Karlsruhe Institute of Technology

On Product Structure of String Graphs

Master's Thesis of

Deborah Haun

At the Department of Informatics
Institute of Theoretical Informatics (ITI)

Reviewers:	PD Dr. Torsten Ueckerdt
	Prof. Dr. Thomas Bläsius
Advisors:	Laura Merker, M.Sc.
	PD Dr. Torsten Ueckerdt

Time Period: 17.01.2026 – 17.07.2026

Statement of Authorship

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, July 16, 2026

Abstract

A graph class $\mathcal{G}$ has *product structure* if there exists a constant t such that every graph $G \in \mathcal{G}$ is a subgraph of the strong product $H \boxtimes P$ for some graph H of treewidth at most t and some path P. A necessary condition for product structure is *linear local treewidth*, where the treewidth of every radius-r neighborhood grows only linear with r. The main goal of this thesis is to identify subclasses of string graphs for which linear local treewidth and product structure are equivalent.

As a potential candidate, we focus on *grounded* $\urcorner$*-graphs*, that is, string graphs admitting a string representation where each string is an $\urcorner$-shape and the bottom endpoints of those $\urcorner$-shapes are on a common horizontal line. We prove upper bounds on the treewidth and the treewidth in the neighborhood of a vertex in terms of the maximum number of intersections that the vertical or horizontal segment of an $\urcorner$-shape is involved in.

An interesting subclass of grounded $\urcorner$-graphs are *permutation graphs*. We show that $K_{k,k}$-free permutation graphs have pathwidth and treewidth $\Theta(k)$. Consequently, for every subclass of permutation graphs, bounded biclique number, bounded pathwidth, bounded treewidth, linear local treewidth, and product structure are all equivalent.

Deutsche Zusammenfassung

Eine Graphklasse $\mathcal{G}$ hat *product structure*, wenn es eine Konstante t gibt, sodass jeder Graph $G \in \mathcal{G}$ ein Teilgraph des starken Produkts $H \boxtimes P$ eines Graphen H mit Baumweite höchstens t und eines Pfades P ist. Eine notwendige Bedingung für product structure ist *linear local treewidth*. Dabei wächst die Baumweite in der r-Nachbarschaft eines jeden Knotens höchstens linear mit r. Ziel dieser Arbeit ist es, Unterklassen von Stringgraphen zu identifizieren, für die linear local treewidth und product structure äquivalent sind.

Als möglichen Kandidaten betrachten wir *grounded* $\urcorner$*-Graphen*, also Stringgraphen, die eine Repräsentation besitzen, in der jeder String die Form eines $\urcorner$ hat und alle unteren Endpunkte dieser $\urcorner$-Strings auf einer gemeinsamen horizontalen Gerade liegen. Wir zeigen obere Schranken an die Baumweite und an die Baumweite in der Nachbarschaft eines Knotens abhängig von der maximalen Anzahl an Schnittpunkten, an denen das vertikale beziehungsweise horizontale Segment eines $\urcorner$-Strings beteiligt ist.

Eine interessante Unterklasse von grounded $\urcorner$-Graphen sind *Permutationsgraphen*. Wir zeigen, dass $K_{k,k}$-freie Permutationsgraphen Pfadweite und Baumweite $\Theta(k)$ haben. Folglich sind für alle Unterklassen von Permutationsgraphen beschränkte Bicliquenzahl, beschränkte Pfadweite, beschränkte Baumweite, linear local treewidth und product structure äquivalent.

Contents

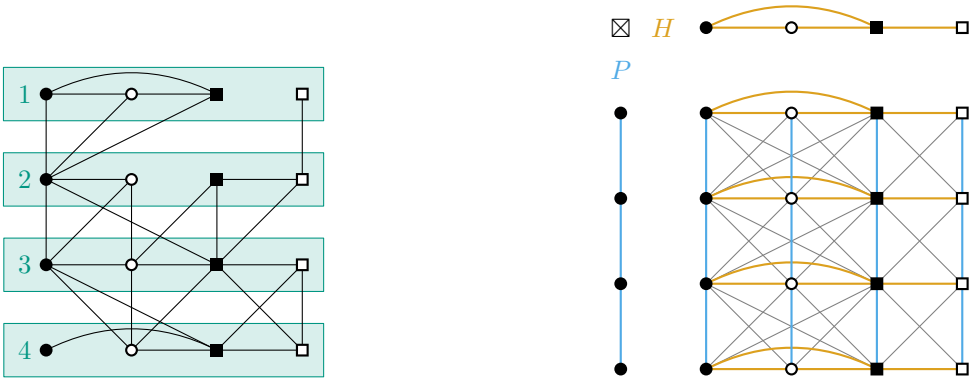
1. Introduction

Product structure decomposes the graphs of a graph class into significantly simpler graphs such as a path and a graph of constant treewidth [DJM+20]. This concept has proved useful for a wide range of applications (see [Duj26] for a survey). Unfortunately, proving product structure for a given graph class is often highly nontrivial, and a variety of advanced techniques have been developed for this purpose over the last couple of years [DMW23, HKW25, HW24].

In contrast, disproving product structure is oftentimes simpler since it suffices to show that a necessary condition, called linear local treewidth, fails. Arguably, it would be very convenient if linear local treewidth was also sufficient for product structure. In general, this is false [BDJ+22b], but for some specific graph classes, for example minor-closed graph classes, linear local treewidth and product structure are equivalent [DJM+20, ISW24]. Our goal is to understand for which other graph classes this equivalence holds.

Formally, a graph class $\mathcal{G}$ admits *product structure* if there exists a constant t such that every graph $G \in \mathcal{G}$ is a subgraph of $H \boxtimes P$ for some graph H of treewidth at most t and some path P [DJM+20]. Here, $H \boxtimes P$ denotes the *strong product* of H and P (see Figure 1.1(b)). For now, it suffices to think of $H \boxtimes P$ as consisting of one copy of H for each vertex of the path P. We refer to these copies as the *rows* of $H \boxtimes P$. Then, all edges in $H \boxtimes P$ are within a row or between two consecutive rows. Hence, for a fixed vertex v of $H \boxtimes P$, every vertex of distance at most d from v lies within the $2d + 1$ rows centered around the row containing v. Since each row is a copy of the graph H, whose treewidth is constant, the treewidth of the d-neighborhood of a vertex grows at most linearly with d, which is precisely the definition of *linear local treewidth*. Therefore, linear local treewidth is necessary for product structure.

Nearly every proof that a graph class does not admit product structure proceeds by rejecting linear local treewidth [BDH+26, HW24, Kar25, MSS+24]. Indeed, there is only a single graph family known that separates linear local treewidth from product structure [BDJ+22b]. Naturally, this raises the question whether linear local treewidth and product structure are equivalent for most interesting graph classes. If true, this would provide an easy way to prove product structure. This question has also been asked by Illingworth, Scott, and Wood [ISW24], and as mentioned before, it is answered affirmatively for minor-closed graph classes [DJM+20, ISW24]. However, the proofs for minor-closed graph classes rely heavily on graph minor theory, e.g., the graph minor structure theorem of Robertson and Seymour [RS03], and cannot simply be extended due to the lack of such a strong tool for other graph classes.



(a) A graph G together with a layering $\delta\colon V(G) \to \{1, 2, 3, 4\}$.

(b) The strong product $H \boxtimes P$.

Figure 1.1: The graph G on the left is a subgraph of the strong product $H \boxtimes P$ on the right. The vertices of G are partitioned into sets (black/white circle/square) that contain at most one vertex per layer. We denote the partition by $\mathcal{P}$. Then, the quotient $G/\mathcal{P}$ is isomorphic to H.

A substantial part of the literature on product structure focuses on string graphs [BDH+26, DHJ+21, DMW23, HW24, Kar25, MSS+24], i.e., intersection graphs of curves in the plane. This is mainly due to their close connection to planar graphs, which were the starting point for all product structure theory [DJM+20]. Indeed, every planar graph is a string graph, and conversely, every string graph admits a natural planarization that allows to transfer product structure theory from planar graphs to bounded-degree string graphs [DMW23, HW24, Kar25] (see Chapter 3).

Furthermore, the construction of Bose, Dujmović, Javarsineh, Morin, and Wood [BDJ+22b] which separates linear local treewidth from product structure, is certainly not a string graph. Their construction contains large complete graphs in which every edge is subdivided at least once. By a result of Ehrlich, Even, and Tarjan [EET76], such subdivisions of non-planar graphs are not string graphs. This makes the question whether linear local treewidth and product structure are equivalent particularly interesting for string graphs and subclasses of string graphs.

Question 1.1. *Are linear local treewidth and product structure equivalent for every family of string graphs?*

To narrow this question down, the goal of this thesis is to identify subclasses of string graphs for which the answer to Question 1.1 is affirmative. In other words, we aim to identify subclasses of string graphs for which linear local treewidth and product structure are equivalent for every family within the subclass. Two examples of such subclasses are Euclidean unit disk graphs and hyperbolic uniform disk graphs with constant disk radius. In both cases, bounded clique number is equivalent to product structure [BDH+26, DHJ+21]. Since bounded clique number is necessary for linear local treewidth, this implies that linear local treewidth and product structure are equivalent for these two subclasses of string graphs.

If linear local treewidth is not sufficient, it is only natural to look for properties between linear local treewidth and product structure. One such candidate is bounded layered treewidth (see Section 2.3 for the definition). However, the construction of Bose, Dujmović, Javarsineh, Morin, and Wood [BDJ+22b] also separates bounded layered treewidth from product structure, whereas it is unclear whether linear local treewidth and bounded layered

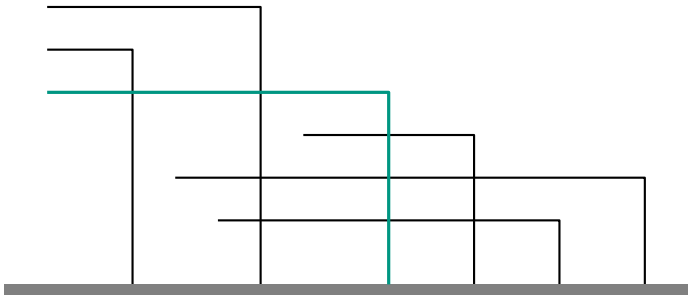


Figure 1.2: Grounded ⅂-shapes. The green ⅂-shape has two intersections in its horizontal segment and three intersections in its vertical segment.

treewidth coincide. Hence, it would be interesting to find out whether linear local treewidth and bounded layered treewidth are equivalent, and to identify additional easy-to-verify properties between linear local treewidth, bounded layered treewidth, and product structure, at least for some specific graph classes.

1.1 Contributions and Outline

In Chapter 2, we define basic concepts and notations and introduce the subclasses of string graphs that we consider throughout this thesis. In Chapter 3, we survey some interesting techniques that have been used in the past to show product structure for various graph classes, and in particular, bounded-degree string graphs. Specifically, we present shortcut systems [DMW23], shallow minors [HW24], and colored planarization [HKW25], focusing on their applications to string graphs.

Then, in Chapter 4 and Chapter 5 we focus on grounded ⅂-graphs, that is, string graphs admitting a representation in which each string is an ⅂-shape and the bottom endpoints of those ⅂-shapes are on a common horizontal line (see Figure 1.2 for an illustration and Section 2.4 for a formal definition). We conjecture that grounded ⅂-graphs are a subclass of string graphs for which Question 1.1 holds true.

Conjecture 1.2. *A family of grounded ⅂-graphs has product structure if and only if it has linear local treewidth.*

As an intermediate step, we introduce grounded k-horizontal-intersection and grounded k-vertical-intersection ⅂-graphs in Chapter 5, i.e., grounded ⅂-graphs where each ⅂-shape has at most k intersections in its horizontal or vertical segment, respectively (see Figure 1.2). Although at first glance these graph classes seem very similar, we show that they are incomparable.

Theorem 1.3 (Corollary 5.5 and Lemma 5.11)**.** *For every $k \geq 0$, there exists a grounded 2-horizontal-intersection ⅂-graph that is not a grounded k-vertical-intersection ⅂-graph, and there exists a grounded 2-vertical-intersection ⅂-graph that is not a grounded k-horizontal-intersection ⅂-graph.*

We conjecture that for grounded ⅂ graphs product structure is equivalent to being k-horizontal-intersection or k-vertical-intersection for some constant value of k.

Conjecture 1.4. *A family $\mathcal{G}$ of grounded ⅂-graphs admits product structure if and only if there exists a constant k such that every graph $G \in \mathcal{G}$ is a grounded k-horizontal-intersection or a grounded k-vertical-intersection ⅂-graph.*

To support Conjecture 1.4, we prove that grounded k-vertical-intersection $\urcorner$-graphs have treewidth in $\mathcal{O}(k)$.

Theorem 1.5 (Theorem 5.4)**.** *For every $k \geq 0$, every grounded k-vertical-intersection $\urcorner$-graph has treewidth at most $3k + 2$.*

Hence, for any fixed k, the class of grounded k-vertical-intersection $\urcorner$-graphs admits product structure, and thus, linear local treewidth, which immediately implies that Conjecture 1.2 holds true for every family of grounded k-vertical-intersection $\urcorner$-graphs. Moreover, we show that the treewidth in the neighborhood of any fixed vertex in a grounded k-horizontal-intersection $\urcorner$-graph is in $\mathcal{O}(k)$, which is a first step towards linear local treewidth and product structure.

Theorem 1.6 (Corollary 5.8)**.** *For every grounded k-horizontal-intersection $\urcorner$-graph G and every vertex $v \in V(G)$, the treewidth in the neighborhood of v is at most $3k + 1$.*

Furthermore, we consider permutation graphs, which is another subclass of grounded $\urcorner$-graphs [JT19] (see Section 2.4). We show that the pathwidth of permutation graphs is at most twice its biclique number.

Theorem 1.7 (Theorem 5.7)**.** *Let G be a permutation graph and let k be the size of the largest biclique contained in G as a subgraph. Then, $k \leq \mathsf{tw}(G) \leq \mathsf{pw}(G) \leq 2k$.*

Since bounded biclique number is necessary for linear local treewidth, this implies that permutation graphs are another positive example for Conjecture 1.2.

Theorem 1.8 (Corollary 5.10)**.** *For every family of permutation graphs, linear local treewidth and product structure are equivalent.*

Finally, in Chapter 6 we conclude this thesis by presenting several open questions which we believe are the next steps towards answering Question 1.1, that is, for which subclasses of string graphs linear local treewidth and product structure are equivalent.

1.2 Related Work

In this section, we first give an overview of the various applications of product structure. We then introduce two variants of product structure, namely BFS structure and geodesic structure, that offer additional insight into the concept. Finally, we conclude this section with some related work on string graphs.

Applications. Product structure has a wide range of applications so that a complete overview would go beyond the scope of this thesis. However, we list some of them here. For a more extensive overview, additionally including an accessible proof of the product structure theorem for planar graphs, we refer to the lecture notes recently provided by Dujmović [Duj26].

Most notably, Dujmović, Joret, Micek, Morin, Ueckerdt, and Wood [DJM+20] introduced product structure as a tool to show that planar graphs have constant queue number, thereby answering a long-standing open question by Heath, Leighton, and Rosenberg [HLR92]. Further graph-theoretic applications include results on track numbers [DJM+20, Pup20],

3-dimensional grid drawings [DJM+20], nonrepetitive colorings [DEJ+20], centered colorings [DFM+21], universal graphs and adjacency labeling schemes [BGP22, DEG+21, EJM23, HMŠ+21], twin-width [BKW25, JP22], and ℓ-vertex ranking [BDJ+22a].

To use product structure algorithmically, one typically first needs to compute a representation of the input graph as a strong product. That is, for a given graph G, one would need to compute a bounded-treewidth graph H and a path P such that G is contained in $H \boxtimes P$, together with an explicit mapping of G into $H \boxtimes P$. For planar graphs, the original proof of product structure by Dujmović, Joret, Micek, Morin, Ueckerdt, and Wood [DJM+20] already yields a quadratic time algorithm for constructing such a representation. This runtime was subsequently improved by Morin [Mor21] to $\mathcal{O}(n \log n)$, and then by Bose, Morin, and Odak [BMO22] to linear time.

Those algorithms immediately yield algorithms for several of the problems listed above (see [BMO22] for details). Furthermore, An, Im, Kim, and Lee [AIK+25] applied product structure to the $\mathcal{F}$-Subgraph-Free Edge Deletion problem on planar graphs and disk graphs of bounded local radius. For this purpose, they also provide a linear-time algorithm that computes, for such a disk graph, a representation as a strong product of a bounded-treewidth graph and a path, building on the linear-time algorithm for planar graphs by Bose, Morin, and Odak [BMO22].

Variants of Product Structure. The rows of the strong product $H \boxtimes P$ can be interpreted as a layering of the graph G, that is, an assignment of vertices to ordered layers such that edges occur only within the same layer or between consecutive layers (see Figure 1.1). In fact, this perspective leads to an alternative definition of product structure. Given a graph G, consider a layering together with a partition $\mathcal{P}$ of $V(G)$ such that each part of $\mathcal{P}$ contains at most one vertex per layer (see Figure 1.1). We then form the quotient graph $G/\mathcal{P}$ as the graph whose vertices correspond to the parts of $\mathcal{P}$, with two parts being adjacent whenever there is at least one edge between them in G. Intuitively, $G/\mathcal{P}$ plays the role of the graph H in the strong product $H \boxtimes P$ (see Figure 1.1). A graph has product structure if and only if there exists such a layering and partition for which the quotient graph has bounded treewidth.

This perspective naturally allows for several refinements of product structure obtained by imposing additional conditions on the layering and the partition. We introduce these refinements here since they frequently occur in constructions, e.g., the one in Chapter 4. In particular, if the layering is required to be a BFS-layering, that is, a layering obtained by a breadth-first search originating from an arbitrary vertex, and the partition is required to be connected, i.e., each part induces a connected subgraph, then we speak of *BFS structure* instead of *product structure*, which a strictly stronger notion [Sch25]. In their seminal work on product structure, Dujmović, Joret, Micek, Morin, Ueckerdt, and Wood [DJM+20] already proved that planar graphs, and more generally, bounded genus graphs admit BFS-structure, not only for some BFS-layering, but for every possible BFS-layering.

It is easy to see that in the case of BFS-structure, every part in the partition is a shortest path, also called a *geodesic*. This leads us to the concept of *geodesic structure*, which predates product structure [PS21]. A graph class $\mathcal{G}$ has geodesic structure if there exists a constant t such that every graph $G \in \mathcal{G}$ has a partition $\mathcal{P}$ into geodesics for which the quotient $G/\mathcal{P}$ has treewidth at most t. It is clear from the definitions that BFS-structure implies geodesic structure. We remark that since this definition does not involve layerings, it is not immediately clear how geodesic structure compares to product structure. Merker, Scherzer, and Schneider [MSS26] showed that product structure does not imply geodesic structure.

String Graphs. String graphs are the intersection graphs of non-self-intersecting continuos curves (strings) in the plane. Equivalently, they can be defined as the intersection graphs of arbitrary connected regions in the plane. While the latter definition makes it more clear that intersection graphs of disks, polygons, or other arbitrary connected shapes in the plane are string graphs, we use the former definition throughout this thesis as it naturally extends to several important subclasses of string graphs that are obtained by imposing geometric constraints on the strings. This includes, for example, *grounded string graphs*, in which each string has one of its endpoints on a common ground line; *segment graphs*, in which each string is a straight-line segment; and *(grounded) ⅂-graphs*, in which each string is a (grounded) ⅂-shape (see Figure 1.2). For a more extensive introduction to string graphs and subclasses of string graphs, we refer to Section 2.4.

Besides these graph classes that are explicitly defined in terms of their string representation, many other well-known graph classes are also string graphs. For example, it is easy to see that every planar graph is a string graph by replacing each vertex by a curve that follows the incident half-edges of a planar embedding. More surprisingly, every planar graph is an ⅂-graph [GIP18]. Similarly, as observed by Jelínek and Töpfer [JT19], permutation graphs are grounded ⅂-graphs. Furthermore, Gavril [Gav74] showed that chordal graphs are exactly the intersection graphs of subtrees in trees, which immediately yields a string representation.

On the other hand, there are also many graphs that are not string graphs. Ehrlich, Even, and Tarjan [EET76] gave the first explicit example by proving that every non-planar graph, e.g., the complete graph K_5, in which each edge is subdivided at least once is not a string graph. Subsequently, Kratochvíl [Kra91a] showed that there are infinitely many *critical non-string graphs*, that is, non-string graphs for which every induced minor is a string graph. More recently, Chudnovsky, Eppstein, and Fischer [CEF25] constructed a finite subcubic family and an infinite triangle-free family of critical non-string graphs. Correspondingly, recognizing string graphs is NP-hard [Kra91b] and can be done in nondeterministic exponential time [SŠ04].

Beyond generalizing planar graphs, string graphs share several structural properties with planar graphs. For example, every string graph with m edges admits a 2/3-balanced separator of size $\mathcal{O}(\sqrt{m}\log m)$ [Mat14], which matches the planar separator theorem [LT79] up to the logarithmic factor. Moreover, Davies [Dav25] and Chang, Conroy, Tan, and Zheng [CCT+26] independently proved that every string graphs admits a constant distortion planar emulator, that is, a planar graph on a superset of the vertices that preserves distances up to a constant factor. Furthermore, every k-degenerate string graph is $2k$-gap-cover planar and has linear expansion [KW26].

Despite these similarities, string graphs differ from planar graphs in fundamental ways. While planar graphs are minor-closed, string graphs are only induced minor-closed. Moreover, unlike planar graphs, string graphs can have arbitrarily large clique number and chromatic number, and the chromatic number is not bounded by a function in the clique number, not even for segment graph [PKK+14]. Nevertheless, the chromatic number of a string graph is bounded by a function of its clique number and the largest Burling graph it contains as an induced subgraph [ABD+26].

2. Preliminaries

In this chapter, we introduce and define all concepts used throughout the thesis. We start with some general notation, before we continue with some basics about graphs such as minors and tree decompositions. Then, in Section 2.3, we define product structure and its two necessary conditions, linear local treewidth and bounded layered treewidth. Finally, in Section 2.4, we give an overview of the subclasses of string graphs considered in this thesis.

2.1 Basic Notation

For an integer n, we denote by $[n]$ the set of integers i with $1 \leq i \leq n$. For two real numbers $a \leq b$, we denote by $[a, b] \coloneqq \{x \in \mathbb{R} \mid a \leq x \leq b\}$ the *(closed) interval* between a and b. The *length* of the interval $[a, b]$ is $b - a$ and its *center* is the point $(a + b)/2$. We say that an interval $[a_1, b_1]$ is *contained* in an interval $[a_2, b_2]$ if it is a *subinterval* of $[a_2, b_2]$, i.e., if $[a_1, b_1] \subseteq [a_2, b_2]$, or equivalently, $a_2 \leq a_1 \leq b_1 \leq b_2$.

Two intervals $[a_1, b_1]$ and $[a_2, b_2]$ *overlap* if $[a_1, b_1] \cap [a_2, b_2] \neq \emptyset$. They are *internally disjoint* if $[a_1, b_1] \cap [a_2, b_2] \subseteq \{a_1, b_1, a_2, b_2\}$. Finally, the intervals *touch* if they overlap and are internally disjoint at the same time, or equivalently, if $\emptyset \subsetneq [a_1, b_1] \cap [a_2, b_2] \subseteq \{a_1, b_1, a_2, b_2\}$.

A *bit string* is a finite word w over the alphabet $\{\mathtt{0}, \mathtt{1}\}$. The *length* of a bit string w is the number of symbols in w. The *empty bit string*, denoted by ε, is the unique bit string of length 0. The *binary value* of a bit string w is the integer value obtained by interpreting w as a binary number. Note that appending a $\mathtt{0}$ to a bit string w corresponds to multiplying its binary value with two.

