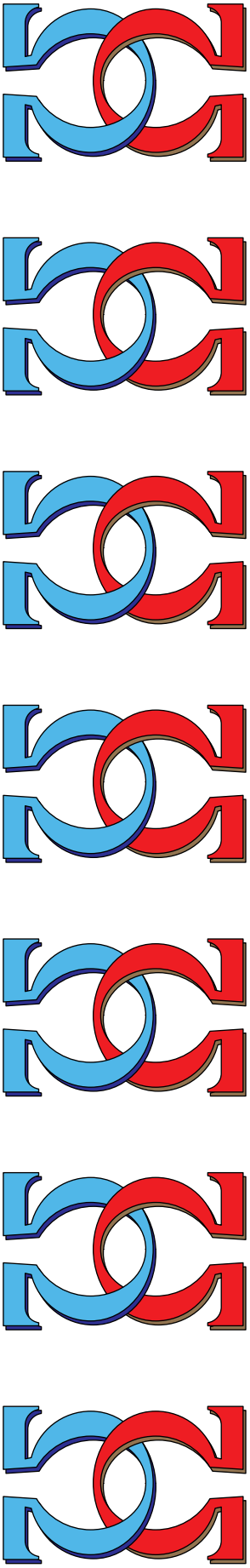


CDMTCS
Research
Report
Series

Obstructions for the Graphs of Vertex Cover Seven

Michael J. Dinneen
Ralph Versteegen

Department of Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-430
December 2012

Centre for Discrete Mathematics and
Theoretical Computer Science

Obstructions for the Graphs of Vertex Cover Seven

Michael J. Dinneen and Ralph Versteegen*

Department of Computer Science, University of Auckland,
Auckland, New Zealand

mjd@cs.auckland.ac.nz rver017@aucklanduni.ac.nz

Abstract

In this paper we present an optimized procedure for computing the minor-order obstructions for graphs of vertex cover at most k. This extends the earlier method and results of Cattell and Dinneen in 1994, for $k \leq 5$. Here we extended the known set of forbidden graphs for $k \leq 7$. To help reduce the size of these sets, we also mention some proposals for finding minimal forbidden graphs for other "stronger" graph partial orders such as the Y-Δ and H-bowtie orders. We briefly mention our plans to incorporate these new features within our distributed computational framework (aka, VACS) for computing obstruction within the universe of bounded pathwidth and treewidth.

1 Introduction

The goal of this paper is to present the characterization of graphs with vertex cover at most 7 by forbidden minors. The general problem of vertex cover asks whether a graph has a set of vertices of size at most k that covers all edges. The *vertex cover* of a graph G (which is NP-complete to decide; see [19]) is the minimum size of a set of vertices $S \subseteq V(G)$ such that every edge of G has at least one endpoint in S. For $k \in \mathbb{N}$, define k-Vertex Cover to be the family of graphs of vertex cover at most k.

Earlier Cattell and Dinneen in [5] classified the families of graphs with vertex cover at most $1 \leq k \leq 5$ by using the computational machinery now described in Cattell et al. [7]; later Dinneen and Xiong in [14] covered the case $k = 6$ by using a more direct brute-force search method based on new bounds on the number of vertices, edges and degree of the forbidden graphs. In this paper we show how we found the obstructions for $k = 7$ by using a combination of new optimization theory and techniques added to the minimal finite-state congruence approach of [5].

The characterization of graph families by forbidden graphs is often illustrated with Kuratowski's famous theorem—where the planar graphs are defined in terms of the two minimal nonplanar graphs in the topological order [1], K_5 and $K_{3,3}$ (the Kuratowski graphs)

[1] The topological and minor orders are defined in Definitions 8 and 9.

[21]. That is, a graph can be drawn in the plane without edge crossings if and only if it does not topologically contain K_5 or $K_{3,3}$. Wagner showed that the Kuratowski graphs are also the minor-minimal nonplanar graphs [28], so they are also called the forbidden minors for the planar graphs.

Another term which is neutral to the graph order considered, and which we will prefer, is to call this set of graphs the *obstruction set* under the minor order to the family of planar graphs, which we now define precisely. We say a set of graphs $\mathcal{F}$ forms a *lower ideal* under, or is *closed* under, an ordering $\preceq$ on graphs if for every $G \in \mathcal{F}$ and graph H, if $H \preceq G$ then $H \in \mathcal{F}$. The obstruction set to a lower ideal $\mathcal{F}$ under a graph ordering $\preceq$, which we write as $\mathcal{O}(\mathcal{F})$ (with $\preceq$ assumed to be clear from the context), is defined as follows. Firstly a graph G is not in $\mathcal{F}$ if and only if there is some obstruction $H \in \mathcal{O}(\mathcal{F})$ such that $H \preceq G$, and secondly, $\mathcal{O}(\mathcal{F})$ contains as few graphs as possible; that is there is no pair of graphs $H_1, H_2 \in \mathcal{O}(\mathcal{F})$ such that $H_1 \preceq H_2$ for H_2 could be removed. Such a set is clearly uniquely defined.

A major result in the last few decades has been to show that these kinds of characterisations in terms of finite sets are in some sense common. The Graph Minor Theorem of Robertson and Seymour states that the minor order is a well-partial-ordering on graphs, or equivalently that every minor order-closed family has a finite set obstruction set [24, 25].

Perhaps the most important application to Computer Science is due to Robertson and Seymour's Graph Minor Algorithm: an $O(n^3)$ time algorithm to check whether a fixed graph is a minor of an input graph [24]. In fact, this algorithm has very recently been improved to $O(n^2)$ running time by Kawarabayashi et al. [20]. Hence, given any minor order lower ideal $\mathcal{F}$, there exists an $O(n^2)$ time decision algorithm for $\mathcal{F}$: test for each of the forbidden minors. The problem (aside from the huge constants hidden by the O notation which already make the decision algorithm impractical [18]) is that the proof of the existence of a finite obstruction set is nonconstructive (the minor embedding algorithm itself is constructive). This has motivated recent researchers to 'constructivise' the Graph Minor Theorem by showing when and how the obstruction sets can be effectively computed. The fundamental problem is to provide a *termination condition* allowing a search procedure to halt.

Fellows and Langston [15] showed that given only a decision oracle for a lower ideal $\mathcal{F}$ it is not possible to compute the obstruction set for $\mathcal{F}$. This negative result was soon followed by Fellows and Langston [16] introducing the technique of using congruences (equivalence relations) on "graph languages", in analogue to the Myhill–Nerode theorem, to prove the following theoretical result (the canonical congruence for $\mathcal{F}$ and refinements will be defined later). This is the theorem on which our computation for minor-order obstructions is fundamentally based:

Theorem 1 (Fellows and Langston [16])**.** *Given:*

1. *a decision algorithm (i.e. a Turing machine) for $\mathcal{F}$ membership,*
2. *a bound on the maximum treewidth or pathwidth of the obstructions to $\mathcal{F}$,*
3. *a decision algorithm for a finite state (index) congruence which refines the canonical congruence for $\mathcal{F}$ on boundaried graphs,*

the obstruction set for $\mathcal{F}$ can be computed.

These results have since been improved and elaborated in many other papers; readers are directed towards Cattell et al. [7] which provides a high level survey on obstruction set computability, and discusses the use of *second order congruences* to provide computation termination conditions.

Courcelle's theorem [10]—which states that for any graph property $\mathcal{P}$ definable in monadic second order logic (MSO), there is a linear time recognition algorithm for $\mathcal{P}$ on graphs of bounded treewidth (in fact, a finite state tree automaton recognising tree decompositions of graphs in $\mathcal{P}$)—is of great importance in this area. It provides a construction for the decision algorithm for item *3* above for any graph ideal definable by a MSO sentence. Recently, Adler et al. [2] explored this alternative approach utilising definability in MSO and the heavy machinery of the Robertson–Seymour graph minor theory, and use it to solve several previously open problems, one of which—computability of the obstruction set of a union of minor ideals—was also further studied by one of the authors in [27].

