Practical Enumeration Methods for Graphs of Bounded Pathwidth and Treewidth

Michael J. Dinneen
Department of Computer Science
University of Auckland
Auckland, New Zealand

CDMTCS-055
September 1997

Practical Enumeration Methods for Graphs of Bounded Pathwidth and Treewidth

Michael J. Dinneen
Department of Computer Science
University of Auckland, Private Bag 92019
Auckland, New Zealand

Abstract. Using an algebraic representation for graphs of bounded pathwidth or treewidth we provide simple methods for generating these families in increasing order of the number of vertices and edges. We also study canonic represention of fixed- and free- boundaried graphs of bounded width.

1 The Graph Theory Setting

Many combinatorial search problems can be limited to a domain of "bounded width" graphs. The goal of this paper is to establish practical methods for generating graphs of this type. For example, the enumeration techniques of this paper have been implimented and successfully utilized in the search for forbidden minor characterizations [CD94, CDF95]. This paper is very much self-contained and concerns the combinatorial notions of pathwidth and treewidth. We define and prove a few basic results about these graph width parameters in Sections 2 and 3 before presenting our simple enumeration methods in Section 4.

The reader is assumed to have a familiarity with graph theory, as may be obtained from any one of the standard introductory graph theory texts (e.g. [BM76, CL86]). A few of our basic graph-theoretical conventions follows, where we restrict our attention to 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.

We study the popular notions of *treewidth* and *pathwidth*, introduced by Robertson and Seymour [RS83, RS84]. 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 $k+1$. We are interested in certain subsets of the set of all finite graphs. In particular, the graph families defined by all graphs with either pathwidth or treewidth at most k have "bounded combinatorial width." In general, 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 study 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 forbidden-minor obstruction set computations [CD94, CDF95]. Note that 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 Introduction to Pathwidth and Treewidth

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 2.3 and [Bod86]). 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 [BGHK95, 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, RS86, RS91, RS], 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 (e.g. see [CD94]). For more examples see [Bod86, 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 [Bod86] 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. Some recently discovered obstruction sets for k–Treewidth, $1 \leq k \leq 3$, are given in [APC87]. Besides the two obstructions, K_3 and $S(K_{1,3})$, for 1–Pathwidth, a set of 110 obstructions for 2–Pathwidth (see [Kin89]) is the only other known forbidden-minor characterization for bounded pathwidth or treewidth.

2.1 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 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 $T_x \cap T_z \subseteq T_y$.*