2.2 Basic Graph Concepts

Graphs. All graphs in this thesis are finite and simple, i.e., without loops and multi-edges. For a graph G, we denote by $V(G)$ the set of vertices and by $E(G) \subseteq \binom{V(G)}{2}$ the set of edges of G. We denote an edge $\{u, v\} \in E(G)$ by uv. A graph G' is a *subgraph* of a graph G, denoted by $G' \subseteq G$, if and only if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. For a set $V \subseteq V(G)$ of vertices of a graph G, we denote by $G[V]$ the subgraph of G *induced by* V, and by $G - V$ the subgraph of G obtained by removing the vertices in V, i.e., $G - V = G[V(G) \setminus V]$.

The *length* $|P|$ of a path P is the number of edges $|E(P)|$. The *distance* between two vertices in a graph is the length of a shortest path between them. For a vertex $v \in V(G)$ and an integer $r \in \mathbb{N}_0$, we denote by $N_r[v]$ the *r-neighborhood* of v, that is, the set of

vertices of distance at most r from v, including v itself. We further set $N[v] \coloneqq N_1[v]$. The *radius* of a graph is the minimum integer r such that each vertex has distance at most r from any other vertex. For a set $V \subseteq V(G)$ of vertices of a graph G, we also sometimes say that its *radius* is the radius of the subgraph $G[V]$ of G induced by V.

A graph G is called *k-degenerate* if every subgraph of G has at least one vertex of degree at most k. The *degeneracy* of G is then the smallest integer k such that G is k-degenerate. Equivalently, a graph G is k-degenerate if and only if there is an ordering of its vertices, called a *degeneracy ordering*, such that each vertex is succeeded by at most k of its neighbors. A graph has degeneracy 1 if and only if it is a forest.

Trees. An acyclic graph is called a *forest* and an acyclic connected graph is called a *tree*. In particular, in a tree there is exactly one path between any two vertices. A *rooted tree* is a tree T together with a designated *root* $t \in V(T)$. In a tree T, the *depth* of a vertex $v \in V(T)$ is its distance to the root t. The *height* of a rooted tree is the maximum depth over all vertices. The *parent* of a vertex $v \neq t$ is the vertex adjacent to v on the unique path from v to t. Conversely, a *child* of a vertex v is a vertex, which has v as its parent. The vertices on the unique path from v to the root t, excluding v itself, are called the *ancestors* of v. Again, a *descendant* of a vertex v is a vertex, which has v as an ancestor.

Minors. For a graph G and an edge $e = uv \in E(G)$, we denote by G/e the graph obtained from G by *contracting* the edge e, that is, by identifying u and v, and, if necessary, removing loops and multiple edges. A graph H is a *minor* of G if it can be obtained from G by a sequence of edge contractions and deletions of edges and vertices. Equivalently, a *model of H in G* is a function $\phi_H \colon V(H) \to 2^{V(G)}$ that maps each vertex of H to a set of vertices of G, called a *branch set*, such that the branch sets are pairwise disjoint, for each vertex $v \in V(H)$, the subgraph $G[\phi_H(v)]$ of G induced by the branch set $\phi_H(v)$ is connected, and for every edge $uv \in E(H)$, there exists an edge in G with one endpoint in $\phi_H(u)$ and the other one in $\phi_H(v)$. Clearly, H is a minor of G if and only if there exists a model of H in G.

For an integer $r \in \mathbb{N}_+$, a minor H of a graph G is an *r-shallow minor* if there exists a model of H in G such that each branch set has radius at most r. Moreover, it is a *weak r-shallow minor* if for each branch set B, there exists a vertex $v \in V(G)$, not necessarily in B, such that each vertex in B has distance at most r from v.

Tree Decompositions. A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \mathsf{bag})$, where T is a tree and $\mathsf{bag} \colon V(T) \to 2^{V(G)}$ is a function that maps every node $t \in V(T)$ to a *bag* of vertices such that:

- $V(G) = \bigcup_{t \in V(T)} \mathsf{bag}(t)$,
- for every vertex $v \in V(G)$, the subgraph of T induced by the set $\{t \in V(T) \mid v \in \mathsf{bag}(t)\}$ of nodes whose bags contain v is a tree, and
- for every edge $uv \in E(G)$, there exists a node $t \in V(T)$ such that $u, v \in \mathsf{bag}(t)$.

The *width* of a tree decomposition is one less than the maximum size of a bag. The *treewidth* $\mathsf{tw}(G)$ of a graph G is the minimum width of a tree decomposition of G. If T is a path, $\mathcal{T}$ is called a *path decomposition*, and the minimum width over all path decompositions is called the *pathwidth* and denoted by $\mathsf{pw}(G)$. If T is a rooted tree, $\mathcal{T}$ is called a *rooted tree decomposition*.

Separators. We say that (A, B) is a *separation* of a graph G if $A \cup B = V(G)$ and there is no edge between $A \setminus B$ and $B \setminus A$. The set $S \coloneqq A \cap B$ is then called a *separator*, and we say that S *separates* $A \setminus B$ from $B \setminus A$. The *order* of a separation is the size of the separator S.

For a value $\alpha \in [0,1]$, we say that a separation (A,B) is *α-balanced* if $|A \setminus B| \leq \alpha \cdot |V(G)|$ and $|B \setminus A| \leq \alpha \cdot |V(G)|$. It is a well-known fact that every graph G has a 2/3-balanced separation (A,B) of order at most $|A \cap B| \leq \mathsf{tw}(G)+1$ (see, e.g., [CFK+15]).

2.3 Product Structure and its Necessary Conditions

A graph class $\mathcal{G}$ has *bounded local treewidth* if there exists a function f such that for every graph $G \in \mathcal{G}$ and every vertex $v \in V(G)$, the treewidth in the r-neighborhood of v, i.e., in $G[N_r[v]]$, is at most $f(r)$. If $f \in \mathcal{O}(r)$, we say that $\mathcal{G}$ has *linear local treewidth*.

A *layering* of a graph G is a function $\delta\colon V(G) \to \{0,1,\dots\}$ that assigns every vertex $v \in V(G)$ a *layer* $\delta(v)$ such that for every edge $uv \in E(G)$ it holds that $|\delta(u)-\delta(v)| \leq 1$. One example for a layering of a graph G would be a *breadth-first search layering*, short *BFS-layering*, which is a layering resulting from a breadth-first search. In particular, if δ is a BFS-layering of G originating from a vertex $v \in V(G)$, then for every vertex $u \in V(G)$, the layer $\delta(u)$ is exactly the distance between u and v. We say that a path $P=(v_0,v_1,\dots,v_k)$ in a graph G is *vertical* with respect to a layering δ if $\delta(v_i)=\delta(v_{i-1})+1$ for every $1 \leq i \leq k$. The *layered treewidth* $\mathsf{ltw}(G)$ of a graph G is the minimum integer k such that G has a tree decomposition, where each bag contains at most k vertices of each layer. A graph class $\mathcal{G}$ has *bounded layered treewidth* if there exists a constant c such that for every graph $G \in \mathcal{G}$, $\mathsf{ltw}(G) \leq c$.

A *partition* of a graph G is a partition of its vertex set $V(G)$. For a graph G and a partition $\mathcal{P}$ of $V(G)$, the *quotient* $G/\mathcal{P}$ is the graph with $\mathcal{P}$ as its vertex set that has an edge between two sets $P_1,P_2 \in \mathcal{P}$ if and only if there is an edge $uv \in E(G)$ with $u \in P_1$ and $v \in P_2$. Hence, the quotient $G/\mathcal{P}$ is the graph obtained from G by identifying all vertices in each set $P \in \mathcal{P}$. Note that if each part $P \in \mathcal{P}$ is connected, then $G/\mathcal{P}$ is a minor of G. In this case, we call $\mathcal{P}$ a *connected partition*.

The *layered width* of a partition $\mathcal{P}$ of a graph G is the minimum integer ℓ such that for some layering $\delta\colon V(G) \to \{0,1,\dots\}$, each part in $\mathcal{P}$ contains at most ℓ vertices from each layer. The *row treewidth* $\mathsf{rtw}(G)$ of a graph G is the minimum k for which G has a partition $\mathcal{P}$ of layered width 1 such that $G/\mathcal{P}$ has treewidth at most k. A graph class $\mathcal{G}$ has *product structure* if there exists a constant c such that every graph $G \in \mathcal{G}$ has row treewidth at most c. In their seminal work on product structure, Dujmović, Joret, Micek, Morin, Ueckerdt, and Wood [DJM+20] proved that in order to show that a graph has bounded row treewidth it is sufficient to show that G has a partition $\mathcal{P}$ of some constant layered width ℓ (instead of layered width 1) such that $G/\mathcal{P}$ has constant treewidth.

Proposition 2.1 ([DJM+20])**.** *If a graph G has a partition $\mathcal{P}$ of layered width ℓ such that $G/\mathcal{P}$ has treewidth at most k, then G has a partition $\mathcal{P}'$ of layered width 1 (w.r.t. the same layering) such that $G/\mathcal{P}'$ has treewidth at most $(k+1)\ell-1$.*

Equivalently, product structure can be defined via the *strong product* of graphs. For two graphs G_1 and G_2, the strong product $G_1 \boxtimes G_2$ is the graph G with vertex set $V(G)=V(G_1)\times V(G_2)$, where $(u_1,u_2),(v_1,v_2) \in V(G)$ are adjacent if and only if either $u_1=v_1$ and $u_2v_2 \in E(G_2)$, $u_1v_1 \in E(G_1)$ and $u_2=v_2$, or $u_1v_1 \in E(G_1)$ and $u_2v_2 \in E(G_2)$ (see Figure 1.1). Then, a graph class $\mathcal{G}$ has product structure if and only if there exists a constant t such that for every graph $G \in \mathcal{G}$ there exists a path P and a graph H with $\mathsf{tw}(H) \leq t$ such that $G \subseteq P \boxtimes H$, and the smallest such t for a graph $G \in \mathcal{G}$ is its row treewidth.

Dujmović, Joret, Micek, Morin, Ueckerdt, and Wood [DJM+20] also showed that bounded layered treewidth is a necessary condition for product structure. It is clear from the definitions that linear local treewidth is necessary for bounded layered treewidth.

Proposition 2.2 ([DJM+20]; see page 3 in [BDJ+22b])**.** *Let G be a graph, $v \in V(G)$ be a vertex, and $r \in \mathbb{N}_0$. Then,* $\mathsf{ltw}(G) \leq \mathsf{rtw}(G) + 1$ *and* $\mathsf{tw}(G[N_r[v]]) \leq \mathsf{ltw}(G) \cdot (2r + 1) - 1$.

In general, the converse does not hold. That is, Bose, Dujmović, Javarsineh, Morin, and Wood [BDJ+22b] showed that the row treewidth is not bounded by a function of the layered treewidth. However, it is unclear whether bounded layered treewidth and linear local treewidth are equivalent.

2.4 String Graphs

We now define string graphs and their subclasses following the terminology of Jelínek and Töpfer [JT19]. An overview of all the graph classes considered in this thesis is illustrated in Figure 2.1. A graph G is called an *intersection graph* if there exists a finite collection $\mathcal{C}$ of sets (e.g., point sets or geometric objects) such that every vertex of G corresponds to a set in $\mathcal{C}$, and two vertices of G share an edge if and only if their corresponding sets intersect. If $\mathcal{C}$ is a finite collection of non-self-intersecting continuous curves in the plane, called *strings*, we say that the intersection graph G is a *string graph* and $\mathcal{C}$ is its *string representation*. We remark that in the literature string graphs are defined similarly for other surfaces [Kar25, KW26], however, in this work, we only consider string graphs in the plane $\mathbb{R}^2$.

A string representation $\mathcal{C}$ is *grounded* if each curve has one of its endpoints, called *ground point*, on a common line, called *grounding line*, and the remaining points of the strings lie all on the same open half-plane defined by the ground line. In the following, we always assume that the ground line is the x-axis, and the strings lie in the half-plane above the x-axis, i.e., all of their points have a non-negative y-coordinate, where the ground point is the only point with y-coordinate 0.

Similarly, a string representation $\mathcal{C}$ is called *outer* if all strings in $\mathcal{C}$ are confined to a disk, called *grounding disk*, and each string has one endpoint, called *ground point*, on the boundary of the disk. It is easy to see that a graph admits a grounded string representation if and only if it admits an outerstring representation. We call these graphs *grounded string graphs* or *outerstring graphs*.

It is known that there are (outer)string graphs, for which the number of string-intersections is exponential in the number of strings in every (outer)string representation [BBD18, KM91]. Hence, a natural way to restrict the class of string graphs would be by allowing only a constant number of pairwise intersections. In particular, for an integer $k \in \mathbb{N}_0$, we call an (outer)string graph *(outer)-k-string* if it admits an (outer)string representation, in which any two distinct strings intersect at most k times.

Another natural way to restrict the class of string graphs is by imposing geometric restrictions on the strings in a string representation. For example, if all strings in the string representation $\mathcal{C}$ are straight-line segments, we call the string graph G a *segment graph*, and if $\mathcal{C}$ is additionally grounded or outer, we call G a *grounded segment graph* or an *outersegment graph*. While clearly every grounded segment graph is also an outersegment graph, the converse does not hold as shown by Cardinal, Felsner, Miltzow, Tompkins, and Vogtenhuber [CFM+18]. If all segments in a grounded segment representation are from d different directions, we call the corresponding grounded segment graph *d-directional*.

Another important type of strings are *$\urcorner$-shapes* or *$\ulcorner$-shapes*. An $\urcorner$-shape is the union of a horizontal and a vertical straight-line segment such that the right endpoint of the horizontal segment and the top endpoint of the vertical segment coincide and an $\ulcorner$-shape is obtained by reflecting an $\urcorner$-shape across the y-axis. The corresponding intersection graphs, *$\urcorner$-graphs*,

or equivalently, $\lceil$*-graphs*, first considered by Gyárfás and Lehel [GL85], are string graphs that admit a string representation, in which every string is an $\urcorner$-shape or an $\lceil$-shape, respectively, while $\{\urcorner,\lceil\}$-graphs allow both orientations in the same representation. A *grounded* $\urcorner$*-graphs* is the intersection graph of grounded $\urcorner$-shapes, i.e., whose bottom endpoints lie on the x-axis. *Grounded* $\lceil$*-graphs* and *grounded* $\{\urcorner,\lceil\}$*-graphs* are defined analogously. The class of grounded $\urcorner$-graphs was first studied by McGuinness [McG96]. In their paper, however, the $\urcorner$-shapes were infinite to the bottom, which is equivalent to them being grounded on the x-axis. It is known that $\urcorner$-graphs are segment graphs and that grounded $\{\urcorner,\lceil\}$-graphs are grounded segment graphs [MP92]. Furthermore, planar graphs are $\urcorner$-graphs [GIP18] and outerplanar graphs are grounded $\urcorner$-graphs [ALD+17].

In this thesis, we also consider *permutation graphs*. A graph G with vertex set $[n]$ is a permutation graph if there exists a permutation $\pi\colon [n] \to [n]$ such that for $1 \leq i < j \leq n$, ij is an edge if and only if $\pi(i) > \pi(j)$. Equivalently, as observed by Jelínek and Töpfer [JT19], permutation graphs are exactly the graphs admitting a grounded $\urcorner$-representation in which the top endpoints of the $\urcorner$-shapes are on a common vertical line (see Figure 2.1). In particular, given such a string representation, the corresponding permutation π is obtained by comparing the left-to-right order of the bottom endpoints of the $\urcorner$-shapes with the bottom-to-top order of their top endpoints.

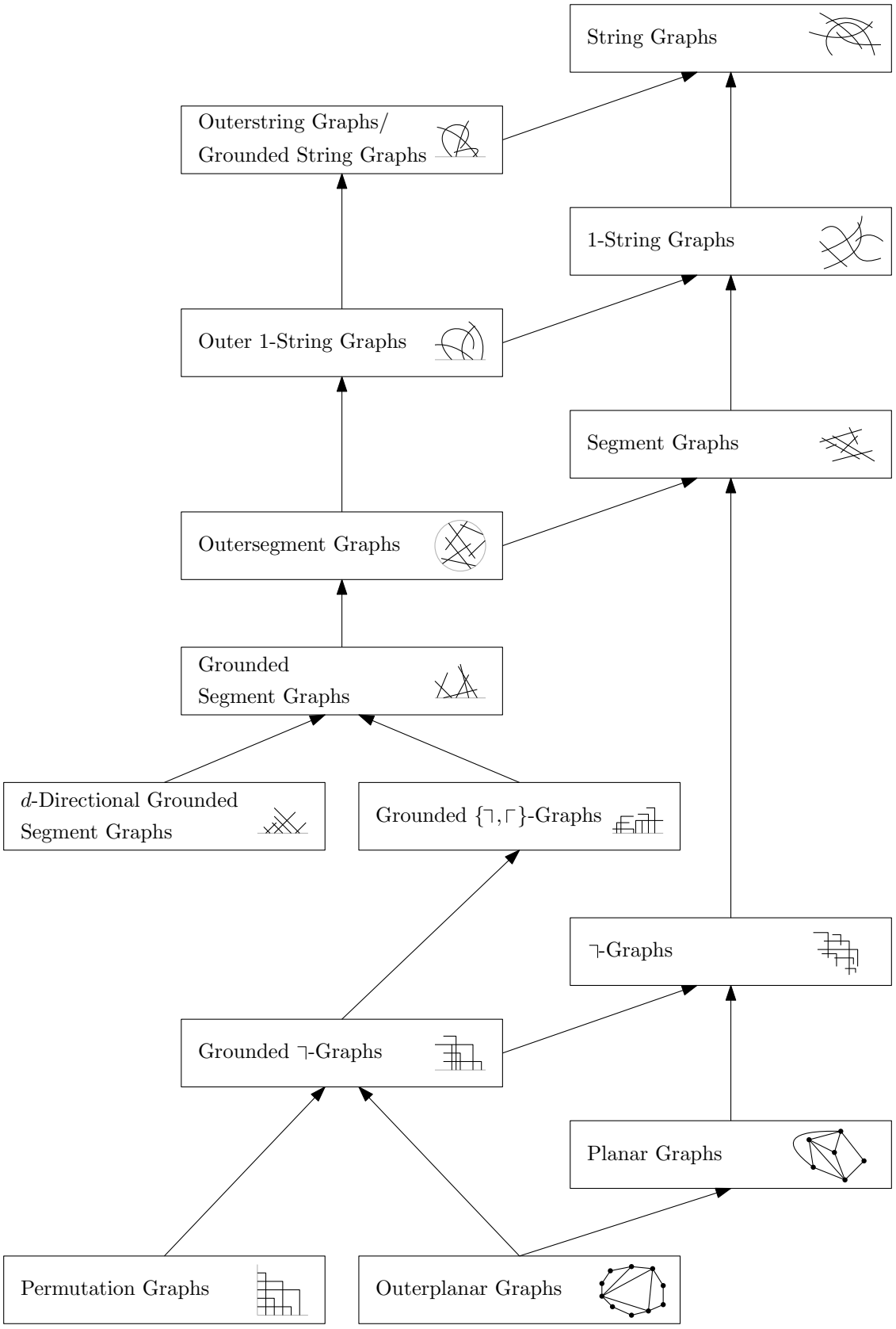


Figure 2.1: An overview of the subclasses of string graphs considered in this thesis, where arrows indicate inclusions.

3. Techniques

Building on earlier work, after introducing product structure, Dujmović, Joret, Micek, Morin, Ueckerdt, and Wood [DJM+20] show that for minor-closed graph classes, product structure, bounded layered treewidth, linear local treewidth, and even bounded local treewidth are all equivalent. Since then, a variety of techniques has been developed to show product structure for graph classes that are not minor-closed. In the following, we survey some of these techniques, focusing on their applications to string graphs.

3.1 Shortcut Systems

Dujmović, Morin, and Wood [DMW23] introduced *shortcut systems*. For a graph G and integers k and d, a (k, d)-shortcut system $\mathcal{S}$ is a non-empty set of paths of length between 1 and k such that each vertex $v \in V(G)$ is an internal vertex in at most d paths of $\mathcal{S}$. Each path $P \in \mathcal{S}$ is called a *shortcut*; if u and v are the endpoints of P, then P is called a *uv-shortcut*. Given a (k, d)-shortcut system $\mathcal{S}$ for a graph G, let $G^{\mathcal{S}}$ denote the supergraph of G obtained from G by adding the edge uv for every uv-shortcut in $\mathcal{S}$.

Dujmović, Morin, and Wood [DMW23] show that if a graph G with a shortcut system $\mathcal{S}$ has product structure, then $G^{\mathcal{S}}$ has product structure as well.

Proposition 3.1 ([DMW23])**.** *Let G be a graph together with a (k, d)-shortcut system $\mathcal{S}$ such that $G \subseteq H \boxtimes P \boxtimes K_\ell$ for some graph H with treewidth at most t and for some path P. Then, $G^{\mathcal{S}} \subseteq J \boxtimes P \boxtimes K_{d\ell(k^3+3k)}$ for some graph J of treewidth at most $\binom{k+t}{t} - 1$ and some path P.*

Subsequently, they show product structure for several graph classes, most notably, for *k-intersection string graphs*, that is, string graphs admitting a string representation, in which each string participates in at most k intersections. Dujmović, Morin, and Wood [DMW23] refer to these graphs simply as k-string graphs. However, we call them k-intersection string graphs to avoid confusion with our notion of *k-string graphs* introduced in Section 2.4, by which we mean string graphs with at most k pairwise intersections.

We remark that a string representation in which each string participates in at most k intersections immediately implies that the corresponding string graph has maximum degree at most k. The converse, however, is considerably less obvious since there exist string graphs, for which the number of intersections in any string representation is exponential

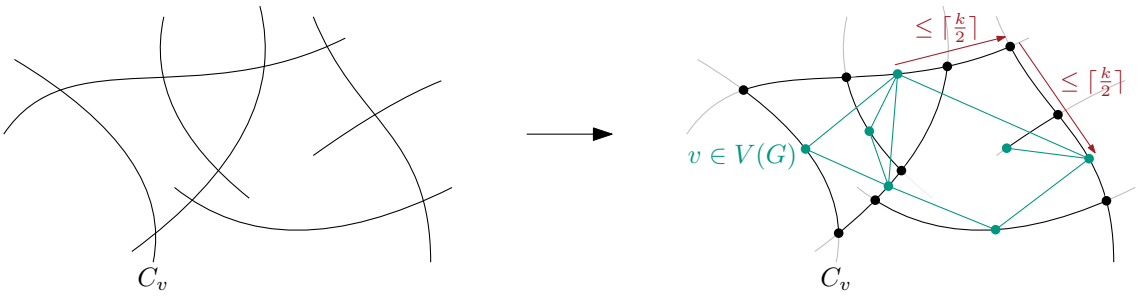


Figure 3.1: A string representation $\mathcal{C}$ (left) of a $(k = 4)$-intersection string graph G, and the corresponding graph $G_0^{\mathcal{S}}$ (right). The planar graph G_0 (right; black and green vertices, black edges) is obtained from planarizing $\mathcal{C}$ and placing each vertex $v \in V(G)$ in the "middle" of its corresponding string $C_v \in \mathcal{C}$ with respect to the dummy vertices along C_v. The green subgraph of $G_0^{\mathcal{S}}$ (right) is exactly the graph G since $\mathcal{S}$ is a $(k+1, k+1)$-shortcut system for G_0 that contains a uv-shortcut for each edge $uv \in E(G)$.

in the number of vertices [KM91]. We remark that Karol [Kar25] later proved that every string graph admits a string representation in which the number of intersections per string depends only on the degree of its corresponding vertex, albeit exponentially. Hence, every string graph with maximum degree Δ is an $f(\Delta)$-intersection string graph for some function f. Together with the proof by Dujmović, Morin, and Wood [DMW23] for k-intersection string graphs based on shortcut systems, this immediately implies product structure for bounded-degree string graphs.

However, for now let us focus on k-intersection string graphs. Specifically, to gain a better understanding on how to use shortcut systems, we now describe how Dujmović, Morin, and Wood [DMW23] apply this technique to show that k-intersection string graphs have product structure. For this, having Proposition 3.1, it only remains to show that every k-intersection string graph G is a subgraph of $G_0^{\mathcal{S}}$ for some planar graph G_0 together with a $(k+1, k+1)$-shortcut system $\mathcal{S}$ for G_0. Since every planar graph is contained in $H \boxtimes P \boxtimes K_3$ for some planar graph H with treewidth 3 and some path P [DJM+20], Proposition 3.1 then immediately implies product structure for k-intersection string graphs.