The algorithm given by the proof of Theorem 1 has proven to be practically implementable; the first author has done this in [12] and computed forbidden minors for several parameterized graph families (see also [5, 6]). This still appears to be the last work on automated computation of complete obstruction sets. The subject of this paper is the continuation of work on this approach, using the *VACS* software package. The implementation of the search procedure employed depends on a number of additional results not of direct interest to the theoreticians.

The primary aim of this research is to compute new obstruction sets for interesting families, such as the graphs of small vertex cover. To aid this, we would like the search algorithm to be as efficient as possible (mainly by decreasing the size of the search space), so that larger and more difficult problems can be attacked. We would also like to consider graph orders other than just the minor order, both so that obstruction sets for different kinds of lower ideals can be computed, and as a means of reducing search spaces by using stronger (extended) orderings.

Today, obstruction sets are still usually found either entirely by hand via long and tedious case analyses, by exhaustive (brute force) computer search generally without a proof that the found set of obstructions is complete, or in some cases by a combination of the two, such as by Archdeacon et al. [3]. The obstruction sets to families embeddable in surfaces, and various variants and restrictions such as outer-planar graphs (which we informally call *surface embedding families*), have historically attracted a great deal of interest. An ultimate goal of this line of research would be to vindicate the approach by computing the complete obstruction sets for surfaces which have proven too large and hard to compute for all human and computer methods attempted. One example is the set of obstructions for the torus, of which over 16000 have been found by brute force search [9]. (By contrast we have found only 2288 torus obstructions so far).

Of course specific obstruction sets are not the only interesting problems. There are still many questions left to answer on theoretical questions of computability.

As an additional motivation, even if a family of graphs $\mathcal{F}$ is not closed under a particular graph ordering it is often possible to define a variant of $\mathcal{F}$ that is and still captures important properties. Consider for example graph coloring. A *t-coloring* of a graph G is a mapping ϕ from the vertices of G to $\{1, \ldots, t\}$ such that if u and v are adjacent vertices in G then they have different *colors* (values under the map ϕ). The

minimum number of colors required to color a graph G is denoted $\chi(G)$. The graphs colorable with at most t colors are not closed under any of the orders defined above except for the subgraph ordering. For example C_4 is two-colorable while its minor C_3 requires three colors. One of the most famous open problems in graph theory concerns coloring (open for $t \geq 7$):

Conjecture 2 (Hadwiger's Conjecture)**.** If $\chi(G) \geq t$ then $K_t \preceq_m G$.

Let $\mathfrak{C}_t$ be the family of graphs G such that $\chi(H) < t$ for every minor H of G (including G). By definition this family is a minor order lower ideal. Suppose G is a minimal (in the minor order) counter example to Hadwiger's conjecture for t. Then G is an obstruction for $\mathfrak{C}_t$ which is not K_t. (The obstructions are called *t-minor-critical* graphs in the literature.) Thus we may have a way of proving or disproving the conjecture for specific t by computer search; though perhaps only in theory. Replacing $\preceq_m$ with $\preceq_t$ we get Hajós' conjecture (open for $t \in \{5, 6\}$ [1]), and replacing it with $\preceq_i$ we get a conjecture of Abu-Khzam and Langston [1] (open for $t \geq 5$). These might be easier targets for attack due to the smaller parameters.

1.1 Paper outline

We first give an extensive background on the graph definitions needed for our computational procedure for computing obstruction sets. Here we give a foundation of the theory of practical computation and on results showing the correctness of the algorithm employed by the current VACS software package. We examine properties of transformations and orderings on boundaried graphs, and show how obstruction sets for lower ideals in several graph orders can be computed. Since most of VACS was left unmodified in the course of this research, we do not entirely repeat what can be found in more detail in [12].

We then provide a brief description of VACS and of the improvements made to it. We demonstrate these by computing for the first time the obstruction set for the 7-Vertex Cover family of graphs (the graphs which can be made edgeless by deleting at most 7 vertices). Previously, computation of vertex cover obstructions was fully implemented in VACS, but it was only feasible to compute up to 5-Vertex Cover because the size of the search space grows dramatically as a function of k. We conclude with a description of further improvements to VACS that we hope to implement in the future.

The appendix contains a listing of all the connected obstructions for 7-Vertex Cover using spring-layout drawings (other electronic formats are available from the first author's website).

2 Background and Definitions

In this section we introduce definitions and notation that will be used through the rest of the paper. It is assumed that the reader is familiar with the basics of graph theory, set theory and relations as well as having a simple understanding of computability. Knowledge of surfaces and graph embeddings is not assumed. We take $\mathbb{N}$ to be the non-negative integers.

In this paper, where not otherwise specified, a graph G is undirected, with vertices $V(G)$ and edges $E(G) \subseteq \{e \subseteq V(G) : |e| = 2\}$, never having multiple edges or loops. The *order* of a graph is the number of vertices.

Denote the degree of vertex v in the graph G as $\deg_G(v)$ (written as $\deg(v)$ where G is clear). For G a graph and $A \subseteq V(G)$, $G[A]$ is the subgraph induced by A. When H_1 and H_2 are both subgraphs of G, $H_1 \cap H_2$ is the subgraph of G with vertices $V(H_1) \cap V(H_2)$ and edges $E(H_1) \cap E(H_2)$, and $H_1 \cup H_2$ is defined similarly.

The Graph Minor series of papers began by defining *tree decompositions* and studying their properties (however the concept of treewidth had existed earlier). Tree decompositions have proven very important for both the theoretical and algorithmic implications of their structure:

Definition 3. A *tree decomposition* of a graph G is a tuple $(\mathcal{T}, \mathcal{B})$, where $\mathcal{T}$ is a tree and $\mathcal{B} = \{X_i\}_{i \in V(\mathcal{T})}$ is a collection of subsets of $V(G)$ (termed *bags*) indexed by the vertices of $\mathcal{T}$, satisfying the following properties:

1. $\bigcup_{i \in V(\mathcal{T})} X_i = V(G)$.
2. For every edge $e = \{u, v\}$ of G there is some $i \in V(\mathcal{T})$ such that $e \subseteq X_i$.
3. For every $v \in V(G)$, the subgraph of $\mathcal{T}$ induced by the *preimage* of v, $B^{-1}(v) := \{i \in V(\mathcal{T}) : v \in X_i\}$, is connected.

A tree decomposition $(\mathcal{T}, \mathcal{B})$ is a *path decomposition of* G if $\mathcal{T}$ is a path.

The *width* of a tree decomposition is the maximum $|X_i| - 1$ taken over all the vertices of $\mathcal{T}$. The *treewidth of a graph* G, denoted $tw(G)$, is the minimum width of any tree decomposition of G. The *pathwidth of* G, denoted $pw(G)$, is defined likewise but restricted to path decompositions.

In the literature the tree $\mathcal{T}$ is frequently considered to be rooted. We sometimes distinguish a root of $\mathcal{T}$, in a context given below, but we will never actually consider $\mathcal{T}$ to be directed.

Definition 4. A *t-boundaried graph* $G = (G', S, f)$ is an *underlying* ordinary graph G' along with a distinguished set $S \subseteq V(G)$ of *boundary* vertices of cardinality t, and a bijection $f : \{0, 1, \ldots, t-1\} \to S$ labelling the boundary. We sometimes consider *t-free-boundaried graphs*, which are tuples (G', S) without a boundary labelling.

Boundaried graphs can be considered as fragments of graphs. We sometimes implicitly forget the boundary of a boundaried graph and treat it as a nonboundaried graph. We call the t-boundaried graph with t vertices (all in the boundary) and no edges the *empty t-boundaried graph*. Note that lower ideals in this paper are *always* of nonboundaried graphs; when we say that a boundaried graph is in a lower ideal, we mean its underlying nonboundaried graph.