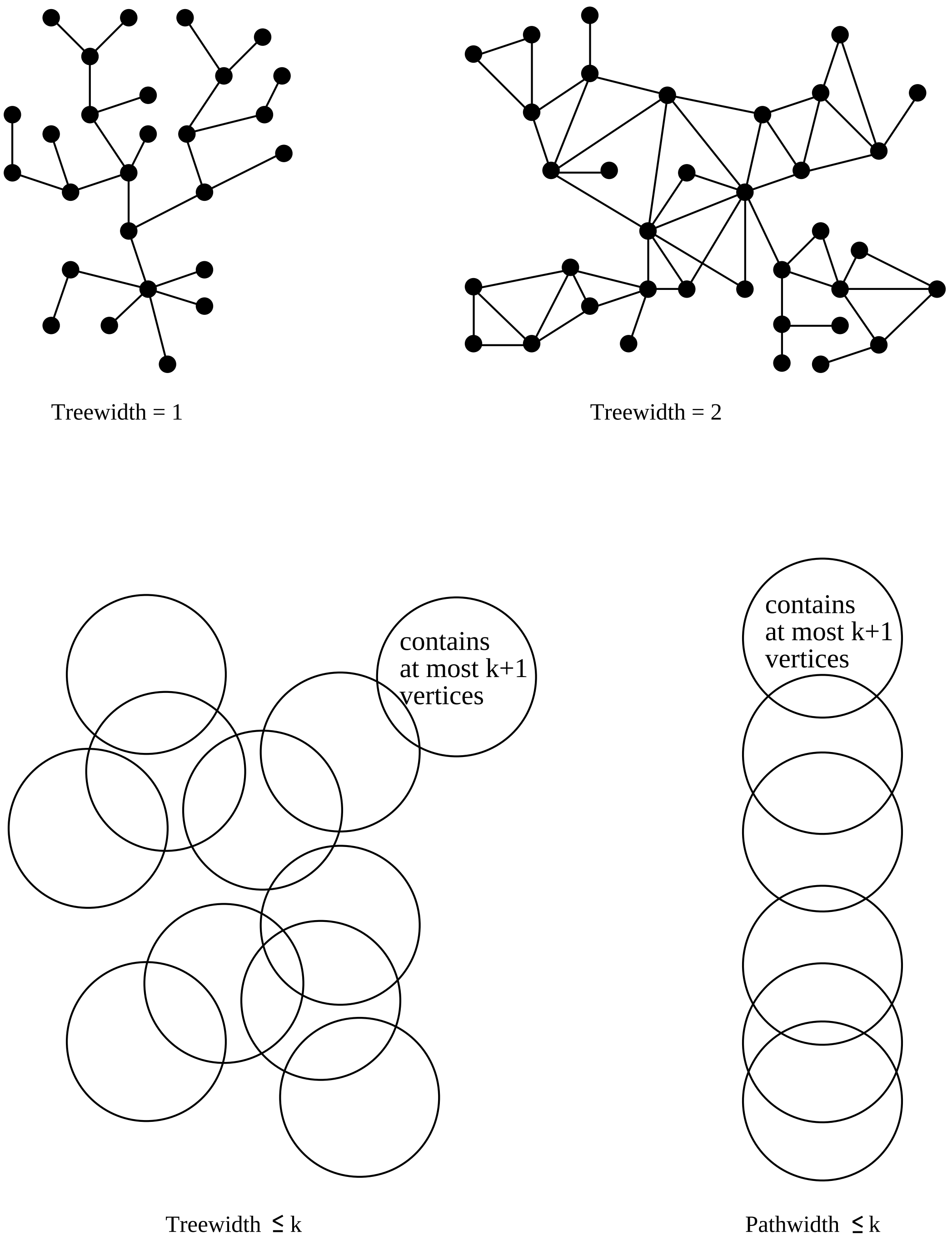


Figure 1: Demonstrating graphs with bounded treewidth (and pathwidth).

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 2.2 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, RS86, Nar89].

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

2.2 Pathwidth and k-paths

To generate graphs of bounded pathwidth (and treewidth) we build from 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 labeled set of boundary vertices with one fewer vertex (see for example [BP71]). Later in this section, Lemmas 9 and 11, we justify these two claims.

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 [Bod93] 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 [RS91]. We use part of the second result when we prove that our algebraic graph representation (given in Section 3) 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.

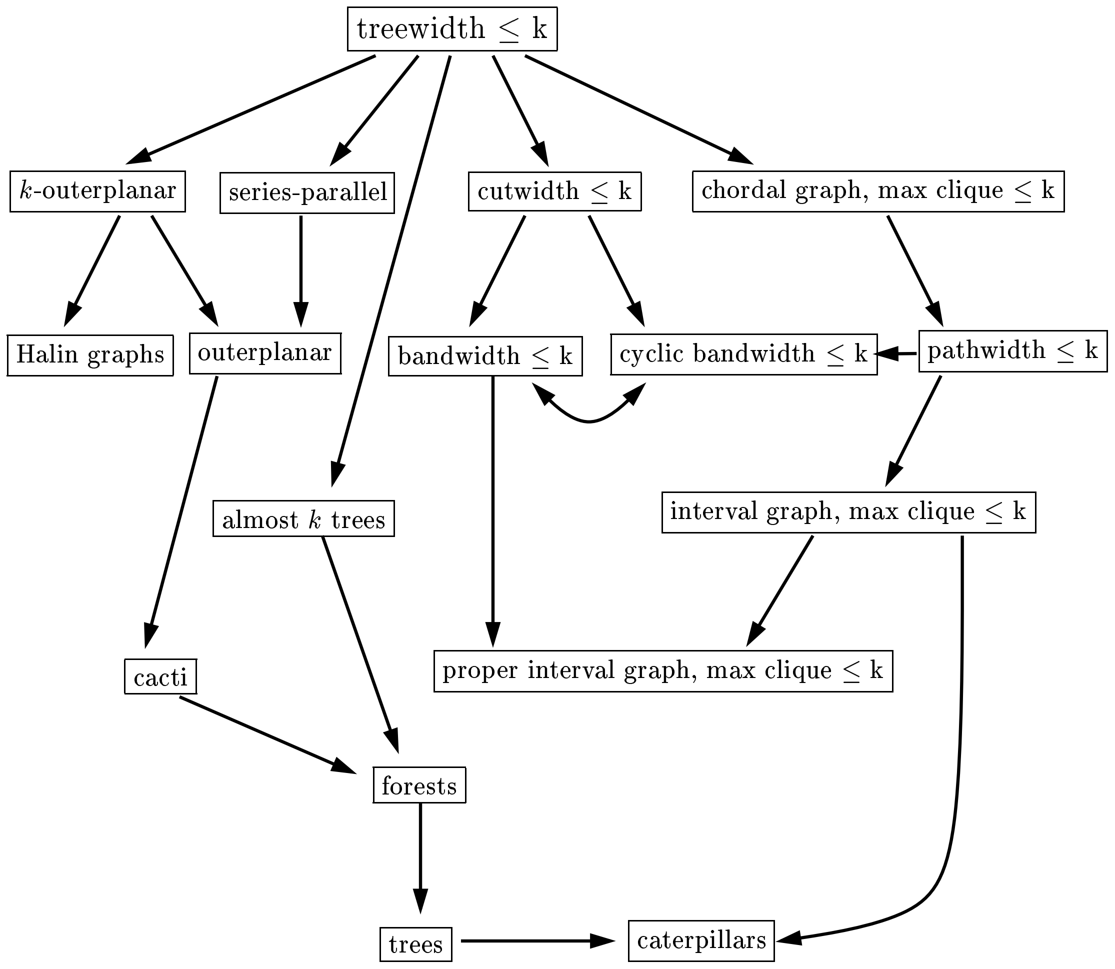
Lemma 11 *Any graph G of pathwidth at most k is a subgraph of a k-path.*

