Enumeration of unlabeled graph classes

A study of tree decompositions and related approaches

Jessica Shi
Advisor: Jérémie Lumbroso
Independent Work Report Fall, 2015

Abstract

In this paper, we study the enumeration of certain classes of graphs that can be fully characterized by tree decompositions; these classes are particularly significant due to the algorithmic improvements derived from tree decompositions on classically NP-complete problems on these classes [12, 7, 17, 35]. Previously, Chauve et al. [6] and Iriza [26] constructed grammars from the split decomposition trees of distance hereditary graphs and 3-leaf power graphs. We extend upon these results to obtain an upper bound grammar for parity graphs. Also, Nakano et al. [25] used the vertex incremental characterization of distance hereditary graphs to obtain upper bounds. We constructively enumerate $(6,2)$*-chordal bipartite graphs, (*C_5*, bull, gem, co-gem)-free graphs, and parity graphs using their vertex incremental characterization and extend upon Nakano et al.'s results to analytically obtain upper bounds of* $\mathrm{O}(7^n)$ *and* $\mathrm{O}(11^n)$ *for* $(6,2)$*-chordal bipartite graphs and (*C_5*, bull, gem, co-gem)-free graphs respectively.*

1. Introduction

1.1. Context

The technique of decomposing graphs into trees has been an object of significant interest due to its applications on classical problems in graph theory. Indeed, many graph theoretic problems are inherently difficult due to the lack of recursive structure in graphs, and recursion has historically offered efficient solutions to challenging problems. In this sense, classifying graphs in terms of trees is of particular interest, since it associates a recursive structure to these graphs.

There are multiple methods for decomposing graphs into trees, which we discuss in Appendix C. For each method, certain graphs, known as *totally decomposable* graphs, can be fully described by restrictions on the labels of the internal nodes in the corresponding trees. Subsequently, certain classically NP-hard problems can be solved in polynomial time on the corresponding trees, leading to efficient solutions on the graphs. For example, Rao [35] constructed polynomial time algorithms for the clique problem, domination-type problems, and the chromatic number problem on parity graphs, using *split decomposition*.

In this paper, we focus primarily on *split decomposition*, which was first introduced by Cunningham [11]. Split decomposition is closely related to the concept of connected components, and indeed split decomposition identifies and "splits" groups of vertices which are connected in certain ways. Cunningham [11] originally created a polynomial time algorithm to compute the split decomposition of a graph; Ma and Spinrad [29] improved upon this algorithm, and Dahlhaus [12] derived a faster, linear time algorithm. This has led to various algorithmic improvements, including

a linear time algorithm to recognize parity graphs [12] and a linear time algorithm to recognize circle graphs [17]. Other notable applications of split decomposition include the proof of the perfect graph theorem [7].

1.2. Our Work

Our main goal in this paper is to gain a better understanding of the totally decomposable graphs in split decomposition by answering two fundamental questions of graph theory, namely counting the number of graphs in a class and describing the structural properties of those graphs. We focus on two main approaches, both of which utilize techniques from analytic combinatorics to fully describe and enumerate certain classes of trees [16].

The first approach involves the work of Gioan and Paul [19], who developed the notion of *graph-labeled trees* and the methodology for creating such trees from the split decomposition operations introduced by Cunningham [11]. The resulting trees have internal nodes which are labeled with certain types of graphs and leaves which represent the vertices of the original graphs. In particular, Chauve *et al.* [6] and Iriza [26] used these results to obtain symbolic grammars and upper bounds on distance hereditary graphs and 3-leaf power graphs. We extend upon their results to obtain a symbolic upper bound grammar on parity graphs.

The second approach involves *vertex incremental* characterizations, which are the necessary and sufficient conditions under which adding a vertex to a graph in a certain class would produce another graph in that class. Gioan and Paul [19] first derived vertex incremental characterizations for distance hereditary graphs and 3-leaf power graphs based on split decomposition, and Nakano *et al.* [25] used these characterizations to derive upper bounds for distance hereditary graphs. Note that the bounds from Chauve *et al.* [6] and Iriza [26] supersede these bounds. We extend upon these results and use vertex incremental characterizations from de Montgolfier and Rao [13] and from Cicerone and Di Stefano [8, 9] to constructively obtain enumerations for $(6,2)$-chordal graphs, (C_5, bull, gem, co-gem)-free graphs, and parity graphs, and to analytically obtain upper bounds for $(6,2)$-chordal graphs and (C_5, bull, gem, co-gem)-free graphs, of $\mathrm{O}\left(7^n\right)$ and $\mathrm{O}\left(11^n\right)$ respectively.

1.3. Outline

In Section 2, we introduce preliminary notations and definitions. In particular, in Section 2.4, we introduce some basic techniques from analytic combinatorics which we use throughout this paper. Also, in Section 2.5, we discuss the dissymmetry theorem, which was first introduced by Otter [33] and later popularized by Bergeron *et al.* [1], and which provides a method for enumerating unrooted trees by considering their rooted counterparts. In Section 3, we discuss split decomposition and derive upper bounds for parity graphs, extending upon the results of Chauve *et al.* [6] and Iriza [26]. In Section 4, we discuss vertex incremental and constructively derive enumerations for $(6,2)$-chordal bipartite graphs, (C_5, bull, gem, co-gem)-free graphs, and parity graphs. We also derive upper bounds for $(6,2)$-chordal bipartite graphs and (C_5, bull, gem, co-gem)-free graphs, extending upon results from Nakano *et al.* [25].

2. Preliminaries and Definitions

2.1. Graph elementary definitions

For the purposes of this paper, every graph G is *simple*, that is to say, undirected, unlabeled, without self-loops, and without multiple edges. We denote its vertex set by $V(G)$ and its edge set by $E(G)$, and for any vertices $x, y \in V(G)$, we denote an edge between x and y as $(x, y) \in E(G)$. Note that since G is undirected, we have $(x, y) = (y, x)$.

The vertices x and y are said to be *adjacent* if there exists an edge between them, namely $(x, y) \in E(G)$. The vertex x is said to have *degree* k if there are exactly k vertices adjacent to x.

Moreover, for a set of vertices $V \subseteq V(G)$, the *neighborhood* of V, denoted by $N(V)$, is defined to be the set of vertices in $V(G) \setminus V$ that are adjacent to at least one vertex in V. For a single vertex $v \in V(G)$, the *neighborhood* of V, denoted by $N(v)$, is similarly defined as the set of vertices adjacent to v.

For a set of vertices $V \subseteq V(G)$, the *closed neighborhood* of V, denoted by $N[V]$, is defined to be $N(V) \cup V$, and similarly, for a vertex $v \in V(G)$, the *closed neighborhood* of v, denoted by $N[v]$, is defined to be $N(v) \cup \{v\}$.

For a set of edges $E \subseteq E(G)$, the *edge-induced subgraph* is the graph H with vertices $V(H)$ and edges $E(H) = E$ such that for any vertices $x, y \in V(G)$, $x, y \in V(H)$ if and only if $(x, y) \in E$. Similarly, for a set of vertices $V \subseteq V(G)$, the *vertex-induced subgraph* is the graph H with vertices $V(H) = V$ and edges $E(H)$ such that for any edge $(x, y) \in E(G)$, $(x, y) \in E(H)$ if and only if $x, y \in V$. In both cases, H is also known simply as the *induced subgraph*, and we say E or V *induce* the subgraph H of G.

2.2. Bipartite graphs

We also refer to a special class of graphs throughout this paper, namely *bipartite* graphs. We define a *bipartition* of the vertices $V(G)$ of a graph G to be a division of the vertices into two subsets, V_1 and V_2, such that V_1 and V_2 are disjoint and $V_1 \cup V_2 = V(G)$.

A *bipartite* graph is a graph G in which there exists a bipartition of the vertices V_1 and V_2 such that every edge connects a vertex in V_1 to a vertex in V_2 (that is to say, for any vertices $x, y \in V_1$, we have $x, y \notin E(G)$, and for any vertices $u, v \in V_2$, we have $u, v \notin E(G)$). A *complete bipartite graph* is a bipartite graph in which given the bipartition mentioned previously, V_1 and V_2, every vertex in V_1 is connected to every vertex in V_2. Figure 1 shows an example of a complete bipartite graph.

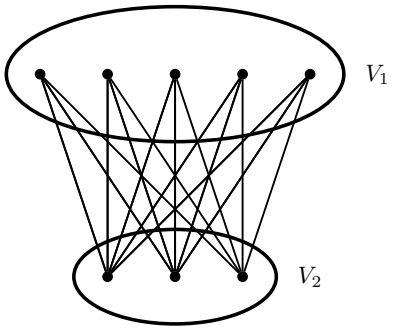

Figure 1. Complete bipartite graph with 5 vertices in V_1 and 3 vertices in V_2.

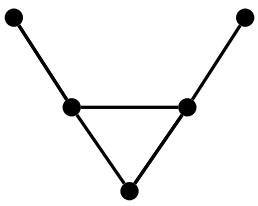

Figure 2. Bull graph.

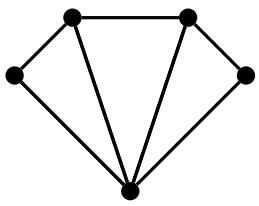

Figure 3. Gem graph.

2.3. Special classes of graphs

We now provide definitions for the main graph classes we consider throughout this paper.

Definition 2.1. A *parity* graph, also known as a *parity perfect* graph, is a graph in which for any pair of vertices, all path lengths between them have the same parity.

Note that in this definition, it is irrelevant whether path length is defined as the number of vertices or the number of edges on a path; for the sake of clarity, we define path length to refer to the number of edges on a path.

Definition 2.2. A $(6,2)$-*chordal bipartite* graph is a graph which is $(6,2)$-chordal and bipartite. A $(6,2)$-*chordal* graph is a graph in which every cycle of length at least 6 has at least 2 *chords*, which are edges that are not part of the cycle but connect vertices in the cycle [3].

Definition 2.3. A *(C_5, bull, gem, co-gem)-free* graph is a graph which does not contain a cycle graph of length 5, a bull graph, a gem graph, or the complement of a gem graph as induced subgraphs. A *cycle* graph of length n is a graph that consists of a single cycle, with n vertices. A *bull* graph is the graph shown in Figure 2. A *gem* graph is the graph shown in Figure 3.

2.4. Analytic combinatorics

We now introduce the analytic combinatorics necessary to enumerate the various classes of trees considered throughout this paper. A *combinatorial class* $\mathcal{A}$, also known simply as *class*, is a countable set of objects with a *size* function $|\cdot| : \mathcal{A} \to \mathbb{Z}_{\geq 0}$ such that for every size (every n in $\mathbb{Z}_{\geq 0}$), there is a finite number of elements. For example, the set of binary trees with a size function that gives the number of internal nodes in each binary tree is a class.

Given a class of objects $\mathcal{A}$, we define an *enumeration* of $\mathcal{A}$ to be the sequence given by $a_n = \#\{\gamma \in \mathcal{A} \mid |\gamma| = n\}$ (in other words, the sequence $\{a_n\}$ in which a_n is the number of objects in $\mathcal{A}$ of size n). The corresponding formal power series $\mathcal{A}(z) = \sum_{n=0}^{\infty} a_n z^n$ is called the *ordinary generating function (OGF)* of A and is used when $\mathcal{A}$ is unlabeled.

The symbolic method, which is what we use in this paper, is based on the idea that there exist symbolic operations that, when applied to combinatorial classes, translate directly to an OGF. We begin with the base elements upon which these operations can be applied. These include the neutral object, ε, which has no size and in an OGF, translates to $z^0 = 1$, and the atomic object, $\mathcal{Z}$, which

Name	**Symbol**	**OGF**
Neutral (element of size 0)	ε	1
Atom (element of size 1)	$\mathcal{Z}$	z
Disjoint Union	$\mathcal{A}+\mathcal{B}$	$\mathcal{A}(z)+\mathcal{B}(z)$
Product	$\mathcal{A}\times\mathcal{B}$	$\mathcal{A}(z)\cdot\mathcal{B}(z)$
Sequence	$\textsc{Seq}(\mathcal{A})$	$1/(1-\mathcal{A}(z))$
Set	$\textsc{Set}(\mathcal{A})$	$\exp(\mathcal{A}(z))$
Pointing	$\Theta\mathcal{A}$	$z\frac{d}{dz}\mathcal{A}(z)$
Substitution	$\mathcal{A}\circ\mathcal{B}$	$\mathcal{A}(\mathcal{B}(z))$

Table 1. Unlabeled combinatorial classes, from Flajolet and Sedgewick [16].

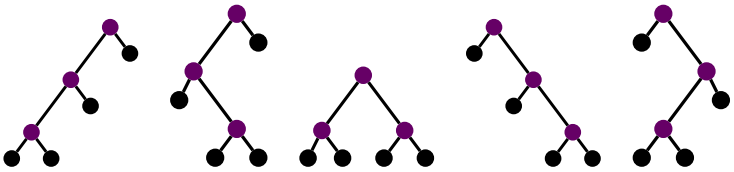

Figure 4. All binary trees of size 3 (with 3 internal nodes, in purple).

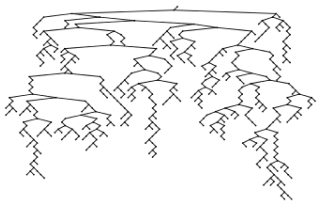

Figure 5. Randomly generated binary tree.

has size 1 and in an OGF, translates to $z^1 = z$. A description of these base elements and some basic symbolic operations are shown in Table 1. For certain classes of graphs, it is possible to express the class entirely in terms of symbolic operations, these base elements, and itself (for recursion), and we call these classes *decomposable*. We call the symbolic representation of such classes a *grammar*.