Given two boundaried graphs with boundaries of the same size, we can 'glue' them together along their boundaries:

Definition 5. Given two t-boundaried graphs $G_1 = (G_1', S_1, f_1)$ and $G_2 = (G_2', S_2, f_2)$ (where $V(G_1'), V(G_2')$ are disjoint), the *boundary join* of G_1 and G_2, denoted $G_1 \oplus G_2$, is the t-boundaried graph $H = (H', S_1, f_1)$ where H' is the disjoint union of G_1', G_2' with $f_1(x)$ and $f_2(x)$ identified for $x \in \{0, 1, \ldots, t-1\}$ (that is, each is replaced with a single new vertex which we continue to call $f_1(x)$ with all edges incident to $f_2(x)$ moved to $f_1(x)$).

For our purposes, we need to extend the concepts of pathwidth and treewidth slightly. Let G be a (possibly boundaried) graph and $S \subseteq V(G)$. Define $\mathrm{CL}(G, S)$ as G with S made into a clique by adding edges.

Definition 6. Let G be a boundaried graph with boundary S. The boundaried-treewidth of G, written $\delta\text{-}tw(G)$, is the minimal width of a tree decomposition of G with S a subset of one of the bags (a *boundaried tree decomposition*). Equivalently, $\delta\text{-}tw(G) = tw(\mathrm{CL}(G, S))$. The boundaried-pathwidth of G, $\delta\text{-}pw(G)$, is the minimal width of a path decomposition $(\mathcal{T}, \mathcal{B})$ of G with S a subset of one of the bags of the two endpoints of the path $\mathcal{T}$ (a *boundaried path decomposition*). We sometimes call the specified vertex $i \in V(\mathcal{T})$ with $S \subseteq X_i$ the *root* of the tree or path $\mathcal{T}$.

The following terminology is from Cattell et al. [7] and is useful for avoiding confusion:

Definition 7. Let t be a positive integer. The *large universe* $\mathcal{U}^t_{large}$ is the set of all t-boundaried graphs. The *small treewidth universe* $\mathcal{U}^t_{tree}$ is the set of all t-boundaried graphs G with $\delta\text{-}tw \leq t-1$. Likewise the *small pathwidth universe* $\mathcal{U}^t_{path}$ is the set of all t-boundaried graphs G with $\delta\text{-}pw \leq t-1$.

Note that the set of underlying nonboundaried graphs of $\mathcal{U}^t_{\mathrm{tree}}$ (respectively $\mathcal{U}^t_{\mathrm{path}}$) is just the set of graphs with treewidth (respectively pathwidth) at most $t-1$ and order at least t.

2.1 Graph Orders

Recall that a relation $\leq$ is a partial order if it is reflexive, anti-symmetric and transitive, but not necessarily total. We say that a partial order $\preceq_a$ *extends* $\preceq_b$ if they have the same domain D, and for every $x, y \in D$, $x \preceq_b y$ implies $x \preceq_a y$. We write $x \prec_a y$ to mean $x \preceq_a y$ and $x \neq y$, and say that x is *properly below* y in $\preceq_a$ (or a proper minor, etc.). A partial order on a domain D is a *well-partial-order* if it is well-founded (any set of elements of D has a least element) and has no infinite anti-chains (a sequence of elements which are pairwise incomparable). We only care that well-partial-ordering implies finite obstruction sets for all lower ideals.

We now define six partial orders on graphs that we are interested in (though the theory will be developed to allow a broad range of graph orders). These are all 'topological' in flavor, that is they are defined by different ways in which one graph can contain another. They are all well-founded: they all have K_1 as the minimum graph. We define them via local transformations on graphs:

Definition 8. If γ is an edge or set of edges of G, *deleting* γ forms the subgraph $G \setminus \gamma$ of G with γ or $\{\gamma\}$ as appropriate removed from the edge set.

If γ is a vertex or set of vertices of G, *deleting* γ forms the subgraph $G \setminus \gamma$ of G induced by $V(G) \setminus \gamma$ or $V(G) \setminus \{\gamma\}$ as appropriate.

If $e = \{u, v\}$ is an edge of G, *contracting* e forms the graph G/e with e deleted and u and v identified.

A pair of adjacent edges $e_1 = \{u, v\}, e_2 = \{v, w\}$ is *lifted* by deleting e_1, e_2 and adding a new edge $\{u, w\}$.

A graph H is a $Y\Delta$-*transformation* of a graph G if H is formed from G by deleting a degree three vertex v of G and adding edges between each pair of the three neighbors of v. Write $G \cup \Delta v$ for the operation of adding edges between neighbors of a degree three vertex v, and $(G \cup \Delta v) \setminus v$ for the $Y\Delta$-transformation of G removing vertex v.

Let G be a graph with a pair of adjacent degree three vertices u, v, and let v, w, x be the neighbors of u and v, y, z be the neighbors of u. A graph K is an $H \bowtie$-*transformation* of G if K is formed from G by deleting vertices u, v and adding a new vertex t and edges $\{w, x\}, \{y, z\}, \{t, w\}, \{t, x\}, \{t, y\}, \{t, z\}$, in the shape of a bowtie.

For clarity (and due to differing definitions of the terms in the literature), performing exactly one of the above transformations once on a graph shall be called a *one-step* transformation. Now for the orders:

Definition 9. The *minor order* on graphs $\preceq_m$ is defined by $G \preceq_m H$ if G can be produced from H by a series of *one-step minor operators* edge deletion, vertex deletion, and edge contraction. In that case G is called a *minor* of H.

If we only allow contraction of edges incident to a vertex of degree one or two we produce instead the *topological order* (or *topological minor order*) on graphs, $\preceq_t$. If $G \preceq_t H$ one says that G is a *topological minor* of H, or H *topologically contains* G. Contracting an edge incident to a degree two vertex v is also called *suppressing* v.

Restricting further and allowing only edge deletions and vertex deletions of course produces the *subgraph order*, $\preceq_s$.

The $Y\Delta$ *ordering* $\preceq_Y$ extends the minor ordering and is defined by $G \preceq_Y H$ if G can be produced from H by a series of edge deletions, vertex deletions, edge contractions, and $Y\Delta$-transformations.

The $H \bowtie$ *ordering* (H-*bowtie*) $\preceq_H$ extends the $Y\Delta$ ordering by also allowing $H \bowtie$-transformations.

The topological order restricted to the class of cubic graphs (those where all vertices have degree 3) is also called the *cubic order*. In that case it can equivalently be defined by edge and vertex deletions followed by suppressing any resulting degree two vertices.

There are several variations on the definition of the *immersion order*, with disagreement in the literature as to which one is the standard. We follow here Abu-Khzam and

Langston [1] and Robertson and Seymour [26]. The *immersion order* on graphs $\preceq_{im}$ is defined by $G \preceq_{im} H$ if G can be produced from H by a series of edge deletions, vertex deletions, and edge liftings; G is said to be *immersed* in H. Note that the immersion order is also an extension of the topological order: suppressing a vertex is a special case of lifting a pair of edges and deleting the resulting isolated vertex.

The $Y\Delta$- and $H \bowtie$-orders are studied perhaps exclusively in the context of graph embeddings.

For convenience we generalize the term 'minor' to other partial graph orders. If $H \preceq G$ call H a *$\preceq$-reduction* (or *reduction* when $\preceq$ is clear) of G, and if H is an appropriate one-step transformation of G, when we call it a *one-step reduction* of G.

Some authors work in the domain of psuedographs, and allow multiple edges and self loops to be formed upon taking transformations of graphs. Note that the difference is not irrelevant: in that domain, the unique forbidden minor (and topological minor) to the family of forests (i.e. acyclic graphs) is the one vertex graph with a self loop, rather than the cycle C_3.

For each graph relation $\preceq_X$ we define a variant on boundaried graphs, written as $\preceq_{\delta X}$ and called the *boundaried* order, as follows: modify each of the above definitions so that G, H are two boundaried graphs, and $G \preceq_{\delta X} H$ if $G' \preceq_X H'$ where G', H' are G, H without boundaries, by a series of one-step transformations not deleting boundary vertices of H' (including indirectly by $Y\Delta$- and $H \bowtie$-transformations) or contracting edges between two boundary vertices—that is, the number of boundary vertices is unchanged between G and H. So considering the underlying graphs, the non-boundaried order is an extension of the boundaried order. We dealing with boundaried graphs we may say that one is a minor of another, etc., referring to the boundaried variant of the order, or say "boundaried minor" to be clear.