To prove that every k-intersection string graph G is a subgraph of $G_0^{\mathcal{S}}$ for some $(k+1, k+1)$-shortcut system $\mathcal{S}$ for some planar G_0, they start with a string representation $\mathcal{C}$ of G in which at most two strings intersect in any point and each string participates in at most k intersections. Then, they *planarize* this representation. That is, they construct a planar graph G_0 from $\mathcal{C}$ by placing a "dummy" vertex of degree four at every intersection point of two strings of $\mathcal{C}$. Then, for every string $C_v \in \mathcal{C}$ corresponding to a vertex $v \in V(G)$, they place a vertex v on C_v so that the dummy vertices along C_v are split as evenly as possible between the two sides of v. The resulting substrings of a string $C \in \mathcal{C}$ between two consecutive vertices on C are then interpreted as edges of G_0. By construction, we have $V(G) \subseteq V(G_0)$, and even $G \subseteq G_0$ (see Figure 3.1).

Now, for each edge $uv \in E(G)$, there is a path of length at most $2\lceil k/2 \rceil \leq k + 1$ in G_0 between u and v. Let $\mathcal{S}$ be the set of all such paths. By construction, no vertex $v \in V(G)$ is an internal vertex of any path $P \in \mathcal{S}$ (see Figure 3.1). Consider the dummy vertex x_{uv} placed in the intersection of two curves C_u and C_v, where $uv \in E(G)$. If x_{uv} is an internal vertex of a path $P \in \mathcal{S}$, then P is either a uw-shortcut for some edge $uw \in E(G)$ incident to u or a vw-shortcut for some edge $vw \in E(G)$ incident to v. Furthermore, if a uw-shortcut passes through x_{uv}, then the dummy vertex x_{uw} placed at the intersection of C_u with the string $C_w \in \mathcal{C}$ that represents the vertex w lies on the same side of u along C_u

as x_{uv}, and the analogue holds for any vw-shortcut that passes through x_{uv}. Since there are at most $\lceil k/2 \rceil$ dummy vertices on either side of u along C_u, and similar for v along C_v, it follows that x_{uv} is an internal vertex of at most $2\lceil k/2 \rceil \leq k+1$ paths in $\mathcal{S}$. Therefore, $\mathcal{S}$ is a $(k+1, k+1)$-shortcut system for G_0.

In fact, Dujmović, Morin, and Wood [DMW23] even prove the same result for string graphs on surfaces with Euler genus at most g, replacing the planar graph G_0 with a graph with Euler genus at most G_0. Furthermore, also using shortcut systems, they show product structure for k-planar graphs, powers of bounded-degree planar graphs, d-map graphs (all three of them on general surfaces with bounded Euler genus), k-nearest-neighbor graphs, and d-framed graphs (see [DMW23] for definitions of these graph classes). The basic proof idea is the same for all these graph classes, the main difference being that they planarize their embedding in the plane instead of their string representation.

3.2 Shallow Minors

As shown by Hickingbotham and Wood [HW24], shortcut systems are limited to graphs with linear crossing number, that is, graphs that admit embeddings in the plane where the number of edge crossings is linear in the number of vertices. By generalizing shortcut systems to *shallow minors*, they are able to overcome this issue, show product structure for several additional beyond-planar graph classes, including (k, p)-cluster planar graphs, fan-planar graphs, and k-fan-bundle-planar graphs, and improve several bounds previously obtained by Dujmović, Morin, and Wood [DMW23], e.g., for powers of bounded-degree planar graphs, k-planar graphs, and k-intersection string graphs on surfaces with bounded Euler genus (see [HW24] for definitions of these graph classes).

Recall that an r-shallow minor of a graph is obtained by contracting subgraphs of radius at most r, and then deleting vertices and edges. Their key insight is that product structure is preserved under taking shallow minors.

Proposition 3.2 ([HW24])**.** *If G is an r-shallow minor of $H \boxtimes P \boxtimes K_\ell$, where H has treewidth at most t and P is a path, then $G \subseteq J \boxtimes P \boxtimes K_{\ell(2r+1)^2}$, where J has treewidth at most $\binom{2r+1+t}{t} - 1$.*

Similarly to the shortcut system approach, their arguments are based on planarization. Again, we outline in more detail how they use Proposition 3.2 to prove product structure for k-intersection string graphs, restricting ourselves to the plane $\mathbb{R}^2$ for simplicity. Having Proposition 3.2, it suffices to show that every k-intersection string graph is a $\lfloor k/2 \rfloor$-shallow minor of $G_0 \boxtimes K_2$ for some planar graph G_0. Actually, Hickingbotham and Wood [HW24] show a slightly stronger statement, replacing the strong product with the *lexicographic product* and the complete graph K_2 with its complement, the edgeless graph $\overline{K_2}$. However, we do not care about this refinement here since we only want to illustrate the general proof idea. Accordingly, we simplify their construction in the following.

Again, let $\mathcal{C}$ be a string representation of a graph G in which at most two strings intersect in any point and each string participates in at most k intersections. Let G_0 denote the planarization of $\mathcal{C}$, obtained from $\mathcal{C}$ by placing dummy vertices in the intersection points and interpreting subcurves between dummy vertices as edges. Without loss of generality, we assume that G does not contain any isolated vertices. Other than for shortcut systems, we do not add any other vertices to the planar graph G_0 here.

Since every string in $\mathcal{C}$ participates in at most k intersections, the set of vertices of G_0 lying on a string $C_v \in \mathcal{C}$ that corresponds to a vertex v induces a connected subgraph of G_0 of radius at most $\lfloor k/2 \rfloor$ (see Figure 3.2). Using this fact, we find G as a shallow minor

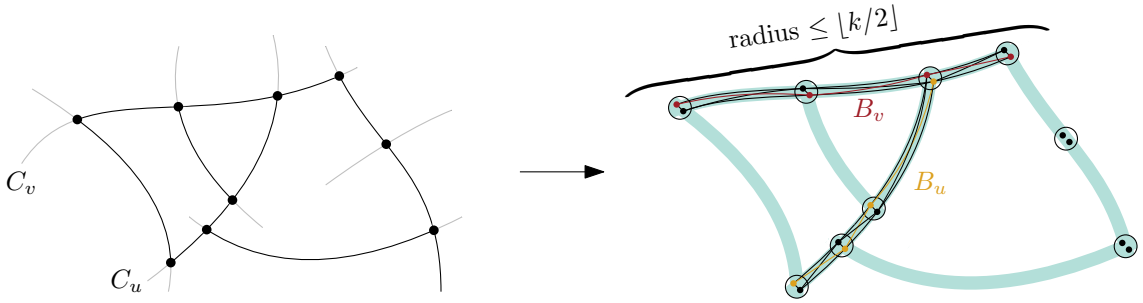


Figure 3.2: Left: The planarization G_0 of the string representation of a $(k = 4)$-intersection string graph. Right: The strong product $G_0 \boxtimes K_2$ (right), where every edge of G_0 is replaced by a complete graph on four vertices. For every dummy vertex $x \in V(G_0)$ lying on a string C_v, either $(x, 1)$ or $(x, 2)$ is assigned to the branch set B_v of v. Since each dummy vertex lies on exactly two curves, the assignments can be chosen so that the branch sets are pairwise disjoint.

in $G_0 \boxtimes K_2$. To see this, we describe the branch sets in the following. Specifically, the goal is to define a branch set $B_v \subseteq V(G_0) \times V(K_2)$ for every vertex $v \in V(G)$ with radius at most $\lfloor k/2 \rfloor$ so that distinct branch sets are pairwise disjoint, and for every edge $uv \in E(G)$, there is an edge between B_u and B_v in $G_0 \boxtimes K_2$. Then, contracting each branch set into a single vertex yields a graph containing G as a subgraph, proving that G is a shallow minor of $G_0 \boxtimes K_2$.

For every vertex $u \in V(G_0)$ lying on the curve C_v corresponding to the vertex $v \in V(G)$, we choose one of the two vertices $(u, 1)$ or $(u, 2)$ in $V(G_0) \times V(K_2)$ and add it to the branch set B_v of v, ensuring that no vertex is assigned to more than one branch set (see Figure 3.2). This assignment is always possible because every vertex of G_0 lies on exactly two strings of $\mathcal{C}$. Since the set of vertices of G_0 lying on a fixed curve $C \in \mathcal{C}$ induces a subgraph of radius at most $\lfloor k/2 \rfloor$, the same holds for each branch set by construction. Furthermore, for every edge $uv \in E(G)$, there are two intersecting curves C_u and C_v in $\mathcal{C}$, and thus, there is a dummy vertex $x_{uv} \in V(G_0)$ that lies on both C_u and C_v. By construction, one of the two vertices $(x_{uv}, 1)$ or $(x_{uv}, 2)$ belongs to B_u, while the other one belongs to B_v. Since these two vertices are adjacent in the strong product $G_0 \boxtimes K_2$, there is an edge between B_u and B_v (see Figure 3.2). Hence, the branch sets form a valid $\lfloor k/2 \rfloor$-shallow minor model of G in $G_0 \boxtimes K_2$.

We remark that this exemplary proof for k-intersection string graphs does not explain how shallow-minors overcome the limitation of shortcut systems to graph classes with linear crossing number. In particular, if a string representation or an embedding in which a string or an edge participates in many crossings is planarized and branch sets are constructed as described above, one can no longer guarantee that the branch sets have small radius, thereby destroying shallowness. In fact, applying Proposition 3.2 to fan-planar graphs, which do not have linear crossing number, is substantially more involved. However, we next describe another approach, called *colored planarization*, that also overcomes this limitation by further generalizing shallow minors to weak shallow minors.

3.3 Colored Planarization

Weak Shallow Minors In order to show product structure for k-matching planar graphs, another graph class without linear crossing number, Hendrey, Karol, and Wood [HKW25] introduce the notion of weak shallow minors. Recall that in a weak r-shallow minor H of a graph G, the branch sets are no longer required to have radius at most r. Instead, they

have *weak radius* at most r, that is, for each branch set $B \subseteq V(G)$, there exists a vertex $v \in V(G)$, not necessarily in B, such that every vertex in B has distance at most r from v.

Hendrey, Karol, and Wood [HKW25] show that an analogue of Proposition 3.2 does not hold for weak shallow minors, i.e., in general, product structure is not preserved under taking weak shallow minors. Nevertheless, the analogue does hold for graphs of bounded Euler genus, and in particular, for planar graphs.

Proposition 3.3 ([HKW25])**.** *There exists a function f such that if G is a weak r-shallow minor of $H \boxtimes K_\ell$, where H has Euler genus g, then* $\mathsf{rtw}(G) \leq f(r, \ell, g)$.

Colored Planarization. Using Proposition 3.3, Hendrey, Karol, and Wood [HKW25] show product structure for k-matching planar graphs, and Karol [Kar25] later apply a similar strategy to bounded-degree string graphs. Again, having Proposition 3.3, it only remains to show that every k-matching planar, or bounded-degree string graph, is a weak shallow minor of a planar graph. Similar to the previous approaches, both Hendrey, Karol, and Wood [HKW25] and Karol [Kar25] achieve this with a construction based on planarization. However, despite the similarity to the earlier techniques, the arguments are substantially more involved. In particular, they rely on a new technique, called *colored planarization*, introduced by Hendrey, Karol, and Wood [HKW25].

We now outline colored planarization using the example of bounded-degree string graphs, following Karol [Kar25]. We remark that on this high level, the proof for k-matching planar graphs given by Hendrey, Karol, and Wood [HKW25] essentially works in the same way. Let $\mathcal{C}$ be a string representation of a bounded-degree string graph G. First, choose a coloring $\phi\colon \mathcal{C} \to \{1, \ldots, t\}$ such that no two strings of the same color intersect, i.e., ϕ induces a proper vertex coloring of G. Assuming that G does not have any isolated vertices, they planarize $\mathcal{C}$ by placing dummy vertices in the intersection points of strings, and adding additional vertices at the endpoints of each string. We denote the resulting planar graph by G_0. The goal is then to show that G is a weak shallow minor of $G_0 \boxtimes K_\ell$ for some constant ℓ.

To that end, the coloring is used to contract certain edges along the strings of $\mathcal{C}$. More precisely, let $C_u \in \mathcal{C}$ be a string and let x_{uv} and x_{uw} be two consecutive dummy vertices along C_u corresponding to intersections of C_u with C_v and C_w, respectively. If both $\phi(C_u) \leq \phi(C_v)$ and $\phi(C_u) \leq \phi(C_w)$, then the edge $x_{uv}x_{uw}$ is contracted.

Clearly, these contractions reduce the radii of the branch sets from the earlier shallow-minor construction, but they also introduce a new difficulty. Previously, every dummy vertex corresponded to exactly two strings, yielding a K_2 factor. However, after the contractions, many strings might correspond to the same contracted dummy vertex, blowing up the necessary clique factor. Thus, the contractions are a trade-off between having small radius and having a small clique factor. Both Hendrey, Karol, and Wood [HKW25] and Karol [Kar25] achieve to show that there is a balance between these competing objectives for k-matching planar and for bounded-degree string graphs, respectively, and for every proper coloring with a constant number of colors, using the fact that only small *weak* radius is required, rather than actual small radius.

4. An Interesting Graph

In this chapter, we construct an interesting example of a family of grounded ⅂-graphs and show that it has product structure. The graph family is inspired by the lower bound construction provided by An, Oh, and Xue [AOX24], which they use to show that their upper bound of $\mathcal{O}(\log n)$ on the treewidth of sparse n-vertex outerstring graphs is tight. While their construction is a 2-directional grounded segment graph (see Figure 4.1), we construct a grounded ⅂-graph with a similar leveled structure that also has bounded degeneracy and $\mathcal{O}(\log n)$ treewidth, as we show later in Section 4.2.

4.1 The Construction

We first construct our graph family independently of any string representation, and later show that they are indeed grounded ⅂-graphs. For $L \geq 3$, we construct the graph G_L with L *levels* and $|V(G_L)| = 2^{L-1}$ vertices together with a labeling $\lambda\colon V(G_L) \to \{\mathtt{0},\mathtt{1}\}^*$ as follows. For each level $0 \leq \ell \leq L-2$, we add a path on 2^ℓ vertices $v_0^\ell, \ldots, v_{2^\ell-1}^\ell$, in this order. Then, for level $\ell = -1$, we add a single vertex v_0^{-1}, which we call the *root* of G_L (see Figure 4.2(a)). We denote by $V_\ell \coloneqq \{v_i^\ell \mid 0 \leq i < 2^\ell\}$ the set of vertices on level ℓ, and by $V_{\leq\ell}$ and $V_{\geq\ell}$ the set of vertices on level at most and least ℓ, respectively.

For $-1 \leq \ell \leq L-2$ and $0 \leq i < 2^\ell$, let $\lambda(v_i^\ell)$ be the bit string of length $\ell+1$ whose binary value is $\lfloor 2^\ell \rfloor + i$. That is, $\lambda(v_0^{-1}) = \varepsilon$ and for $0 \leq \ell \leq L-2$ and $0 \leq i < 2^\ell$, $\lambda(v_i^\ell) = \mathtt{1} \cdot w$, where w is the bit string of length ℓ representing i (see Figure 4.2). We observe that all bit string labels are unique, i.e., λ is an injective function. Moreover, $\lambda(V(G_L)) = \{\varepsilon\} \cup \{\mathtt{1}\} \cdot \bigcup_{\ell=0}^{L-2} \{\mathtt{0},\mathtt{1}\}^\ell$

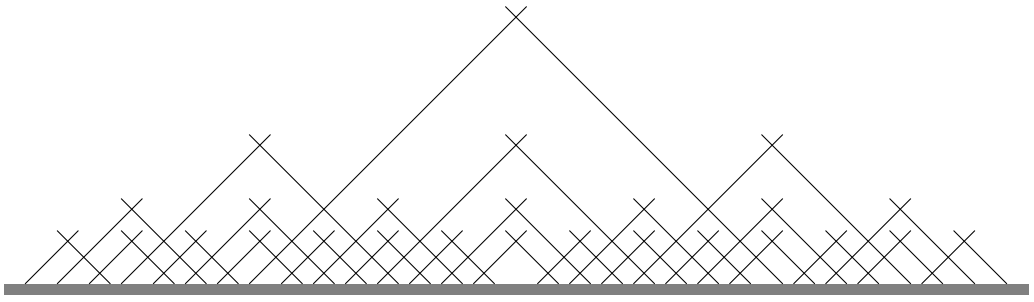
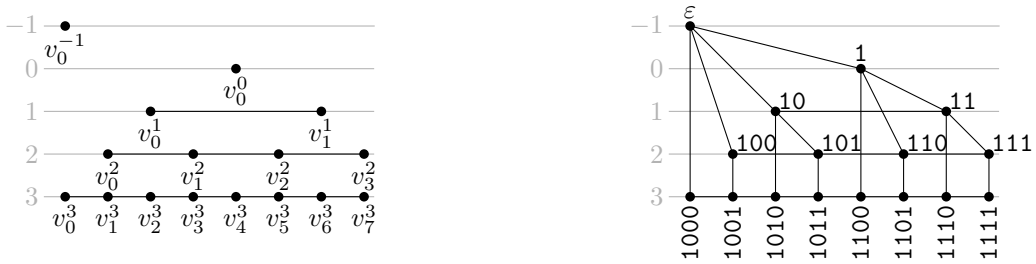


Figure 4.1: The string representation of the 2-directional grounded segment graph constructed by An, Oh, and Xue [AOX24].



(a) The paths induced by single levels of G_5.

(b) The graph G_5 with bit string labels λ.

Figure 4.2: The construction of G_5.

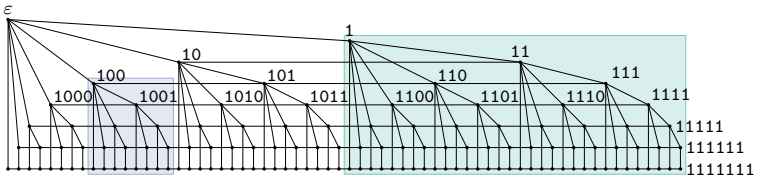


Figure 4.3: The graph G_8, where two subgraphs isomorphic to G_7 (green) and G_5 (blue), respectively, are highlighted. Deleting the leading `1` (or `100`) from the bit string labels of the vertices in this G_7 (or G_5) subgraph yields the correct labeling of G_7 (or G_5).

Now, we add edges between vertices from different levels. Let $-1 \leq \ell \leq L-2$ and let v be a vertex on level ℓ. For every $k \geq 0$ with $\ell + k \leq L-2$, we add an edge between v and the unique vertex u with $\lambda(u) = \lambda(v) \cdot \mathtt{1} \cdot \mathtt{0}^k$ (see Figures 4.2(b) and 4.3).

Note that a vertex is on level ℓ if and only if its bits string label has length $\ell + 1$, and there is an edge between two vertices on level ℓ, if and only if the binary values of their bit string labels differ by exactly 1. Let u and v be two vertices on the same level. We say that u is the *left neighbor* of v, and conversely, v is the *right neighbor* of u if and only if the binary value of $\lambda(u)$ is exactly one less than the binary value of $\lambda(v)$. More generally, we say that u is *(j steps) to the left of* v and v is *(j steps) to the right of* u if and only if the binary value of $\lambda(u)$ is exactly j less than the binary value of $\lambda(v)$. Furthermore, a vertex v lies *between* two vertices u and w if u is to the left of v and w is to the right of v, or $v = u$ or $v = w$, and v lies *strictly between* u and w if it lies between u and w, $v \neq u$, and $v \neq w$.

For two vertices u and v, where u is on level ℓ_u and v is on level ℓ_v, we say that v is on a *smaller* level than u, and u is on a *larger* level than v if and only if $\ell_v < \ell_u$. Thus, for example, a vertex v of G_8 is on a smaller level than a vertex u of G_8 if and only if it is further to the top in Figure 4.3.

We observe that every vertex has exactly one neighbor on a smaller level and exactly one neighbor on every larger level (see Figures 4.2 and 4.3). Hence, the graph induced by the edges between different levels is a tree, which we call T_L in the following. We therefore adopt standard tree terminology to describe G_L. For a vertex u and its unique neighbor v on a smaller level, we say that v is the *parent* of u and u is the *child* of v. The notions of *ancestor* and *descendant* are used as defined for trees in Chapter 2. For a vertex v, we denote by T_v the subtree of T_L rooted at v. Then, v is an ancestor of u, and conversely, u is a descendant of v, if and only if $u \in V(T_v) \setminus \{v\}$.

It is easy to see that G_L contains $G_{L'}$ for every $3 \leq L' \leq L$ as a subgraph (see Figure 4.3). In particular, for every level ℓ and every vertex v on level ℓ, the subgraph $G_L[V(T_v)]$ induced by v and its descendants is isomorphic to $G_{L-\ell-1}$. Moreover, deleting the prefix $\lambda(v)$ from every vertex in $V(T_v)$ yields the correct labeling of $G_{L-\ell-1}$.

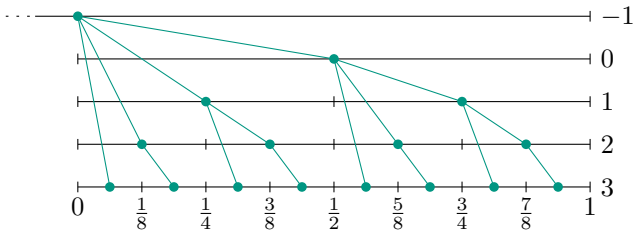


Figure 4.4: The tree T_5 (green), where each vertex is assigned an interval (black) as described in Section 4.1. Two vertices of G_5 are left/right neighbors of each other if and only if their intervals have the same length and touch. A vertex v is a parent of a vertex u if and only if the interval u begins in the center of the interval of v.

The Interval Perspective.

Another way to view the graph G_L is by associating every vertex with an interval. We do this in a way such that the intervals of vertices on the same level have the same length, and they touch if and only if the vertices are left and right neighbors of each other. Moreover, a vertex v is a parent of a vertex u if and only if the interval u begins exactly in the center of the interval of v (see Figure 4.4).

This interval perspective is particularly useful for constructing a path decomposition of G_L of width $\mathcal{O}(L)$. Indeed, we can just take one node p_i for each vertex v_i^{L-2} on the largest level $L-2$ and put every vertex whose interval overlaps with the interval of v_i^{L-2} into the bag of p_i. It is easy to see that this yields a valid path decomposition of G_L of width $\mathcal{O}(L)$. Formally, we prove the correctness of this construction in Section 4.2. Moreover, this interval perspective is also useful for constructing a grounded $\urcorner$-representation of G_L.

Let us now construct the interval assignment $\mathcal{I}\colon V(G_L) \to \{[a,b] \mid a,b \in [-1,1] \text{ and } a \leq b\}$ of G_L formally and prove its correctness. For a vertex v_i^ℓ with $0 \leq \ell \leq L-2$ and $0 \leq i < 2^\ell$, we define the interval $\mathcal{I}(v_i^\ell) \coloneqq \left[2^{-\ell} \cdot i, 2^{-\ell} \cdot (i+1)\right]$. Furthermore, we define the interval $\mathcal{I}(v_0^{-1}) \coloneqq [-1,1]$ (see Figure 4.4).

First, we observe that the interval of every vertex on a fixed level ℓ has length $2^{-\ell}$. Moreover, consecutive vertices on the same level have consecutive intervals. Thus, as desired, two vertices are the left and right neighbor of each other if and only if their intervals have the same length and they touch.

Observation 4.1. *Let $u, v \in V(G_L)$ be two vertices with intervals $\mathcal{I}(u) = [a_u, b_u]$ and $\mathcal{I}(v) = [a_v, b_v]$. Then, u is the left neighbor of v if and only if $|b_u - a_u| = |b_v - a_v|$ and $b_u = a_v$.*

Now, we show that the children of a vertex v are exactly those vertices whose interval begins in the center of the interval of v.