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 $G_r \supseteq G$. □

2.3 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 [Wim87] 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.

We briefly describe two of the bounded width families, given in Figure 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 $f(i) \leq k < f(j)$



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

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 [EST87].

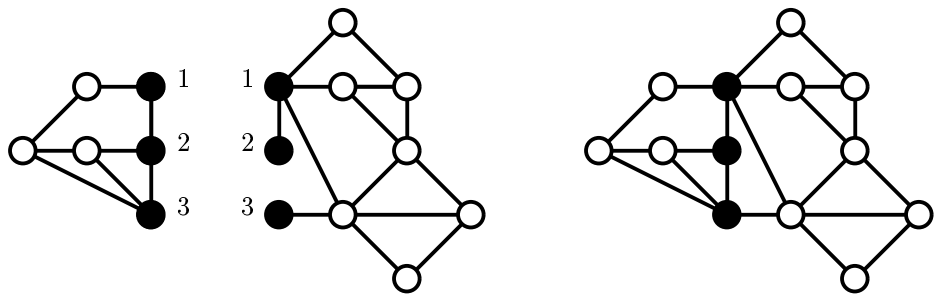
The definition of interval graphs may be found in most introductory graph theory books. We conclude this section with a description for two of the other less known bounded-width families. A *Halin graph* is a planar graph $G = T \cup C$, where T is a tree of order at least 4 with no degree two vertices 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.

3 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 $G = A \oplus B$.*



A $\qquad$ B $\qquad$ $G = A \oplus B$

3.1 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* $\Sigma_t = V_t \cup E_t$:

$$\begin{aligned} V_t &= \{\textcircled{0}, \ldots, \textcircled{t}\} \quad \text{and} \\ E_t &= \{\boxed{i\ j} \mid 0 \le i < j \le t\}. \end{aligned}$$

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 $\boxed{i\ j}$, $0 \le i < j \le t$.*

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 $\partial_S(i) = \textcircled{i}$, $0 \le i \le t$, 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 study of graphs of bounded width, 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 $0 \le i \le t$, 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 $j < k \le n$.

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 section, we mainly limit ourselves to graphs of bounded pathwidth. 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 $G_m = [g_1, g_2, \ldots, g_m]$, $1 \le m \le n$.*

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 $\textcircled{0}$, $\textcircled{1}$, $\ldots$, $\textcircled{t}$ appear in G_n we have $\max_{1 \le i \le |I|} \{|X_i|\} = t + 1$. 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.

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

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

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

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. $\square$

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.