If $\preceq_a$ is an extension of $\preceq_b$, and $\mathcal{F}$ is a family closed under both $\preceq_a$ and $\preceq_b$, then it is easy to see that the obstruction set for $\mathcal{F}$ under $\preceq_a$ is subset of the obstruction set under $\preceq_b$. So the Kuratowski graphs are easily seen to be the minimal non-planar graphs under the topological, minor, $Y\Delta$ and $H \bowtie$ orders, since the planar graphs are closed under all of them and neither of K_5 or $K_{3,3}$ is a reduction of the other in any of these orders. But the planar graphs are not closed under the immersion order: every graph has an immersion in some planar graph, by drawing it in the plane and adding vertices at edge crossings.

As stated, the minor order is a well-partial-order by the Graph Minor Theorem.

Lemma 10 (Robertson and Seymour [26])**.** *Let $\preceq$ be a well-partial-order and let $\mathcal{F}$ be a lower ideal in $\preceq$. Then there are a finite number of δ-obstructions for $\mathcal{F}$ in $\preceq_\delta$.*

However it is not hard to prove this directly and constructively for specific orders. The boundaried minor order, as well as the immersion and boundaried immersion orders, have also been proven to be well-partial-ordered by Robertson and Seymour [26]. (In fact, they show that the boundaried variant of any well-partial-order on graphs is a well-partial-order). In addition, by the argument in the preceding paragraph, an extension of a well-partial-order is a well-partial-order, so the $Y\Delta$ and $H \bowtie$ orders are too.

However, the subgraph and the topological orders are not well-partial-orders (consider the set of all cycles $\{C_k\}_{k\geq 3}$). However, we do have the following known result.

Lemma 11. *If $\mathcal{F}$ is a lower ideal in the minor order, then $\mathcal{F}$ is a lower ideal in the topological order with a finite obstruction set.*

2.2 t-parses

In this section we describe a way (from [12, 5]) of representing graphs in $\mathcal{U}^t_{\text{path}}$ and $\mathcal{U}^t_{\text{tree}}$, and more generally graphs of bounded pathwidth or treewidth, using strings of unary operators from an operator set along with the boundary join operator. This will be used to provide a generation scheme for these families of graphs.

There are many other operator sets that can be used to generate $\mathcal{U}^t_{\text{path}}$ or $\mathcal{U}^t_{\text{tree}}$. See [12] for a discussion. This one is used due to its simplicity and more importantly its prefix properties, which will be explained later.

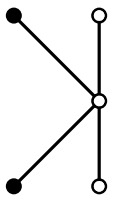
The operator set Σ_t is composed of the following operators: *vertex operators* $v_0, \ldots, v_t$ and *edge operators* $e_{i,j}$ for $0 \leq i < j \leq t$. Let $G = (G', S, f)$ be a $(t+1)$-boundaried graph. The effects of the operators (applied left-to-right) are defined as follows:

- $G\ v_i$: Add a new vertex u to G', remove $f(i)$ from S and add u, and set $f(i) := u$.
- $G\ e_{i,j}$: Add an edge to G' between $f(i)$ and $f(j)$.

Definition 12. A *pathwidth t-parse* is a string $w \in \Sigma_t^*$ of length at least $t+1$, where the first $t+1$ operators are $v_0, v_1, \ldots, v_t$, called the *preamble*. The boundaried graph corresponding to (or, *generated by*) a pathwidth t-parse $v_0, v_1, \ldots, v_t, w$, where $w \in \Sigma_t^*$, is the graph obtained by applying the operators in w one at a time starting with the empty $(t+1)$-boundaried graph.

Definition 13. A *treewidth t-parse* is an expression composed of pathwidth t-parses and $\oplus$ operators. We use brackets to display the grouping of the operators as necessary.

For example, the 2-parse $(v_0, v_1, v_2, e_{0,1}, v_0, e_{0,1}) \oplus (v_0, v_1, v_2, e_{1,2}, v_2, e_{0,1})$ is the boundaried graph displayed in Figure 1.



$(v_0, v_1, v_2, e_{0,1}, v_0, e_{0,1}) \oplus (v_0, v_1, v_2, e_{1,2}, v_2, e_{0,1})$

Figure 1: An example of a treewidth 2-parse

We may write just *t-parse* where making a generalized argument about either pathwidth or treewidth t-parses (of course pathwidth t-parses are a special case of treewidth t-parses, but this is a subtly different kind of generalization). Our definitions of pathwidth and treewidth t-parses differ slightly from those in [12], as we require for convenience that every pathwidth t-parse have a preamble, rather than allowing the initial vertex operators anywhere. The difference is insignificant, as the initial vertex operators can simply

be moved to the front without changing the graph generated, and while enumerating graphs, all t-parses have this form anyway.

We will frequently conflate pathwidth or treewidth t-parses and graphs in $\mathcal{U}^{t+1}_{\text{path}}$ or $\mathcal{U}^{t+1}_{\text{tree}}$. Of course there is an important distinction between the two: there are usually multiple t-parses for every graph in $\mathcal{U}^{t+1}_{\text{path}}$ or $\mathcal{U}^{t+1}_{\text{tree}}$ so we only fail to distinguish between the two when the difference does not matter.

Of course, the reason for using t-parses is that they allow simple representation and generation of (boundaried) bounded treewidth and pathwidth graphs. The following theorems should be easy to see by building a t-parse while visiting the vertices of a tree decomposition by depth-first-search in one direction, and by building a decomposition with one bag per vertex operator in the other:

Theorem 14 (Dinneen [12], Theorem 21)**.** *The family of graphs representable as pathwidth t-parses equals the family of graphs of pathwidth at most t and order at least $t+1$.*

Theorem 15 (Dinneen [12], Theorem 23)**.** *The family of graphs representable by treewidth t-parses equals the family of graphs of treewidth at most t and order at least $t+1$.*

3 Computing Obstruction Sets

In this section we present (much of) and extend in places the theory of Fellows and Langston [16] and Dinneen [12], to a class of graph orders including the ones which were introduced in Chapter 1. It has been noted in the literature that the Fellows-Langston method can be used to compute obstruction sets under orders other than the minor order (e.g. the immersion order), which is usually the only one seriously considered, but as far as we know the details have never been given. We find that it is nontrivial to apply the search procedure of [12] due to complications (though at the more abstract level of [16] there is not really any difference). We also slightly strengthen some lemmas of [12] by refining one-step reductions to *primitive* reductions, and discuss other theoretical and practical optimisations.

Throughout the rest of this paper $\preceq$ refers to some unspecified partial order on graphs, and $\preceq_\delta$ refers to the boundaried version of $\preceq$.

3.1 Primitive Reductions

Let G be a possibly boundaried graph. Let us say a proper reduction H of G is a *primitive reduction of G* (or *primitive* where G is clear) if there is no K such that $H \prec K \prec G$. Clearly a primitive reduction is one-step. We mention primitive reductions because they are in a sense fundamental (so restricting to them can improve algorithmic efficiency), but there is no actual need to use primitive reductions anywhere; one-step reductions suffice.

Lemma 16 (Versteegen [27], Lemma 15)**.** *Let H be a below G in the $H \bowtie$ order (or any weaker one).*

- *If $H = G \setminus v$ for some vertex v, then it is primitive if and only if v in G is degree 0.*

- *If $H = G \setminus e$ for some edge e, then it is primitive.*
- *If $H = G \,/\, e$ for some edge $e = \{u, v\}$, then it is primitive if and only if u, v (either have degree at least two or are boundary vertices) and have no common neighbors.*
- *If $H = G \cup \Delta v \setminus v$ for some vertex v, then it is primitive if and only if the neighbors of v are pairwise non-adjacent.*
- *If H is an $H \bowtie$-transformation of G removing vertices u, v (having neighbors $\{v, w, x\}$ and $\{u, y, z\}$ respectively) then it is primitive if and only if the neighbors of u other than v, and also the neighbors of v other than u, are non-adjacent, and $|\{w, x, y, z\}| > 2$.*

Furthermore, all of the above are true when we consider the corresponding boundaried orders.