Lemma 4.2. *Let $u, v \in V(G_L)$ be two vertices with intervals $\mathcal{I}(u) = [a_u, b_u]$ and $\mathcal{I}(v) = [a_v, b_v]$. Then, v is the parent of u if and only if $a_u = (a_v + b_v)/2$.*

Proof. We prove both directions separately. Assume that v is the parent of u. First, consider the case that $v = v_0^{-1}$. Then $\mathcal{I}(v) = \mathcal{I}(v_0^{-1}) = [-1,1]$, and by construction, its children are exactly the vertices v_0^ℓ for $0 \leq \ell \leq L-2$. Hence, $u = v_0^\ell$ for some $0 \leq \ell \leq L-2$, and $\mathcal{I}(u) = [0, 2^{-\ell}]$. Thus, $a_u = 0 = (-1+1)/2$.

Now, suppose $v \neq v_0^{-1}$, say $v = v_i^{\ell_v}$ and $u = v_j^{\ell_u}$ for some $0 \leq \ell_v < \ell_u \leq L-2$, $0 \leq i < 2^{\ell_v}$, and $0 \leq j < 2^{\ell_u}$. Recall that $\lambda(v)$ and $\lambda(u)$ have lengths $\ell_v + 1$ and $\ell_u + 1$, respectively, and the binary values of the ℓ_v and ℓ_u least significant bits of $\lambda(v)$ and $\lambda(u)$ are exactly i and j, respectively. Furthermore, since u is a child of v, we have that $\lambda(u) = \lambda(v) \cdot \mathtt{10}^k$ for some $k \geq 0$. Then, $\ell_u = \ell_v + k + 1$ and $j = i \cdot 2^{k+1} + 2^k$.

By the definition of the intervals, $[a_v, b_v] = \mathcal{I}(v) = \mathcal{I}(v_i^{\ell_v}) = [2^{-\ell_v} \cdot i, 2^{-\ell_v} \cdot (i+1)]$ and $[a_u, b_u] = \mathcal{I}(u) = \mathcal{I}(v_j^{\ell_u}) = \mathcal{I}(v_{i \cdot 2^{k+1}+2^k}^{\ell_v+k+1})$ with

$$\begin{aligned} a_u &= 2^{-\ell_v-k-1} \cdot (i \cdot 2^{k+1} + 2^k) = 2^{-\ell_v-k-1} \cdot 2^k \cdot (2i+1) \\ &= 2^{-\ell_v-1} \cdot (2i+1) = \frac{2^{-\ell_v} \cdot (2i+1)}{2} = \frac{a_v + b_v}{2}. \end{aligned}$$

For the other direction, assume that $a_u = (a_v + b_v)/2$. Then, clearly, v is on a smaller level than u. We show that v is the parent of u. Since every vertex has a child on every larger level, v has indeed a child u' on the same level as u. We now argue that $u' = u$.

From the first direction, the interval $\mathcal{I}(u') = [a_{u'}, b_{u'}]$ of the child u' of v satisfies $a_{u'} = (a_v + b_v)/2$. Recall that for every level, all intervals of vertices on that level have distinct left endpoints (see Figure 4.4). Hence, for a fixed v, there is at most one vertex per level whose interval starts at $(a_v + b_v)/2$. It follows that $u = u'$. □

Let $uv \in E(G_L)$ be an edge. Then, either u is a neighbor of v, u is a child of v, or the roles are reversed. If u is the left neighbor of v, then by Observation 4.1, the right endpoint of $\mathcal{I}(u)$ is the left endpoint of $\mathcal{I}(v)$. If u is the child of v, then by Lemma 4.2, the left endpoint of $\mathcal{I}(u)$ is the center of $\mathcal{I}(v)$. In both cases, their intervals overlap.

Corollary 4.3. *Let $uv \in E(G_L)$ be an edge. Then, $\mathcal{I}(u) \cap \mathcal{I}(v) \neq \emptyset$.*

We remark that the converse does not hold: There are non-adjacent vertices whose corresponding intervals overlap. Hence, this interval assignment is not an interval representation in the classical sense, where two vertices are adjacent if and only if their intervals overlap. Indeed, such a representation would imply that G_L is an interval graph, which is not the case. For example, G_L contains induced cycles of length five (see Figure 4.3).

For the path decomposition of G_L that we construct in Section 4.2, we put all the vertices whose intervals overlap with the interval of a fixed vertex on the largest level into the same bag. To obtain an upper bound on the width of this path decomposition, we need an upper bound on the number of vertices whose intervals overlap with the interval of the fixed vertex. Note that the intervals get larger for smaller levels, and each two intervals on the same level are internally disjoint. Therefore, for every vertex v, at most two vertices on each smaller level have intervals that overlap $\mathcal{I}(v)$.

Observation 4.4. *Let $v \in V(G_L)$ be a vertex. Then, there are at most two vertices on every smaller level whose intervals overlap with $\mathcal{I}(v)$.*

The Grounded ⅂-Graph Perspective.

We now show that G_L is a grounded ⅂-graph by constructing a grounded ⅂-representation (see Figure 4.5). As usual, we consider the x-axis to be the grounding line, and we want all ⅂-shapes to lie above the x-axis and only touch the x-axis with their bottom endpoint. We say that the *height* of an ⅂-shape is the difference of the y-coordinates of the two endpoints, and the *width* of an ⅂-shape is the difference of the x-coordinates of the two

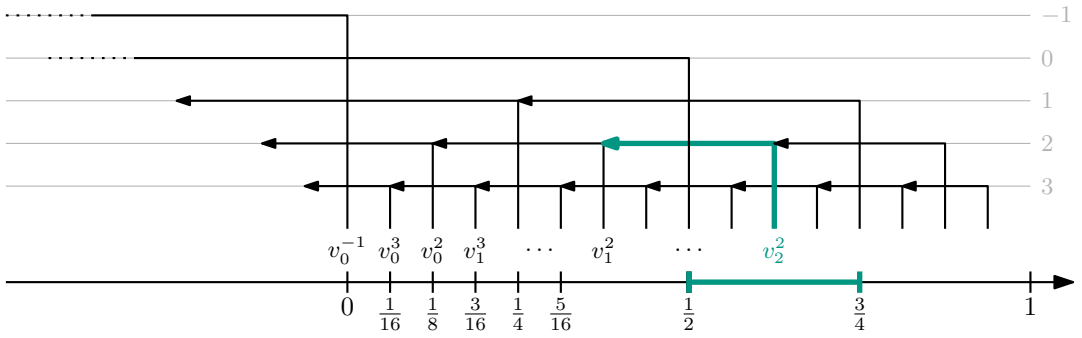


Figure 4.5: A grounded ⅂-representation of the graph G_5, where the arrow tips indicate when an ⅂-shape ends. The interval $\mathcal{I}(v_2^2)$ and the ⅂-shape corresponding to v_0^2 are both highlighted in green.

endpoints. Note that a grounded ⅂-shape is uniquely determined by the x-coordinate of its bottom endpoint, its height, and its width.

For $-1 \leq \ell \leq L-2$ and $0 \leq i < 2^\ell$, the vertex v_i^ℓ is represented by an ⅂-shape of height $L-\ell-1$ and width $2^{-\ell}$. If $v_i^\ell \neq v_0^{-1}$, the bottom endpoint of corresponding ⅂-shape has $2^{-\ell} \cdot i + 2^{-\ell-1}$ as its x-coordinate, while for v_0^{-1} the bottom endpoint is placed at $x = 0$. Recall that the interval $\mathcal{I}(v_i^\ell)$ is defined as $[2^{-\ell} \cdot i, 2^{-\ell} \cdot (i+1)]$, and $\mathcal{I}(v_0^{-1}) = [-1, 1]$. Hence, for every $-1 \leq \ell \leq L-2$ and $0 \leq i < 2^\ell$ the bottom endpoint of the ⅂-shape representing v_i^ℓ lies at the center of the interval $\mathcal{I}(v_i^\ell)$. Since $\mathcal{I}(v_i^\ell)$ has length $2^{-\ell}$, equal to the width of the corresponding ⅂-shape, ⅂-shapes corresponding to consecutive vertices on the same level touch, whereas all other ⅂-shapes of vertices on the same level are disjoint (see Figure 4.5).

Furthermore, by Lemma 4.2, a vertex v is the parent of a vertex u if and only if $\mathcal{I}(u)$ starts at the center of $\mathcal{I}(v)$. In this case, we observe that the ⅂-shape of u contains the left endpoint of $\mathcal{I}(u)$, and therefore also the center of $\mathcal{I}(v)$. Since the bottom endpoint of the ⅂-shape representing the parent v is placed at the center of $\mathcal{I}(v)$, and since the ⅂-shape of v is taller than that of its child u, the two ⅂-shapes intersect.

Conversely, the x-coordinates of the bottom endpoints are all distinct, and thus, each ⅂-shape intersects at most one ⅂-shape on every larger level (see Figure 4.5). By the argument above, these intersections correspond exactly to parent-child edges. In particular, apart from these, there are no additional intersections between ⅂-shapes corresponding to vertices on different levels. Hence, the constructed representation is a valid grounded ⅂-representation of G_L, and therefore, G_L is a grounded ⅂-graph.

4.2 Treewidth and Pathwidth

As mentioned before, An, Oh, and Xue [AOX24] proved that every bounded-degeneracy n-vertex outerstring graph has treewidth $\mathcal{O}(\log n)$. It is easy to see that G_L has degeneracy at most 2 since every vertex has at most one right neighbor and at most one parent. Indeed, ordering the vertices level by level from largest to smallest, and within each level from left to right, yields an ordering in which each vertex has at most two forward neighbors, thereby witnessing degeneracy 2. Since grounded ⅂-graphs are outerstring, it follows from the result of An, Oh, and Xue [AOX24] that G_L has treewidth in $\mathcal{O}(\log |V(G_L)|)$.

In fact, both the treewidth and the pathwidth of G_L are in $\Theta(\log |V(G_L)|)$, as we show in the following. Thus, G_L matches the aforementioned upper bound of An, Oh, and Xue [AOX24] on the treewidth of sparse outerstring graphs, analogous to their construction of a family of 2-directional grounded segment graphs.

Lemma 4.5. *For every $L \geq 3$, it holds that $L - 1 = \log_2 |V(G_L)| \leq \mathsf{tw}(G_L) \leq \mathsf{pw}(G_L) \leq 2 \log_2 |V(G_L)| = 2L - 2$.*

Proof. For the lower bound, recall that every vertex has a neighbor on every larger level. Thus, if we contract each level, we obtain a clique of size $L = \log_2 |V(G_L)| + 1$, certifying that $\mathsf{pw}(G_L) \geq \mathsf{tw}(G_L) \geq \log_2 |V(G_L)| = L - 1$.

For the upper bound, we construct a path decomposition $\mathcal{P} = (P, \mathsf{bag})$ of G_L of width $2L-2 = 2\log_2(|V(G_L)|)$. Since the treewidth of a graph is upper-bounded by its pathwidth, the claim follows. Let $P = (p_0, p_1, \ldots, p_{2^{L-2}-1})$ be a path on 2^{L-2} vertices, i.e., P has the same length as the path induced by the vertices on the largest level $L-2$ of G_L. We associate each node p_i of P with the vertex v_i^{L-2} (for $0 \leq i < 2^{L-2}$) by putting v_i^{L-2} into $\mathsf{bag}(p_i)$. Further, we add every vertex $u \in V(G_L)$ into $\mathsf{bag}(p_i)$ whose interval overlaps with the interval $\mathcal{I}(v_i^{L-2})$. By Observation 4.4, $\mathsf{bag}(p_i)$ contains at most $2L-1$ vertices, that is, at most three vertices from level $L-2$, including v_i^{L-2}, at most one vertex from levels -1 and 0, and at most two from every remaining level. Thus, the width of $\mathcal{P}$ is at most $2L-2$.

Now, we show that $\mathcal{P}$ is indeed a path decomposition of G_L, i.e., for every vertex $u \in V(G_L)$, there is a node $p \in V(P)$ with $v \in \mathsf{bag}(p)$, the subgraph of P induced by the nodes whose bags contain v is again a path, and for every edge $uv \in E(G_L)$, there is a node $p \in V(P)$ with $u, v \in V(P)$.

First, note that the intervals of vertices on any level, and in particular on level $L-2$, cover the entire interval $[0,1]$, i.e.,

$$\bigcup_{i=0}^{2^{L-2}-1} \mathcal{I}(v_i^{L-2}) \supseteq [0,1]. \tag{4.1}$$

Also note that for every vertex $u \in V(G_L)$, $\mathcal{I}(u) \cap [0,1] \neq \emptyset$. Therefore, the interval of every vertex $u \in V(G_L)$ overlaps with the interval of some vertex v_i^{L-2} on level $L-2$, implying that $u \in \mathsf{bag}(p_i)$. Furthermore, if the interval of u overlaps with the intervals of more than one vertex on level $L-2$, then those vertices are consecutive. Hence, the subgraph of P induced by the nodes whose bags contain u is a path.

It remains to show that all edges of G_L are covered by $\mathcal{P}$. To do so, let $uv \in E(G_L)$ be an edge. By Corollary 4.3, the corresponding intervals $\mathcal{I}(u)$ and $\mathcal{I}(v)$ overlap, i.e., there is an $x \in \mathcal{I}(u) \cap \mathcal{I}(v)$. Since every interval corresponding to a vertex except for $\mathcal{I}(v_0^{-1})$ is contained in $[0,1]$, it follows that $x \in [0,1]$. Then, by Equation (4.1), there is an index i with $0 \leq i < 2^{L-2}$ such that $x \in \mathcal{I}(v_i^{L-2})$, and thus, $u, v \in \mathsf{bag}(p_i)$. Thus, every edge of G_L is covered by $\mathcal{P}$, and hence, $\mathcal{P}$ is a valid path decomposition of G_L of width $2L-2$. □

4.3 Product Structure

The main goal of this section is to show that G_L has product structure. Specifically, we show the following bound on the row treewidth of G_L.

Lemma 4.6. *Let $L \geq 3$. Then, there exists a partition $\mathcal{P}$ of G_L of layered width 1 such that $G_L/\mathcal{P}$ has treewidth at most 23. In particular, $\mathsf{rtw}(G_L) \leq 23$.*

As the layering we use the BFS-layering δ originating from v_0^{-1}. In the following section, we first study the structure of shortest paths from an arbitrary vertex to v_0^{-1}. This yields a better understanding of the layering δ, which subsequently allows us to construct a partition of G_L with bounded layered width with respect to δ in Section 4.3.2.

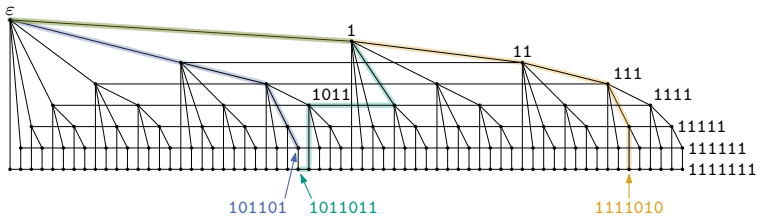


Figure 4.6: The graph G_8 with three canonical paths highlighted.

4.3.1 Canonical Shortest Paths

In this section, for every vertex $v \in V(G_L)$, we define a specific kind of path, called the *canonical path*, from v to v_0^{-1}. Specifically, depending on the bit string label of the current vertex, this path goes either from left to right or from a child to its parent, but never the other way around. We then prove that these paths are indeed shortest paths.

For two paths $P = (p_0, p_1, \ldots, p_n)$ and $Q = (q_0, q_1, \ldots, q_m)$ with $n, m \geq 0$ and $p_n = q_0$, we define their *concatenation* $P \circ Q \coloneqq (p_0, p_1, \ldots, p_n = q_0, q_1, \ldots, q_m)$. For a vertex $v \in V(G_L)$, let r_v denote the right neighbor of v and p_v its parent. We define the *canonical path* $P_c(v)$ from v to v_0^{-1} recursively as follows:

$$P_c(v) = \begin{cases} (v) & \text{if } v = v_0^{-1} \\ (v, r_v) \circ P_c(r_v) & \text{if } \lambda(v) = w \cdot \mathtt{01}^k \text{ for some } w \in \{\mathtt{0},\mathtt{1}\}^* \text{ and } k \geq 2 \\ (v, p_v) \circ P_c(p_v) & \text{otherwise} \end{cases} \tag{4.2}$$

Thus, each step of the canonical path is determined solely be the suffix of the bit string label of the current vertex: If $\lambda(v)$ ends with $\mathtt{01}^k$ for some $k \geq 2$, the canonical path proceeds to the right neighbor of v, otherwise to the parent (see Figure 4.6).

As illustrated in Figure 4.6, the canonical paths of a child and its parent may be internally disjoint, i.e., they only meet in the final vertex v_0^{-1}. Nevertheless, we show that the lengths of the canonical paths of a child and its parent differ by at most 1. As will become clear later, this is the key argument for proving that canonical paths are shortest paths.

Lemma 4.7. *Let $L \geq 3$ and let $u, v \in V(G_L)$ be two vertices such that v is the parent of u. Then, $|P_c(v)| \leq |P_c(u)| \leq |P_c(v)| + 1$.*

Proof. Let $P_c(u) = (u_0 = u, u_1, \ldots, u_{|P_c(u)|} = v_0^{-1})$ and $P_c(v) = (v_0 = v, v_1, \ldots, v_{|P_c(v)|} = v_0^{-1})$ be the canonical paths from u and v to v_0^{-1}, respectively. If u falls into the third case of Equation (4.2), then $P_c(u) = (u, v) \circ P_c(v)$, and hence $|P_c(u)| = |P_c(v)| + 1$. Thus, it suffices to consider the case that $\lambda(u) = w \cdot \mathtt{01}^k$ for some $w \in \{\mathtt{0},\mathtt{1}\}^*$ and $k \geq 2$. Since v is the parent of u, we have $\lambda(v) = w \cdot \mathtt{01}^{k-1}$. We distinguish the two cases $k \geq 3$ and $k = 2$.

Case 1. Suppose $k \geq 3$. We show that $P_c(u)$ and $P_c(v)$ meet after two steps, i.e., that $u_2 = v_2$, certifying that $|P_c(u)| = |P_c(v)|$ in this case (see Figure 4.7). First, since $\lambda(u) = w \cdot \mathtt{01}^k$ and $k \geq 3$, the first step from u goes to its right neighbor u_1, and thus, $\lambda(u_1) = w \cdot \mathtt{10}^k$. Hence, the next step from u_1 goes to its parent u_2, and thus, $\lambda(u_2) = w$.

Second, since $\lambda(v) = w \cdot \mathtt{01}^{k-1}$ and $k - 1 \geq 2$, the first step from v goes to its right neighbor v_1, and thus, $\lambda(v_1) = w \cdot \mathtt{10}^{k-1}$. Then, the next step from v_1 goes to its parent v_2, and thus, $\lambda(v_2) = w$. By uniqueness of bit string labels, $u_2 = v_2$, and hence $|P_c(u)| = |P_c(v)|$.

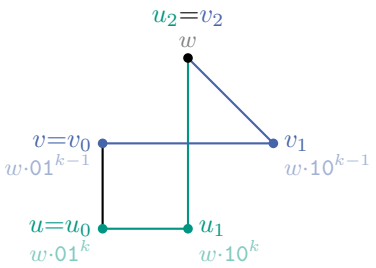


Figure 4.7: Canonical shortest paths of a parent v (blue) and its child u (green) in Case 1: If $\lambda(u) = w \cdot \mathtt{01}^k$ for some $w \in \{\mathtt{0},\mathtt{1}\}^*$ and $k \geq 3$, then $P_c(u)$ and $P_c(v)$ meet in the second vertex $u_2 = v_2$.

Case 2. Next, we consider the case that $k = 2$, i.e., $\lambda(u) = w \cdot \mathtt{011}$ and $\lambda(v) = w \cdot \mathtt{01}$. Let $a \geq 1$ be maximal such that $\lambda(u) = w' \cdot (\mathtt{011})^a$ for some $w' \in \{\mathtt{0},\mathtt{1}\}^*$, and consequently, $\lambda(v) = w' \cdot (\mathtt{011})^{a-1} \cdot \mathtt{01}$. As visualized in Figure 4.6, this is exactly the hard case, where the canonical paths of a child and its parent are internally disjoint until after the $(\mathtt{011})^a$- and $(\mathtt{011})^{a-1} \cdot \mathtt{01}$-suffixes are processed, respectively. In particular, u falls into the second case of Equation (4.2), and thus, $P_c(u)$ first goes one step to the right and then up to the parent, while v falls into the third case of Equation (4.2), and thus, $P_c(v)$ goes directly to the parent of v. Hence, for $P_c(u)$, after two steps the suffix $\mathtt{011}$ is removed and the process repeats, while the suffixes on the path $P_c(v)$ alternate between $\mathtt{01}$ and $\mathtt{10}$ and $P_c(v)$ always goes to the parent of the current vertex (see Figure 4.8(a)).

We now argue that this results in $\lambda(u_{2a}) = w'$ and $\lambda(v_{2a-1}) = w' \cdot \mathtt{0}$. Then, we need to trace the two paths $P_c(u)$ and $P_c(v)$ only a few steps further to obtain that either $u_{2a+1} = v_{2a}$ or $u_{2a+2} = v_{2a+2}$, certifying that either $|P_c(u)| = |P_c(v)| + 1$ or $|P_c(u)| = |P_c(v)|$, respectively (see Figure 4.8(a)).

Claim 4.8. *It holds that $\lambda(u_{2a}) = w'$ and $\lambda(v_{2a-1}) = w' \cdot \mathtt{0}$. In particular, $|P_c(u)|, |P_c(v)| \geq 2a$.*

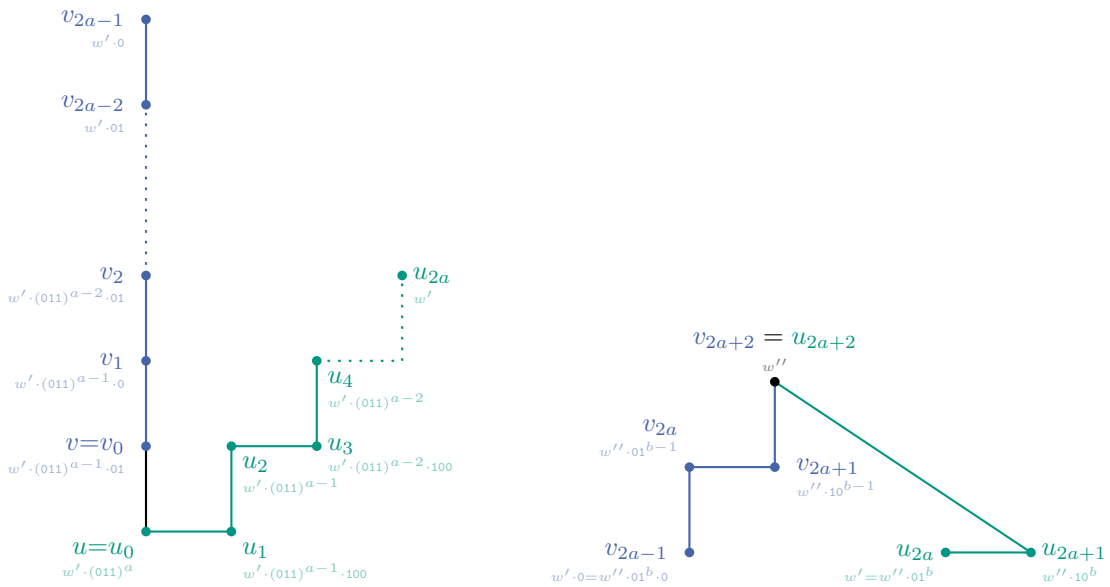
Proof of the claim. We show the following stronger statement by induction on j: For every $j = 0, \ldots, a$, $\lambda(u_{2j}) = w' \cdot (\mathtt{011})^{a-j}$, and for every $j = 0, \ldots, a-1$, $\lambda(v_{2j}) = w' \cdot (\mathtt{011})^{a-j-1} \cdot \mathtt{01}$. Then, it follows that $\lambda(u_{2a}) = w'$ and $\lambda(v_{2a-2}) = w' \cdot \mathtt{01}$, and thus, $\lambda(v_{2a-1}) = w' \cdot \mathtt{0}$. First, let $j = 0$. Then, $\lambda(u_0) = \lambda(u) = w' \cdot (\mathtt{011})^{a-0}$ and $\lambda(v_0) = \lambda(v) = w' \cdot (\mathtt{011})^{a-0-1} \cdot \mathtt{01}$.