For example, let $\mathcal{B}$ be the class of binary trees where the size function gives the number of internal nodes. All binary trees of size 3 are shown in Figure 4. Recursively, binary trees are either a leaf (no size contribution) or an internal node (1 size contribution) with two binary subtrees. As such, the grammar for this class is given by

$$B = \varepsilon + B \times Z \times B.$$

This results in the functional equation

$$B(z) = 1 + B(z) \cdot z \cdot B(z),$$

and solving the quadratic, we receive

$$B(z) = \frac{1 - \sqrt{4z-1}}{2}.$$

It is then possible to derive a formal power series for $\mathcal{B}(z)$ to receive the OGF. There are various computer algebra systems that automate this process, that is, given a grammar, can evaluate and return an OGF. For the purposes of this paper, we use Maple with the `combstruct` package to evaluate symbolic grammars. In the case of $\mathcal{B}$, this gives us

$$B(z) = z^1 + 2z^2 + 5z^3 + 14z^4 + \ldots.$$

Note that it is also possible to obtain a random sampler from a grammar, through the recursive method or through Boltzmann samplers. The details of this is beyond the scope of this paper. We provide an example of a randomly generated binary tree in Figure 5.

2.5. Dissymmetry theorem

Note that binary trees are rooted at a vertex, that is to say, they have a "distinguished" vertex from which they can be recursively decomposed into smaller binary trees. In general, the above techniques can easily enumerate such classes of trees, with a "distinguished" vertex, edge, or other feature from which they can be either recursively decomposed into smaller trees of the same class or decomposed into trees of other classes. Such classes of trees are known as *rooted* trees, and the "distinguished" vertex, edge, or feature is known as the *root*.

However, in subsequent sections, we focus on trees that do not have a root, known as *unrooted* trees. This poses a nontrivial problem for unlabeled trees. We consider here one technique for addressing this problem, namely the dissymmetry theorem, which relates an unrooted class of trees to its rooted versions. The dissymmetry theorem was first developed by Otter [33] and later popularized by Bergeron *et al.* [1].

Theorem 2.1 (Dissymmetry theorem)**.** *Let $\mathcal{A}$ be an unrooted class of trees, and let $A^{\bullet}$, $A^{\bullet-\bullet}$, and $A^{\bullet\to\bullet}$ be the corresponding classes of trees rooted at a vertex, an edge, and a directed edge respectively. Then, there is a bijection*

$$A + A^{\bullet\to\bullet} \simeq A^{\bullet} + A^{\bullet-\bullet}$$

in the sense that for a given n, there are the same number of objects of size n in both sets. In other words,

$$a_n = a_n^{\bullet} + a_n^{\bullet-\bullet} - a_n^{\bullet\to\bullet}.$$

Note that Chapuy *et al.* have previously used the dissymmetry theorem to characterize classes of trees that are obtained from decomposing graphs [5]. Also, Chauve *et al.* [6] proved that it is possible to consider only internal nodes when applying the dissymmetry theorem.

Lemma 2.2. *The dissymmetry theorem remains true when the three rooted classes are restricted to only contain those trees rooted at internal nodes or edges between two internal nodes.*

We use this lemma in conjunction with symbolic grammars to enumerate parity graphs in Section 3.2.

3. Enumerating graphs exactly using split decomposition

In this section, we derive upper bounds for parity graphs, extending upon the results of Chauve *et al.* [6] and Iriza [26]. Specifically, Gioan and Paul [19] developed a method to represent graphs as *graph-labeled trees*, using a technique called *split decomposition*. We apply this to parity graphs and then enumerate the resulting graph-labeled trees using techniques introduced in Sections 2.4 and 2.5.

3.1. Split decomposition

Gioan and Paul [19] developed the notion of representing certain classes of graphs using *graph-labeled trees*, based on tree decomposition techniques, including *split decomposition*. We begin by reproducing their characterizations and results here.

We first introduce Gioan and Paul's [19] notions of graph-labeled trees.

Definition 3.1. A *graph-labeled tree* $(T, \mathcal{F})$ is a tree T in which every internal node u is labeled by a graph $G_u \in \mathcal{F}$ such that there is a bijection ρ_u from the tree edges of T incident to u to $V(G_u)$. We call the vertices of G_u *marker-vertices*.

Figure 6 depicts a graph-labeled tree T in which the leaves are denoted by small shaded circles, the internal nodes are denoted by large circles, and the marker-vertices are denoted by small solid circles. The bijection ρ_u is denoted by the edges from the marker-vertices to the other nodes of

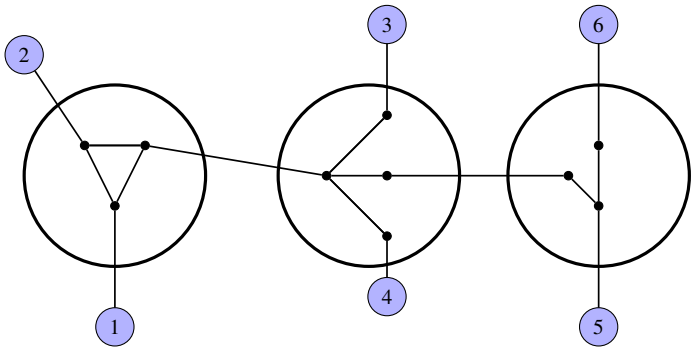

Figure 6. Example of a graph-labeled tree.

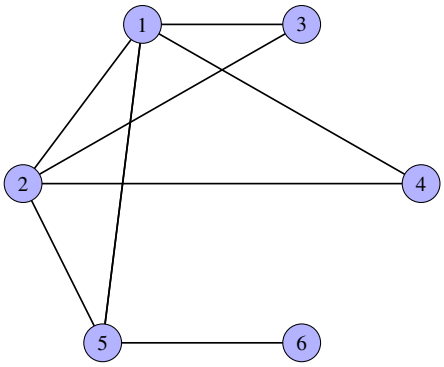

Figure 7. Accessibility graph for the graph-labeled tree in Figure 6.

the tree. Note that the leaves are numbered for convenience in discussing them, and the tree T is actually unlabeled.

In order to connect the concept of graph-labeled trees with graphs, we introduce Gioan and Paul's [19] notion of *accessibility*.

Definition 3.2. Let $(T,\mathcal{F})$ be a graph-labeled tree and let u,v be leaves of T. We say that u and v are *accessible* if it is possible to draw a path from u to v by traversing no more than on edge in each internal graph label. More formally, u and v are accessible if there exists a path from u to v such that for any pair of tree edges $e=(x,y)$ and $e'=(y,x')$ on the path, we have $(\rho_y(e),\rho_y(e'))\in E(G_y)$.

For example, in the graph-labeled tree T in Figure 6, leaf 2 and leaf 3 are accessible while leaf 3 and leaf 6 are not accessible.

Definition 3.3. The *accessibility graph*, also known as the *original graph*, of a graph-labeled tree $(T,\mathcal{F})$ is the graph G whose vertex set is the leaf set of T, and for $x,y\in V(G)$, we have $(x,y)\in E(G)$ if and only if x and y are accessible in T.

Figure 7 depicts the accessibility graph G corresponding to the graph-labeled tree T shown in Figure 6, where the vertices of G are labeled to correspond with the leaves in T.

In particular, given a connected graph G, Gioan and Paul [19] use split decomposition to represent the graph as a graph-labeled tree T in which the accessibility graph of T is exactly G. The following definitions regarding split decomposition were originally derived from Cunningham [11].

Definition 3.4. A *split* of a graph G is a bipartition (V_1,V_2) of $V(G)$ such that

1. $|V_1|\geq 2$ and $|V_2|\geq 2$, and
2. the edges between V_1 and V_2 induce a complete bipartite graph.

Note that the second statement in the definition is equivalent to the statement, every vertex of $N(V_1)$ is adjacent to every vertex of $N(V_2)$.

Definition 3.5. A graph is *degenerate* if every bipartition of its vertices into two sets of size ≥ 2 is a split, and a graph is *prime* if it has no split. The only degenerate graphs are known to be cliques and stars.

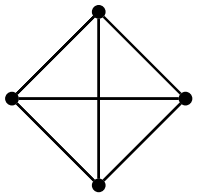

Figure 8. Clique with 4 vertices.

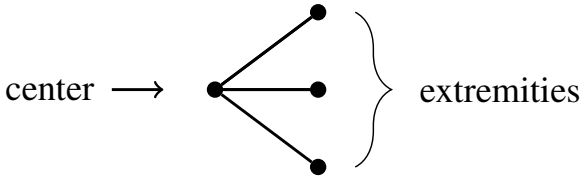

Figure 9. Star with 4 vertices.

A *clique* is a graph in which every pair of vertices is connected by an edge, and a *star* is a graph with n vertices such that some vertex $x \in V(G)$ is adjacent to the other $n-1$ vertices and there are no other edges (more formally, for some $x \in V(G)$, we have $E(G) = \{(x,y) \mid y \in V(G) \setminus \{x\}\}$). In a star, we refer to the vertex x as its *center* and the other $n-1$ vertices as its *extremities*. Figure 8 shows an example of a clique and Figure 9 shows an example of a star.

In a graph-labeled tree, an internal node labeled with a clique is called a *clique node*, an internal node labeled with a star is called a *star node*, and an internal node labeled with a prime graph is called a *prime node*. In general, if we have an internal node labeled with a graph of class $\mathcal{A}$, we call the node a $\mathcal{A}$ *node*. Gioan and Paul [19] apply this concept of a split to create a graph-labeled tree in the following manner. Given a connected graph G, we begin with a graph labeled tree $(T,\mathcal{F})$ such that T has exactly one internal node labeled with G, and as such, $|V(G)|$ leaves connected to the internal node. Necessarily, $\mathcal{F} = \{G\}$. We then repeatedly apply the following *node-split operation* until the internal nodes of T are labeled with graphs which are either degenerate or prime:

Definition 3.6. Given an internal node g of T with graph label G and a split (V_1,V_2) of G, we can consider the complete bipartite graph H induced by the edges between V_1 and V_2. Let X_1 denote the set of all vertices x_1 of H such that $x_1 \in V_1$ and let X_2 denote the set of all vertices x_2 of H such that $x_2 \in V_2$. In the *node-split operation*, we replace g with two internal nodes, g_1 and g_2, labeled with G_1 and G_2 respectively, where G_1 and G_2 are the subgraphs of G induced by V_1 and V_2 respectively. We add a vertex a_1 to G_1, and for each vertex $x_1 \in X_1$, we add the edge (a_1,x_1) to G_1. Similarly, we add a vertex a_2 to G_2, and for each vertex $x_2 \in X_2$, we add the edge (a_2,x_2) to G_2. Finally, we add an edge to T connecting these two new internal nodes, between a_1 and a_2.

An example of a node-split operation is shown in Figure 10. On the left is a graph-labeled tree T before the split operation and on the right is the tree after the split operation, in which the internal nodes are denoted by large circles, the leaves are denoted by small shaded circles (shaded red or

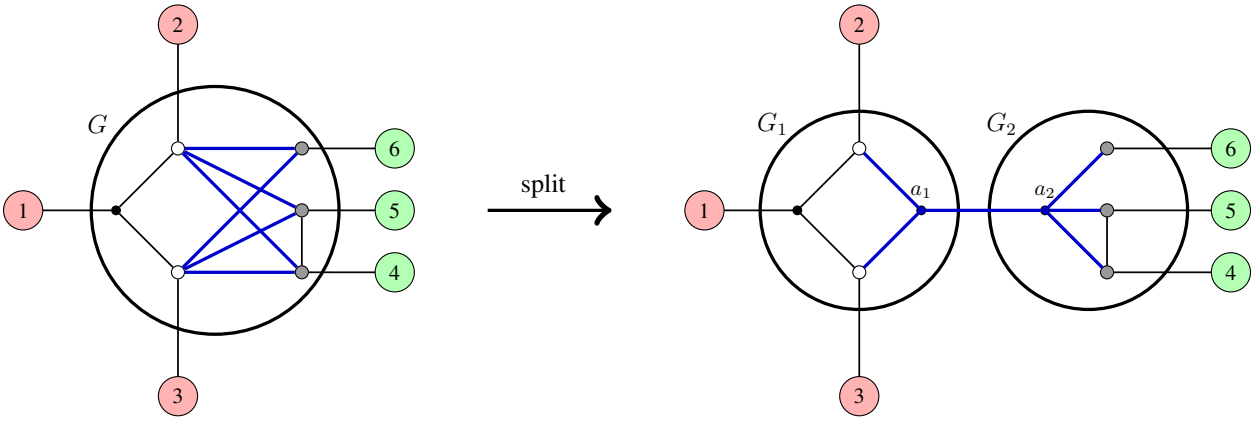

Figure 10. Example of a node-split operation on a graph-labeled tree.

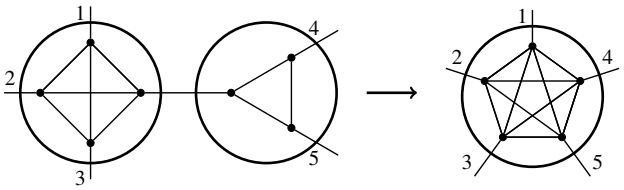

Figure 11**.** Reduction of two adjacent clique nodes in a split tree to a single clique (recreated from [26]).

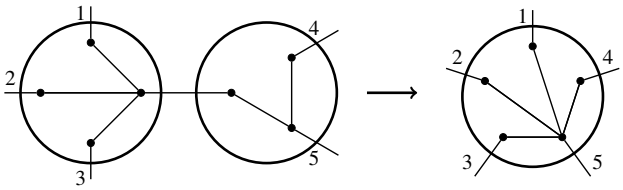

Figure 12**.** Reduction of two adjacent start nodes in a split tree to a single star (recreated from [26]).

green), and the marker-vertices are denoted by smaller circles (shaded black, gray, or white). Note that the leaves are labeled for convenience, and are not actually labeled in T. The leaves colored in red represent the vertices of G associated with V_1 and the leaves colored in green similarly represent V_2. The edges of H are colored blue, and the white marker-vertices represent X_1 while the gray marker-vertices represent X_2.