3.2 Apex-treewidth

The following definitions are useful for defining tighter bounds on classes of graphs, which can translate to significantly smaller class, as will be seen in Section 4:

Definition 17. Say that a graph has *apex-pathwidth* (x, w) if after deleting some set of x *apex* vertices the remaining graph has pathwidth exactly w. Define *apex-treewidth* similarly, and define the *boundaried apex-pathwidth* and *boundaried apex-treewidth* of boundaried graphs by replacing pathwidth and treewidth with their boundaried versions in the definition.

Lemma 18. *An apex-treewidth (x, w) graph has treewidth at most $x+w$ and a boundaried apex-treewidth (x, w) graph has boundaried treewidth at most $x + w$. The same is true for treewidth replaced with pathwidth.*

Proof. Consider the boundaried apex-treewidth case. Let $G = (G', S, f)$ be a boundaried graph, and $A \subseteq V(G)$ be the x apex vertices. Take a width w tree decomposition $(\mathcal{T}, \mathcal{B})$ of $G \setminus A$ which contains $S \setminus A$ in some bag $B_i \in \mathcal{B}$. Then add A to every bag to form a width $x + w$ tree decomposition for G which has a bag B'_i with $S \subseteq B'_i = B_i \cup A$.

The other cases are all proved in the same way, ignoring the boundary argument in the nonboundaried cases. □

The treewidth 0 and pathwidth 0 families of graphs are identical and are exactly the edgeless graphs. The treewidth 1 graphs are the forests, and the pathwidth 1 graphs are the unions of caterpillar graphs. $(x, 0)$-apex-treewidth and $(x, 0)$-apex-pathwidth graphs are again identical and are graphs of vertex cover at most x (see Section 4). $(x, 1)$-apex-treewidth graphs are those of feedback-vertex-set at most x, i.e. those graphs which can be made acyclic by removing at most x vertices.

We note that we could also consider the set of graphs within k edges of treewidth or pathwidth w. Say that a graph has *almost-treewidth* (a, w) if after deleting a edges the remaining graph has treewidth w. Since we can choose one endpoint of each edge, an (a, w)-almost-treewidth graph has $([x{=}]a, w)$-apex-treewidth. *Boundaried almost-pathwidth*, etc., are defined similarly and have similar relations to the corresponding apex-pathwidth/treewidth.

3.3 Transformations on t-parses

It is well known and easy to prove that:

Lemma 19. *If G is below H in the subgraph, topological or minor order, then $tw(G) \leq tw(H)$ and $pw(G) \leq pw(H)$.*

However, things are more complicated when we consider the other graph orders, none of which are closed in the same way. See for example Figure 2, a counter example for the immersion order ($pw(G) = tw(G) = 2$ but $pw(H) = tw(H) = 3$).

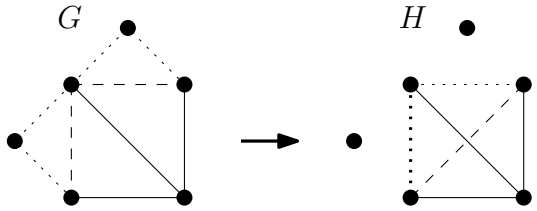


Figure 2: An example showing that an immersed graph may have a larger treewidth than the graph it is immersed in.

Graphs with boundaries add even more complication. Let $G \in \mathcal{U}^{t+1}_{\text{tree}}$. A one-step reduction H of G is in $\mathcal{U}^{t+1}_{\text{large}}$, since it has the same boundary, but we can not conclude that it is in $\mathcal{U}^{t+1}_{\text{tree}}$ (in fact not for many of our graph transformations) and therefore that it has a t-parse (recall t-parses generate $t+1$ boundaried graphs). This makes it more difficult to continue to use t-parses with fixed t to represent all the graphs we need to! But we will show how this can done anyway, so that we can keep the algorithms to compute congruences as simple as possible (as will be explained in the next section).

The proof of the following result, answering this question for the minor order, is by description of an algorithm to compute the t-parse of the reduction. It can be proved perhaps more easily using tree decompositions directly, but in practice we do need such an algorithm acting on t-parses.

Lemma 20 (Dinneen [12], Lemma 60, Observation 61)**.** *Let G be a pathwidth (treewidth) t-parse, and suppose H is a boundaried minor of G. Then H has a representation as a pathwidth (respectively treewidth) t-parse.*

Instead of generalizing Lemma 20 directly, we opt to first rephrase it in the following way:

Lemma 21. *If G, H are t-boundaried graphs with G below H in the boundaried minor, topological or subgraph order, then δ-$tw(G) \leq \delta$-$tw(H)$ and δ-$pw(G) \leq \delta$-$pw(H)$.*

We can extend to the other graph orders, albeit limited to a single one-step transformation:

Lemma 22 (Versteegen [27], Lemma 21)**.** *If G, H are t-boundaried graphs with G one step below H in the boundaried immersion, $Y\Delta$, or $H \bowtie$ order, then δ-tw$(G) \leq \delta$-tw$(H)+1$ and δ-pw$(G) \leq \delta$-pw$(H)+1$.*

In fact the bounds in the above lemma are tight. We could of course prove similar, more complex bounds for boundaried apex-treewidth graphs, but this is not useful for our purposes.

Before continuing, we introduce the following needed definitions, which are used for our workaround:

Definition 23. Let $c \in \mathbb{N}$. A c-expanded t-parse is a $(c+t)$-parse where the boundary vertices numbered $t+1, \ldots, t+c$ are isolated.

Definition 24. Let $n \in \mathbb{N}$. If $G = (G', S, f)$ is a t-boundaried graph, define $\mathrm{SP}(G, n)$ to be G with n isolated boundary vertices $\{v_1, \ldots, v_n\}$ added to $V(G)$ and S, and boundary labelling $f' : \{0, \ldots, t+n-1\} \to V(G) \cup \{v_1, \ldots, v_n\}$ given by

$$f'(x) := \begin{cases} v_{x-t+1} & \text{if } x \geq t \\ f(x) & \text{otherwise.} \end{cases}$$

If G is a treewidth t-parse, then define $\mathrm{SP}(G, n)$ to be the n-expanded t-parse which is G with n new vertex operators $v_{t+1}, v_{t+2}, \ldots, v_{t+n}$ appended to G's preambles (that is, the preamble of every pathwidth t-parse in G).

We are now ready to prove an analogue of Lemma 20, still restricted to one-step transformations, though the algorithm is not given completely explicitly:

Lemma 25. *Let G be a pathwidth (treewidth) t-parse generating the $(t+1)$-boundaried graph G'. Suppose K is a $(t+2)$-boundaried graph one-step below $\mathrm{SP}(G', 1)$ in the boundaried immersion, $Y\Delta$, or $H \bowtie$ order. Then K has a representation as a 1-expanded pathwidth (respectively treewidth) t-parse.*

Proof. Let v be the new boundary of $\mathrm{SP}(G', 1)$ not in G', and S the boundary of G'.

The one-step transformation used to produce K does not touch v, so v is an isolated boundary vertex in K and $K = \mathrm{SP}(J, 1)$ for some $(t+1)$-boundaried graph J, which is a one-step reduction of G'. Since G' has boundaried pathwidth (treewidth) at most t, by Lemma 22 J has a boundaried path (tree) decomposition $(\mathcal{T}, \mathcal{B})$ of width at most $t+1$. This tree decomposition corresponds to a $(t+1)$-parse J' of J (by the proof of Lemma 14 or 15). The boundary of J' is a superset of S with one additional vertex, labelled $t+1$, since we assume the labels of the boundary vertices of G are preserved. By appending the vertex operator v_{t+1} to J' we get a 1-expanded t-parse which generates K. □

We can go directly from G to J' by relabelling vertex and edge operators to make use of the spare $(t+2)$th boundary vertex. The details of the algorithm, which is linear time in t and the length of G, can be derived by examining the proof of Lemma 22.