Now, let $0 < j \leq a$. First, we consider u_{2j}. By induction, we have $\lambda(u_{2j-2}) = w' \cdot (\mathtt{011})^{a-(j-1)}$. By the definition of canonical paths, it follows that u_{2j-1} is the right neighbor of u_{2j-2}, and thus, $\lambda(u_{2j-1}) = w' \cdot (\mathtt{011})^{a-j} \cdot \mathtt{100}$. Then, again by the definition of canonical paths, u_{2j} is the parent of u_{2j-1} and $\lambda(u_{2j}) = w' \cdot (\mathtt{011})^{a-j}$.

Next, we consider v_{2j}, assuming $j < a$. By induction, we have $\lambda(v_{2j-2}) = w' \cdot (\mathtt{011})^{a-j} \cdot \mathtt{01}$. Again, tracing the canonical path $P_c(v)$ further, we obtain that v_{2j-1} is the parent of v_{2j-2}, $\lambda(v_{2j-1}) = w' \cdot (\mathtt{011})^{a-j} \cdot \mathtt{0}$, v_{2j} is the parent of v_{2j-1}, and $\lambda(v_{2j}) = w' \cdot (\mathtt{011})^{a-j-1} \cdot \mathtt{01}$. $\lhd$

Recall that the bit string label of the parent of a vertex is determined by removing the $\mathtt{10}^b$ suffix from the child's bit string label for some $b \geq 0$. Since removing the $\mathtt{10}^b$ suffix from both $\lambda(u_{2a}) = w'$ and $\lambda(v_{2a-1}) = w' \cdot \mathtt{0}$ yields the same bit string, it follows that u_{2a} and v_{2a-1} have the same parent. Since $\lambda(v_{2a-1}) = w' \cdot \mathtt{0}$, the canonical path $P_c(v)$ proceeds from v_{2a-1} to its parent, and hence, v_{2a} is the common parent of u_{2a} and v_{2a-1}.

We now distinguish into which case of Equation (4.2) u_{2a} falls. First, recall that $\lambda(u) = w' \cdot (\mathtt{011})^a$ for some $a \geq 1$. Since every bit string label except $\lambda(v_0^{-1})$ starts with $\mathtt{1}$,



(a) Claim 4.8: $\lambda(u_{2a}) = w'$ and $\lambda(v_{2a-1}) = w' \cdot \mathtt{0}$.

(b) If u_{2a} falls into the second case of Equation (4.2), then $\lambda(u_{2a}) = w'' \cdot \mathtt{01}^b$ for some $w'' \in \{\mathtt{0},\mathtt{1}\}^*$ and $b \geq 3$. In this case, $P_c(u)$ and $P_c(v)$ meet in the vertex $v_{2a+2} = u_{2a+2}$, and thus, $|P_c(u)| = |P_c(v)|$.

Figure 4.8: Canonical shortest paths of a parent v (blue) and its child u (green) in Case 2. Note: These figures are not true to scale. In particular, how far a vertex is up or down in these figures does not necessarily depend on its level. However, right neighbors are always to the right of their left neighbor and parents are always to the top of their children.

$\lambda(u) = w' \cdot (\mathtt{011})^a$ starts with $\mathtt{1}$. Thus, $\lambda(u_{2a}) = w'$ starts with $\mathtt{1}$, and in particular, is not empty. Hence, $u_{2a} \neq v_0^{-1}$ and u_{2a} falls into the second or third case of Equation (4.2).

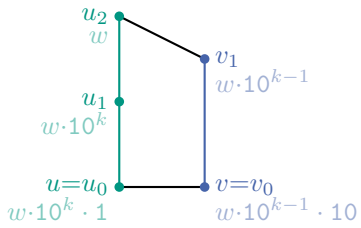
If u_{2a} falls into the third case, then u_{2a+1} is the parent of u_{2a}, i.e., $u_{2a+1} = v_{2a}$, and hence, $|P_c(u)| = |P_c(v)| + 1$. Otherwise, u_{2a} falls into the second case, and thus, $\lambda(u_{2a}) = w' = w'' \cdot \mathtt{01}^b$ for some $w'' \in \{\mathtt{0},\mathtt{1}\}$ and $b \geq 2$. Due to the maximality of a, it follows that $b \neq 2$, i.e., $b \geq 3$. Then, u_{2a+1} is the right neighbor of u_{2a}, $\lambda(u_{2a+1}) = w'' \cdot \mathtt{10}^b$, u_{2a+2} is the parent of u_{2a+1}, and $\lambda(u_{2a+2}) = w''$ (see Figure 4.8(b)).

Furthermore, we have that $\lambda(v_{2a-1}) = w' \cdot \mathtt{0} = w'' \cdot \mathtt{01}^b \cdot \mathtt{0}$. As observed before, v_{2a} is the parent of v_{2a-1}, and thus, $\lambda(v_{2a}) = w'' \cdot \mathtt{01}^{b-1}$. Since $b \geq 3$, it follows that v_{2a+1} is the right neighbor of v_{2a}, $\lambda(v_{2a+1}) = w'' \cdot \mathtt{10}^{b-1}$, v_{2a+2} is the parent of v_{2a+1}, and $\lambda(v_{2a+2}) = w''$. Due to the uniqueness of bit string labels, it follows that $u_{2a+a} = v_{2a+2}$, and thus, $|P_c(u)| = |P_c(v)|$. □

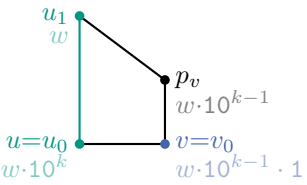
In order to show that canonical paths are shortest paths, we first need to show that the lengths of the canonical paths of left and right neighbors differ by at most 1, similarly as for parents and children. Surprisingly, this essentially follows from the result for parents and children, i.e., from Lemma 4.7.

Lemma 4.9. *For every $L \geq 3$ and every edge $uv \in E(G_L)$, $|P_c(u)|$ and $|P_c(v)|$ differ by at most 1.*

Proof. Let $L \geq 3$ and let $uv \in E(G_L)$ be an edge. Recall that all edges of G_L are either between a child and its parent or between left and right neighbors. The case that uv is a



(a) Case 1: If $\lambda(u)$ ends with $\mathtt{1}$, u_2 is the parent of v_1.



(b) Case 2: If $\lambda(u)$ ends with $\mathtt{0}$, u_1 is the grandparent of v.

Figure 4.9: If u is the left neighbor of v and $P_c(u)$ does not start to the right (i.e., $u_1 \neq v$), then either u_1 or u_2 is the grandparent of v. In both cases, we can apply Lemma 4.7 to obtain that the lengths of $P_c(u)$ and $P_c(v)$ differ by at most 1.

parent-child edge is already handled by Lemma 4.7, and hence, we only need to consider the case that u is the left neighbor of v.

Let $P_c(u) = (u_0 = u, u_1, \ldots, u_{|P_c(u)|} = v_0^{-1})$ and $P_c(v) = (v_0 = v, v_1, \ldots, v_{|P_c(v)|} = v_0^{-1})$. If u falls into the second case of Equation (4.2), then $P_c(u) = (u, v) \circ P_c(v)$, and hence $|P_c(u)| = |P_c(v)| + 1$. Thus, we can assume that u falls into the third case, i.e., $\lambda(u)$ does not end with $\mathtt{01}^k$ for any $k \geq 2$. We distinguish two cases depending on the last bit of $\lambda(u)$.

Case 1. First, suppose $\lambda(u)$ ends with $\mathtt{1}$. As every bit string label except $\lambda(v_0^{-1})$ starts with $\mathtt{1}$, so does $\lambda(u)$, and since u is not the rightmost vertex on its level (as it is the left neighbor of v), $\lambda(u)$ contains at least one $\mathtt{0}$. Hence, we can write $\lambda(u)$ as $\lambda(u) = w \cdot \mathtt{10}^k \cdot \mathtt{1}$ for some $w \in \{\mathtt{0},\mathtt{1}\}^*$ and $k \geq 1$, and consequently, $\lambda(v) = w \cdot \mathtt{10}^{k-1} \cdot \mathtt{10}$.

Tracing the canonical paths yields that u_1 is the parent of $u_0 = u$, and thus, $\lambda(u_1) = w \cdot \mathtt{10}^k$ (see Figure 4.9(a)). Furthermore, u_2 is the parent of u_1, and thus, $\lambda(u_2) = w$. Similarly, it follows that v_1 is the parent of $v_0 = v$ and $\lambda(v_1) = w \cdot \mathtt{10}^{k-1}$. Thus, the parent of v_1 is the unique vertex with label w, that is, u_2. Note that v_1 might fall into the second case of Equation (4.2), and hence, the canonical path $P_c(v)$ might proceed from v_1 to its right neighbor and not to its parent u_2. Nevertheless, we can apply Lemma 4.7 to u_2 and v_1, which yields

$$|P_c(v)| = |P_c(v_1)| + 1 = |P_c(u_2)| + 1 + \Delta = |P_c(u)| - 1 + \Delta,$$

where $\Delta \in \{0, 1\}$. Hence, $|P_c(u)|$ and $|P_c(v)|$ differ by at most 1.

Case 2. Now, we consider the case that $\lambda(u)$ ends with $\mathtt{0}$. Again, since every bit string label except $\lambda(v_0^{-1}) = \varepsilon$ contains a $\mathtt{1}$, we have that $\lambda(u) = w \cdot \mathtt{10}^k$ for some $w \in \{\mathtt{0},\mathtt{1}\}^*$ and $k \geq 1$, and consequently $\lambda(v) = w \cdot \mathtt{10}^{k-1} \cdot \mathtt{1}$. Tracing the canonical path $P_c(u)$, we obtain that u_1 is the parent of $u_0 = u$, and thus, $\lambda(u_1) = w$ (see Figure 4.9(b)). Let p_v denote the parent of v. Again, note that p_v is not necessarily contained in $P_c(v)$ since v might fall into the second case of Equation (4.2), and hence $P_c(v)$ might start to the right neighbor of v. Nevertheless, we have $\lambda(p_v) = w \cdot \mathtt{10}^{k-1}$, and the parent of p_v is the unique vertex with label w, that is, u_1.

Again, applying Lemma 4.7, first to p_v and u_1, and then to v and p_v, yields

$$|P_c(u_1)| \leq |P_c(p_v)| \leq |P_c(v)| \leq |P_c(p_v)| + 1 \leq |P_c(u_1)| + 2.$$

Since $|P_c(u_1)| = |P_c(u)| - 1$, this implies $|P_c(u)| - 1 = |P_c(u_1)| \leq |P_c(v)| \leq |P_c(u_1)| + 2 = |P_c(u)| + 1$. □

Finally, we are ready to show that canonical paths are shortest paths.

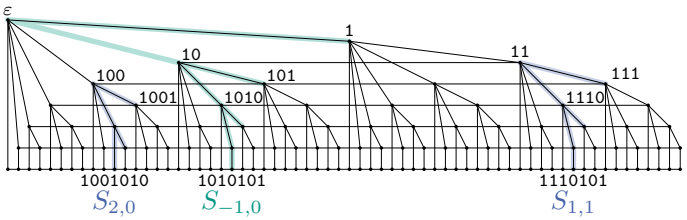


Figure 4.10: The set of vertices with alternating bit string labels (green) is a separator of G_L. This construction can be repeated recursively inside the resulting subgraphs (blue).

Lemma 4.10. *For every $L \geq 3$ and every vertex $v \in V(G_L)$, the canonical path $P_c(v)$ is a shortest path from v to v_0^{-1}.*

Proof. We prove the claim by induction on the layer $\delta(v)$ in the BFS-layering δ originating from the root v_0^{-1}. Recall that the layer $\delta(v)$ of a vertex $v \in V(G_L)$ is exactly its distance from v_0^{-1}. In particular, if $\delta(v) = 0$, then $v = v_0^{-1}$ and $P_c(v) = (v)$ has length zero. Hence, it is a shortest path.

Now, let $\delta(v) > 0$ and suppose the statement holds for all vertices with layer at most $\delta(v) - 1$. Let $P_c(v) = (p_0 = v, p_1, \ldots, p_{d'} = v_0^{-1})$, i.e., d' is the length of $P_c(v)$, and suppose for contradiction that $d' > \delta(v)$. Let $Q = (q_0 = v, q_1, \ldots, q_{\delta(v)} = v_0^{-1})$ be a shortest path from v to v_0^{-1}. Then, Q is a vertical path, and in particular, $\delta(q_1) = \delta(q_0) - 1$. Hence, by induction, $P_c(q_1)$ is a shortest path from q_1 to v_0^{-1}, and thus, $|Q| = 1 + |P_c(q_1)|$.

Since $q_0q_1 \in E(G_L)$, by Lemmas 4.7 and 4.9, the lengths of the paths $P_c(q_0)$ and $P_c(q_1)$ differ by at most 1. It follows that $\delta(v) = |Q| = 1 + |P_c(q_1)| \geq |P_c(q_0)| = |P_c(v)| = d'$, contradicting $d' > \delta(v)$. □

4.3.2 Product Structure

In this section, we finally show that G_L has product structure, that is, we prove Lemma 4.6. We use the BFS-layering δ originating from the root v_0^{-1} and construct a partition $\mathcal{P}$ that has layered width at most 2 with respect to the layering δ such that $G_L/\mathcal{P}$ has treewidth at most 11. By Proposition 2.1, this implies row treewidth at most 23 for G_L.

As a first step, we find a separator hierarchy of the graph G_L, which we later use to construct a tree decomposition of G_L in which each bag contains only a constant number of separators from that hierarchy. Those separators will be the parts of the partition $\mathcal{P}$. We then observe that each of those separators contains at most two vertices from each layer of δ, implying that $\mathcal{P}$ has layered width at most 2. Moreover, we observe that those separators, and thus, the parts of $\mathcal{P}$ are connected. Hence, $G_L/\mathcal{P}$ is a minor of G_L. Therefore, the tree decomposition of G_L with a constant number of separators per bag yields a valid constant-width tree decomposition of $G_L/\mathcal{P}$.

First, we observe that the set of vertices whose labels are alternating bit strings, that is, bit strings in $\{\texttt{10}\}^* \cdot \{\texttt{1}, \varepsilon\}$, forms a separator of G_L (see Figure 4.10). More generally, this construction can be applied recursively inside connected components. Given a connected subgraph G of G_L, starting from one of the smallest-level vertices v in G, the vertices whose bit string labels have $\lambda(v)$ as a prefix and whose remaining suffix consists of alternating `1`s and `0`s (starting with `1`), form a separator within G (see Figure 4.10). Repeating this process yields a partition of $V(G_L)$ into such separators.

Formally, for every level ℓ and every vertex v_i^ℓ on level ℓ, we define the *alternating separator rooted at* v_i^ℓ as $S_{\ell,i} \coloneqq \{v \in V(G_L) \mid \lambda(v) = \lambda(v_i^\ell) \cdot a_k \text{ for some } k \geq 0\}$, where a_k is the alternating bit string of length k starting with $\mathtt{1}$, i.e.,

$$a_k \coloneqq \begin{cases} (\mathtt{10})^{k/2} & \text{if } k \text{ is even} \\ (\mathtt{10})^{\lfloor k/2 \rfloor} \cdot \mathtt{1} & \text{otherwise.} \end{cases}$$

Thus, $S_{\ell,i}$ consists of all descendants of v_i^ℓ whose bit string labels extend $\lambda(v_i^\ell)$ by an alternating sequence of bits starting with $\mathtt{1}$ (see Figure 4.10). In particular, $S_{\ell,i}$ contains exactly one vertex on every level larger than ℓ. We say that v_i^ℓ is the *head* of $S_{\ell,i}$. We observe that every alternating separator $S_{\ell,i}$ is indeed a separator in the graph $G_L[V_{\geq \ell}]$ (see Figure 4.10).

Observation 4.11. *For every level $-1 \leq \ell \leq L-2$ and $0 \leq i < 2^\ell$, the alternating separator $S_{\ell,i}$ is a separator in the graph $G_L[V_{\geq \ell}]$ that separates the vertices to the left of $S_{\ell,i}$ from the vertices to the right of $S_{\ell,i}$.*

Consider the unique vertex v on the largest level $L-2$ in $S_{\ell,i}$. Recall that the canonical path $P_c(v)$ from v to v_0^{-1} goes to the parent in each step until it reaches v_i^ℓ since the bit string label ends with an alternating bit sequence (except if $\ell = L-1$, but then $S_{\ell,i}$ consists of only two vertices). Since the bit string label of a parent of a vertex v is determined by removing the $\mathtt{10}^k$-suffix from $\lambda(v)$ (for some $k \geq 0$), all vertices on this subpath of $P_c(v)$ again extend the bit string label of v_i^ℓ by an alternating bit string. Thus, all these vertices on the canonical path from v to v_0^{-1} that are on levels at least ℓ are contained in $S_{\ell,i}$. Additionally, for each such vertex (except v itself), $S_{\ell,i}$ contains its child on one level larger. Since canonical paths are shortest paths from an arbitrary vertex to the root v_0^{-1}, canonical paths are vertical paths with respect to the BFS-layering δ originating from v_0^{-1}. It follows that $S_{\ell,i}$ contains at most two vertices from each layer of the BFS-layering δ.

Our goal is to construct a partition $\mathcal{P}$ of G_L from this hierarchy of alternating separators. Now, clearly, these alternating separators are not pairwise disjoint, and thus, do not form a partition of G_L. For example, the alternating separator rooted at a vertex v contains the entire alternating separator rooted at its child two levels larger. We iteratively define the following partition $\mathcal{P}$ consisting of alternating separators and singleton sets of vertices: First, we put $S_{-1,0}$ into $\mathcal{P}$. Then, for every odd level ℓ, considered in ascending order, and every vertex v_i^ℓ not yet covered by any alternating separator in $\mathcal{P}$, we add the alternating separator $S_{\ell,i}$ to $\mathcal{P}$. Finally, for every vertex $v \in V(G_L)$ not contained in any alternating separator already added to $\mathcal{P}$, we add the singleton set $\{v\}$ to $\mathcal{P}$ (see Figure 4.11). By construction, the singletons only appear on even levels. We further observe that every other vertex, starting with the leftmost, on every even level is a singleton (see Figure 4.11).

By construction, every vertex $v \in V(G_L)$ is contained in some part of $\mathcal{P}$. Moreover, the parts of $\mathcal{P}$ are pairwise disjoint. Indeed, we only add alternating separators $S_{\ell,i}$ rooted at every other level to $\mathcal{P}$, and we skip any alternating separators whose root is already covered by a previously added part of $\mathcal{P}$. To see that this suffices, observe that an alternating separator rooted at a vertex on level $\ell + 2k$ is either completely contained in an alternating separator rooted at a vertex on level ℓ, or it is disjoint from all such separators (see Figure 4.11). Hence, $\mathcal{P}$ is a partition of G_L, and as observed before, $\mathcal{P}$ has layered width at most two.

Furthermore, for every odd level ℓ and for every vertex v_i^ℓ on this level, the alternating separator $S_{\ell,i}$ is either contained in $\mathcal{P}$ itself or it is a subset of some set in $\mathcal{P}$. Accordingly, for odd ℓ and $0 \leq i < 2^\ell$, we denote by $S'_{\ell,i}$ the set contained in $\mathcal{P}$ that is a (not necessarily

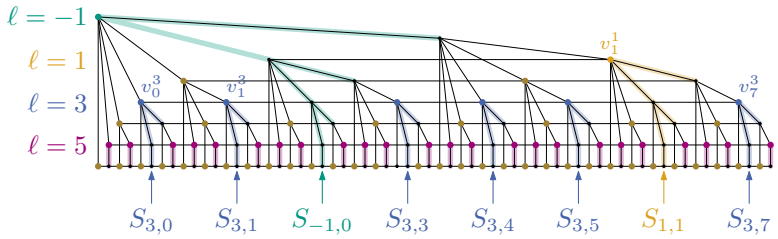


Figure 4.11: The partition $\mathcal{P}$ of the graph G_8. First, $S_{-1,0}$ (green) is added to $\mathcal{P}$. Then, $S_{1,1}$ (orange) but not $S_{1,0}$ since $S_{1,0} \subseteq S_{-1,0}$. Then, the sets $S_{3,j}$ (blue) for $j = 0, \dots, 7$ are added to $\mathcal{P}$ except for $S_{3,2}$ and $S_{3,6}$ since $S_{3,2} \subseteq S_{-1,0}$ and $S_{3,6} \subseteq S_{1,1}$. Similarly, the separators rooted at vertices on level $\ell = 5$ (purple) are added, and finally, the remaining vertices (brown) as singleton sets.

strict) superset of $S_{\ell,i}$. Finally, for a vertex $v \in V(G_L)$, we denote by $S(v)$ the unique part of $\mathcal{P}$ that contains v.

Our goal is to construct a tree decomposition $\mathcal{T}$ of G_L, in which every bag contains vertices from at most 12 parts of $\mathcal{P}$. Then, this induces a tree decomposition of the quotient $G_L/\mathcal{P}$ of width at most 11, certifying that G_L has row treewidth at most 23. The high-level idea is that we consider the subgraphs of G_L that lie between two alternating separators $S_{\ell,i-1}$ and $S_{\ell,i}$ rooted at two consecutive vertices v_{i-1}^ℓ and v_i^ℓ for every level ℓ. We then show that all such subgraphs of G_L admit a tree decomposition whose width is upper-bounded by an absolute constant.

First, we formally define the set of vertices between two alternating separators $S_{\ell,i-1}$ and $S_{\ell,i}$ rooted at two consecutive vertices v_{i-1}^ℓ and v_i^ℓ on an odd level ℓ. For this, recall that each alternating separator contains exactly one vertex on each level larger than ℓ. Then, for $1 \leq i \leq 2^\ell - 1$, we define $V_{\ell,i}$ as follows: For $k \geq 0$, $V_{\ell,i}$ contains exactly those vertices on level $\ell + k$ that lie between the unique vertices in $S_{\ell,i-1}$ and $S_{\ell,i}$ on level $\ell + k$ (see Figure 4.12). We further define $V_{\ell,0}$ to contain for every level $\ell + k$ all vertices on level $\ell + k$ that lie to the left of the unique vertex in $S_{\ell,0}$ on level $\ell + k$. Similarly, for $i = \max\{1, 2^\ell\}$ (i.e., $\ell = -1$ and $i = 1$ or $\ell > -1$ and $i = 2^\ell$) we define $V_{\ell,i}$ to contain for every level $\ell + k$ all vertices on level $\ell + k$ that are to the right of the unique vertex in $S_{\ell,i}$ on level $\ell + k$ (see Figure 4.12).

In the following, we refer to the sets $V_{\ell,i}$ as *blocks*. If $i = 0$, we say that $V_{\ell,0}$ is a *left block*, if $i = \max\{1, 2^\ell\}$, we say that $V_{\ell,i}$ is a *right block*, and otherwise, it is a *middle block*. For the sake of convenience, we set $S_{\ell,i} := \emptyset$ for all values of ℓ and i, for which $S_{\ell,i}$ is not yet defined. We now show that vertices strictly between $S_{\ell,i-1}$ and $S_{\ell,i}$, that is, the vertices in $V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$, have at most one neighbor outside $V_{\ell,i}$ (see Figure 4.12).