Note that the node-split operation does not modify the accessibility graph of the graph-labeled tree. Since the graph-labeled tree we started with had accessibility graph G, the resulting graph-labeled tree after repeated applications of the node-split operations also has accessibility graph G. Thus, we obtain from a graph G, a graph-labeled tree $(T,\mathcal{F})$ such that the graph labels of T are all either degenerate or prime.

Definition 3.7. A graph-labeled tree $(T,\mathcal{F})$ is *reduced* if

1. every $v \in V(T)$ has degree ≥ 3,
2. there does not exist $(u,v) \in E(T)$ such that their corresponding labels G_u and G_v are both cliques, and
3. there does not exist $e = (u,v) \in E(T)$ such that their corresponding labels G_u and G_v are both stars where $\rho_u(e)$ is the center of G_u and $\rho_v(e)$ is an extremity of G_v.

Every graph-labeled tree can be reduced, because any two adjacent clique nodes is the result of a node-split operation on a single internal node labeled with a clique, and any two adjacent star nodes (in the manner described) is the result of a node-split operation on a single internal node labeled with a star (see Figures 11 and 12 respectively). The graph-labeled tree obtained earlier can as such be reduced, and in fact Gioan and Paul [19] prove that such a graph-labeled tree is unique.

Theorem 3.1. *For any connected graph G, there exists a unique reduced graph-labeled tree $(T,\mathcal{F})$ such that G is the accessibility graph of $(T,\mathcal{F})$ and every node label $G_v \in \mathcal{F}$ is either prime or degenerate. We call this graph-labeled tree the* split tree *of G, and denote it by $ST(G)$.*

Note that the graph-labeled tree in Figure 6 is actually the split tree of the accessibility graph in Figure 7.

3.2. Results: Parity graphs

Of particular interest is the following characterization of parity graphs by certain conditions on their split trees, proven by Cicerone and Di Stefano [9].

Theorem 3.2. *A graph G is a parity graph if and only if its split tree has only clique and connected bipartite nodes.*

Note that a star graph is a bipartite graph, so it is redundant to include star nodes. Throughout this paper, we use *bipartite node* interchangeably with *connected bipartite node*, for brevity. We now note that in a reduced split tree for parity graphs, no two adjacent internal nodes can be bipartite nodes. We prove this result below:

Theorem 3.3. *In a graph-labeled tree $(T,\mathcal{F})$, any two adjacent bipartite nodes can be reduced to one bipartite node such that the accessibility graph is preserved.*

Proof. Consider two adjacent bipartite nodes in $(T,\mathcal{F})$, say a and b with edge $e=(a,b)$, labeled with bipartite graphs A and B respectively. We can consider a bipartition of A, say A_1 and A_2, such that every edge in A connects a vertex in A_1 to a vertex in A_2. We consider a similar bipartition of B, say B_1 and B_2.

Let $x_a=\rho_a(e)$ and $x_b=\rho_b(e)$. Without loss of generality, let $x_a\in A_2$ and $x_b\in B_2$. Now, let $X_a\subseteq A_1$ such that $X_a=\{x\mid(x,x_a)\in E(A)\}$. Similarly, let $X_b\subseteq B_1$ such that $X_b=\{x\mid(x,x_b)\in E(B)\}$. Note that X_a is disjoint from A_2 and X_b is disjoint from B_2 by construction.

Remove the internal nodes a and b from T and replace them with an internal node c with graph label C, where C is the disjoint union of graphs A and B (so $V(C)=V(A)\sqcup V(B)$ and $E(C)=E(A)\sqcup E(B)$). Call this new graph-labeled tree $(T',\mathcal{F}')$. In C, remove vertices x_a and x_b. Also, connect every vertex in X_a to every vertex in X_b, so $X_a\cup X_b$ induces a complete bipartite graph from C. Note that as such, $A_1\cup B_2$ and $A_2\cup B_1$ is a bipartition of C such that every edge in C connects a vertex in one set to a vertex in the other. Thus, C is a bipartite graph and c is a bipartite node.

Moreover, note that by construction, $V(A)$ and $V(B)$ is a split, and we can apply the split operation to c in $(T',\mathcal{F}')$ to receive the original graph-labeled tree $(T,\mathcal{F})$. As such, since the node-split operation preserves the accessibility graph, the accessibility graph of $(T,\mathcal{F})$ and $(T',\mathcal{F}')$ is the same. Thus, we have reduced two adjacent bipartite nodes to one bipartite node while preserving the accessibility graph. □

See Figure 13 for an example of this reduction.

Based on Theorem 2.1, Lemma 2.2, Theorem 3.2, and Theorem 3.3, we obtain the following

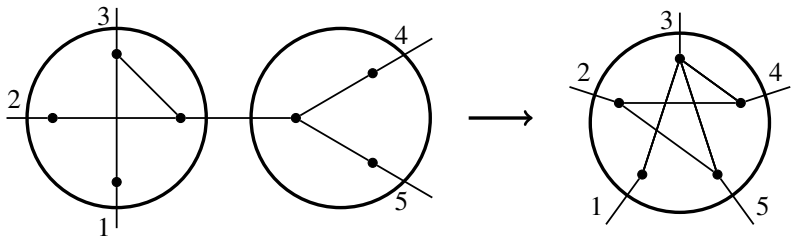

Figure 13**.** Reduction of two adjacent bipartite nodes in a split tree to a single bipartite node.

upper bound grammar for connected parity graphs, $\mathcal{PG}_{\mathcal{C}}$:

$$\begin{aligned}
\mathcal{PG}_{\mathcal{C}} + \mathcal{T}_{\mathcal{K}-\mathcal{P}} &\simeq \mathcal{T}_{\mathcal{K}} + \mathcal{T}_{\mathcal{P}} \\
\mathcal{T}_{\mathcal{K}} &= \textsc{Set}_{\geq 3}\left(\mathcal{Z} + \mathcal{P}\right) \\
\mathcal{T}_{\mathcal{P}} &\leq \left(\mathcal{Z} + \mathcal{K}\right) \circ \mathcal{B} \\
\mathcal{T}_{\mathcal{K}-\mathcal{P}} &= \mathcal{K} \times \mathcal{P} \\
\mathcal{K} &= \textsc{Set}_{\geq 2}\left(\mathcal{Z} + \mathcal{P}\right) \\
\mathcal{P} &\leq \left(\mathcal{Z} + \mathcal{K}\right) \circ \Theta\mathcal{B} \\
\mathcal{P} &\geq \textsc{Set}_{=n}\left(\mathcal{Z} + \mathcal{K}\right) \odot \mu\left(\mathcal{B}\right) \\
B &= z + z^2 + z^3 + 3z^4 + 5z^5 + 17z^6 + 44z^7 + \ldots
\end{aligned}$$

and the upper ground grammar for parity graphs in general, $\mathcal{PG}$:

$$\begin{aligned}
\mathcal{PG} + \mathcal{T}_{\mathcal{K}-\mathcal{P}} &\simeq \mathcal{T}_{\mathcal{K}} + \mathcal{T}_{\mathcal{P}} \\
\mathcal{T}_{\mathcal{K}} &\leq \textsc{Set}\left(\mathcal{Z} + \mathcal{Z} \times \mathcal{Z} + \textsc{Set}_{\geq 3}\left(\mathcal{Z} + \mathcal{P}\right)\right) \\
\mathcal{T}_{\mathcal{P}} &\leq \textsc{Set}\left(\mathcal{Z} + \mathcal{Z} \times \mathcal{Z} + \left(\mathcal{Z} + \mathcal{K}\right) \circ \mathcal{B}\right) \\
\mathcal{T}_{\mathcal{K}-\mathcal{P}} &\geq \textsc{Set}\left(\mathcal{K} \times \mathcal{P}\right) \\
\mathcal{K} &= \textsc{Set}_{\geq 2}\left(\mathcal{Z} + \mathcal{P}\right) \\
\mathcal{P} &\leq \left(\mathcal{Z} + \mathcal{K}\right) \circ \Theta\mathcal{B} \\
\mathcal{P} &\geq \textsc{Set}_{=n}\left(\mathcal{Z} + \mathcal{K}\right) \odot \mu\left(\mathcal{B}\right) \\
B &= z + z^2 + z^3 + 3z^4 + 5z^5 + 17z^6 + 44z^7 + \ldots
\end{aligned}$$

Note that we introduce two non-canonical symbolic operations, $\mu\left(\mathcal{A}\right)$ and $\textsc{Set}_{=n}\left(\mathcal{A}\right) \odot \mathcal{B}$. For $\mu\left(\mathcal{A}\right)$, we define the corresponding OGF to be $z^{-1}\mathcal{A}(z)$. For $\textsc{Set}_{=n}\left(\mathcal{A}\right) \odot \mathcal{B}$, if we let $A_n(z)$ be the OGF of $\textsc{Set}_{=n}\left(\mathcal{A}\right)$, we replace each term $b_n z^n$ of the OGF of $\mathcal{B}$ with $b_n \cdot A_n(z)$. Also, note that we substitute the lower bound for $\mathcal{T}_{\mathcal{K}-\mathcal{P}}$ on the left hand side of the isomorphism and the upper bound for $\mathcal{T}_{\mathcal{K}}$ and $\mathcal{T}_{\mathcal{P}}$ on the right hand side of the isomorphism to receive an upper bound for $\mathcal{PG}_{\mathcal{C}}$. The following list explains the symbols for each class:

- $\mathcal{B}$: A connected bipartite graph; the generating function for connected bipartite graphs, developed by Hanlon [21], is complex and difficult to express as a symbolic grammar. As such, we use a partial generating function, generated by Sloane [37].
- $\mathcal{K}$: A clique node with one of its incident subtrees having been removed
- $\mathcal{P}$: A bipartite node with one of its incident subtrees having been removed (no constraint on which incident subtree is removed)
- $\mathcal{T}_{\mathcal{K}}$: A parity split tree rooted at a clique node
- $\mathcal{T}_{\mathcal{P}}$: A parity split tree rooted at a bipartite node
- $\mathcal{T}_{\mathcal{K}-\mathcal{P}}$: A parity split tree rooted at an undirected edge connecting a clique node and a bipartite node

Proof. The mutually-recursive expressions for $\mathcal{K}$ and $\mathcal{P}$ encode the requirements that the connected parity split tree must have all nodes of degree at least 3, no adjacent clique nodes, and no adjacent bipartite nodes (by Theorem 3.3). By Theorem 2.1 and Lemma 2.2, we have

$$\mathcal{PG}_{\mathcal{C}} + \mathcal{T}_{\mathcal{K}\to\mathcal{K}} + \mathcal{T}_{\mathcal{P}\to\mathcal{P}} + \mathcal{T}_{\mathcal{K}\to\mathcal{P}} + \mathcal{T}_{\mathcal{P}\to\mathcal{K}} \simeq \mathcal{T}_{\mathcal{K}} + \mathcal{T}_{\mathcal{P}} + \mathcal{T}_{\mathcal{K}-\mathcal{K}} + \mathcal{T}_{\mathcal{P}-\mathcal{P}} + \mathcal{T}_{\mathcal{K}-\mathcal{P}}.$$

Note that $\mathcal{T}_{\mathcal{K}\to\mathcal{K}}$, $\mathcal{T}_{\mathcal{P}\to\mathcal{P}}$, $\mathcal{T}_{\mathcal{K}-\mathcal{K}}$, and $\mathcal{T}_{\mathcal{P}-\mathcal{P}}$ are all empty, since clique nodes cannot be adjacent in a split tree and bipartite nodes cannot be adjacent in a split tree. Also, $\mathcal{T}_{\mathcal{K}\to\mathcal{P}} \simeq \mathcal{T}_{\mathcal{P}\to\mathcal{K}} \simeq \mathcal{T}_{\mathcal{K}-\mathcal{P}}$, so it follows that

$$\mathcal{PG}_{\mathcal{C}} + \mathcal{T}_{\mathcal{K}-\mathcal{P}} \simeq \mathcal{T}_{\mathcal{K}} + \mathcal{T}_{\mathcal{P}}.$$

□

Using Maple with the `combstruct` package, we receive the following enumeration:

Corollary 3.4. *The first few terms of the upper bound OGF of the class of connected parity graphs is*

$$\mathcal{PG}_{\mathcal{C}}(z) \leq z + z^2 + 2z^3 + 6z^4 + 21z^5 + 90z^6 + 432z^7 + \ldots.$$

Note that in Section 4.2 we obtain an exact enumeration of connected parity graphs, specifically

$$\mathcal{PG}_{\mathcal{C}}(z) = z + z^2 + 2z^3 + 6z^4 + 18z^5 + 75z^6 + 318z^7 + \ldots,$$

so the first few terms of the upper bound OGF of connected parity graphs appear to be correct.

Corollary 3.5. *The first few terms of the upper bound OGF of the class of parity graphs is*

$$\mathcal{PG}(z) \leq 2z + 4z^2 + 6z^3 + 14z^4 + 38z^5 + 197z^6 + 639z^7 + \ldots.$$

Note that in Section 4.2 we obtain an exact enumeration of parity graphs, which matches the enumeration generated by Hougardy [23], specifically

$$\mathcal{PG}(z) = z + 2z^2 + 4z^3 + 11z^4 + 31z^5 + 116z^6 + 466z^7 + \ldots,$$

so the first few terms of the upper bound OGF of parity graphs appear to be correct.

In addition, Iriza [26] wrote a parsing module that translates a split tree string into the corresponding 3-leaf power graph or distance-hereditary graph. We have extended his parsing module to translate a parity split tree string into the corresponding parity graph. The code can be found in Appendix B.