3.4 Congruence and Minimality

Now we finally define the canonical congruence, and explain the proof of Theorem 1, as well as the most important lemmas from [12]. A congruence is an equivalence relation (in this context, on $\mathcal{U}^t_{\text{tree}}$ or $\mathcal{U}^t_{\text{path}}$), though we are interested only in refinements of the canonical congruence for a graph family:

Definition 26. Let $\mathcal{F}$ be a lower ideal in $\preceq$, and let the universe $\mathcal{U}^t$ be either $\mathcal{U}^t_{\text{tree}}$ or $\mathcal{U}^t_{\text{path}}$. Let $G, H \in \mathcal{U}^t$. G and H are *$\mathcal{F}$-congruent in $\mathcal{U}^t$*, written $G \sim_{\mathcal{F}} H$, if for every $X \in \mathcal{U}^t$, $G \oplus X \in \mathcal{F} \iff H \oplus X \in \mathcal{F}$ (recall that we ignore boundaries). $\sim_{\mathcal{F}}$ is the *canonical congruence for $\mathcal{F}$ in $\mathcal{U}^t$*.

Definition 27. Let $\sim$, $\sim'$ be congruences on $\mathcal{U}^t$. $\sim'$ *refines* $\sim$ if $A \sim' B$ $(A, B \in \mathcal{U}^t)$ implies $A \sim B$.

The *index* of a congruence is the cardinality of the equivalence classes.

For the rest of this section we generally drop reference to $\mathcal{U}^t_{\text{tree}}$ and $\mathcal{U}^t_{\text{path}}$, and to a flavor of t-parse; nearly all of the following definitions (eg. whether a boundaried graph is minimal) depend on the universe we are working in, but we always use $\mathcal{U}^t_{\text{path}}$ with pathwidth t-parses and $\mathcal{U}^t_{\text{tree}}$ with treewidth t-parses, so we will cease mentioning this is background information.

Unfortunately the following standard terminology is slightly confusing; when we say "minimal" from here on we mean in the following sense, not in terms of the graph order $\preceq_\delta$.

Definition 28. A t-parse G is *nonminimal* for $\mathcal{F}$ if there is a proper boundaried reduction H of G such that $G \sim_{\mathcal{F}} H$. G is *minimal* for $\mathcal{F}$ if it is not nonminimal. G is a *δ-obstruction* if it is minimal and not in $\mathcal{F}$. Let $\mathcal{O}^t(\mathcal{F})$ be the set of δ-obstructions of $\mathcal{F}$.

δ-obstructions are generally easier to recognise than other minimal graphs because we need only consider membership of boundaried reductions, not congruences.

It is obvious that if G is an obstruction for $\mathcal{F}$ under $\preceq$ with pathwidth or treewidth t, then G with an added boundary (according to any minimal path or tree decomposition) is an element of $\mathcal{O}^t(\mathcal{F})$ (in fact there is a copy for every permutation of the boundary labels). So if we have a bound B on the maximum pathwidth or treewidth of an obstruction for $\mathcal{F}$, the obstruction set for $\mathcal{F}$ is a subset of the underlying nonboundaried graphs in $\bigcup_{1 \le i \le B} \mathcal{O}^i(\mathcal{F})$. It might seem that we can consider just $\mathcal{O}^B(\mathcal{F})$, but this only includes graphs with at least $B+1$ vertices. Recall, as discussed in Section 2.1, that this set is finite.

Now we finally consider other graph orders. Informally we could say that one of the qualities of graph orders meeting the following definition is that reductions under the graph order are 'local'. It is not strictly needed for computing obstructions, but in practice it is probably required for feasible computation. All of our six graph orders have this property.

Definition 29. Let $\preceq_\delta$ be a partial ordering on boundaried graphs. Say that $\preceq_\delta$ is *algebraic* if whenever G, H, K are t-boundaried graphs and $G \preceq_\delta H$, we have $G \oplus K \preceq_\delta H \oplus K$.

Lemma 30. *For $\preceq_\delta$ an algebraic ordering and $\mathcal{F}$ a lower ideal for $\preceq_\delta$, a t-boundaried graph G is $\mathcal{F}$-minimal if and only if it is not $\mathcal{F}$-congruent to any $\preceq_\delta$-primitive reduction.*

This lemma, proven in [12] (Lemma 71) for the boundaried minor order, and trivially generalizing to the topological and subgraph orders, also generalizes to other algebraic orders; the proof given below is essentially identical.

Proof. Let G be nonminimal. Then there is a $K \prec_\delta G$ such that $K \sim_{\mathcal{F}} G$. There is an H primitive below G such that $K \preceq_\delta H \prec_\delta G$.

Let $Z \in \mathcal{U}^t$. If $G \oplus Z \in \mathcal{F}$ then $H \oplus Z \in \mathcal{F}$ as $\preceq$ is algebraic and $\mathcal{F}$ is a lower ideal. If $G \oplus Z \notin \mathcal{F}$ then because $K \sim_{\mathcal{F}} G$ we have $K \oplus Z \notin \mathcal{F}$. But since $K \oplus Z \preceq_\delta H \oplus Z$ by algebraicity, also $H \oplus Z \notin \mathcal{F}$. Thus $H \sim_{\mathcal{F}} G$. The other direction is true by definition. □

The following lemma is the central one; working towards its generalization has been our main goal. But note for the first time we are considering only pathwidth t-parses, at least partially in order to reason about prefixes of t-parses. Note that the Prefix Lemma can be generalized to treewidth t-parses. We only consider prefixes which contain a full preamble of vertex operators; otherwise the prefix would not be a t-parse.

Lemma 31 (Prefix Lemma, Dinneen [12] Lemma 72)**.** *Let $\mathcal{F}$ be a lower ideal in the minor order. If $G = g_1, g_2, \ldots, g_n$ is a minimal pathwidth t-parse for $\mathcal{F}$, then any prefix t-parse of G is also minimal.*

We could properly generalize the Prefix Lemma by defining a parameter of a graph order which abstracts the results of Lemmas 21 (e.g., for $\preceq$ one of $\preceq_s, \preceq_t, \preceq_m$ define the constant $c_\preceq := 0$), and 22 (for $\preceq$ one of $\preceq_Y, \preceq_H, \preceq_{im}$ define $c_\preceq := 1$). However we would also need Lemma 25 to go through for a graph ordering in order to do so, so we only consider the three additional graph orders we are familiar with. First we make the following technical definition: say that a pathwidth $(t+1)$-parse G is a *widening of H* (or is a *widening*) if it is equal to $\mathrm{SP}(H, 1)$ for some pathwidth t-parse H.

Lemma 32 (Generalized Prefix Lemma)**.** *Let $\preceq$ be one of $\preceq_Y, \preceq_H, \preceq_{im}$, and $\mathcal{F}$ a lower ideal for $\preceq$. If G is a t-parse which is a widening and is minimal, then any prefix t-parse of G is also minimal.*

Proof. Note that any prefix t-parse of G is also a widening since the spare boundary vertex is completely unused anywhere in G. to to $\mathrm{SP}(K', 1)$ for some pathwidth t-parse K'. So it suffices to show that if G is a widening and is nonminimal, then any extension G' of G which is a widening is also nonminimal; and in particular we can consider just the case that G' is an extension of G by a single operator $g \in \Sigma_t$ (g must not be v_t or an operator for an edge incident to the $(t+1)$-th boundary vertex). By Lemma 30, there is a $(t+1)$-boundaried graph H primitive below G such that $G \sim_{\mathcal{F}} H$. By Lemma 25, H is representable as a pathwidth t-parse (in fact a 1-expanded one).