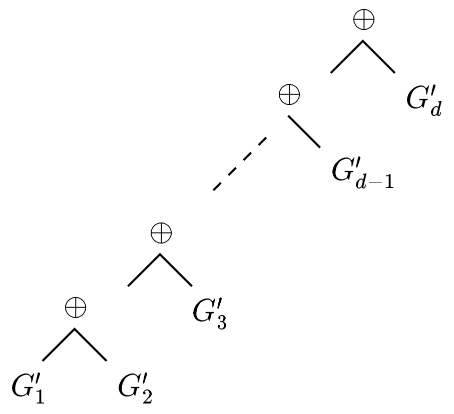
If G has $t+1$ vertices (i.e., a clique K_{t+1}) then the construction given in Lemma 20 for the t-path case is sufficient. We now recursively build a t-parse for G based on the degree $d \geq 1$ of a root vertex r. Let $v_1, \ldots, v_d$ be the neighbors of vertex r in T. For each $1 \leq i \leq d$ 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.

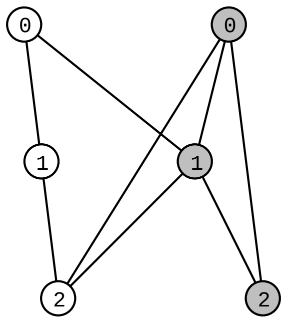


Note that multi-edges created by the repeated $\oplus$ operators are ignored. □

3.2 Some t-parse examples

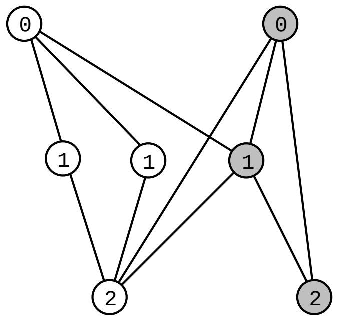
The next three examples illustrate our algebraic representation of t-boundaried graphs of bounded width. The first two t-parse examples are graphs of pathwidth 2 which are then combined with the circle plus operator to form a graph of treewidth 2.

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



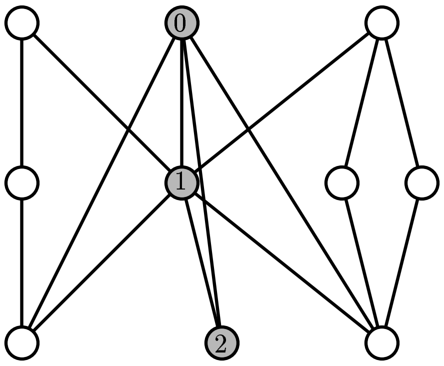
$$[\textcircled{0}, \textcircled{1}, \textcircled{2}, \boxed{0\ 1}, \boxed{1\ 2}, \textcircled{1}, \boxed{0\ 1}, \boxed{1\ 2}, \textcircled{0}, \boxed{0\ 1}, \boxed{0\ 2}, \textcircled{2}, \boxed{0\ 2}, \boxed{1\ 2}]$$

Example 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).*



3.3 Other operator sets

The 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, Wim87]). 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].

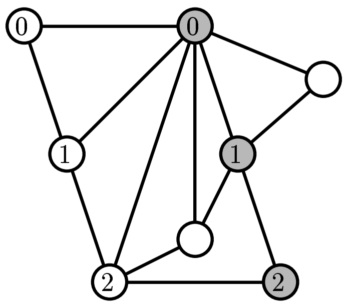
A constant sized operator set Ψ_k, for graphs with pathwidth at most k, consists of two boundary permutation operators p_1 and p_2 and the two t-parse graph building operators $\textcircled{0}$ and $\boxed{0\ 1}$. The domain for these operators is again $(k+1)$-boundaried graphs. The permutation operators 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.

$G\ \boxed{\mathrm{i}}$	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, $1 \le p < k$.

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



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

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 $s_r = \boxed{\text{i}}$ 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 $\bigcup_{1 \leq j \leq i} X_j$. The current boundary of P_i will be $\partial(P_i) = X_i \cap X_{i+1}$, for $1 \leq i < r$. 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

$$P_1 = [\textcircled{0}, \textcircled{1}, \boxed{0\ 1}, \textcircled{2}, \boxed{0\ 2}, \boxed{1\ 2}, \textcircled{3}, \boxed{0\ 3}, \boxed{1\ 3}, \boxed{2\ 3}, \ldots,$$
$$\textcircled{k\text{-}1}, \boxed{0\ k{-}1}, \boxed{1\ k{-}1}, \ldots, \boxed{k{-}2\ k{-}1}, \boxed{\{0, 1, \ldots, k{-}1\}}]$$

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 $1 \leq i < r$. We have two cases to consider when constructing P_{i+1} from P_i, for $1 \leq i \leq r-2$.
If $new_i = old_{i+1}$ then

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

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

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

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

One of the reasons why we picked the 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. For example, for obstruction set computions, 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 for obstruction set searching [Din95].