3.3. Next steps

The difficulty with the grammar presented in Section 3.2 is that because it uses a partial version of the connected bipartite graph generating function, it cannot consider isomorphisms arising from the structure of connected bipartite graphs, hence the reason why the grammar merely gives an upper bound. As previously mentioned, Hanlon [21] has developed a generating function for connected bipartite graphs, which could potentially be used to construct an exact grammar for parity graphs. However, the generating function is highly complex and difficult to integrate with the grammar developed in Section 3.2.

Also, Iriza [26] used the dissymmetry theorem to build a Boltzmann sampler for distance hereditary graphs and 3-leaf power graphs. We may be able to extend this result to parity graphs as well, and obtain a Boltzmann sampler for parity graphs.

4. Enumerating graphs constructively using vertex incremental

In this section, we constructively derive enumerations for $(6,2)$-chordal bipartite graphs,(C_5, bull, gem, co-gem)-free graphs, and parity graphs, using a technique known as veretex incremental and based on results from Cicerone and Di Stefano [9] and de Montgolfier and Rao [13]. We also derive upper bounds for $(6,2)$-chordal bipartite graphs and (C_5, bull, gem, co-gem)-free graphs using analytic combinatorics, based on results from Nakano *et al.* [25].

4.1. Vertex incremental

We begin by reviewing some concepts and notation presented in Section 2. Considering a graph G and its vertex set $V(G)$, for a set of vertices $V \subseteq V(G)$, the *neighborhood* of V, denoted by $N(V)$, is defined to be the set of vertices $V(G) \setminus V$ that are adjacent to at least one vertex in V. For a vertex $v \in V(G)$, the *neighborhood* of v, denoted by $N(v)$, is defined similarly to be the set of vertices $V(G) \setminus \{v\}$ that are adjacent to v. Also, for a set of vertices $V \subseteq V(G)$, the *closed neighborhood* of V, denoted by $N[V]$, is defined to be $N(V) \cup V$, and similarly, for a vertex $v \in V(G)$, the *closed neighborhood* of v, denoted by $N[v]$, is defined to be $N(v) \cup \{v\}$.

We now consider *vertex incremental* characterizations. A *vertex incremental* characterization of a class of graphs $\mathcal{A}$ is the necessary and sufficient conditions under which adding a vertex v to a graph from $\mathcal{A}$ would produce another graph in $\mathcal{A}$. The characterizations are often written as a set of operations, which when repeatedly applied to a starting graph of one vertex (either in a specified order or in any order), would produce all graphs in $\mathcal{A}$ (considering all possible combinations of applying these operations). Moreover, the characterizations are derived from various types of tree decompositions, including modular decomposition, split decomposition, and bi-join decomposition.

For the purposes of this paper, we focus on the vertex incremental characterizations presented by Cicerone and Di Stefano [9] and de Montgolfier and Rao [13], which fully describe $(6,2)$-chordal bipartite graphs, (C_5, bull, gem, co-gem)-free graphs, and parity graphs. All of these are derived from the split decomposition [9], except for (C_5, bull, gem, co-gem)-free graphs, which is derived from the bi-join decomposition [13]. Their characterizations involve the following operations:

Pick a vertex x in G, add a new vertex x' to G, and

- *Pendant*: add edge x–x'.

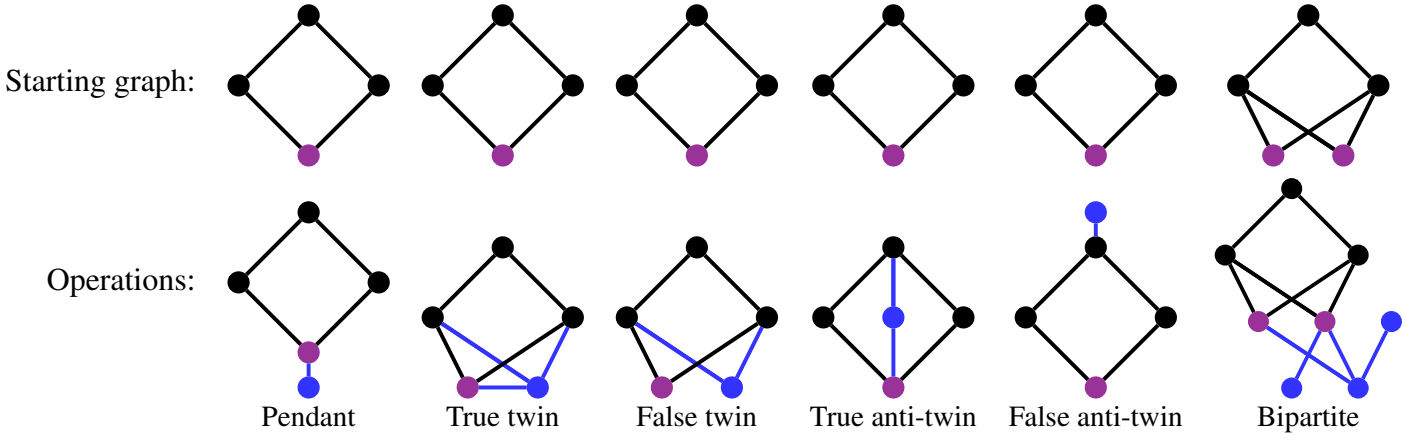

Table 2. Vertex incremental operations; the vertex/vertices in purple is the vertex x/vertices $\{x_1,\ldots,x_n\}$ (the vertex/vertices picked), and the vertices/edges in blue are the vertices/edges added due to the operation.

Graph	Pendant	True twin	False twin	True anti-twin	False anti-twin	Bipartite
3-leaf[19]	1	2				
Cograph[25]		X	X			
Distance-hereditary [25]	X	X	X			
(C_5, bull, gem, co-gem)-free[13]		X	X	X	X	
(6, 2)-chordal bipartite[8]	X		X			
Parity [9]		X	X			X

Table 3. Vertex incremental descriptions of certain graph classes; the numbers denote operations that must be performed in a certain order (e.g., 3-leaf power graphs are characterized by some number of pendant operations followed by some number of true twin operations), while the Xs denote operations that can be performed in any order.

- *True twin*: add edge x–x', and for all $y \in N(x)$, add edge x'–y.
- *False twin*: for all $y \in N(x)$, add edge x'–y.
- *True anti-twin*: add edge x–x', and for all $y \notin N(x)$, add edge x'–y.
- *False anti-twin*: for all $y \notin N(x)$, add edge x'–y.

Pick a set of false twins $X = \{x_1, x_2, \ldots, x_n\}$ in G (where x_i,$x_j \in X$ if $N(x_i) = N(x_j)$ and $(x_i, x_j) \notin E(G)$) and

- *Bipartite*: add a bipartite graph $B = V_1 \cup V_2$ to G (where V_1 and V_2 is a bipartition of B such that every edge in B connects a vertex from V_1 to a vertex from V_2), identifying certain vertices in V_1 with the false twins.

Table 2 contains examples of each of the operations. Table 3 describes which operations characterize which graph classes. Note that in [9], Cicerone and Di Stefano incorrectly claim that the vertex incremental characterization for $(6,2)$-chordal bipartite graphs is for $(6,2)$-chordal graphs (while in [8], Cicerone and Di Stefano correctly attribute the characterization to $(6,2)$-chordal bipartite graphs). We ascertained this by comparing our enumeration derived from the vertex incremental characterization to our enumeration of distance-hereditary bipartite graphs (which is equivalent to $(6,2)$-chordal bipartite graphs).

4.2. Results: Enumerations

Lumbroso [28] wrote code to generate and enumerate 3-leaf power graphs and distance-hereditary graphs based on their vertex incremental characterizations. We extend upon his code to generate and enumerate $(6,2)$-chordal bipartite graphs, (C_5, bull, gem, co-gem)-free graphs, and parity graphs. The code and generated graphs can be found in Appendix B. Table 4 contains all of the generated enumerations. Note that Hougardy [23] has previously generated the enumeration for parity graphs, and our derivation matches this enumeration.

n	1	2	3	4	5	6	7	8	9	10
Connected $(6,2)$-chordal bipartite	1	1	1	3	5	15	37	119	365	1258
$(6,2)$-chordal bipartite	1	2	3	7	13	33	79	229	671	
Connected (C_5, bull, gem, co-gem)-free	1	1	2	6	18	73	308	1484		
(C_5, bull, gem, co-gem)-free	1	2	4	11	31	114	454	2078		
Connected parity	1	1	2	6	18	75	318	1599	8439	
Parity	1	2	4	11	31	116	466	2207	11258	

Table 4. Enumeration of various graph classes, derived from vertex incremental characterizations.

4.3. Results: Vertex incremental trees

Nakano *et al.* [25] obtained upper bounds on the number of distance-hereditary graphs by creating trees from the vertex incremental characterization, labeling the internal nodes of the trees with the type of operation used to construct each graph. They also normalized these trees and evaluated the upper bound space complexity of such a tree to obtain an upper bound on the number of distance-hereditary graphs. We make an argument parallel to this for $(6,2)$-chordal bipartite graphs and (C_5, bull, gem, co-gem)-free graphs.

4.3.1. (6,2)-chordal bipartite graphs We begin with some concepts and lemmas that Nakano *et al.* [25] introduce. Specifically, given a graph G such that $V(G) \geq 3$, they define the following vertex sets:

$$\mathcal{S} = \{S \mid x,y \in S \text{ if } N[x] = N[y] \text{ and } |S| \geq 2\}$$
$$\mathcal{W} = \{W \mid x,y \in W \text{ if } N(x) = N(y), |W| \geq 2, \text{ and } |N(x)| = |N(y)| \geq 1\}$$
$$\mathcal{P} = \{P \mid x,y \in P \text{ if } x \text{ is a pendant vertex and } y \text{ is its neck}\}.$$

Note that $\mathcal{S}$ corresponds with vertices generated by the true twin operation, also known as *strong twins*, $\mathcal{W}$ corresponds with vertices generated by the weak twin operation, also known as *weak twins*, and $\mathcal{P}$ corresponds with vertices generated by the pendant operation.

Nakano *et al.* [25] also prove the following three lemmas:

Lemma 4.1. *For each P in $\mathcal{P}$, P contains exactly one neck with associated pendants.*

Lemma 4.2. *Let v be any vertex in a distance-hereditary graph G where $|V(G)| \geq 3$. Then, v belongs to either*

1. *exactly one set in $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$, or*
2. *no set in the families.*

Lemma 4.3. *For any distance-hereditary graph G, $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \emptyset$.*

Note that $(6,2)$-chordal bipartite graphs are a subset of distance-hereditary graphs [15], so all of the above lemmas hold. Moreover, under the assumption that $V(G) \geq 3$, we have $\mathcal{S} = \emptyset$ by the vertex incremental characterization.

Construction of a C-tree. Nakano *et al.* [25] use these lemmas to construct a *DH-tree*, to describe distance hereditary graphs. Here, we construct a *C-tree* in a similar manner. Note that a C-tree is rooted, ordered, and each internal node of the C-tree is labeled.

We begin with three basic cases for a $(6,2)$-chordal bipartite graph G:

1. G is a single vertex: the C-tree is defined to be a single root with no label
2. G is a star of size $n = 2$: the C-tree is defined to be a single root with label p and 2 leaves with no labels. The leaves can be in any order (note that Lemma 4.1 does not apply here, since $n < 3$).
3. G is a star of size $n \geq 3$: the C-tree is a tree with a single root with label p and n leaves with no labels. Note that the tree is ordered, so the leftmost child of the node with label p indicates the neck; in this case, the leftmost child corresponds to the center of the star, and the other children correspond to the extremities.

Note that the number of leaves of the tree is the number of vertices in G, and this fact remains invariant for all C-trees. Now, we consider the general case. Given a $(6,2)$-chordal bipartite graph G with $|V(G)| = n \geq 3$, we define its C-tree $\mathcal{T}$. Assume that G is not a star with n vertices. We start with n leaves of $\mathcal{T}$ which are initially independent, and we identify each leaf with a vertex $v \in G$. More formally, we construct an injection $\phi : V(G) \to V(\mathcal{T})$ which maps $v \in V(G)$ to a unique leaf of $\mathcal{T}$.

By Lemmas 4.2 and 4.3, we can group the vertices of G into three families, $\mathcal{S}$, $\mathcal{W}$, and $\mathcal{P}$ with $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \emptyset$. We place the vertices that do not belong in one of these three families into a set N. Since by the vertex incremental characterization of G there is no true twin operation, $\mathcal{S} = \emptyset$. Thus, we group the vertices of G into two disjoint families, $\mathcal{W}$ and $\mathcal{P}$, and one set disjoint from all elements in the two families, N.

For any set $W \in \mathcal{W}$, we consider the leaves associated with W in $\mathcal{T}$ $(\{\phi(w) \mid w \in W\})$ and make a common parent of these leaves with label w, where the leaves are placed in any arbitrary order. Similarly, for any set $P \in \mathcal{P}$, we consider the leaves associated with P in $\mathcal{T}$ $(\{\phi(p) \mid p \in P\})$ and make a common parent of these leaves with label p. By Lemma 4.1, P has a unique neck v, so we make the associated leaf $\phi(v)$ the leftmost child of the parent. The rest of the leaves are placed in an arbitrary order to the right of v.

Next, we construct a set of vertices $V \subset V(G)$. For every vertex set $W \in \mathcal{W}$, we choose any vertex $w \in W$ and put it in V. Similarly, for every vertex set $P \in \mathcal{P}$, since by Lemma 4.1 P has a unique neck v, we put v in V. For every vertex $u \in N$, we place u in V. Note that the subgraph H of G induced by V is still a $(6,2)$-chordal bipartite graph. This arises directly from the definition; any subgraph of a bipartite graph is bipartite, since if we consider a bipartition of G, say G_1 and G_2, such that every edge connects a vertex from G_1 to a vertex in G_2, we can consider the bipartition of H given by $H_1 = \{g \in H \mid g \in G_1\}$ and $H_2 = \{g \in H \mid g \in G_2\}$ and necessarily, every edge in H connects a vertex from H_1 to a vertex in H_2. Moreover, if there exists a cycle of length at least 5 in H, the cycle must have at least 2 chords in G, say edges $e_1 = (x_1, y_1)$ and $e_2 = (x_2, y_2)$ where $x_1, y_1, x_2, y_2 \in V = V(H)$. Since V induces H, necessarily $e_1, e_2 \in H$, so the cycle has at least 2 chords in H. Thus, H is a $(6,2)$-chordal bipartite graph.