Note that $H\ g$ is a one-step boundaried reduction of $G' = G\ g$ (this requires some inspection). Let $Z \in \mathcal{U}^{t+1}_{\mathrm{path}}$. $G' \oplus Z = G \oplus (Z\ g) \in \mathcal{F}$ if and only if $H \oplus (Z\ g) = H' \oplus Z \in \mathcal{F}$, since $G \sim_{\mathcal{F}} H$. Therefore G' is nonminimal. □

3.5 Search Algorithm

The original prefix lemma enables a search procedure for $\mathcal{O}^t$ by building a search tree where the nodes are t-parses, beginning with the empty t-parse $v_0\, v_1 \ldots v_t$. For every minimal t-parse that we encounter, we consider its extensions by a single operator. which are minimal to the search tree. The only nodes in the search tree which are not in the family $\mathcal{F}$ are the δ-obstructions. We prove that a t-parse G is minimal by building t-parses of all the primitive reductions (actually, minors) of G and testing whether they are congruent to G.

Using the generalized prefix lemma, we do the same, except that we use $(t+1)$-parses to search for δ-obstructions in $\mathcal{O}^t$, and we make fewer extensions in order to produce only $(t+1)$-parses that are widenings. The primitive reductions are also representable as $(t+1)$-parses, and have an 'expanded' boundary vertex just as G does.

Definition 33. A width k tree decomposition $(\mathcal{T}, \mathcal{B})$, where $\mathcal{B} = \{X_i\}_{i \in V(\mathcal{T})}$, is *smooth* if for every $i \in V(\mathcal{T})$ $|X_i| = k$ and for every adjacent $i, j \in V(\mathcal{T})$ $|X_i \cap X_j| = k - 1$.

It is easy to see that every smooth tree decomposition can be transformed into a smooth one of the same width ([4] provides a simple procedure).

Let G be a graph, and $t \geq \mathrm{pw}(G)$. There typically is more than one non-isomorphic smooth tree decomposition of width t of G. There is also usually more than one t-parse corresponding to each tree decomposition. For example, swapping two adjacent edge operators produces the same boundaried path/tree decomposition.

Cattell *et al.* [5, 8] introduce a *canonic* total ordering $<_c$ on t-parses (defined in Dinneen [12]). The *canonic t-parse* for a t-free-boundaried graph is the minimal one in the canonical ordering which generates the graph. The definition of canonicity is such to ensure a vital property: any prefix t-parse of a canonic t-parse is also canonic [12, Lemma 36], which allows all non-canonic t-parses to be pruned. In this way it would be possible to find the obstructions using a depth-first search and minimal memory (see McKay [22]) which was however never actually implemented in VACS.

An algorithm to test the canonicity of a t-parse is presented in [12], however this is very slow. A dramatically faster approach is to compute canonical representations (i.e. canonical graph labellings) of the t-boundaried graphs generated by all minimal t-parses of each length, compare them (using a hash set) and choose the minimal t-parse under $<_c$. Of course only one minimality test need be performed per isomorphism class. The canonical labellings can be computed very quickly using a library such as NAUTY [23] since all graphs are very small. This approach trades space resources for time.

Modifying the above procedure to search amongst the graphs of (x, w)-apex-pathwidth is simple: t-parses are grown using only edge operators and vertex operators $v_0, \ldots, v_{x-1}$.

Theorem 34. *The above algorithm computes $\mathcal{O}^t(\mathcal{F})$ when $\mathcal{O}^t(\mathcal{F})$ is (x, w)-apex-pathwidth.*

□

3.6 Quick-canonicity

There are several *quick* t-parse canonicity tests which some (in fact most) non-canonic t-parses fail:

Lemma 35 (Dinneen [12], Lemma 34, 35)**.** *A t-parse $G = o_1, \ldots, o_n$ is not canonic if*

1. *G has adjacent edge operators $o_i = e_{a,b}, o_{i+1} = e_{c,d}$ (recall $a < b$ and $c < d$) such that $ab > cd$ in the lexicographical ordering (i.e. $a > c$ or $a = c$ and $b > d$), or*

2. *G has adjacent vertex operators $o_i = v_a, o_{i+1} = v_b$ such that $a > b$, or*

3. *G has an edge operator $o_i = e_{a,b}$ such that the next vertex operator is $o_j = v_c$ $(i < j)$ with $c \neq a$ and $c \neq b$.*

In fact pruning rules 1 and 2 above are still valid if we use the lexicographic ordering on t-parses instead of $<_c$ to define canonic t-parses.

Using these tests greatly reduces the number of isomorphism tests that must be performed to determine canonicity.

What may be unexpected is that using these tests also greatly reduces the number of non-minimal nodes found (and thus number of minimality tests performed). The reason for this is that the parent of the canonical t-parse G of a non-minimal node may be non-minimal even if some of the non-canonical isomorphs of G have parents which are minimal nodes. In this case we avoid a minimality test for the isomorphism class of G if every isomorph of G which is encountered is determined to be non-canonical.

For example, for 6-Vertex Cover (at pathwidth 7), using these quick canonic tests reduced the number of graphs tested for minimality, and running times, by a factor of about five.

4 Computing Vertex Cover Obstructions

In this section we describe simple specialisations of the search algorithm for computing vertex cover obstructions and present our results.

The set of graphs of vertex cover at most k, k-Vertex Cover, is easily seen to be a lower ideal in the minor order. In fact, by the following theorem, we need only to test a fewer set of primitive reductions.

Theorem 36 (Dinneen and Xiong [14] Theorem 2.1)**.** *The obstructions to k-Vertex Cover under the minor order are the same as the obstructions to k-Vertex Cover under the subgraph order.*

Relatedly, k-Vertex Cover is also well-quasi-ordered under the induced subgraph order by a special case of Ding's Bounded Paths Theorem ([11], see also [17]).

It is easy to see that the obstructions to k-Vertex Cover have pathwidth at most $k + 1$. In fact it follows from the definitions that:

Lemma 37. *The minor order obstructions to k-Vertex Cover have apex-pathwidth $(k + 1, 0)$.*

We restrict our attention to connected obstructions, as the disconnected obstructions are just unions of connected obstructions as the following lemma shows.

Lemma 38 (Cattell and Dinneen [5] Lemma 13)**.** *If the disjoint union of two graphs $O = C_1 \cup C_2$ is an obstruction for k-*Vertex Cover*, then C_1 and C_2 are obstructions for k'-*Vertex Cover *and k''-*Vertex Cover *for some k', k'' such that $0 < k', k'' < k$ and $k' + k'' = k - 1$.*

The following result provides an additional search space pruning rule:

Theorem 39 (Dinneen and Lai [13] Theorem 2.12)**.** *For any connected obstruction $G \in \mathcal{O}(k$–*Vertex Cover*), $|V(G)| \leq 2k - \max_{v \in G} \deg(v) + 3$.*

4.1 Implementation: VACS

The *VACS* software system (VLSI Automated Compilation System) was first developed by Cattell and Dinneen for computing the k-Vertex Cover obstructions, $k \leq 5$, in [5]. It enables distributed computation of obstruction sets (a number of families are currently supported), with a central process which grows the search tree and manages databases and the time consuming minimality and family membership tests delegated to worker processes possibly on other machines.

Table 1 shows how the search space grows very quickly as k increases, as well as demonstrating the effectiveness of restricting the search space using 39. "Search nodes, total" includes t-parses which are non-minimal or pruned by an isomorphism test or one of the rules in the previous subsection.

For comparison, computing 5-Vertex Cover obstructions with the previous version of VACS (using the previous search algorithm) took 50 hours of CPU time on the same hardware (and a week on a cluster of machines in 1994).

4.2 Results

Computation of k-Vertex Cover (except for the above search strategy) was already fully implemented in VACS, and it was originally used to compute the obstruction sets for vertex covers up to 5 by Cattell and Dinneen in [5]. Several years later, Dinneen and Xiong in [14] computed the obstructions to 6-Vertex Cover using a brute force approach (with a large number of pruning rules to reduce the search space). It still seemed infeasible to compute obstruction sets for $k \geq 6$ using VACS just by running longer computations even after 15 years of Moore's Law. But with a few computational optimizations we have now succeeded in computing the obstructions to 7-Vertex Cover. Our results are shown in Table 2. At the same time, we confirmed the 260 obstructions found by [14].