Lemma 4.12. *Let $L \geq 3$, $-1 \leq \ell \leq L - 2$ be odd, and let $0 \leq i \leq \max\{1, 2^\ell\}$. If $\ell \neq -1$ and $V_{\ell,i}$ is a left or middle block, then there exists at most one vertex $p_i^\ell \in V(G_L) \setminus V_{\ell,i}$ that has a neighbor in $V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$, and p_i^ℓ is the parent of v_i^ℓ. If $\ell = -1$ or $V_{\ell,i}$ is a right block, no such vertex exists.*

Proof. We prove the following equivalent statement: Every neighbor of a vertex $v \in V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$ is either in $V_{\ell,i}$ or is the parent p_i^ℓ of v_i^ℓ, where the latter case only applies if $\ell \neq -1$ and if $V_{\ell,i}$ is a left or middle block. By Observation 4.11, the alternating separators $S_{\ell,i-1}$ and $S_{\ell,i}$ are separators in $G_L[V_{\geq \ell}]$ that together separate the vertices in between them from the vertices to their left and right. Thus, no vertex $v \in V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$ has a neighbor outside $V_{\ell,i}$ on a level ℓ or larger. Hence, it only remains to show that the parent of every vertex $v \in V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$ is either on level ℓ or larger, and thus in $V_{\ell,i}$, or it is

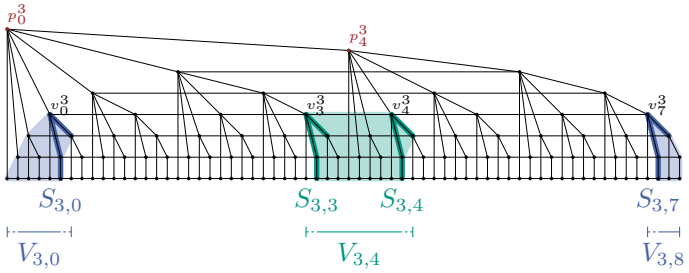


Figure 4.12: The graph G_8 with the three vertex sets $V_{3,0}$, $V_{3,4}$, and $V_{3,8}$ highlighted. Note that the parent p_0^3 of v_0^3 is the only vertex outside the left block $V_{3,0}$ that is adjacent to vertices inside $V_{3,0} \setminus S_{3,0}$, and similarly, the parent p_4^3 of v_4^3 is the only vertex outside the middle block $V_{3,4}$ that is adjacent to vertices inside $V_{3,4} \setminus (S_{3,3} \cup S_{3,4})$. For the right block $V_{3,8}$, there exists no such vertex.

the parent p_i^ℓ of v_i^ℓ, where the latter is only applicable for $\ell \neq -1$ and if $V_{\ell,i}$ is a left or middle block.

Note that v_{i-1}^ℓ and v_i^ℓ are the only vertices on level ℓ that are in $V_{\ell,i}$ (see Figure 4.12). Since $v_{i-1}^\ell \in S_{\ell,i-1}$ and $v_i^\ell \in S_{\ell,i}$, we only need to consider vertices $v \in V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$ that are on levels larger than ℓ, i.e., on some level $\ell + k$ for $k \geq 1$.

Furthermore, note that for $\ell = -1$, the statement follows trivially since every vertex of G_L is on level $\ell = -1$ or larger. Thus, from now on, we can assume that $\ell \neq -1$. We distinguish the three different cases whether $V_{\ell,i}$ is a left, middle, or right block.

Case 1. First, assume that $V_{\ell,i}$ is a left block i.e., $i = 0$, and let $v \in V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i}) = V_{\ell,0} \setminus S_{\ell,0}$ be a vertex on level $\ell + k$ for some $k \geq 1$. Recall that the head of $S_{\ell,0}$ is the vertex v_0^ℓ with bit string label $\lambda(v_0^\ell) = \mathtt{10}^\ell$, and its parent is $p_0^\ell = v_0^{-1}$. Hence, the unique vertex in $S_{\ell,0}$ on level $\ell + k$ is the vertex with bit string label $\mathtt{10}^\ell \cdot a_k$, where a_k is the alternating bit string of length k. Since v lies to the left of this vertex by definition of $V_{\ell,0}$, the binary value of $\lambda(v)$ is smaller than the binary value of $\mathtt{10}^\ell \cdot a_k$. It follows that $\lambda(v) = \mathtt{10}^\ell \cdot w$, where $w \in \{\mathtt{0},\mathtt{1}\}^*$ is a bit string of length k whose binary value is smaller than that of a_k. If $w = \mathtt{0}^k$, then $v_0^{-1} = p_0^\ell$ is the parent of v. Otherwise, the bit string w contains a $\mathtt{1}$. Since removing the $\mathtt{10}^*$ suffix from $\lambda(v)$ yields the bit string label of the parent of v, it follows that the bit string label of the parent of v has length at least $\ell + 1$. Thus, the parent of v is on level ℓ or larger.

Case 2. Next, we assume that $V_{\ell,i}$ is a middle block. Let again $v \in V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$ be a vertex on level $\ell + k$ for some $k \geq 1$. By definition of $V_{\ell,i}$, v lies strictly between the two unique vertices in $S_{\ell,i-1}$ and $S_{\ell,i}$ on level $\ell + k$, which have bit string labels $\lambda(v_{i-1}^\ell) \cdot a_k$ and $\lambda(v_i^\ell) \cdot a_k$, respectively. Since the binary value of v lies between those of $\lambda(v_{i-1}^\ell) \cdot a_k$ and $\lambda(v_i^\ell) \cdot a_k$, and a_k starts with $\mathtt{1}$, $\lambda(v)$ has either $\lambda(v_{i-1}^\ell) \cdot \mathtt{1}$, $\lambda(v_i^\ell) \cdot \mathtt{0}$, or $\lambda(v_i^\ell) \cdot \mathtt{1}$ as a prefix. If $\lambda(v)$ has $\lambda(v_{i-1}^\ell) \cdot \mathtt{1}$ or $\lambda(v_i^\ell) \cdot \mathtt{1}$ as a prefix, then v is a descendant of v_{i-1}^ℓ or v_i^ℓ, respectively. Thus, the parent of v is on level ℓ or larger.

Otherwise, $\lambda(v) = \lambda(v_i^\ell) \cdot w$, where $w \in \{\mathtt{0},\mathtt{1}\}^*$ is a bit string of length k that starts with $\mathtt{0}$. Then, similarly to Case 1, if $w = \mathtt{0}^k$, v and v_i^ℓ have the same parent p_i^ℓ, and if w contains a $\mathtt{1}$, then the parent of v lies on a level larger than ℓ.

Case 3. Finally, we consider the case that $V_{\ell,i}$ is a right block, i.e., $i = 2^\ell$. Then, the head of $S_{\ell,i-1}$ is the vertex v_{i-1}^ℓ with bit string label $\lambda(v_{i-1}^\ell) = \lambda(v_{2^\ell-1}^\ell) = \mathtt{1}^{\ell+1}$. Let $v \in V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$ be a vertex on level $\ell + k$ for some $k \geq 1$. Then, by definition of $V_{\ell,i}$, the binary value of the bit string label of v is larger than that of $\mathtt{1}^{\ell+1} \cdot a_k$, where a_k

starts with a 1. Since removing the 10^* suffix from $\lambda(v)$ yields the bit string label of the parent of v, v has v_{i-1}^ℓ as an ancestor, and thus, the parent of v is on level ℓ or larger. □

For a vertex v_i^ℓ, let p_i^ℓ be its parent. Note that if $\ell = -1$, then p_0^{-1} does not exist, and if $V_{\ell,i}$ is a right block, i.e., $i = 2^\ell$, then v_i^ℓ does not exist. We now define

$$\tilde{V}_{\ell,i} \coloneqq \begin{cases} V_{\ell,i} \cup \{p_i^\ell\} & \text{if } v_i^\ell \text{ and } p_i^\ell \text{ exist} \\ V_{\ell,i} & \text{otherwise} \end{cases}$$

and $G_{\ell,i} \coloneqq G_L[\tilde{V}_{\ell,i}]$. Recall that by Observation 4.11, every alternating separator $S_{\ell,i}$ is a separator in the induced subgraph $G_L[V_{\geq \ell}]$ that separates the vertices to its left from the vertices to its right. Hence, two consecutive alternating separators $S_{\ell,i-1}$ and $S_{\ell,i}$ together separate $V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$ from $V_{\geq \ell} \setminus V_{\ell,i}$. Now, turning from $G_L[V_{\geq \ell}]$ to G_L, by Lemma 4.12, the only vertex in $V(G_L) \setminus V_{\ell,i}$ that can have a neighbor in $V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$, thereby destroying the separator property, is the parent p_i^ℓ of v_i^ℓ. Therefore, the two alternating separators $S_{\ell,i-1}$ and $S_{\ell,i}$ together with p_i^ℓ form a separator in G_L that separates $V_{\ell,i} \setminus (S_{\ell,i-1} \cup S_{\ell,i})$ from $V(G_L) \setminus (V_{\ell,i} \cup \{p_i^\ell\})$ (see Figure 4.12). We call this set the *boundary* $B_{\ell,i}$ of $G_{\ell,i}$. Formally, we define

$$B_{\ell,i} \coloneqq \begin{cases} S_{\ell,i-1} \cup S_{\ell,i} \cup \{p_i^\ell\} & \text{if } v_i^\ell \text{ and } p_i^\ell \text{ exist} \\ S_{\ell,i-1} \cup S_{\ell,i} & \text{otherwise.} \end{cases}$$

Then, $B_{\ell,i}$ is a separator that separates $V_{\ell,i} \setminus B_{\ell,i}$ from $V(G_L) \setminus (V_{\ell,i} \cup B_{\ell,i})$.

Observation 4.13. *Let $L \geq 3$, $-1 \leq \ell \leq L-2$, and $0 \leq i < j \leq \max\{1, 2^\ell\}$. Let $uv \in E(G_L)$ be an edge with $u \in \tilde{V}_{\ell,i}$ and $v \in \tilde{V}_{\ell,j}$. Then, $u \in B_{\ell,i} \cap B_{\ell,j}$ and $v \in B_{\ell,i} \cap B_{\ell,j}$.*

Furthermore, $B_{\ell,i}$ contains vertices from at most three parts of $\mathcal{P}$, namely $S'_{\ell,i-1}$, $S'_{\ell,i}$, and $S(p_i^\ell)$. Here, recall that $S'_{\ell,i-1}$ and $S'_{\ell,i}$ denote the alternating separators contained in $\mathcal{P}$ that are supersets of $S_{\ell,i-1}$ and $S_{\ell,i}$, respectively, and that $S(p_i^\ell)$ denotes the alternating separator in $\mathcal{P}$ that contains p_i^ℓ.

Observation 4.14. *Let $L \geq 3$, $-1 \leq \ell \leq L-2$, and $0 \leq i \leq \max\{1, 2^\ell\}$. Then, $B_{\ell,i}$ contains vertices from at most three parts of $\mathcal{P}$.*

We further observe that two subgraphs $G_{\ell,i}$ and $G_{\ell,j}$ with $i \neq j$ are internally disjoint by construction of the blocks $V_{\ell,i}$ and $V_{\ell,j}$, which are separated by $B_{\ell,i}$ and $B_{\ell,j}$.

Observation 4.15. *Let $L \geq 3$, $-1 \leq \ell \leq L-2$, and $0 \leq i < j \leq \max\{1, 2^\ell\}$. Then, $\tilde{V}_{\ell,i} \cap \tilde{V}_{\ell,j} \subseteq B_{\ell,i} \cap B_{\ell,j}$.*

Our goal is to construct a tree decomposition of G_L, where each bag contains at most four such boundaries $B_{\ell,i}$. Since each such boundary contains vertices from at most three parts of $\mathcal{P}$, every bag of the tree decomposition contains vertices from at most twelve parts of $\mathcal{P}$. Recall that the partition $\mathcal{P}$ consists of alternating separators rooted at odd levels together with some singleton parts (see Figure 4.11). Let us first describe the high-level construction here before we take care of some remaining technicalities in Lemmas 4.16 and 4.17. The proof of Lemma 4.18 is a formal version of the following proof idea.

We construct the tree decomposition inductively, that is, by induction on (odd) ℓ we show that each subgraph $G_{\ell,i}$ admits such a tree decomposition. To that end, we additionally

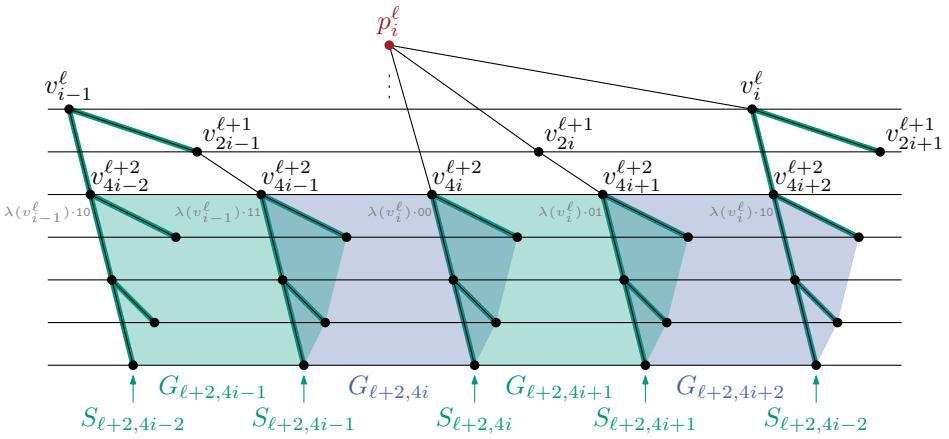


Figure 4.13: The graph $G_{\ell,i}$, obtained by taking the union of the four graphs $G_{\ell+2,4i-2+j}$ for $j = 1, \dots, 4$ and adding the vertices v_{i-1}^ℓ and $v_{2i+1}^{\ell+1}$ together with their corresponding edges. The vertices p_i^ℓ $(= p_{4i}^{\ell+2})$, v_i^ℓ $(= p_{4i+2}^{\ell+2})$, $v_{2i-1}^{\ell+1}$ $(= p_{4i-1}^{\ell+2})$, and $v_{2i}^{\ell+1}$ $(= p_{4i+1}^{\ell+2})$ are already contained in one of the subgraphs.

maintain as an invariant that the tree decomposition of $G_{\ell,i}$ has a node whose bag contains $B_{\ell,i}$. In the end, we combine the tree decompositions of $G_{-1,0}$ and $G_{-1,1}$ to a tree decomposition of G_L using that the set $S_{-1,0}$ of vertices with alternating bit string labels (the green set in Figure 4.10) is a separator that separates $G_{-1,0}$ and $G_{-1,1}$.

For the base case $\ell = L - 2$ (the largest level), we observe that $G_{L-2,i}$ contains only two vertices v_{i-1}^{L-2} and v_i^{L-2}, assuming $L - 2$ is odd, which makes the construction of a tree decomposition with the required properties trivial. For the inductive step, i.e., for odd $\ell \leq L - 4$, we consider the two vertices v_{i-1}^ℓ and v_i^ℓ that define $G_{\ell,i}$ (if $1 \leq i \leq 2^\ell - 1$, i.e., if $V_{\ell,i}$ is a middle block). Both vertices have a child on every larger level, and in particular, on level $\ell + 2$. Recall that the bit string label of a child two levels larger is obtained by appending `10`, i.e., the bit string labels of these children are $\lambda(v_{i-1}^\ell) \cdot \texttt{10}$ and $\lambda(v_i^\ell) \cdot \texttt{10}$, respectively. Since appending `10` corresponds to multiplying the index by four and then adding two, the children of v_{i-1}^ℓ and v_i^ℓ are $v_{4i-2}^{\ell+2}$ and $v_{4i+2}^{\ell+2}$, respectively. We observe that there are exactly three vertices on level $\ell + 2$ between them (see Figure 4.13). In Lemma 4.16, we show that the vertex set $\tilde{V}_{\ell,i}$ of $G_{\ell,i}$ is a subset of the union of the vertex sets $\tilde{V}_{\ell+2,4i-2+j}$ of $G_{\ell+2,4i-2+j}$ for $j = 1, \dots, 4$ together with the two additional vertices v_{i-1}^ℓ and $v_{2i+1}^{\ell+1}$ (see Figure 4.13).

We construct a tree decomposition of $G_{\ell,i}$ by combining the tree decompositions of the four subgraphs $G_{\ell+2,4i-2+j}$ (for $j = 1, \dots, 4$). Assuming inductively that each of them admits a tree decomposition, where each bag contains at most four of boundaries and one bag contains the entire boundary $B_{\ell+2,4i-2+j}$, we can do this while preserving the required properties. For this, it remains to show that the new boundary $B_{\ell,i}$ of $G_{\ell,i}$ can also be obtained by taking a subset of the union of the four boundaries $B_{\ell+2,4i-2+j}$ for $j = 1, \dots, 4$ and adding the same two additional vertices as for $\tilde{V}_{\ell,i}$, which we do in Lemma 4.17 (see Figure 4.13).

The remaining cases, i.e., for left and right blocks, are similar (see Figure 4.14). For left blocks, $G_{\ell,0}$ consists of all vertices to the left of $S_{\ell,0}$. The child of v_0^ℓ on level $\ell + 2$ is the vertex $v_2^{\ell+2}$, and there are exactly two vertices to its left, namely, $v_0^{\ell+2}$ and $v_1^{\ell+2}$ (see Figure 4.14). In this case, we combine the tree decompositions of the three subgraphs $G_{\ell+2,0}$, $G_{\ell+2,1}$, and $G_{\ell+2,2}$ to a tree decomposition of $G_{\ell,0}$, which works under the same assumptions as for middle blocks.

Similarly, for right blocks, $G_{\ell,i}$ consists of all vertices to the right of $S_{\ell,i}$. The child of v_{i-1}^{ℓ} on level $\ell+2$ is the vertex $v_{2^{\ell+2}-2}^{\ell+2}$ and there is one vertex to the right of it, namely $v_{2^{\ell+2}-1}^{\ell+2}$ (see Figure 4.14). In this case, we combine the tree decompositions of the two subgraphs $G_{\ell+2,2^{\ell+2}-1}$ and $G_{\ell+2,2^{\ell+2}}$.

Before we are able to formalize this proof, there are two things that we need to take care of. That is, for both the vertex set $\tilde{V}_{\ell,i}$ and the boundary $B_{\ell,i}$ of $G_{\ell,i}$, we need to show that it can be obtained from the vertex sets and boundaries of the subgraphs as described before. We do this in the following two lemmas, starting with the vertex set.

Lemma 4.16. *Let $L \geq 3$, $-1 \leq \ell \leq L-2$, and $1 \leq i \leq 2^{\ell}-1$. Then,*

$$\tilde{V}_{\ell,i} = \{v_{i-1}^{\ell}, v_{2i+1}^{\ell+1}\} \cup \bigcup_{j=1}^{4} \tilde{V}_{\ell+2,4i-2+j}.$$

Furthermore,

$$\begin{aligned}
\tilde{V}_{-1,0} &= \{v_0^0\} \cup \tilde{V}_{1,0},\\
\tilde{V}_{\ell,0} &= \{v_1^{\ell+1}\} \cup \bigcup_{j=0}^{2} \tilde{V}_{\ell+2,j} \text{ (for } \ell \neq -1\text{), and}\\
\tilde{V}_{\ell,\max\{1,2^{\ell}\}} &= \{v_{\max\{0,2^{\ell}-1\}}^{\ell}\} \cup \tilde{V}_{\ell+2,2^{\ell+2}-1} \cup \tilde{V}_{\ell+2,2^{\ell+2}}.
\end{aligned}$$

Proof. We only consider the case $1 \leq i \leq 2^{\ell}-1$ as the other three cases follow with similar arguments (see Figure 4.14). First, we restrict ourselves to vertices on levels at least $\ell+2$. Let v be a vertex on level $\ell+k \geq \ell+2$. By construction, the binary value of $\lambda(v)$ lies between those of $\lambda(v_{i-1}^{\ell}) \cdot a_k = \lambda(v_{4i-2}^{\ell+2}) \cdot a_{k-2}$ and $\lambda(v_i^{\ell}) \cdot a_k = \lambda(v_{4i+2}^{\ell+2}) \cdot a_{k-2}$ if and only if it lies between the binary values of $\lambda(v_{4i-3+j}^{\ell+2}) \cdot a_{k-2}$ and $\lambda(v_{4i-2+j}^{\ell+2}) \cdot a_{k-2}$ for some $j = 1, \ldots, 4$ (see Figure 4.13). Hence, for a vertex v on a level at least $\ell+2$, we have $v \in \tilde{V}_{\ell,i}$ if and only if $v \in \tilde{V}_{\ell+2,4i-2+j}$ for some $j = 1, \ldots, 4$.

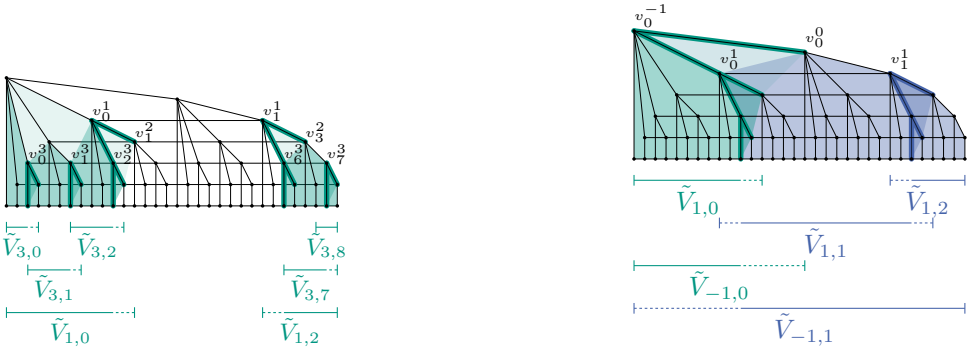
Next, we consider vertices on levels smaller than $\ell+2$. There are exactly six such vertices in $\tilde{V}_{\ell,i}$ (see Figure 4.13):

- the two vertices v_{i-1}^{ℓ} and v_i^{ℓ},
- three vertices on level $\ell+1$, namely $v_{2i-1}^{\ell+1}$, $v_{2i}^{\ell+1}$, and $v_{2i+1}^{\ell+1}$, and
- the special vertex p_i^{ℓ}.

On the right-hand side of the equation, there are also six vertices on levels smaller than $\ell+2$, namely, v_{i-1}^{ℓ}, $v_{2i+1}^{\ell+1}$, and the four special vertices $p_{4i-2+j}^{\ell+2}$ for $j = 1, \ldots, 4$. It therefore suffices to show that those four special vertices are the same vertices as the remaining vertices v_i^{ℓ}, $v_{2i-1}^{\ell+1}$, $v_{2i}^{\ell+1}$, and p_i^{ℓ} from $\tilde{V}_{\ell,i}$ (see Figure 4.13).

First, recall that v_i^{ℓ} is the parent of $v_{4i+2}^{\ell+2}$, and hence, $p_{\ell+2,4i+2} = v_i^{\ell}$ by Lemma 4.12. Similarly, $v_{2i-1}^{\ell+1}$ and $v_{2i}^{\ell+1}$ are the parents of $v_{4i-1}^{\ell+2}$ and $v_{4i+1}^{\ell+2}$, respectively, implying that $p_{\ell+2,4i-1} = v_{2i-1}^{\ell+1}$ and $p_{\ell+2,4i+1} = v_{2i}^{\ell+1}$, again by Lemma 4.12. Finally, since $\lambda(v_{4i}^{\ell+2}) = \lambda(v_i^{\ell}) \cdot \texttt{00}$, the vertices $v_{4i}^{\ell+2}$ and v_i^{ℓ} have the same parent. Hence, by Lemma 4.12, $p_{\ell+2,4i} = p_{\ell,i}$, proving the first equality. □

Next, we show that the boundary $B_{\ell,i}$ of $G_{\ell,i}$ can also be obtained by taking a subset of the union of the boundaries over the corresponding subgraphs and adding at most two additional vertices.



(a) Example for $\ell = 1$: $\tilde{V}_{1,0}$ can be obtained by taking the union of $\tilde{V}_{3,0}$, $\tilde{V}_{3,1}$, and $\tilde{V}_{3,2}$ and adding the vertex v_1^2, and $\tilde{V}_{1,2}$ can be obtained by taking the union of $\tilde{V}_{3,7}$ and $\tilde{V}_{3,8}$ and adding the vertex v_1^1.