We now construct an injection $\phi' : V(H) \to V(\mathcal{T})$ which maps each vertex $v \in V(H) = V$ to the parent of $\phi(v)$ if it exists, and otherwise to $\phi(v)$. We can now repeat all of the above steps, except

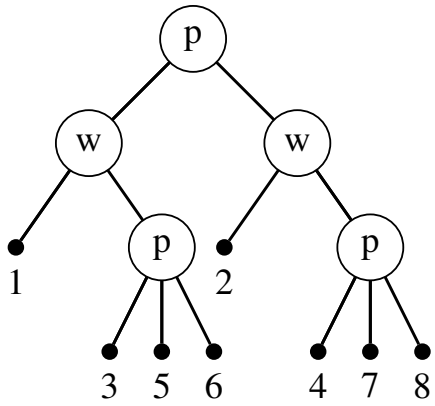

Figure 14. Example of a C-tree.

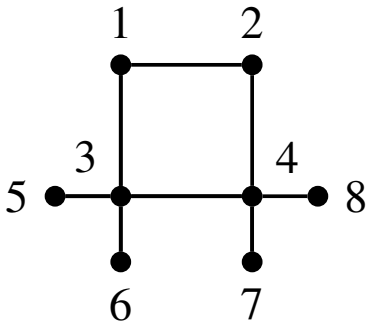

Figure 15. $(6,2)$-chordal bipartite graph for the C-tree in Figure 14.

using H and ϕ' instead of G and ϕ. We stop when H matches one of the basic cases mentioned initially, which we use to complete the tree $\mathcal{T}$. See Figure 14 for an example of a C-tree and Figure 15 for the corresponding $(6,2)$-chordal bipartite graph.

Theorem 4.4. *We can construct the C-tree $\mathcal{T}$ derived from graph G if and only if G is a $(6,2)$-chordal graph.*

Proof. From Lemmas 4.2 and 4.3, and the above constructive description of the C-tree, it is clear that given a $(6,2)$-chordal graph G, we can construct a corresponding C-tree $\mathcal{T}$. Thus, we only need to prove that given a C-tree $\mathcal{T}$, we can construct the corresponding $(6,2)$-chordal graph. Consider any C-tree $\mathcal{T}$ and a level order tree traversal $\{t_1,\ldots,t_m\}$ where $t_i \in V(\mathcal{T})$ and $m = |V(\mathcal{T})|$. Let n be the number of leaves in $\mathcal{T}$. We begin constructing graph G, where G starts with n vertices and no edges. Define a surjection $\rho : V(\mathcal{T}) \to V(G)$ which maps each leaf in $\mathcal{T}$ to a unique vertex in $V(G)$ and each internal node in $\mathcal{T}$ to $\rho(\ell)$ where ℓ is the leftmost child of that internal node.

Now, consider t_1 and its children $\{c_1,\ldots,c_s\}$ where c_1 is its leftmost child. We define the following operations, corresponding to the label of t_1:

- p: add edge $(\rho(c_1),\rho(c_i))$ to G for all $1 < i \leq s$.
- w: add edge $(c,\rho(c_i))$ to G for all $c \in N(\rho(c_1))$ and $1 < i \leq s$.

Apply the operation corresponding with the label of t_1 to G, and then repeat this process for all t_i such that t_i is not a leaf, in the level order given.

Since we have essentially repeatedly applied the vertex incremental operations for $(6,2)$-chordal bipartite graphs in this process, G is a $(6,2)$-chordal bipartite graph. It is also clear that G is the graph corresponding to the C-tree $\mathcal{T}$ by our earlier constructive description of the C-tree, by the vertex incremental operations used to create G. □

Corollary 4.5. *For any $(6,2)$-chordal bipartite graph G, the C-tree derived from G is uniquely determined.*

Proof. This follows from the fact that the families $\mathcal{W}$ and $\mathcal{P}$ are uniquely determined. □

Lemma 4.6. *Let G be a $(6,2)$-chordal bipartite graph that contains at least 2 vertices and let $\mathcal{T}$ be the corresponding C-tree. Then, we have the following:*

1. *Each internal node has at least two children.*
2. *Each inner node has a label, either p or w, and each leaf has no label.*
3. *The label of the root is p.*

Proof. All three statements follow immediately from the construction of the C-tree. □

Note that C-trees may be redundant, so we introduce here some rules for a normalized C-tree, which match the rules introduced by Nakano *et al.* [25] for normalized DH-trees. Specifically, we have the following two rules for a C-tree $\mathcal{T}$, which preserve the corresponding $(6,2)$-chordal bipartite graph G:

1. Given a node labeled p, say node p_1, with a non-leftmost child labeled w, say node w_1, which in turn has children $\{c_1,\ldots,c_n\}$, we can remove node w_1 and make p_1 the parent of the children $\{c_1,\ldots,c_n\}$. Thus, in a normalized C-tree, any non-leftmost child of a node with label p is not labeled w. See Figure 16 for an example.
2. Given a node labeled w, say node w_1, with a child labeled w, say node w_2, which in turn has children $\{c_1,\ldots,c_n\}$, we can remove node w_2 and make w_1 the parent of the children $\{c_1,\ldots,c_n\}$. Thus, in a normalized C-tree, any child of a node with label w is not labeled w. See Figure 17 for an example.

Based on Theorem 4.4, Corollary 4.5, and Lemma 4.6, we obtain the following grammar for C-trees $\mathcal{C}_\mathcal{T}$, which is also an upper bound grammar for connected $(6,2)$-chordal bipartite graphs:

$$\begin{aligned}\mathcal{C}_\mathcal{T} &= \mathcal{P}\\ \mathcal{P} &= (\mathcal{P}+\mathcal{W}+\mathcal{Z})\times \textsc{Set}_{\geq 1}(\mathcal{P}+\mathcal{Z})\\ \mathcal{W} &= \textsc{Set}_{\geq 2}(\mathcal{P}+\mathcal{Z})\end{aligned}$$

The following list explains the symbols for each class:

- $\mathcal{P}$: An internal node with label p
- $\mathcal{W}$: An internal node with label w

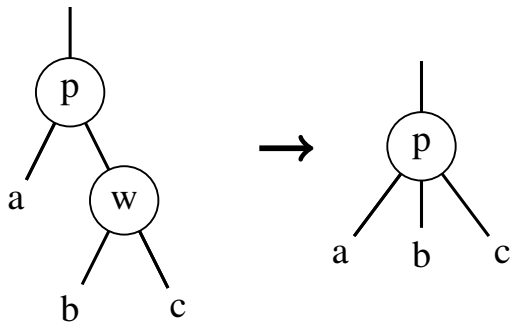

Figure 16**.** Normalizing a node labeled p with a non-leftmost child labeled w.

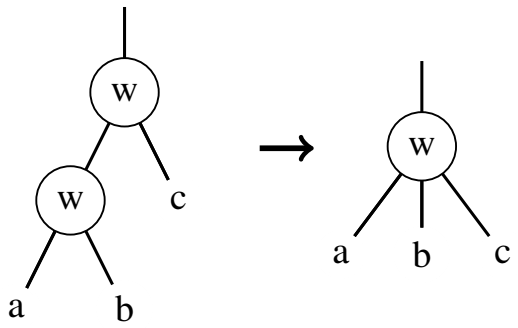

Figure 17**.** Normalizing a node labeled w with a child labeled w.

Note that the grammar for a multiply rooted C-tree $\mathcal{C}_{\mathcal{RT}}$, which is also an upper bound grammar for $(6,2)$-chordal bipartite graphs, uses the same symbols as above and is given by $\mathcal{C}_{\mathcal{RT}} = \textsc{Set}(\mathcal{Z} + \mathcal{P})$.

Proof. These follow directly from the construction of a normalized C-tree and the requirements in Lemma 4.6. Nodes labeled with p cannot have a non-leftmost child labeled w, and nodes labeled with w cannot have a child labeled w. □

Using Maple with the `combstruct` package, we receive the following enumeration:

Corollary 4.7. *The first few terms of the OGF of the class of C-trees, which is also an upper bound OGF of the class of connected $(6,2)$-chordal bipartite graphs, is*

$$\mathcal{C}_{\mathcal{T}}(z) = z + z^2 + 4z^3 + 16z^4 + 70z^5 + 322z^6 + 1556z^7 + 7757z^8 + 39698z^9 + 207217z^10 + \ldots.$$

Note that in Section 4.2 we obtain an exact enumeration of connected $(6,2)$-chordal bipartite graphs $\mathcal{C}_{\mathcal{C}}$, specifically

$$\mathcal{C}_{\mathcal{C}}(z) = z + z^2 + 1z^3 + 3z^4 + 5z^5 + 15z^6 + 37z^7 + 119z^8 + 365z^9 + 1258z^10 + \ldots,$$

so the first few terms of the upper bound OGF of connected $(6,2)$-chordal bipartite graphs appear to be correct.

Corollary 4.8. *The first few terms of the OGF of the class of multiply rooted C-trees, which is also an upper bound OGF of the class of $(6,2)$-chordal bipartite graphs, is*

$$\mathcal{C}_{\mathcal{RT}}(z) = z + 2z^2 + 6z^3 + 23z^4 + 97z^5 + 446z^6 + 2140z^7 + 10662z^8 + 54482z^9 + \ldots.$$

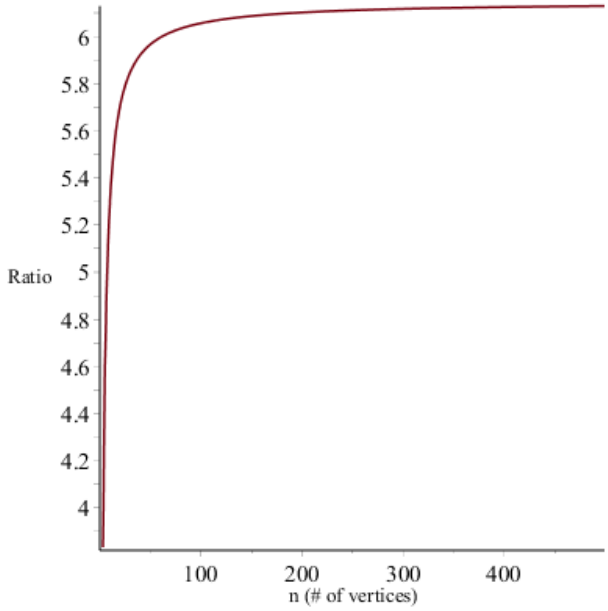

Figure 18**.** Plot of the ratio between consecutive terms of the generating function for $(6,2)$-chordal bipartite graphs.

Note that in Section 4.2 we obtain an exact enumeration of $(6,2)$-chordal bipartite graphs $\mathcal{C}$, specifically

$$\mathcal{C}(z) = z + 2z^2 + 3z^3 + 7z^4 + 13z^5 + 33z^6 + 79z^7 + 229z^8 + 671z^9 + \ldots,$$

so the first few terms of the upper bound OGF of $(6,2)$-chordal bipartite graphs appear to be correct.

Corollary 4.9. *Analytically, an upper bound on the number of $(6,2)$-chordal bipartite graphs is $O(7^n)$.*

Proof. Using Maple, we plotted the ratio between the first 500 consecutive terms of the generating function obtained in Corollary 4.8. The plot is shown in Figure 18, and it indicates an asymptote below 7. As such, we have an analytically obtained upper bound of $O(7^n)$. □

4.3.2. (C_5 bull, gem, co-gem)-free graphs We begin with some of the same constructions used in Section 4.3.1. Given a graph G such that $V(G) \geq 3$, we define four vertex sets. For clarity, we rename the families $\mathcal{S}$ and $\mathcal{W}$, and introduce two new families as follows:

$$\begin{aligned}
\mathcal{ST} &= \{S \mid x,y \in S \text{ if } N[x] = N[y] \text{ and } |S| \geq 2\} \\
\mathcal{WT} &= \{W \mid x,y \in W \text{ if } N(x) = N(y) \text{ and } |W| \geq 2\} \\
\mathcal{SA} &= \{\{W_1,W_2\} \mid x \in W_1 \text{ and } y \in W_2 \text{ if } N(x) = V(G) \setminus N(y) \text{ and } |W_1|,|W_2| \geq 1\} \\
\mathcal{WA} &= \{\{S_1,S_2\} \mid x \in S_1 \text{ and } y \in S_2 \text{ if } N[x] = V(G) \setminus N[y] \text{ and } |S_1|,|S_2| \geq 1\}
\end{aligned}$$

Note that $\mathcal{ST}$ corresponds with vertices generated by the true twin operation, $\mathcal{WT}$ corresponds with vertices generated by the false twin operation, $\mathcal{SA}$ corresponds with vertices generated by the true anti-twin operation, and $\mathcal{WA}$ corresponds with vertices generated by the false anti-twin operation. Also note that by construction, for any $\{W_1,W_2\} \in \mathcal{SA}$, we must have $W_1,W_2 \in \mathcal{WT}$ if $|W_1|,|W_2| \geq 2$. Similarly, for any $\{S_1,S_2\} \in \mathcal{WA}$, we must have $S_1,S_2 \in \mathcal{ST}$ if $|S_1|,|S_2| \geq 2$.