4 Simple Enumeration Schemes

As indicated by the topic of this paper, the two main invariants that we study are the pathwidth and the treewidth metrics. Many combinatorial graph search problems can be restricted to domains of bounded width (e.g., obstruction set searches). To take advantage of a particular bounded invariant a practical method is needed to generate all these restricted graphs. How to do this smartly is the topic of this section.

As seen by the examples in Section 3, 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.

There exists known polynomial-time algorithms to determine whether two graphs of bounded treewidth are isomorphic [Bod90] (also see [YBdFT97]). However, we would still like to avoid the isomorphism problem (as much as possible, anyway).

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

4.1 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. $\square$

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, $1 \le m < n$, 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 $\textcircled{j}$. $\square$

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 $H_{n-1} \not<_c G_{n-1}$. $\square$

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 $k \le n$ (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. $\square$

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.*

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

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 $\{i,j,k\} \setminus \{a,b\}$. $\square$

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. Our implementation uses a variation of the one presented in [SD76]; but there exists theoretically more efficient algorithms (e.g. [Bod90]).

4.2 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:*

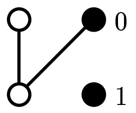
1. *If $G = G_1 \cdot [\boxed{i\ j}]$ then $s(G) = s(G_1)$.*
2. *If $G = G_1 \cdot [\textcircled{i}]$ then $s(G) = [v(G), s(G_1)]$.*

3. If $G = G_1 \oplus G_2$, where without loss of generality $s(G_1) \le s(G_2)$, then $s(G) = [s(G_1), s(G_2)]$.

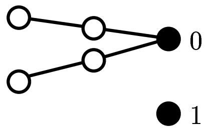
A signature s_1 is less than a signature s_2 if $|s_1| < |s_2|$, or $|s_1| = |s_2|$ and $s_1 < s_2$ in lexicographic order.

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.

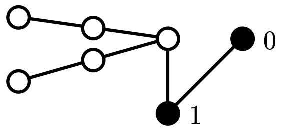
Example 42 *A treewidth* 1*-parse is shown below for $S(K_{1,3})$, a subdivided $K_{1,3}$.*



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



$A \oplus A$



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

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. $\square$

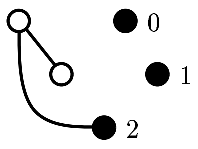
The above lemma allows us to do computations (enumerations) for pathwidth t-parses and then reuse (if using the lex-canonic pathwidth scheme) any partial results for these t-parses when computing within the treewidth t-parse domain. For example, for the obstruction set search method described in [Din95] we could reuse any proofs of graph minimality or nonminimality from a previous search that was restricted to pathwidth t-parses.

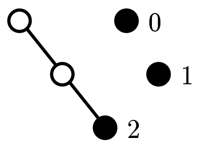
Our treewidth analog to Lemma 36's "prefix of canonic is canonic" is given below.

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. $\square$

As a consequence of the above lemma there is a simple enumeration scheme (as was given in Section 4.1 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.

$G \cdot Z =$


$\simeq_\partial$


$= H$

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

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.

Acknowledgement

Most of the pathwidth t-parse theory presented here was obtained during several tea-time discussions with Kevin Cattell with regards to feasible obstruction set search strategies (e.g., [CD94]). Of course, I take all responsibility for the correctness of the material that is presented here.

References

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8:277–284, April 1987.

[AP89] Stefan Arnborg and Andrzej Proskurowski. Linear algorithms for $\mathcal{NP}$-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, pages 11–24, 1989.

[APC87] Stefan Arnborg, Andrzej Proskurowski, and Derek G. Corneil. Minimal forbidden minor characterization of a class of graphs. *Colloquia Mathematica Societatis János Bolyai*, 52:49–62, 1987.

[Arn85] Stefan Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT*, 25:2–23, 1985. Invited paper.

[BC87] Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical System Theory*, 20:83–127, 1987.

[BGHK95] Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, and minimum elimination tree height. *Journal of Algorithms*, 18:238–255, 1995.

[BH80] Terry Beyer and Sandra Mitchell Hedetniemi. Constant time generation of rooted trees. *SIAM Journal on Computing*, 9:706–712, 1980.