(b) The vertex set $\tilde{V}_{-1,0}$ can be obtained by taking $\tilde{V}_{1,0}$ and adding the vertex v_0^0 (green). The vertex set $\tilde{V}_{-1,1}$ can be obtained by taking the union of $\tilde{V}_{1,1}$ and $\tilde{V}_{1,2}$ and adding the vertex v_0^{-1}.

Figure 4.14: The edge cases $\ell > -1$ and $i = 0$ or $i = 2^\ell$ (left), and $\ell = -1$ and $i = 0$ or $i = 1$ (right).

Lemma 4.17. *Let $L \geq 3$, $-1 \leq \ell \leq L-2$, and $1 \leq i \leq 2^\ell - 1$. Then,*

$$B_{\ell,i} \subseteq \{v_{i-1}^\ell, v_{2i+1}^{\ell+1}\} \cup \bigcup_{j=1}^{4} B_{\ell+2,4i-2+j}.$$

Furthermore,

$$B_{-1,0} \subseteq \{v_0^0\} \cup B_{1,0},$$
$$B_{\ell,0} \subseteq \{v_1^{\ell+1}\} \cup \bigcup_{j=0}^{2} B_{\ell+2,j} \text{ (for } \ell \neq -1\text{), and}$$
$$B_{\ell,\max\{1,2^\ell\}} \subseteq \{v^\ell_{\max\{0,2^\ell-1\}}\} \cup B_{\ell+2,2^{\ell+2}-1} \cup B_{\ell+2,2^{\ell+2}}.$$

Proof. Again, we only consider the first equation, i.e., the case $1 \leq i \leq 2^\ell - 1$ since the other three cases follow with similar arguments (see Figure 4.14). Note that in this case, the vertex v_i^ℓ and its parent p_i^ℓ exists. Recall that $B_{\ell,i} = S_{\ell,i-1} \cup S_{\ell,i} \cup \{p_i^\ell\}$. Since v_{i-1}^ℓ and v_i^ℓ are the parents of $v_{4i-2}^{\ell+2}$ and $v_{4i+2}^{\ell+2}$, respectively, and are two levels smaller, it follows that $S_{\ell+2,4i-2} \subseteq S_{\ell,i-1}$ and $S_{\ell+2,4i+2} \subseteq S_{\ell,i}$ (see Figure 4.13). More precisely, $S_{\ell,i-1} = \{v_{i-1}^\ell, v_{2i-1}^{\ell+1}\} \cup S_{\ell+2,4i-2}$ and $S_{\ell,i} = \{v_i^\ell, v_{2i+1}^{\ell+1}\} \cup S_{\ell+2,4i+2}$. Thus,

$$\begin{aligned} B_{\ell,i} &= S_{\ell,i-1} \cup S_{\ell,i} \cup \{p_i^\ell\} \\ &= S_{\ell+2,4i-2} \cup S_{\ell+2,4i+2} \cup \{p_i^\ell, v_{i-1}^\ell, v_{2i-1}^{\ell+1}, v_i^\ell, v_{2i+1}^{\ell+1}\} \\ &\subseteq B_{\ell+2,4i-1} \cup B_{\ell+2,4i+2} \cup \{p_i^\ell, v_{i-1}^\ell, v_{2i-1}^{\ell+1}, v_i^\ell, v_{2i+1}^{\ell+1}\}. \end{aligned}$$

Hence, it remains to show that $p_i^\ell, v_{2i-1}^{\ell+1}, v_i^\ell \in \bigcup_{j=1}^4 B_{\ell+2,4i-2+j}$. By looking at the bit string labels, we conclude that the parent p_i^ℓ of v_i^ℓ is also the parent of $v_{4i}^{\ell+2}$ (see Figure 4.13). Hence, $p_i^\ell = p_{4i}^{\ell+2} \in B_{\ell+2,4i}$. Similarly, we observe that $v_{2i-1}^{\ell+1}$ is the parent of $v_{4i-1}^{\ell+2}$, i.e., $v_{2i-1}^{\ell+1} = p_{4i-1}^{\ell+2} \in B_{\ell+2,4i-1}$. Finally, we know that v_i^ℓ is the parent of $v_{4i+2}^{\ell+2}$, and thus, $v_i^\ell = p_{4i+2}^{\ell+2} \in B_{\ell+2,4i+2}$. □

Now, we are finally ready to construct a tree decomposition $\mathcal{T}$ of G_L such that each bag of $\mathcal{T}$ contains vertices from at most 12 parts in $\mathcal{P}$. Since the parts in $\mathcal{P}$ are connected, i.e., $G_L/\mathcal{P}$ is a minor of G_L, this implies that there exists a tree decomposition of $G_L/\mathcal{P}$ of constant width, which we formally prove afterward in the proof of Lemma 4.6.

Lemma 4.18. *Let $L \geq 3$. There exists a partition $\mathcal{P}$ of G_L into parts of layered width at most 2 with respect to the BFS-layering δ from v_0^{-1} such that each part induces a connected subgraph of G_L, and there exists a tree decomposition $\mathcal{T} = (T, \mathsf{bag})$ of G_L such that each bag of $\mathcal{T}$ contains vertices from at most 12 parts of $\mathcal{P}$.*

Proof. As G_L is a subgraph of G_{L+1}, without loss of generality, we can assume that L is odd. Let $\mathcal{P}$ be the partition of G_L defined above. That is, $\mathcal{P}$ consists of alternating separators rooted at odd levels together with some singleton parts (see Figure 4.11). Hence, $\mathcal{P}$ has layered width at most 2 with respect to the BFS-layering δ. Recall that $\mathcal{P}$ is additionally connected, which implies that $G_L/\mathcal{P}$ is a minor of G_L. By Observation 4.14, each $B_{\ell,i}$ contains vertices from at most three parts of $\mathcal{P}$. Hence, constructing a tree decomposition of G_L in which each bag contains vertices from at most four boundaries $B_{\ell,i}$ yields the desired tree decomposition in which each bag contains vertices from at most 12 parts of $\mathcal{P}$.

We prove the following statement by induction on ℓ: For every odd $-1 \leq \ell \leq L-2$ and $0 \leq i \leq \max\{1, 2^\ell\}$, the graph $G_{\ell,i}$ admits a tree decomposition $\mathcal{T} = (T, \mathsf{bag})$ such that

(a) each bag contains vertices from at most twelve parts of $\mathcal{P}$, and

(b) there exists a node $t \in V(T)$ such that $B_{\ell,i} \subseteq \mathsf{bag}(t)$.

For the induction base, let $\ell = L-2$ be the largest level and $0 \leq i \leq \max\{1, 2^{L-2}\}$. Then, $\tilde{V}_{L-2,i} = B_{L-2,i} = \{v_{i-1}^{L-2}, v_i^{L-2}\}$. Hence, the tree decomposition consisting of a single node t with $\mathsf{bag}(t) = \{v_{i-1}^{L-2}, v_i^{L-2}\}$ satisfies all the requirements.

Now, let $\ell \leq L-4$ be odd. We only show the induction statement for middle blocks explicitly as the cases of left and right blocks follow with similar arguments. In particular, for $1 \leq i \leq 2^\ell - 1$, we show that $G_{\ell,i}$ has a tree decomposition with the required properties. By induction, the graphs $G_{\ell+2,4i-2+j}$ (for $j = 1, \ldots, 4$) each admit a tree decomposition $\mathcal{T}_j = (T_j, \mathsf{bag}_j)$ that satisfies the required properties. For $1 \leq j \leq 4$, let $t_j \in V(T_j)$ be the node with $B_{\ell+2,4i-2+j} \subseteq \mathsf{bag}(t_j)$. We construct a tree decomposition $\mathcal{T} = (T, \mathsf{bag})$ of $G_{\ell,i}$ as follows.

First, we take the disjoint union of the $\mathcal{T}_j$ for $j = 1, \ldots, 4$. Then, we add two nodes t and t', an edge between t and t', and edges between t' and t_j for $j = 0, \ldots, 3$. Finally, we set $\mathsf{bag}(t) \coloneqq B_{\ell,i}$ and $\mathsf{bag}(t') \coloneqq \bigcup_{j=0}^{3} B_{\ell+2,4i-2+j} \subseteq \bigcup_{j=0}^{3} \mathsf{bag}_j(t_j)$. Clearly, $\mathcal{T}$ has a node, namely t, with $B_{\ell,i} \subseteq \mathsf{bag}(t)$, and since each boundary contains vertices from at most three parts of $\mathcal{P}$, $\mathcal{T}$ contains vertices from at most twelve parts of $\mathcal{P}$. Hence, properties (a) and (b) are satisfied.

Next, we argue that $\mathcal{T}$ is a valid tree decomposition of $G_{\ell,i}$. First, we show that every vertex $v \in \tilde{V}_{\ell,i}$ is contained in some bag of $\mathcal{T}$. By Lemma 4.16, we know that $\tilde{V}_{\ell,i} = \{v_{i-1}^{\ell}, v_{2i+1}^{\ell+1}\} \bigcup_{j=1}^{3} \tilde{V}_{\ell+2,4i-2+j}$. Hence, every vertex of $G_{\ell,i}$ except v_{i-1}^{ℓ} and $v_{2i+1}^{\ell+1}$ is already captured by some node in one of the subtrees $\mathcal{T}_j$. Moreover, $v_{i-1}^{\ell} \in S_{\ell,i-1} \subseteq B_{\ell,i}$ and $v_{2i+1}^{\ell+1} \in S_{\ell,i} \subseteq B_{\ell,i}$, and thus, $v_{i-1}^{\ell}, v_{2i+1}^{\ell+1} \in \mathsf{bag}(t)$.

Next, we argue that for every edge $uv \in E(G_{\ell,i})$ there is a bag in $\mathcal{T}$ that contains both u and v. So, let $uv \in E(G_{\ell,i})$ be an edge. If $u, v \in \tilde{V}_{\ell+2,4i-2+j}$ for some $1 \leq j \leq 4$, then $uv \in E(G_{\ell+2,4i-2+j})$, and since $\mathcal{T}_j$ is a valid tree decomposition of $G_{\ell+2,4i-2+j}$, both u and v are in the same bag of the subtree $\mathcal{T}_j$. If $u \in \tilde{V}_{\ell+2,4i-2+j}$ and $v \in \tilde{V}_{\ell+2,4i-2+j'}$ for some $1 \leq j < j' \leq 4$, then by Observation 4.13, u or v is contained in $B_{\ell+2,4i-2+j} \cap B_{\ell+2,4i-2+j'} \subseteq \tilde{V}_{\ell+2,4i-2+j} \cap \tilde{V}_{\ell+2,4i-2+j'}$. Finally, we observe that the only vertices in $\tilde{V}_{\ell,i}$ to which the two new vertices v_{i-1}^{ℓ} and $v_{2i+1}^{\ell+1}$ are adjacent, are $v_{4i-2}^{\ell+2}$, $v_{2i-1}^{\ell+1}$, and v_i^{ℓ}, and these five vertices are all contained in $\mathsf{bag}(t)$.

It remains to show that for every vertex $v \in \tilde{V}_{\ell,i}$, the subtree of T induced by the nodes whose bags contain v is connected. By induction, this property is satisfied for the subtrees

T_j. Moreover, by Observation 4.15, the vertex sets of the subgraphs only intersect in their boundaries, and these boundaries are completely contained in their respective top-level node t_j and in $\mathsf{bag}(t')$. Furthermore, by Lemma 4.17, $\mathsf{bag}(t) = B_{\ell,i} \subseteq \bigcup_{j=1}^{4} B_{\ell+2,4i-2+j} = \mathsf{bag}(t')$. Hence, the two additional nodes t and t' do not break the connectivity property as well, which finishes the induction.

In particular, it follows that the graphs $G_{-1,0}$ and $G_{-1,1}$ have such tree decompositions $\mathcal{T}_0$ and $\mathcal{T}_1$. By construction, it holds that $V(G_L) = \tilde{V}_{-1,0} \cup \tilde{V}_{-1,1}$ (see Figure 4.10), $B_{-1,0} = B_{-1,1} = S_{-1,0}$ and $S_{-1,0}$ is a separator that separates $G_{-1,0}$ and $G_{-1,1}$. Thus, we can combine the tree decompositions $\mathcal{T}_0$ and $\mathcal{T}_1$ to a tree decomposition of G_L that satisfies properties (a) and (b) in the same way as in the induction, and the claim follows. □

Finally, we are ready to prove the main goal of this section, that is, that G_L has row treewidth at most 23, and thus, product structure.

Lemma 4.6. *Let $L \geq 3$. Then, there exists a partition $\mathcal{P}$ of G_L of layered width 1 such that $G_L/\mathcal{P}$ has treewidth at most 23. In particular, $\mathsf{rtw}(G_L) \leq 23$.*

Proof. Let δ be the BFS-layering from v_0^{-1} and let $\mathcal{P}$ be the partition from Lemma 4.18 of layered width at most two with respect to δ. Let $\mathcal{T} = (T, \mathsf{bag})$ be the tree decomposition of G_L from Lemma 4.18, where each bag contains vertices from at most twelve parts of $\mathcal{P}$. We now construct a tree decomposition $\mathcal{T}_{G_L/\mathcal{P}} = (T, \mathsf{bag}_{G_L/\mathcal{P}})$ of $G_L/\mathcal{P}$ of width at most 11 using the same tree T.

Recall that the quotient $G_L/\mathcal{P}$ has $\mathcal{P}$ as its vertex set. For every node $t \in V(T)$, we set $\mathsf{bag}_{G_L/\mathcal{P}}(t) = \{P \in \mathcal{P} \mid P \cap \mathsf{bag}(t) \neq \emptyset\}$. Since each bag of $\mathcal{T}$ contains vertices from at most 12 parts of $\mathcal{P}$, $\mathcal{T}_{G_L/\mathcal{P}}$ has width at most 11. It remains to show that $\mathcal{T}_{G_L/\mathcal{P}}$ is a valid tree decomposition of $G_L/\mathcal{P}$. Since no part in $\mathcal{P}$ is empty, every part $P \in \mathcal{P}$ is contained in some bag of $\mathcal{T}$.

For the connectivity property of $\mathcal{T}$, we first observe that for any set $S \subseteq V(G_L)$ that induces a connected subgraph $G_L[S]$ of G_L, the set of nodes $t \in V(T)$ with $S \cap \mathsf{bag}(t) \neq \emptyset$ induces a connected subtree of T. Since the parts of our partition from Lemma 4.18 are connected, the set $\{t \in V(T) \mid P \cap \mathsf{bag}(t) \neq \emptyset\} = \{t \in V(T) \mid P \in \mathsf{bag}_{G_L/\mathcal{P}}(t)\}$ induces a subtree in T. Finally, for every edge in $G_L/\mathcal{P}$ between two parts $P_1, P_2 \in \mathcal{P}$, there exists an edge $uv \in E(G_L)$ with $u \in P_1$ and $v \in P_2$. Since $\mathcal{T}$ is a valid tree decomposition of G_L, there exists a node $t \in V(T)$ with $u, v \in \mathsf{bag}(t)$, and hence, $P_1, P_2 \in \mathsf{bag}_{G_L/\mathcal{P}}(t)$.

It follows that G_L has a partition $\mathcal{P}$ of layered width two with respect to δ such that $G_L/\mathcal{P}$ has treewidth at most 11. By Proposition 2.1, G_L has a partition $\mathcal{P}'$ of layered width one with respect to the same layering δ such that $G_L/\mathcal{P}'$ has treewidth at most $2(11+1)-1 = 23$. □

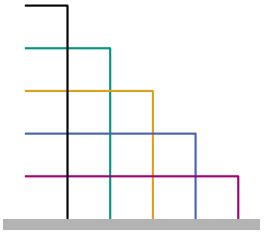
5. Bounded Horizontal & Vertical Intersections

Now that we have shown product structure for a specific family of grounded ⅂-graphs, let us return to general grounded ⅂-graphs. It is easy to see that in general, grounded ⅂-graphs do not admit product structure. For instance, arbitrarily large cliques and bicliques are grounded ⅂-graphs (see Figures 5.1(a) and 5.1(b)).

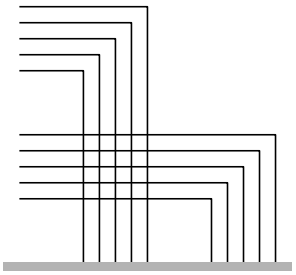
Moreover, linear local treewidth, and thus, product structure can be destroyed by adding a universal vertex, i.e., a vertex that is adjacent to every vertex of the graph, to the graphs of a graph family with unbounded treewidth. Indeed, given a grounded ⅂-representation $\mathcal{L}$ of a graph G, an ⅂-shape representing, such a universal vertex can easily be added: Take an ⅂-shape of height smaller than the height of all ⅂-shapes in $\mathcal{L}$, place its bottom endpoint to the right of all bottom endpoints in $\mathcal{L}$, and choose its width so that it spans all bottom endpoints of $\mathcal{L}$ (see Figure 5.1(c)). If the graph family has unbounded treewidth, then the neighborhood of the universal vertex has unbounded treewidth. Consequently, the resulting graph family does not have linear local treewidth, and therefore, it does not have product structure. Specifically, by Lemma 4.5, the graph G_L constructed in Chapter 4 has treewidth $\Theta(\log |V(G_L)|)$. Hence, adding a universal vertex to G_L yields a grounded ⅂-graph without product structure (see Figure 5.1(c)).

Apart from not having product structure or even linear local treewidth, these three graphs share another property that distinguishes them from the graph G_L: In each of their grounded ⅂-representations there is at least one ⅂-shape whose horizontal segment is intersected many times, and there is at least one ⅂-shape whose vertical segment is intersected many times (see Figure 5.1). We will later see that this is indeed necessary in every grounded ⅂-representation of these three graphs. For a fixed ⅂-shape, we call the intersections of its horizontal segment *horizontal intersections*, and the intersections of its vertical segment *vertical intersections*.

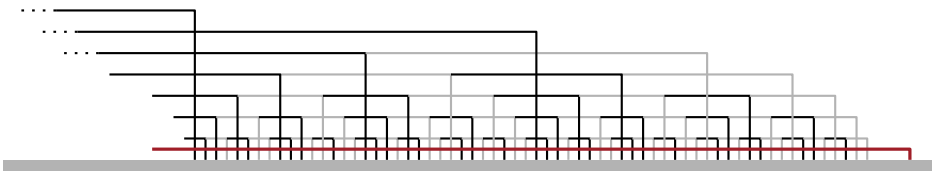
We now define *grounded k-horizontal-intersection ⅂-graphs* and *grounded k-vertical-intersection ⅂-graphs* as the classes of graphs that admit grounded ⅂-representations in which each ⅂-shape has at most k horizontal or k vertical intersections, respectively. We remark that the graph G_L constructed in Chapter 4 is a grounded 2-horizontal-intersection ⅂-graph and a grounded $(L-1)$-vertical-intersection ⅂-graph (see Figure 5.2). The complete graph K_k is both a grounded $(k-1)$-horizontal-intersection ⅂-graph and a grounded $(k-1)$-vertical-intersection ⅂-graph (see Figure 5.1(a)), and the complete bipartite graph $K_{k,k}$ is



(a) A grounded ⅂-representation of the complete graph K_5.



(b) A grounded ⅂-representation of the complete bipartite graph $K_{5,5}$.



(c) A grounded ⅂-representation of the graph G_L constructed in Chapter 4 together with a universal vertex (red). The different colors (black and gray) help to distinguish the ⅂-shapes.

Figure 5.1: Examples of grounded ⅂-graphs that do not have product structure.

both a grounded k-horizontal-intersection ⅂-graphs and a grounded k-vertical-intersection ⅂-graph (see Figure 5.1(b)).

We remark that any graph class whose graphs admit grounded ⅂-representations with both the number of horizontal and vertical intersections per ⅂-shape bounded, has bounded degree, and thus, by the result of Karol [Kar25], has product structure. In contrast, if neither the number of horizontal nor the number of vertical intersections is bounded, then we can construct arbitrarily large cliques and bicliques, and therefore, the graph class does not have product structure. Moreover, by adding a universal vertex to the construction G_L, we obtain a (bi)clique-free graph family without product structure.

This raises the question of what happens when only one of the two quantities, the number of horizontal or the number of vertical intersections per ⅂-shape, is bounded. More precisely, we are asking whether grounded k-horizontal-intersection or grounded k-vertical-intersection ⅂-graphs admit product structure, where the row treewidth may depend on k. In Section 5.2, we answer this question affirmatively for grounded k-vertical-intersection ⅂-graphs. In fact, we prove an even stronger result: Every grounded k-vertical-intersection ⅂-graphs has treewidth in $\mathcal{O}(k)$.

Then, in Section 5.3, we consider grounded k-horizontal-intersection ⅂-graphs. We show that the treewidth in the neighborhood of any vertex in a grounded k-horizontal-intersection ⅂-graph is in $\mathcal{O}(k)$. Hence, there are no such graphs with unbounded treewidth and a universal vertex. In particular, this implies that our construction G_L together with a universal vertex requires an unbounded number of horizontal intersections.

Incidentally, we are also interested in understanding how different the graph classes of grounded k-horizontal-intersection and grounded k-vertical-intersection ⅂-graphs are. Specifically, we are asking if there are functions f and g such that every grounded k-horizontal-intersection ⅂-graph is a grounded $f(k)$-vertical-intersection ⅂-graph, and symmetrically, such that every grounded k-vertical-intersection ⅂-graph is a grounded $g(k)$-horizontal-intersection ⅂-graph. If such functions existed, then results for one graph class would automatically transfer to the other, up to the corresponding function.

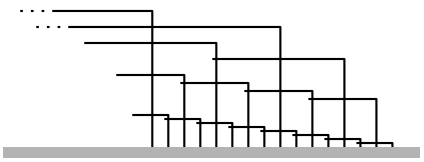


Figure 5.2: A grounded ⅂-representation of G_5, where the ⅂-shapes are of distinct heights. Note that each ⅂-shape has at most two horizontal intersections.

As we show in Section 5.1, such an equivalence is true for $k = 1$. However, due to the aforementioned bound on the treewidth of grounded k-vertical-intersection ⅂-graphs, G_L is already a counterexample for the first inclusion, even for $k = 2$. More precisely, we later show that grounded 1-horizontal-intersection and grounded 1-vertical-intersection ⅂-graphs are exactly the same, whereas there exist grounded 2-horizontal-intersections ⅂-graphs requiring arbitrarily many vertical intersections, and grounded 2-vertical-intersection ⅂-graphs requiring arbitrarily many horizontal intersections.

5.1 Basic Observations

Without loss of generality, we can assume that the ⅂-shapes of any grounded ⅂-representation are of distinct heights. Indeed, given a grounded ⅂-representation $\mathcal{L}$ that contains $p \geq 2$ ⅂-shapes of the same height h, we can do the following: Let h' be the largest height of any ⅂-shape in $\mathcal{L}$ strictly smaller than h, or $h' = 0$ if h is the minimum height. Let $L_0, \ldots, L_{p-1}$ be the ⅂-shapes of height h, ordered from left to right according to the x-coordinate of their bottom endpoints. Then, we choose a value $y > 0$ such that $y \cdot (p-1) < h - h'$ and decrease the height of every ⅂-shape L_i by $y \cdot i$ for $i = 0, \ldots, p-1$. This process clearly does not create or destroy any intersections of ⅂-shapes and applying it to every set of at least two ⅂-shapes of the same height yields a grounded ⅂-representation of the same graph, where all ⅂-shapes have distinct heights (see Figure 5.2).

Under this assumption, we observe that for a fixed ⅂-shape, its horizontal intersections are exactly those intersections with taller ⅂-shapes, and its vertical intersections are exactly those intersections with shorter ⅂-shapes. In particular, it follows that the classification of the intersections of a fixed ⅂-shape into horizontal and vertical intersections is a partition.