We now prove the following observations:

Lemma 4.10. *Let v be any vertex in a (C_5, bull, gem, co-gem)-free graph G where $|V(G)| \geq 3$. Then v belongs to*

1. *exactly one set in $\mathcal{ST} \cup \mathcal{WT} \cup \left(\bigcup_{W \in \mathcal{SA}} W\right) \cup \left(\bigcup_{S \in \mathcal{WA}} S\right)$, or*
2. *exactly two sets, one in $\mathcal{ST}$ and one in an element of $\mathcal{WA}$, or*
3. *exactly two sets, one in $\mathcal{WT}$ and one in an element of $\mathcal{SA}$, or*
4. *no set in the families.*

Proof. The proof is very similar to that of Lemma 4.2, which can be found in Nakano *et al.*'s work [25]. The full proof is in Appendix A.1. □

Lemma 4.11. *For any (C_5, bull, gem, co-gem)-free graph G,*

$$\mathcal{ST} \cup \mathcal{WT} \cup \Big(\bigcup_{W \in \mathcal{SA}} W\Big) \cup \Big(\bigcup_{S \in \mathcal{WA}} S\Big) \neq \emptyset.$$

Proof. This follows directly from the vertex incremental characterization of (C_5, bull, gem, co-gem)-free graphs given in Table 3. Necessarily, such a graph G can be generated by a sequence of operations, and based on the last operation in that sequence, at least one of the sets in $\mathcal{ST}$, $\mathcal{WT}$, $\bigcup_{W \in \mathcal{SA}} W$, or $\bigcup_{S \in \mathcal{WA}} S$ must be nonempty. □

Construction of a B-tree. Now, note that the construction of a *B-tree* is similar to the construction of the DH-tree by Nakano *et al.* [25] and to the construction of the C-tree in Section 4.3.1. As such, we defer a full construction to Appendix A.1. Note that a B-tree is rooted, ordered, and each internal node of the B-tree is labeled with *st*, *wt*, *sa*, or *wa*.

Theorem 4.12. *We can construct the B-tree $\mathcal{T}$ derived from graph G if and only if G is a (C_5, bull, gem, co-gem)-free graph.*

Proof. The proof is very similar to that of Theorem 4.4. As such, we defer a full proof to Appendix A.1. □

Corollary 4.13. *For any (C_5, bull, gem, co-gem)-free graph G, the B-tree derived from G is uniquely determined.*

Proof. This follows from the fact that the families $\mathcal{ST}$, $\mathcal{WT}$, $\mathcal{SA}$, and $\mathcal{WA}$ are uniquely determined. □

Lemma 4.14. *Let G be a (C_5, bull, gem, co-gem)-free graph that contains at least 2 vertices and let $\mathcal{T}$ be the corresponding B-tree. Then, we have the following:*

1. *Each internal node has at least two children.*
2. *Each internal node has label st, wt, sa, or wa, and each leaf is unlabeled.*
3. *Each internal node labeled sa or wa has exactly two children.*
4. *The label of the root is st.*

Proof. All of these statements follow immediately from the construction of the B-tree. □

Note that B-trees may be redundant, so we introduce here some rules for a normalized B-tree, which match the rules introduced by Nakano *et al.* [25] for normalized DH-trees. Specifically, we have the following four rules for a B-tree $\mathcal{T}$, which preserve the corresponding (C_5, bull, gem, co-gem)-free graph G:

1. Given a node labeled *st*, say node s_1, with a child labeled *st*, say node s_2, which in turn has children $\{c_1, \ldots, c_2\}$, we can remove node s_2 and make s_1 the parent of the children $\{c_1, \ldots, c_2\}$. Thus, in a normalized B-tree, any child of a node with label *st* is not labeled *st*. See Figure 19 for an example.
2. Given a node labeled *wt*, say node w_1, with a child labeled *wt*, say node w_2, which in turn has children $\{c_1, \ldots, c_2\}$, we can remove node w_2 and make w_1 the parent of the children $\{c_1, \ldots, c_2\}$. Thus, in a normalized B-tree, any child of a node with label *wt* is not labeled *wt*. See Figure 20 for an example.

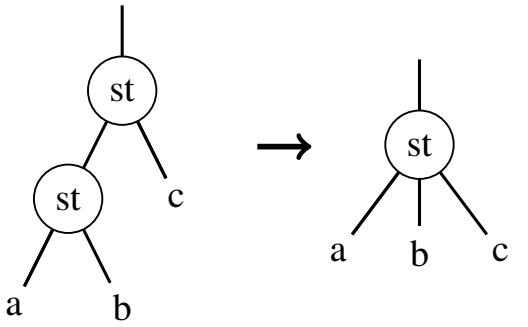

Figure 19**.** Normalizing a node labeled *st* with a child labeled *st*.

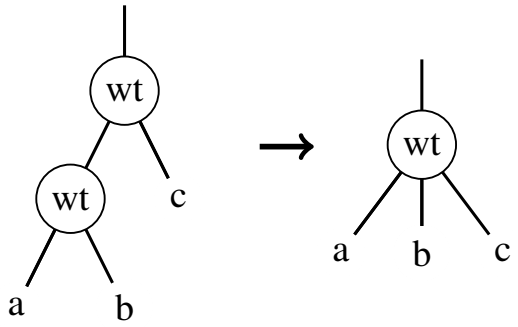

Figure 20**.** Normalizing a node labeled *wt* with a child labeled *wt*.

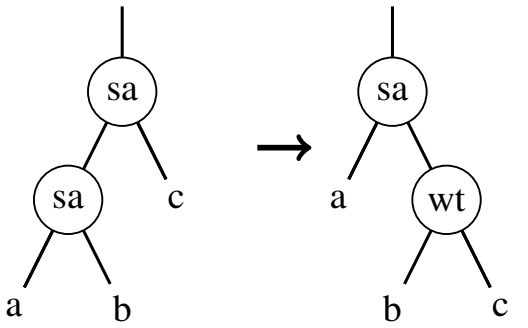

Figure 21**.** Normalizing a node labeled *sa* with a child labeled *sa*.

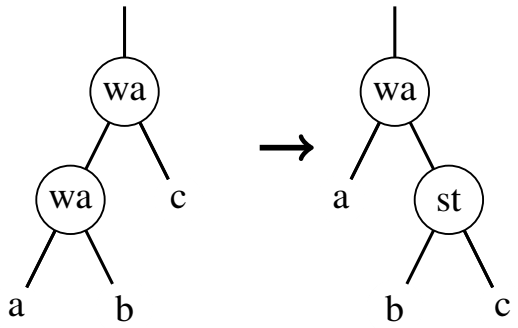

Figure 22**.** Normalizing a node labeled *wa* with a child labeled *wa*.

3. Consider a node labeled *sa* with two children, nodes w_1 and w_2, where node w_1 has label *sa*. Let node w_1 have two children, nodes c_1 and c_2, where node c_1 is the left child. Then, we can replace node w_1 with its left child c_1 and replace node w_2 with an internal node labeled *wt*, with two children c_2 and w_2. Thus, in a normalized B-tree, any child of a node with label *sa* is not labeled *sa*. See Figure 21 for an example.
4. Consider a node labeled *wa* with two children, nodes s_1 and s_2, where node s_1 has label *wa*. Let node s_1 have two children, nodes c_1 and c_2, where node c_1 is the left child. Then, we can replace node s_1 with its left child c_1 and replace node s_2 with an internal node labeled *st*, with two children c_2 and s_2. Thus, in a normalized B-tree, any child of a node with label *wa* is not labeled *wa*. See Figure 22 for an example.

Based on Theorem 4.12, Corollary 4.13, and Lemma 4.14, we obtain the following grammar for B-trees $\mathcal{B}_{\mathcal{T}}$, which is also an upper bound grammar for connected (C_5, bull, gem, co-gem)-free

graphs:

$$\begin{aligned}
\mathcal{B}_{\mathcal{T}} &= \mathcal{ST} \\
\mathcal{ST} &= \textsc{Set}_{\geq 2}\left(\mathcal{WT} + \mathcal{SA} + \mathcal{WA} + \mathcal{Z}\right) \\
\mathcal{WT} &= \textsc{Set}_{\geq 2}\left(\mathcal{ST} + \mathcal{SA} + \mathcal{WA} + \mathcal{Z}\right) \\
\mathcal{SA} &= \textsc{Seq}_{=2}\left(\mathcal{ST} + \mathcal{WT} + \mathcal{WA} + \mathcal{Z}\right) \\
\mathcal{WA} &= \textsc{Seq}_{=2}\left(\mathcal{ST} + \mathcal{WT} + \mathcal{SA} + \mathcal{Z}\right)
\end{aligned}$$

The following list explains the symbols for each class:

- $\mathcal{ST}$: An internal node with label *st*
- $\mathcal{WT}$: An internal node with label *wt*
- $\mathcal{SA}$: An internal node with label *sa*
- $\mathcal{WA}$: An internal node with label *wa*

Note that the grammar for a multiply rooted B-tree $\mathcal{B}_{\mathcal{RT}}$, which is also an upper bound grammar for (C_5, bull, gem, co-gem)-free graphs, uses the same symbols as above and is given by $\mathcal{B}_{\mathcal{RT}} = \textsc{Set}\left(\mathcal{Z} + \mathcal{ST}\right)$.

Proof. These follow directly from the construction of a normalized B-tree and the requirements in Lemma 4.14. Nodes labeled $\mathcal{ST}$ and $\mathcal{WT}$ cannot have children labeled $\mathcal{ST}$ and $\mathcal{WT}$ respectively. Similarly, nodes labeled $\mathcal{SA}$ and $\mathcal{WA}$ cannot have children labeled $\mathcal{SA}$ and $\mathcal{WA}$. □

Using Maple with the `combstruct` package, we receive the following enumeration:

Corollary 4.15. *The first few terms of the OGF of the class of B-trees, which is also an upper bound OGF of the class of connected (C_5, bull, gem, co-gem)-free graphs, is*

$$\mathcal{B}_{\mathcal{T}}(z) = z + z^2 + 4z^3 + 26z^4 + 174z^5 + 1318z^6 + 10326z^7 + 84195z^8 + \ldots.$$

Note that in Section 4.2 we obtain an exact enumeration of connected (C_5, bull, gem, co-gem)-free graphs $\mathcal{B}_{\mathcal{C}}$, specifically

$$\mathcal{B}_{\mathcal{C}}(z) = z + z^2 + 2z^3 + 6z^4 + 18z^5 + 73z^6 + 308z^7 + 1484z^8 + \ldots,$$

so the first few terms of the upper bound OGF of (C_5, bull, gem, co-gem)-free graphs appear to be correct.

Corollary 4.16. *The first few terms of the OGF of the class of multiply rooted B-trees, which is also an upper bound OGF of the class of (C_5, bull, gem, co-gem)-free graphs, is*

$$\mathcal{B}_{\mathcal{RT}}(z) = z + 2z^2 + 6z^3 + 33z^4 + 211z^5 + 1566z^6 + 12174z^7 + 98771z^8 + \ldots.$$

Note that in Section 4.2 we obtain an exact enumeration of (C_5, bull, gem, co-gem)-free graphs $\mathcal{B}$, specifically

$$\mathcal{B}(z) = z + 2z^2 + 4z^3 + 11z^4 + 31z^5 + 114z^6 + 454z^7 + 2078z^8 + \ldots,$$

so the first few terms of the upper bound OGF of (C_5, bull, gem, co-gem)-free graphs appear to be correct.

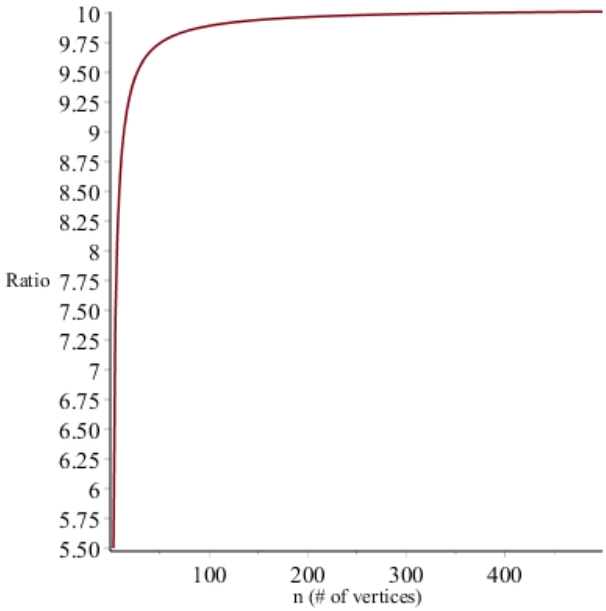

Figure 23**.** Plot of the ratio between consecutive terms of the generating function for (C_5, bull, gem, co-gem)-free graphs.

Corollary 4.17. *Analytically, an upper bound on the number of (C_5, bull, gem, co-gem)-free graphs is* $\mathrm{O}\left(11^n\right)$.

Proof. Using Maple, we plotted the ratio between the first 500 consecutive terms of the generating function obtained in Corollary 4.16. The plot is shown in Figure 23, and it indicates an asymptote below 11. As such, we have an analytically obtained upper bound of $\mathrm{O}\left(11^n\right)$. □

4.4. Next steps

It may be possible to obtain upper bounds on the number of parity graphs by extending the results of Nakano *et al.* [25], and in particular, obtain tighter bounds by using symbolic grammars. The main difficulty with parity graphs is the bipartite operation, which may be difficult to normalize in a tree.

Also, Nakano *et al.* [25] obtained an upper bound for the number of distance hereditary graphs by considering a compact encoding of the corresponding DH-tree. Such a compact encoding of the C-trees or B-trees considered in this section may provide a better bound than the analytic one obtained from constructing a grammar.

5. Conclusion

In this work, we have studied parity graphs through their split trees and have obtained an upper bound grammar. We have also studied $(6,2)$-chordal bipartite graphs, (C_5, bull, gem, co-gem)-free graphs, and parity graphs through their vertex incremental characterizations and have constructively obtained an enumeration for each graph class. Moreover, we have constructed upper bound grammars for $(6,2)$-chordal bipartite graphs and (C_5, bull, gem, co-gem)-free graphs, and have analytically obtained upper bounds of $\mathrm{O}\left(7^n\right)$ and $\mathrm{O}\left(11^n\right)$ respectively.