[Bie91] Daniel 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.

[BM76] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* MacMillan, 1976.

[Bod86] Hans 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).

[Bod88a] Hans 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.

[Bod88b] Hans L. Bodlaender. Some classes of graphs with bounded treewidth. *Bulletin of the EATCS*, 36:116–126, 1988.

[Bod90] Hans L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11:631–644, 1990.

[Bod93] Hans 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.

[Bor88] Richard Brian Borie. *Recursively Constructed Graph Families: Membership and Linear Algorithms.* Ph.D. thesis, Georgia Institute of Technology, School of Information and Computer Science, 1988.

[BP71] L.W. Beineke and R.E. Pippert. Properties and characterizations of k-trees. *Mathematika*, 18:141–151, 1971.

[CD94] 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 on Computer Science*, pages 86–99. Springer-Verlag, July 1994.

[CDF95] Kevin Cattell, Michael J. Dinneen, and Michael 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.

[CL86] Gary Chartrand and Linda Lesniak. *Graphs and Digraphs.* Wadsworth Inc., 1986.

[CM93] Bruno Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109:49–82, 1993.

[Din95] Michael J. Dinneen. *Bounded Combinatorial Width and Forbidden Substructures.* Ph.D. dissertation, Dept. of Computer Science, University of Victoria, P.O. Box 3055, Victoria, B.C., Canada V8W 3P6, December 1995.

[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.

[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.

[Fel] Michael R. Fellows. private communication. Dept. of Computer Science, University of Victoria.

[FL92] Michael R. Fellows and Michael 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.

[Kin89] Nancy G. Kinnersley. *Obstruction set isolation for layout permutation problems.* Ph.D. Thesis, Dept. of Computer Science, Washington State University, Pullman, WA 99164, 1989.

[KS93] Haim Kaplan and Ron 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.

[KT92] András Kornai and Zsolt Tuza. Narrowness, pathwidth, and their application in natural language processing. *Discrete Applied Mathematics*, 36:87–92, 1992.

[Lu93] Xiuyan 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.

[MM91] E. Mata-Montero. Resilience of partial k-tree networks with edge and node failures. *Networks*, 21:321–344, 1991.

[Möh90] Rolf H. Möhring. Graph problems releted to gate matrix layout and PLA folding. In G. Tinhofer, E. Mayr, H. Noltemeier, and M. Syslo (in cooperation with R. Albrecht), editors, *Computational Graph Theory, Computing Supplementum 7*, pages 17–51. Springer-Verlag, 1990.

[Nar89] Chandrasekharan Narayanan. *Fast parallel algorithms and enumeration techniques for partial k-trees.* Ph.D. dissertation, Dept. of Computer Science, Clemson University, August 1989.

[Ros73] Donald J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7:317–322, 1973.

[RS] Neil Robertson and Paul D. Seymour. Graph Minors. XX. Wagner's conjecture. in progress.

[RS83] Neil Robertson and Paul D. Seymour. Graph Minors. I. Excluding a Forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[RS84] Neil Robertson and Paul D. Seymour. Graph Minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36:49–64, 1984.

[RS86] Neil Robertson and Paul D. Seymour. Graph Minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[RS91] Neil Robertson and Paul D. Seymour. Graph Minors. X. Obstructions to tree-decompositions. *Journal of Combinatorial Theory, Series B*, 52:153–190, 1991.

[SD76] D. C. Schmidt and L. E. Druffel. A fast backtracking algorithm. *Journal of the Association Computing Machinery*, 23(3):?–445, July 1976.

[WHL85] T. V. Wimer, S. T. Hedetniemi, and R. Laskar. A methodology for constructing linear time graph algorithms. *Congressus Numerantium*, 50:43–60, 1985.

[Wil89] Herbert 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.

[Wim87] T. V. Wimer. *Linear algorithms on k-terminal graphs.* Ph.D. dissertation, Dept. of Computer Science, Clemson University, August 1987. Report No. URI-030.

[WROM86] Robert A. Wright, Bruce Richard, Andrew Odlyzko, and Brendand D. McKay. Constant time generation of free trees. *SIAM Journal on Computing*, 15:540–548, 1986.

[YBdFT97] Koichi Yamazaki, Hans L. Bodlaender, Babette de Fluiter, and Dimitrios M. Thilikos. Isomorphism for graphs of bounded distance width. Technical report 25, Dept. of Computer Science, University of Utrecht, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands, 1997.