The 2250 connected graphs of the obstruction set for 7-Vertex Cover are included in the appendix of this paper. We provide small scalable vector drawings that should be zoomable with a proper PDF viewer. We also provide links on the VACS webpage `http://www.cs.auckland.ac.nz/~mjd/vacs/info.html` to download these obstructions in t-parse and adjacency matrix representations for easier computer manipulation.

Table 1: Rates of growth of search trees and search times for k-Vertex Cover, $1 \leq k \leq 7$.

k	Pathwidth	Search nodes, total	Search nodes, minimal	CPU time
$k+1$ pathwidth search tree				
2	3	272	47	0.04s
3	4	2 429	337	0.23s
4	5	39 657	4 553	4.6s
5	6	1256 804	122 044	261.4s
$(k+1,0)$-apex-pathwidth search tree				
2	3	111	34	0.04s
3	4	647	178	0.1s
4	5	5 442	1 659	0.9s
5	6	86 065	31 015	24.6s
6	7	4 266 172	1 265 128	35 minutes
7	8	357 466 818	113 678 571	209 hours
$(k+1,0)$-apex-pathwidth, applying Theorems 36 & 39				
6	7	2 630 396	1 156 152	23 minutes
7	8	223 142 515	107 634 497	91 hours

4.3 Future Research and Improvements

We now briefly describe some planned, in progress, and prospective changes to VACS.

One obvious change that could be made to the current VACS system would be to implement partial k-tree generation. Bounds on the treewidth of obstructions are more common than pathwidth bounds since every pathwidth bound is also a treewidth bound. It is also worth noting that tree decompositions and treewidth have received far more study than path decompositions and pathwidth. However it is not obvious what the best way to grow a search space of treewidth t-parses is; and adding boundary join operators would add a great deal of complexity throughout VACS, so it would not easily be attempted.

Recall that for each boundaried graph G we need to compare the congruence class $C(G'_m)$ of each primitive reduction G' of G with $C(G)$.

Assuming that our implementation of an algorithm to compute a congruence state really is a finite state automaton one notes that a great deal of duplicate work is performed. When we compute $S(G_n)$ for a t-parse G_n in the search tree, we do so by computing $S(G_{n-1})$ (where G_{n-1} is G_n with the last operator removed) and then computing from it the next state (or altering it in-place). But we have already computed $S(G_{n-1})$ for the parent node G_{n-1}. Clearly we could compute the congruence state for every node in the search tree without redundant computation. However, the majority of time is in fact spent computing congruences for reductions of t-parses.

Using a cache is the most straight forward way to deal with this problem. A preliminary version was implemented, but it turned out to perform badly, with only on the

Table 2: Number of obstructions for k-Vertex Cover, $1 \leq k \leq 7$.

k	Connected obstructions	Disconnected obstructions	Total obstructions
1	1	1	2
2	2	2	4
3	3	5	8
4	8	10	18
5	31	25	56
6	188	72	260
7	**1930**	320	**2250**
8		**2544**	

order of 30% cache hits. Edge deletions are still likely to share large prefixes with other t-parses, but the t-parses for contractions are not.

5 Conclusions

This paper has described some improvements and simplification to the boundaed-pathwidth obstruction set computation algorithm from [16] and first elaborated into a concrete algorithm in [5, 6]. This has allowed us to compute obstructions for k-Vertex Cover for $k = 6, 7$ using this algorithm ($k = 7$ for the first time using any method), which was previously infeasible.

6 Acknowledgment

The authors want to acknowledge the support of the University of Auckland FRDF grant 9843/3626216 which funded a nice computational server for calculating our results.

References

[1] Faisal Abu-Khzam and Michael Langston. Graph coloring and the immersion order. In Tandy Warnow and Binhai Zhu, editors, *Computing and Combinatorics*, volume 2697 of *Lecture Notes in Computer Science*, pages 394–403. Springer Berlin / Heidelberg, 2003.

[2] Isolde Adler, Martin Grohe, and Stephan Kreutzer. Computing excluded minors. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 641–650, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[3] Dan Archdeacon, C. Paul Bonnington, Nathaniel Dean, Nora Hartsfield, and Katherine Scott. Obstruction sets for outer-cylindrical graphs. *Journal of Graph Theory*, 38(1):42–64, 2001.

[4] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *Proceedings of the ACM Symposium on the Theory of Computing*, 25, 1993.

[5] Kevin Cattell and Michael J. Dinneen. A characterization of graphs with vertex cover up to five. In Vincent Bouchitte and Michel Morvan, editors, *Orders, Algorithms and Applications, ORDAL'94*, volume 831 of *Lecture Notes in Computer Science*, pages 86–99. Springer-Verlag, 1994.

[6] Kevin Cattell, Michael J. Dinneen, and Michael R. Fellows. Obstructions to within a few vertices or edges of acyclic. In *Proceedings WADS'95, Springer-Verlag, Lecture Notes in Computer Science*, pages 415–427. Springer-Verlag, 1995.

[7] Kevin Cattell, Michael J. Dinneen, Rodney G. Downey, Michael R. Fellows, and Michael A. Langston. On computing graph minor obstruction sets. *Theoretical Computer Science A*, 233:107–127, 1997.

[8] Kevin Cattell, Michael J. Dinneen, and Michael R. Fellows. Forbidden minors to graphs with small feedback sets. *Discrete Mathematics*, 230:215–252, March 2001.

[9] John Chambers. Hunting for torus obstructions. Master's thesis, Department of Computer Science, University of Victoria, 2002.

[10] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990.

[11] Guoli Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16(5): 489–502, 1992.

[12] Michael J. Dinneen. *Bounded Combinatorial Width and Forbidden Substructures*. PhD thesis, Dept. of Computer Science, University of Victoria, Canada, 1995.

[13] Michael J. Dinneen and Rongwei Lai. Properties of vertex cover obstructions. *Discrete Mathematics*, 307:2484–2500, 2007. URL `http://www.cs.auckland.ac.nz/CDMTCS/researchreports/254rongwei.pdf`.

[14] Michael J. Dinneen and Liu Xiong. Minor-order obstructions for the graphs of vertex cover 6. *J. Graph Theory*, 41(3):163–178, November 2002.

[15] Michael R. Fellows and Michael A. Langston. On search decision and the efficiency of polynomial-time algorithms. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, STOC '89, pages 501–512, New York, NY, USA, 1989. ACM.

[16] Michael R. Fellows and Michael A. Langston. An analogue of the myhill-nerode theorem and its use in computing finite-basis characterizations. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, pages 520–525, Washington, DC, USA, 1989. IEEE Computer Society.

[17] Michael R. Fellows, Danny Hermelin, and Frances A. Rosamond. Well-quasi-orders in subclasses of bounded treewidth graphs. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation*, pages 149–160. Springer-Verlag, Berlin, Heidelberg, 2009.

[18] Harvey Friedman, Neil Robertson, and Paul Seymour. The meta-mathematics of the graph minor theorem. In *Logic and combinatorics*, volume 65 of *Contemp. Math.*, pages 229–261, Providence, RI, 1987. Amer. Math. Soc.

[19] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.

[20] K. Kawarabayashi, Y. Kobayashi, and B. Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424 – 435, 2012.

[21] K. Kuratowski. Sur le Probléme des Courbes Gauches en Topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[22] Brendan D McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26 (2):306 – 324, 1998.

[23] Brendan D. McKay. nauty Users Guide (version 2.4). 2009.

[24] N. Robertson and P. D. Seymour. Graph minors – a survey. In I. Anderson, editor, *Surveys in Combinatorics*, pages 153–171. Cambridge University Press, 1985.

[25] Neil Robertson and P. D. Seymour. Graph Minors XX. Wagner's conjecture. *J. Comb. Theory Ser. B*, 92(2):325–357, November 2004.

[26] Neil Robertson and Paul Seymour. Graph Minors XXIII. Nash-Williams' immersion conjecture. *J. Comb. Theory Ser. B*, 100(2):181–205, March 2010.

[27] Ralph Versteegen. Computing obstruction sets. Master's thesis, Dept. of Computer Science, University of Auckland, New Zealand, 2012.

[28] K. Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114:570–590, 1937.

A Connected obstructions to 7-Vertex Cover