Note that we have not obtained an exact grammar for parity graphs due to the complexity of the generating function for bipartite graphs, as given by Hanlon [21]. It may be possible to obtain an

exact grammar if we can successfully integrate this generating function. Moreover, in the future, we aim to obtain a Boltzmann sampler for parity graphs by further extending upon Iriza's work [26].

We also aim to obtain a tree for parity graphs derived from its vertex incremental characterization, in a similar manner to our derivations of the C-tree and B-tree in Section 4.3 and to Nakano *et al.*'s [25] derivation of the DH-tree. In addition, Nakano *et al.* [25] considered a compact encoding of the DH-tree to obtain an upper bound on the number of distance hereditary graphs; it may be useful to consider a compact encoding of the C-tree and B-tree, and compare the upper bound obtained with the analytically derived upper bounds from Section 4.3.

6. Acknowledgments

I would like to thank my advisor Jérémie Lumbroso for his invaluable help and guidance in this work. I would also like to acknowledge the computing resources provided by the Princeton University Computer Science Department.

This paper represents my own work in accordance with University regulations. /s/ Jessica Shi.

References

[1] F. Bergeron, G. Labelle, and P. Leroux, *Combinatorial Species and Tree-like Structures*. Cambridge University Press, 1997.

[2] A. Bouchet, "Reducing prime graphs and recognizing circle graphs," *Combinatorica*, vol. 7, no. 3, pp. 243–254, 1987.

[3] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A Survey*. Philadelphia PA: Society for Industrial and Applied Mahtematics, 1999.

[4] C. Capelle, "Décomposition de graphes et permutations factorisantes," Ph.D. dissertation, Univ. de Montpellier II, 1997.

[5] G. Chapuy *et al.*, "A complete grammar for decomposing a family of graphs into 3-connected components," *The Electronic Journal of Combinatorics*, vol. 15, 2008.

[6] C. Chauve, Éric Fusy, and J. Lumbroso, "An enumeration of distance-hereditary and 3-leaf power graphs," 2014, Preprint presented at ICGT 2014.

[7] M. Chudnovsky *et al.*, "The strong perfect graph theorem," *Annals of Mathematics*, vol. 164, no. 1, pp. 51–229, 2006.

[8] S. Cicerone and G. D. Stefano, "Graph classes between parity and distance-hereditary graphs," *Discrete Applied Mathematics*, vol. 95, pp. 197–216, 1999.

[9] S. Cicerone and G. D. Stefano, "On the extension of bipartite to parity graphs," *Discrete Applied Mathematics*, vol. 95, pp. 181–195, 1999.

[10] C. Crespelle and C. Paul, "Fully dynamic algorithm for recognition and modular decomposition of permuation graphs," in *Graph-theoretic concepts in computer science*, D. Kratsch, Ed. Springer Berlin Heidelberg, 2005, pp. 38–48.

[11] W. H. Cunningham, "Decomposition of directed graphs," *SIAM Journal on Algebraic and Discrete Methods*, vol. 3, no. 2, pp. 214–228, 1982.

[12] E. Dahlhaus, "Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition," *Journal of Algorithms*, vol. 36, pp. 205–240, 2000.

[13] F. de Montgolfier and M. Rao, "The bi-join decomposition," *Electronic Notes in Discrete Mathematics*, vol. 22, pp. 173–177, 2005.

[14] F. de Montgolfier and M. Rao, "Bipartitive families and the bi-join decomposition," 2005, submitted to special issue of Discrete Mathematics devoted to ICGT05.

[15] de Ridder, H. N. *et al.*, "Information System on Graph Classes and their Inclusions (ISGCI)," http://www.graphclasses.org/classes/gc_78.html, 2015, (6,2)-chordal bipartite graph entry.

[16] P. Flajolet and R. Sedgewick, *Analytic Combinatorics*. Cambridge University Press, 2009.

[17] C. P. Gabor, K. J. Supowit, and W.-L. Hsu, "Recognizing circle graphs in polynomial time," *Journal of the Association for Computing Machinery*, vol. 36, no. 3, pp. 435–473, 1989.

[18] T. Gallai, "Transitiv orientierbare graphen," *Acta Mathematica Acad. Sci. Hungar.*, vol. 18, pp. 25–66, 1967.

[19] E. Gioan and C. Paul, "Split decomposition and graph-labelled trees: Characterizations and fully dynamic algorithms for totally decomposable graphs," *Discrete Applied Mathematics*, vol. 160, no. 6, pp. 708–733, 2012.

[20] M. Habib and C. Paul, "A simple linear time algorithm for cograph recognition," *Discrete Applied Mathematics*, vol. 145, no. 2, pp. 183–197, 2005.

[21] P. Hanlon, "The enumeration of bipartite graphs," *Discrete Mathematics*, vol. 28, pp. 49–57, 1979.

[22] P. Hell, R. Shamir, and R. Sharan, "A fully dynamic algorithm for recognizing and representing proper interval graphs," in *Algorithms — ESA '99*, J. Nešetřil, Ed. Springer Berlin Heidelberg, 1999, pp. 527–539.

[23] S. Hougardy, "The on-line encyclopedia of integer sequences," 2006, sequence A123443.

[24] W.-L. Hsu and T.-H. Ma, "Substitution decomposition on chordal graphs and applications," in *ISA'91 Algorithms*, W.-L. Hsu and R. C. T. Lee, Eds. Springer Berlin Heidelberg, 1991, pp. 52–60.

[25] S. ichi Nakano, R. Uehara, and T. Uno, "A new approach to graph recognition and applications to distance-hereditary graphs," *Journal of Computer Science and Technology*, vol. 24, no. 3, pp. 517–533, 2009.

[26] A. Iriza, "Enumeration and random generation of unlabeled classes of graphs: A practical study of cycle pointing and the dissymmetry theorem," Ph.D. dissertation, Princeton University, 2015.

[27] L. Lovász, "A characterization of perfect graphs," *Journal of Combinatorial Theory Series B*, vol. 13, pp. 95–98, 1972.

[28] J. Lumbroso, "Graphs by tree decomposition: Vertex incremental," unpublished code.

[29] T. H. Ma and J. Spinrad, "An $o(n^2)$ algorithm for undirected split decomposition," *Journal of Algorithms*, vol. 16, no. 1, pp. 145–160, 1994.

[30] R. M. McConnell and J. P. Spinrad, "Modular decomposition and transitive orientation," *Discrete Mathematics*, vol. 201, pp. 189–241, 1999.

[31] R. H. Möhring and F. J. Radermacher, "Substitution decomposition for discrete structures and connections with combinatorial optimization," *Annals of Discrete Mathematics*, vol. 19, pp. 257–356, 1984.

[32] S. D. Nikolopoulos, L. Palios, and C. Papadopoulos, "A fully dynamic algorithm for the recognition of p4-sparse graphs."
[33] R. Otter, "The number of trees," *Annals of Mathematics*, vol. 49, no. 3, pp. 583–599, 1948.
[34] C. Paul, "Split decomposition and circle graph recognition in quasi-linear time," Available at http://www.lirmm.fr/~paul/Talks/talk-09-agt.pdf.
[35] M. Rao, "Solving some np-complete problems using split decomposition," *Discrete Applied Mathematics*, vol. 156, pp. 2768—-2780, 2008.
[36] V. Ravelomanana and L. Thimonier, "Asymptotic enumeration of cographs," *Electronic Notes in Discrete Mathematics*, vol. 7, pp. 58–61, 2001.
[37] N. J. A. Sloane, "The on-line encyclopedia of integer sequences," 1995, sequence A005142.
[38] J. Spinrad, "Recognition of circle graphs," *Journal of Algorithms*, vol. 16, pp. 264–282, 1994.

Appendix A. Proofs

In this appendix section, we list some proofs that were omitted from the paper.

A.1. Results: Vertex incremental trees

Lemma 4.10 Let v be any vertex in a (C_5, bull, gem, co-gem)-free graph G where $|V(G)| \geq 3$. Then v belongs to

1. exactly one set in $\mathcal{ST} \cup \mathcal{WT} \cup \left(\bigcup_{W \in \mathcal{SA}} W\right) \cup \left(\bigcup_{S \in \mathcal{WA}} S\right)$, or
2. exactly two sets, one in $\mathcal{ST}$ and one in an element of $\mathcal{WA}$, or
3. exactly two sets, one in $\mathcal{WT}$ and one in an element of $\mathcal{SA}$, or
4. no set in the families.

Proof. If $v \in V(G)$ belongs to no set, then we are done. Thus, we consider the case where v belongs to at least one set. We now have four cases.

1. v is in a set S in $\mathcal{ST}$. We first assume that there is some distinct set $S' \in \mathcal{ST}$ such that $v \in S$. Then, for any $s \in S$ and $s' \in S'$, we have $N[s] = N[v] = N[s']$, so $S = S'$, which is a contradiction. So, for all $S' \in \mathcal{ST}$ where $S' \neq S$, we have $v \notin S'$.

   Now, we assume that there is a set $W \in \mathcal{WT}$ such that $v \in W$. By assumption, $N[v] = N[s]$ for some $s \in S$, and $N(v) = N(w)$ and $(v,w) \notin E(G)$ for some $w \in W$. Since $N[v] = N[s]$, we have $(s,v) \in E(G)$. Since $N(v) = N(w)$, we have $s \in N(v) = N(w)$, so $(s,w) \in E(G)$. Thus, $w \in N[s] = N[v]$, so we have $(v,w) \in E(G)$, which is a contradiction. So, for all $W \in \mathcal{WT}$, we have $v \notin W$.

   Finally, we assume that for some set $A = \{A_1, A_2\} \in \mathcal{SA}$, $v \in A_1$. By assumption, $N[v] = N[s]$ for some $s \in S$ and $N(v) = V(G) \setminus N(a)$ for some $a \in A_2$, which implies that $N[v] \cap N[a] = \{v,a\}$. Note that since $N[v] = N[s]$, we have $s \in N[s] = N[v]$. Since $N[v] \cap N[a] = \{v,a\}$, we have $(v,a) \in E(G)$, so $a \in N[v] = N[s]$. Thus, since $a \in N[s]$, we have $(s,a) \in E(G)$, so $s \in N[a]$. Since $N[v] \cap N[a] = \{v,a\}$, we have $s \notin N[v]$, which is a contradiction. So, for all $A = \{A_1, A_2\} \in \mathcal{SA}$, we have $v \notin A_1$.

2. v is in a set W in $\mathcal{WT}$. Assume that there is some distinct set $W' \in \mathcal{WT}$ such that $v \in W'$. Then, for any $w \in W$ and $w' \in W'$, we have $N(w) = N(v) = N(w')$, so $W = W'$, which is a contradiction. So, for all $W' \in' mathcalWT$ where $W' \neq W$, we have $v \notin W'$.

   By 1, for all $S \in \mathcal{ST}$, we have $v \notin S$.

   Now, assume that there is a set $A = \{A_1, A_2\} \in \mathcal{WA}$ such that $v \in A_1$. By assumption, $N(v) = N(w)$ for some $w \in W$ and $N[v] = V(G) \setminus N[a]$ for some $a \in A_2$. Note that since $N(v) = N(w)$, we have $w \notin N(w) = N(v)$, so $w \notin N[v]$. Since $N[v] = V(G) \setminus N[a]$, we have $w \in N[a]$, so $(w,a) \in E(G)$. Thus, $a \in N(w) = N(v)$, so $(v,a) \in E(G)$. Since $N[v] = V(G) \setminus N[a]$, and since $a \in N[a]$, we have $a \notin N[v]$ and $(v,a) \notin E(G)$, which is a contradiction. Thus, for all $A = \{A_1, A_2\} \in \mathcal{WA}$, we have $v \notin A_1$.

3. For some $W = \{W_1, W_2\} \in \mathcal{SA}$, v is in W_1. Note that it is impossible for v to be in W_2, since in that case $N(v) = V(G) \setminus N(v)$, but $v \notin N(v)$ and $v \in V(G) \setminus N(v) = N(v)$, which is a contradiction. So, consider the case where for some distinct $W' = \{W'_1, W'_2\} \in \mathcal{SA}$, $v \in W'_1$. Thus, for all $w \in W_2$, we have $N(w) = V(G) \setminus N(v)$, and for all $w' \in W'_2$, we have

$N(w') = V(G) \setminus N(v)$. Note that necessarily, $W_2 = \{w \in V(G) \mid N(w) = V(G) \setminus N(v)\}$, since any $v' \in W_1$ must be a false twin of v by definition, and similarly, $W_2' = \{w' \in V(G) \mid N(w') = V(G) \setminus N(v)\}$. Thus, $W_2 = W_2'$, and necessarily, $W_1 = W_1'$, so $W = W'$, which is a contradiction. So, for all $W' = \{W_1', W_2'\} \in \mathcal{SA}$ where $W' \neq W$, we have $v \notin W_1'$.

By 1, for all $S \in \mathcal{ST}$, we have $v \notin S$.

Now, assume that there is a set $S = \{S_1, S_2\} \in \mathcal{WA}$ such that $v \in S_1$. Thus, we have for some $w \in W_2$, $N(v) = V(G) \setminus N(w)$, which implies $(w,v) \in E(G)$, and for some $s \in S_2$, $N[v] = V(G) \setminus N[s]$, which implies $(s,v) \notin E(G)$. Since $(w,v) \in E(G)$, we have $w \in N[v]$, so $w \notin N[s]$ and $(w,s) \notin E(G)$. Since $(s,v) \notin E(G)$, we have $s \notin N(v)$, so $s \in N(w)$ and $(w,s) \in E(G)$, which is a contradiction. Thus, for all $S = \{S_1, S_2\} \in \mathcal{WA}$, we have $v \notin S_1$.

4. For some $S = \{S_1, S_2\} \in \mathcal{WA}$, v is in S_1. Note that it is impossible for v to be in S_2, since in that case $N[v] = V(G) \setminus N[v]$, but $v \in N[v]$ and $v \notin V(G) \setminus N[v] = N[v]$, which is a contradiction. So, consider the case where for some distinct $S' = \{S_1', S_2'\} \in \mathcal{WA}$, $v \in S_1'$. Thus, for all $s \in S_2$, we have $N[s] = V(G) \setminus N[v]$, and for all $s' \in S_2'$, we have $N[s'] = V(G) \setminus N[v]$. Note that necessarily, $S_2 = \{s \in V(G) \mid N[s] = V(G) \setminus N[v]\}$, since any $v' \in S_1$ must be a true twin of v by definition, and similarly, $S_2' = \{s' \in V(G) \mid N[s'] = V(G) \setminus N[v]\}$. Thus, $S_2 = S_2'$, and necessarily, $S_1 = S_1'$, so $S = S'$, which is a contradiction. So, for all $S' = \{S_1', S_2'\} \in \mathcal{WA}$ where $S' \neq S$, we have $v \notin S_1'$.

   By 2, for all $W \in \mathcal{WT}$, we have $v \notin W$.

   By 3, for all $W = \{W_1, W_2\} \in \mathcal{SA}$, we have $v \notin W_1$.

This covers all of the cases. □

Construction of a B-tree, from Section 4.3.2 We construct a *B-tree* in a similar manner to the DH-tree constructed by Nakano *et al.* [25] and to the C-tree constructed previously in Section 4.3.1. Note that a B-tree is rooted, ordered, and each internal node of the B-tree is labeled. We begin with two basic cases for a (C_5, bull, gem, co-gem)-free graph G:

1. G is a single vertex: the B-tree is defined to be a single root with no label.
2. G is a clique of size $n \geq 2$: the B-tree is defined by a single root with label st and n leaves with no labels (in any order).

Note that the number of leaves of the tree is the number of vertices in G, and this fact remains invariant for all B-trees. Now, consider the general case. Given a (C_5, bull, gem, co-gem)-free graph G with $|V(G)| \geq 3$, we define its B-tree $\mathcal{T}$. Assume that G is not a clique. We start with n leaves of $\mathcal{T}$ which are initially independent, and we identify each leaf with a vertex $v \in G$. More formally, we construct the injection $\phi : V(G) \to V(\mathcal{T})$ which maps each $v \in V(G)$ to a unique leaf of $\mathcal{T}$.

By Lemmas 4.10 and 4.11, we can group the vertices of G into four families, $\mathcal{ST}$, $\mathcal{WT}$, $\mathcal{SA}$, and $\mathcal{WA}$, such that they are not all empty. We place vertices that do not belong in one of those families into a set N. Note that by Lemma 4.10, the familes are all disjoint, except for the pairs $\mathcal{SA}$ and $\mathcal{WT}$, and $\mathcal{WA}$ and $\mathcal{ST}$.

For any set $S \in \mathcal{ST}$, we consider the leaves associated with S in $\mathcal{T}$ ($\{\phi(s) \mid s \in S\}$) and make a common parent of these leaves with label st. Similarly, for any set $W \in \mathcal{WT}$, we consider the leaves associated with W in $\mathcal{T}$ ($\{\phi(w) \mid w \in W\}$) and make a common parent of these leaves with the label wt.

Now, consider any set $W = \{W_1, W_2\} \in \mathcal{SA}$. If $|W_1| = |W_2| = 1$, we consider the leaves associated with $w_1 \in W_1$ and $w_2 \in W_2$ (namely, $\phi(w_1)$ and $\phi(w_2)$ respectively) and make a common parent of those leaves with label *sa*. If, without loss of generality, $|W_1| = 1$ and $|W_2| > 1$, where $w_1 \in W_1$, we place w_1 into N.

Similarly, we consider any set $S = \{S_1, S_2\} \in \mathcal{WA}$. If $|S_1| = |S_2| = 1$, we consider the leaves associated with $s_1 \in S_1$ and $s_2 \in S_2$ (namely, $\phi(s_1)$ and $\phi(s_2)$ respectively) and make a common parent of those leaves with label *wa*. If, without loss of generality, $|S_1| = 1$ and $|S_2| > 1$, where $s_1 \in S_1$, we place s_1 into N. Note that in all of these cases, the leaves can be arranged in any order.

Next, we construct a set of vertices $V \subset V(G)$. For each parent node created in the previous step, we choose the leftmost child c and place $\phi^{-1}(c)$ in V (this inverse necessarily exists, by our construction). We can in fact use any arbitrary child c if the parent node is labeled *st* or *wt*, but the ordering is important for a parent node labeled *sa* or *wa*. For every vertex $u \in N$, we place u in V.

If $|V| = 2$, say $a_1, b_1 \in V$, and $(a_1, b_1) \notin E(G)$, then we consider the parent nodes of $\phi(a_1)$ and $\phi(b_1)$, each of which must have another child, say $\phi(a_2)$ and $\phi(b_2)$ respectively. Necessarily, we must have for some $1 \leq i, j \leq 2$, $(a_i, b_j) \in E(G)$. This is easy to show by casework, considering all possible labels of the parent nodes of $\phi(a_1)$ and $\phi(b_1)$. We move $\phi(a_i)$ and $\phi(b_j)$ so that they are the leftmost children of their respective parent nodes, and we let $V = \{a_i, b_j\}$.

Note that the subgraph H of G induced by V is still a (C_5, bull, gem, co-gem)-free graph, since if G does not contain a cycle graph of size 5, a bull graph, a gem graph, or the complement of a gem graph, then neither does H.

We now construct an injection $\phi' : V(H) \to V(\mathcal{T})$ which maps each vertex $v \in V(H) = V$ to the parent of $\phi(v)$ if it exists, and otherwise to $\phi(v)$. We can now repeat all of the above steps, except using H and ϕ' instead of G and ϕ. We stop when H matches one of the basic cases mentioned initially, which we use to complete the tree $\mathcal{T}$. See Figure 24 for an example of a B-tree and Figure 25 for the corresponding (C_5, bull, gem, co-gem)-free graph.

Theorem 4.12 We can construct the B-tree $\mathcal{T}$ derived from graph G if and only if G is a (C_5, bull, gem, co-gem)-free graph.

Proof. From Lemmas 4.10 and 4.11, and the above constructive description of the B-tree, it is clear that given a (C_5, bull, gem, co-gem)-free graph, we can construct a corresponding B-tree $\mathcal{T}$. Thus, we only need to prove that given a B-tree $\mathcal{T}$, we can construct the corresponding (C_5, bull, gem, co-gem)-free graph. Note that this construction is identical to the one in Theorem 4.4,

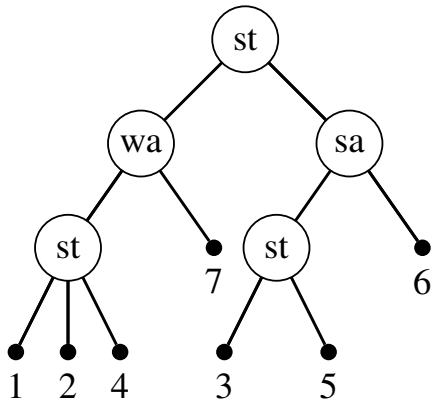

Figure 24**.** Example of a B-tree.

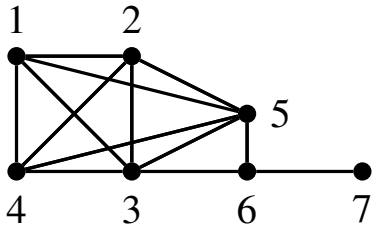

Figure 25**.** (C_5, bull, gem, co-gem)-free graph for the B-tree in Figure 24.

with the exception of the operations corresponding to the label of each $t_i \in V(\mathcal{T})$. Thus, we list the operations here and refer the reader to Theorem 4.4 for the full proof. Given $t_i \in V(\mathcal{T})$ and its children $\{c_1, \ldots, c_s\}$ where c_1 is its leftmost child, we define the following operations, corresponding to the label of t_i:

- *st*: add edges $(\rho(c_1), \rho(c_i))$ and $(c, \rho(c_i))$ to G for all $c \in N(\rho(c_1))$ and $1 < i \leq s$.
- *wt*: add edge $(c, \rho(c_i))$ to G for all $c \in N(\rho(c_1))$ and $1 < i \leq s$.
- *sa*: there can only be 2 children (by definition), c_1 and c_2. Add edges $(\rho(c_1), \rho(c_2))$ and $(c, \rho(c_2))$ for all $c \notin N(\rho(c_1))$.
- *wa*: there can only be 2 children (by definition), c_1 and c_2. Add edge $(c, \rho(c_2))$ for all $c \notin N(\rho(c_1))$.

Note that as in Theorem 4.4, these operations are essentially the same as the vertex incremental operations for (C_5, bull, gem, co-gem)-free graphs. □

Appendix B. Relevant code

All relevant code can be found here: `https://github.com/PrincetonUniversity/graphs-by-tree-decomposition.git`. Note that the git repository is currently private; access can be granted upon request. The code has also been attached as a supplemental file (sans generated graphs, due to space constraints).

Iriza [26] wrote a parsing module to translate a split tree string into its corresponding 3-leaf power graph or distance hereditary graph. We have extended the parsing module to translate a parity split tree string into its corresponding parity graph, found in `unlabeled-graph-samplers`.

Lumbroso [28] wrote a program to constructively generate and enumerate 3-leaf power graphs and distance hereditary graphs using their corresponding vertex incremental characterizations. We have extended the program to generate and enumerate $(6,2)$-chordal bipartite graphs, (C_5, bull, gem, co-gem)-free graphs, and parity graphs. The code and generated graphs, in both image format and graph6 format, can be found in `vertex-incremental`.

Appendix C. Survey of tree decompositions

C.1. Modular decomposition

Modular decomposition, also known as *substitution decomposition*, was first introduced by Gallai [18]. Its most notable applications are on classes of perfect graphs, including cographs, P_4-sparse graphs, permutation graphs, and interval graphs, and in particular, Lovász used modules in his proof of the perfect graph theorem [27]. We give a cursory description of modular decompsition here and refer to Möhring and Radermacher's survey of modular decomposition [31] and McConnell and Spinrad's linear time algorithm for modular decomposition [30] for details.

Modular decomposition is essentially a generalization of connected components, and in a modular decomposition tree, the degenerate graphs are known as parallel (if the component is connected) or series (if the complement of the component is connected). Table 5 contains a list of graph classes associated with modular decomposition, information about the prime nodes relating to those classes (if available), and additional constraints on a modular decomposition tree.

We list supplemental definitions here:

Graph class	Prime nodes	Constraints
Cograph (P_4-free graph) [20]	None	None
P_4-sparse graph [32]	Prime spider graph (see Definition C.1)	• No vertex of $S \cup K$ (for any prime spider with spider-partition (S,K,R)) is an internal node
Proper interval [10]	Representation maintained in [22]	• The root r is a parallel node • Every non-leaf children of r is either a prime node or a series node, with at most one non-leaf child, which must be a prime node • The non-leaf children of any prime nodes are series nodes with only leaf children
Permutation [18, 10]	Permutation graph	• Prime permutation graphs are unique up to a realizer (up to reversal)

Table 5. Survey of modular decomposition.

Definition C.1. A *spider* graph is a graph G such that $V(G)$ admits a partition into sets S, K, and R such that:

1. $|S| = |K| \geq 2$, the set S is an independent (stable) set, and the set K is a clique;
2. all of the vertices in R are adjacent to all of the vertices in K and no vertices in S;
3. there exists a bijection $f : S \to K$ such that one of the following statements holds:
   (a) for each vertex $v \in S$, $N(v) \cap K = \{f(v)\}$, or
   (b) for each vertex $v \in S$, $N(v) \cap K = K - \{f(v)\}$.

The triple (S,K,R) is a *spider-partition*. A graph G is a *prime spider* if G is a spider with $|R| \leq 1$.

Note that in addition to the graph classes listed in Table 5, Hsu and Ma [24] have developed a linear time algorithm for modular decomposition on chordal graphs. This algorithm arises from Capelle's work [4], in which he proved that computing a modular decomposition tree and computing a *factoring permutation* are equivalent.

Moreover, Ravelomanana and Thimonier [36] have generated a symbolic grammar for cographs using its modular decomposition tree, and have obtained asymptotic bounds from the grammar.

C.2. Split decomposition

We have fully explained the *split decomposition*, also known as the *join decomposition*, in Section 3.1. Importantly, split decomposition is a generalization of modular decomposition. Table 6 contains a list of graph classes associated with split decomposition, information about the prime nodes related to those classes (if available), and additional constraints on a split tree (we presuppose the contraints given in the definition of a split tree, in Theorem 3.1).

Graph class	Prime nodes	Constraints
Distance-hereditary [19]	None	None
Cograph [19]	None	• Contains a clique node or a tree-edge towards which all star nodes are oriented
3-leaf power [19]	None	• The set of star nodes forms a connected subtree • If u is a star node, then the tree-edge e such that $\rho_u(e)$ is the center of the star is incident to a leaf or a clique node
Circle	Circle graph [2, 17]	• Prime circle graphs are unique up to a realizer [34] • Algorithm for detecting prime circle graphs (and circle graphs in general), in $O(n^2)$ time [38]
Parity [9]	Bipartite graph	None

Table 6. Survey of split decomposition.

C.3. Bijoin decomposition

Bijoin decomposition was first introduced by Montgolfier and Rao [13] as a new generalization of modular decomposition. The degenerate graphs are complete bipartite graphs or a disjoint union of two complete graphs, and the completely decomposable graphs with respect to the bi-join decomposition are (C_5, bull, gem, co-gem)-free [13, 14]. Montgolfier and Rao [13] also give a linear time algorithm for computing the bi-join decomposition.