For another perspective on these types of intersections, consider the order of the ⅂-shapes from left to right according to the x-coordinates of their bottom endpoints. Then, for a fixed ⅂-shape, its horizontal intersections are exactly those intersections with ⅂-shapes that are to the left of it, and its vertical intersections are exactly those intersections with ⅂-shapes that are to the right of it. For a fixed representation, we transfer this order to the corresponding vertices, and say that a vertex u is *to the left of* a vertex v, and conversely, v is *to the right of* u, if the bottom endpoint of the ⅂-shape of u is to the left of the bottom endpoint of the ⅂-shape of v. If additionally uv is an edge, we say that u is a *left neighbor* of v and v is a *right neighbor* of u. Hence, the left neighbors of a vertex u correspond precisely to the horizontal intersections of the ⅂-shape of u, and similarly, the right neighbors of u correspond to the vertical intersections of the ⅂-shape of u.

From this perspective it is easy to see that for every $k \geq 0$, every grounded k-horizontal-intersections and every grounded k-vertical-intersection ⅂-graph G has degeneracy at most k. Indeed, ordering the vertices of G from left to right according to the bottom endpoints of their ⅂-shapes yields a degeneracy ordering witnessing degeneracy at most k since each vertex has at most k right neighbors or at most k left neighbors, respectively.

Observation 5.1. *For every $k \geq 0$, every grounded k-horizontal-intersection and every grounded k-vertical-intersection ⅂-graph has degeneracy at most k.*

Since the complete graph K_k and the complete bipartite graphs $K_{k,k}$ have degeneracy $k-1$ and k, respectively, $(k-1)$ or k horizontal and vertical intersections are indeed necessary, respectively.

Corollary 5.2. *For every $k \geq 1$, in every grounded $\sqcap$-representation of the complete graph K_k, there is an $\sqcap$-shape with $k-1$ horizontal (vertical) intersections, and in every grounded $\sqcap$-representation of the complete bipartite graph $K_{k,k}$, there is an $\sqcap$-shape with k horizontal (vertical) intersections.*

In particular, the complete graph K_k is neither a grounded $(k-2)$-horizontal-intersection nor a grounded $(k-2)$-vertical-intersection $\sqcap$-graph, and the complete bipartite graph $K_{k,k}$ is neither a grounded $(k-1)$-horizontal-intersection nor a grounded $(k-1)$-vertical-intersection $\sqcap$-graph. We further remark that Observation 5.1 also implies that grounded 0-horizontal-intersection and grounded 0-vertical intersection $\sqcap$-graphs are edgeless and grounded 1-horizontal-intersection and grounded 1-vertical-intersection $\sqcap$-graphs are forests. We now show that every forest is also a grounded 1-horizontal-intersection and a grounded 1-vertical-intersection $\sqcap$-graph.

Lemma 5.3. *The following are equivalent for a graph G:*

(a) G is a forest.

(b) G is a grounded 1-horizontal-intersection $\sqcap$-graph.

(c) G is a grounded 1-vertical-intersection $\sqcap$-graph.

Proof. By Observation 5.1, it follows that both (b) and (c) imply (a), respectively. Hence, it only remains to show that (a) implies (b) and (c). Moreover, it clearly suffices to prove the claim for trees since $\sqcap$-representations of different connected components can be placed side by side without creating additional intersections. We therefore show that every tree T admits both a grounded 1-horizontal-intersection and a grounded 1-vertical-intersection $\sqcap$-representation.

To this end, let T be a rooted tree. We inductively construct both a grounded 1-horizontal-intersection and a grounded 1-vertical-intersection $\sqcap$-representation of T. As invariants, we require that in our constructed grounded 1-horizontal-intersection $\sqcap$-representation, the root of T corresponds to the leftmost (and thus, tallest) $\sqcap$-shape and does not have any horizontal intersections. Similarly, we require that in the grounded 1-vertical-intersection $\sqcap$-representation, the root corresponds to the rightmost (and thus, shortest) $\sqcap$-shape and does not have any vertical intersections. Note that it is indeed necessary to have two distinct representations. Indeed, if a single representation would satisfy that both the number of horizontal and vertical intersections of each $\sqcap$-shape is at most 1, the represented graph would have maximum degree at most 2, and hence, would be a path.

For the base case, suppose that T has height 0. Then, T consists of a single vertex, which can be represented by a single $\sqcap$-shape in both settings.

Now, suppose that T has height $h \geq 1$. Removing the root of T yields rooted subtrees $T_1, \ldots, T_p$, each of height smaller than h. By induction, each T_i admits both a grounded 1-horizontal-intersection and a grounded 1-vertical-intersection $\sqcap$-representation. We now combine those representations to obtain the desired representations of T, starting with the 1-horizontal case.

Let $\mathcal{L}_1, \ldots, \mathcal{L}_p$ denote the grounded 1-horizontal-intersection $\sqcap$-representations of $T_1, \ldots, T_p$, respectively. First,we scale the representation so that every $\sqcap$-shape in $\mathcal{L}_i$ is taller than every

(a) Combining grounded 1-horizontal-intersection ⅂-representations of the subtrees $T_1, \ldots, T_p$.

(b) Combining grounded 1-vertical-intersection ⅂-representations of the subtrees $T_1, \ldots, T_p$.

Figure 5.3: Every tree is a grounded 1-horizontal-intersection and a grounded 1-vertical-intersection ⅂-graph.

⅂-shape in $\mathcal{L}_{i-1}$ for each $2 \leq i \leq p$ (see Figure 5.3(a)). Next, we place the representations from left to right so that every ⅂-shape in $\mathcal{L}_i$ lies to the right of every ⅂-shape in $\mathcal{L}_{i-1}$. We then add an ⅂-shape representing the root of T that is taller than and to the left of any other previously placed ⅂-shape. Finally, for every $1 \leq i \leq p$, we extend the horizontal segment of the ⅂-shape corresponding to the root of T_i until it intersects the ⅂-shape representing the root of T (see Figure 5.3(a)).

By the invariant, the root of each subtree is represented by its leftmost ⅂-shape. Hence, due to the scaling and ordering of the representations, the roots of the subtrees T_i are each taller than any ⅂-shape to their left, except for the one representing the root of T. It follows that extending the widths of these ⅂-shapes does not create any unintended intersections. Therefore, the represented graph is indeed the tree T. Moreover, since the roots of the subtrees did not have any horizontal intersections before, they now have exactly one horizontal intersection each. Hence, the representation is still grounded 1-horizontal, and the invariant is clearly satisfied.

Similarly, let $\mathcal{L}'_1, \ldots, \mathcal{L}'_p$ denote the grounded 1-vertical-intersection ⅂-representations of $T_1, \ldots, T_p$, respectively. Again, we first scale the representations and then place them next to each other. However, in this case, we scale them so that all ⅂-shapes in $\mathcal{L}_i$ are shorter than all ⅂-shapes in $\mathcal{L}_{i-1}$ for $2 \leq i \leq p$ (see Figure 5.3(b)). Then, we place them next to each other so that $\mathcal{L}_i$ is strictly to the right of $\mathcal{L}_{i-1}$. The ⅂-shape representing the root of T is now shorter than any other ⅂-shape and completely to the right of the representation. Again, we extend the width of the ⅂-shapes corresponding to the roots of the subtrees T_i. However, this time we keep the position of the top left endpoint and move the ground point to the right until it intersects the ⅂-shape corresponding to the root of T. We do this in a way so that the ⅂-shapes corresponding to the roots of the subtrees T_i nest, and in particular, do not intersect each other. Again, it is easy to see that due to the invariant, the scaling, and the ordering, this indeed yields a valid grounded 1-vertical-intersection ⅂-representation of T satisfying the required invariant. □

In contrast to this result, we later show that already for $k = 2$, there are grounded 2-horizontal-intersection ⅂-graphs that require arbitrarily many vertical intersections in every representation (see Corollary 5.5), and conversely, there are grounded 2-vertical-intersection ⅂-graphs that require arbitrarily many horizontal intersections in every representation (see Lemma 5.11).

5.2 Bounded Vertical Intersections

In this section, we consider grounded ⅂-graphs, where the number of vertical intersections is bounded. Recall that this condition implies that every vertex has only a bounded number

of right neighbors. We show that every graph admitting a grounded ⅂-representation in which each ⅂-shape has at most k vertical intersections has treewidth $\mathcal{O}(k)$.

The key idea of the proof is to consider the tallest ⅂-shape in a given representation together with all of its right neighbors. We observe that this set forms a separator between the ⅂-shapes lying to the left of the tallest ⅂-shape and those lying to its right. Indeed, any ⅂-shape on the right that intersects an ⅂-shape on the left also intersects the vertical segment of the tallest ⅂-shape, and therefore, belongs to the separator as a right neighbor of the tallest ⅂-shape. The remaining technical challenge is then to show that this process can be repeated recursively, yielding a tree decomposition of bounded width.

Theorem 5.4. *For every $k \geq 0$, every grounded k-vertical-intersection ⅂-graph has treewidth at most $3k + 2$.*

Proof. We describe a recursive procedure that, given a graph G together with a grounded k-vertical-intersection ⅂-representation $\mathcal{L}$ of G, constructs a tree decomposition of G of width at most $3k + 2$. For the sake of convenience, we assume that G contains two isolated vertices v_L and v_R whose ⅂-shapes are both taller than any other ⅂-shape in $\mathcal{L}$ such that the ⅂-shape corresponding to v_L lies completely to the left of any other ⅂-shape in $\mathcal{L}$, and the ⅂-shape corresponding to v_R lies completely to and right of any other ⅂-shapes in $\mathcal{L}$.

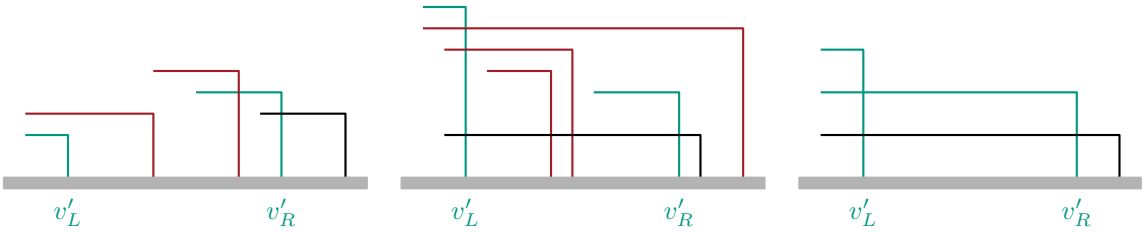
The recursive procedure is additionally given a set S' consisting of two vertices v'_L and v'_R, where v'_L lies to the left of v'_R. Here, recall that a vertex lies u to the left of another vertex v if the bottom endpoint of the ⅂-shape corresponding to u lies to the left of the bottom endpoint of the ⅂-shape corresponding to v. Given S', we define two other sets $S(S')$ and $V(S')$ as follows. The set $S(S')$ consists of v'_L and v'_R and all of their right neighbors, and $V(S')$ consists of $S(S')$ together with all vertices whose ⅂-shapes lie between those of v'_L and v'_R.

The procedure works only under the assumption that S' satisfies the following invariants that are illustrated in Figure 5.4. Firstly, the ⅂-shape of v'_R is taller than any other ⅂-shape corresponding to a vertex in $V(S')$, except possibly that of v'_L (see Figure 5.4(b)). Secondly, the ⅂-shape of v'_L is taller than any other ⅂-shape corresponding to a vertex in $V(S')$, except possibly that of v'_R and those of right neighbors of v'_R (see Figure 5.4(a)). In particular, the ⅂-shape corresponding to v'_L is taller than every ⅂-shape corresponding to vertices strictly between v'_L and v'_R.

The main idea behind these invariants is that the set $S(S')$ is a separator, defined by two vertices v'_L and v'_R with rather tall ⅂-shapes, that separates $G[V(S') \setminus S(S')]$ from the rest of G. We remark that v'_L and v'_R may have common right neighbors, and even v'_R itself may be a right neighbor of v'_L (see Figures 5.4(b) and 5.4(c)). We further remark that every vertex in $V(S') \setminus S'$ lies strictly between v'_L and v'_R or is a right neighbor of v'_R.

Given such a set S', the algorithm outputs a rooted tree decomposition $\mathcal{T}(S')$ of $G[V(S')]$ of width at most $3k + 2$ whose root bag contains $S(S')$. The initial call uses $S' = \{v_L, v_R\}$. Since v_L and v_R are isolated and their ⅂-shapes lie on opposite sides of $\mathcal{L}$, we have $S(S') = \{v_L, v_R\}$ and $V(S') = V(G)$. Furthermore, the invariants are satisfied due to the assumption that the ⅂-shapes corresponding to v_L and v_R are taller than any other ⅂-shape in $\mathcal{L}$.

We now describe how the algorithm proceeds, given a set $S' = \{v'_L, v'_R\}$ that satisfies the required invariants. For the base case, if $S(S') = V(S')$, then $\mathcal{T}(S')$ consists of a single node t whose bag contains $V(S')$. Since every vertex in $S(S')$, except v'_L and v'_R themselves, is a right neighbor of either v'_L or v'_R, and each vertex has at most k right neighbors, it holds that $|V(S')| = |S(S')| \leq 2k + 2$. Hence, $\mathcal{T}(S')$ has width at most $2k + 1$.



(a) The ⅂-shapes corresponding to v'_R and its right neighbors are the only ones that are allowed to be taller than the ⅂-shape corresponding to v'_L.

(b) The ⅂-shape corresponding to v'_L is the only ⅂-shape that is allowed to be taller than the ⅂-shape of v'_R. The vertices v'_L and v'_R are allowed to share left and right neighbors.

(c) The vertex v'_R may be a right neighbor of v'_L. In this case, they may still share a right neighbor.

Figure 5.4: Examples of choices for the set $S' = \{v'_L, v'_R\}$, where the red ⅂-shapes would violate the invariants.

Now, suppose $S(S') \subsetneq V(S')$. Let $v \in V(S') \setminus S'$ be the vertex with the tallest ⅂-shape that is not a right neighbor of v'_R. In particular, v lies strictly between v'_L and v'_R, and it may be a right neighbor of v'_L. We define $S'_L \coloneqq \{v'_L, v\}$ and $S'_R \coloneqq \{v, v'_R\}$. By the choice of v, both S'_L and S'_R satisfy the invariant above, and therefore, the recursive calls produce tree decompositions $\mathcal{T}(S'_L)$ and $\mathcal{T}(S'_R)$ of width at most $3k + 2$.

To construct $\mathcal{T}(S')$, we take the disjoint union of $\mathcal{T}(S'_L)$ and $\mathcal{T}(S'_R)$, and add two new vertices r and t, where r becomes the root of $\mathcal{T}(S')$. We add an edge between r and t, and between t and the two roots of $\mathcal{T}(S'_L)$ and $\mathcal{T}(S'_R)$. The bag of r is $S(S')$ and the bag of t is $S(S'_L) \cup S(S'_R)$. We observe that the latter set consists of the vertices v'_L, v, and v'_R, together with all of their right neighbors. Since each vertex has at most k right neighbors, we have $|S(S'_L) \cup S(S'_R)| \leq 3k + 3$. Since $S(S') \subseteq S(S'_L) \cup S(S'_R)$, the resulting tree decomposition has width at most $3k + 2$.

We now prove correctness of the construction. First, we show that every vertex in $V(S')$ is contained in some bag of $\mathcal{T}(S')$. It suffices to show that $V(S') = V(S'_L) \cup V(S'_R)$, since then, every vertex in $V(S')$ is already covered by either $\mathcal{T}(S'_L)$ or $\mathcal{T}(S'_R)$. Recall that since v is not a right neighbor of v'_R, v lies strictly between v'_L and v'_R. Hence, any other vertex of $V(S')$ that lies between v'_L and v'_R lies either between v'_L and v, and thus, in $V(S'_L)$, or between v and v'_R, and thus, in $V(S'_R)$. Furthermore, every vertex in $V(S')$ that is not between v'_L and v'_R is a right neighbor of v'_R, and therefore included in $S(S'_R) \subseteq V(S'_R)$.

Next, we argue that for every vertex $u \in V(S')$, the set of nodes of $\mathcal{T}(S')$ whose bags contain u induces a connected subtree. By induction, this property holds for both $\mathcal{T}(S'_L)$ and $\mathcal{T}(S'_R)$. Therefore, it suffices to consider vertices that are contained in both $V(S'_L)$ and $V(S'_R)$. Let $u \in V(S'_L) \cap V(S'_R)$. We argue that $u \in S(S'_L) \cap S(S'_R)$. From this, it follows that u is contained in the root bags of both $\mathcal{T}(S'_L)$ and $\mathcal{T}(S'_R)$, as well as in the bag of the newly added vertex t, which is adjacent to both roots. In particular, t connects the two connected components of $\mathcal{T}(S'_L)$ and $\mathcal{T}(S'_R)$ induced by the nodes whose bags contain u to one single connected component of $\mathcal{T}(S')$.

To show that $u \in S(S'_L) \cap S(S'_R)$, we first show that $u \in S(S'_L)$. Suppose for contradiction that $u \notin S(S'_L)$. Then, since $u \in V(S'_L)$, u lies strictly between v'_L and v. However, every vertex in $V(S'_R)$, except v itself, lies to the right of v, contradicting that $u \in V(S'_R)$.

Similarly, suppose for contradiction that $u \notin S(S'_R)$. Then, u lies strictly between v and v'_R. However, the only vertices of $V(S'_L)$ that are to the right of v are the right neighbors of v. Since $u \in V(S'_L)$, it follows that u is a right neighbor of v, contradicting $u \notin S(S'_R)$.

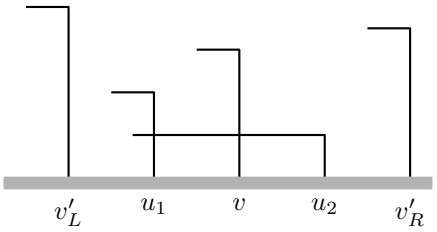


Figure 5.5: If there is an edge u_1u_2, where v lies between u_1 and u_2, then u_2 is a right neighbor of v.

Finally, it remains to argue that for every edge $u_1u_2 \in E(G[V(S')])$, there exists a bag of $\mathcal{T}(S')$ containing both u_1 and u_2. Again, by induction, every edge with both endpoints in $V(S'_L)$ or both endpoints in $V(S'_R)$ is already covered by $\mathcal{T}(S'_L)$ or $\mathcal{T}(S'_R)$, respectively. Therefore, it suffices to consider edges u_1u_2 with $u_1 \in V(S'_L)$ and $u_2 \in V(S'_R)$. If $u_1 \in S(S'_L)$ and $u_2 \in S(S'_R)$, then both vertices are contained in the bag of the newly added node t, and thus, the edge is covered.

Now, suppose that $u_1 \notin S(S'_L)$, i.e., u_1 lies between v'_L and v. Since $u_2 \in V(S'_R)$, either $u_2 = v$ or u_2 lies to the right of v. In both cases, u_2 lies to the right of u_1, and since the edge u_1u_2 exists, u_2 is a right neighbor of u_1. It follows that the $\urcorner$-shape of u_2 is shorter than the $\urcorner$-shape of u_1. Note that this also implies that $u_2 \neq v$ by the choice of v. Thus, v lies between u_1 and u_2 and its $\urcorner$-shape is taller than those of u_1 and u_2. It follows that the $\urcorner$-shape of u_2 intersects the $\urcorner$-shape of v on its way to the $\urcorner$-shape of u_1 (see Figure 5.5). Therefore, u_2 is a right neighbor of v, and thus, $u_2 \in S(S'_L) \subseteq V(S'_L)$, and the edge u_1u_2 is covered by $\mathcal{T}(S'_L)$.

Similarly, suppose that $u_2 \notin S(S'_R)$, i.e., u_2 lies between v and v'_R. If $u_1 = v$ or u_1 is a right neighbor of v, then $u_1 \in S(S'_R) \subseteq V(S'_R)$, and the edge u_1u_2 is covered by $\mathcal{T}(S'_R)$. Hence, we may assume that $u_1 \neq v$ and that u_1 is not a right neighbor of v. It follows that u_1 lies to the left of both v and u_2, and therefore, either $u_1 = v'_L$ or u_1 lies between v'_L and v. In the first case, since u_1u_2 is an edge and u_2 is to the right of $u_1 = v'_L$, u_2 is a right neighbor of v'_L. It follows that $u_2 \in S(S'_L) \subseteq V(S'_L)$, and thus, the edge u_1u_2 is covered by $\mathcal{T}(S'_L)$. In the latter case, when u_1 lies between v'_L and v, then again, the $\urcorner$-shape of v is between those of u_1 and u_2 and it is taller (see Figure 5.5). Consequently, u_2 is a right neighbor of v, which implies that $u_2 \in S(S'_R)$, contradicting our assumption. □

By Lemma 4.5, the graph G_L has treewidth at least $L-1$ for every $L \geq 3$. Therefore, if G_L admits a grounded k-vertical-intersection $\urcorner$-representation, then $L - 1 \leq \mathsf{tw}(G) \leq 3k + 2$, and hence, $k \geq (L-3)/3$. Recall that G_L is a grounded 2-horizontal-intersection $\urcorner$-graph. Consequently, there exist grounded $\urcorner$-graphs that require only few horizontal intersections per $\urcorner$-shape, but arbitrarily many vertical intersections per $\urcorner$-shape in every representation.

Corollary 5.5. *For every $L \geq 3$, the graph G_L is not a grounded k-vertical-intersection $\urcorner$-graph for any $k < (L-3)/3$. In particular, for every $k \geq 0$, there exists a grounded 2-horizontal-intersection $\urcorner$-graph that is not a grounded k-horizontal-intersection $\urcorner$-graph.*

In the following section, we show that the converse holds as well. That is, there exist grounded $\urcorner$-graphs that require only a few vertical intersections per $\urcorner$-shape, but arbitrarily many horizontal intersections per $\urcorner$-shape in every representation (see Lemma 5.11).

We further remark that an analogue of Theorem 5.4 does not hold for the pathwidth. In particular, by Lemma 5.3, every tree is a grounded 1-vertical-intersection $\urcorner$-graph.

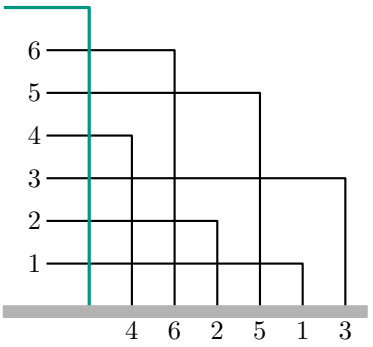


Figure 5.6: The vertices corresponding to the right neighborhood of the green ⅂-shape form a permutation graph.

Since trees can have arbitrarily large pathwidth, it follows that the pathwidth of grounded k-vertical-intersection ⅂-graphs is not bounded by a function in k.

Corollary 5.6. *For every $k \geq 0$, there exists a grounded 1-vertical-intersection ⅂-graph G with $\mathsf{pw}(G) \geq k$.*

5.3 Bounded Horizontal Intersections

In this section, we investigate grounded ⅂-graphs, where the number of horizontal intersections per ⅂-shape is bounded. Recall that if the number of horizontal intersections per ⅂-shape is unbounded, one can always add a universal vertex to the graph (see Figure 5.1(c)). For graphs with unbounded treewidth, this destroys linear local treewidth, and thus, product structure.

Now, we are asking whether we can apply the same trick while keeping the number of horizontal intersections bounded. More precisely, are there grounded k-horizontal-intersection ⅂-graphs that have both unbounded treewidth and a universal vertex? In the following, we show that this is not the case.

For this, note that a universal vertex with only few horizontal intersections has many right neighbors. We observe that the right neighborhood of any vertex induces a permutation graph (see Figure 5.6). Indeed, recall that permutation graphs are exactly those grounded ⅂-graphs, where the top endpoints lie on a common vertical axis. Let L be the ⅂-shape representing the universal vertex. Since every intersection between two right neighbors of L occurs strictly to the right of L, we can crop their ⅂-shapes so that each of them only barely intersects L. Then, their top endpoints lie on a common vertical axis, and we did not create or destroy any intersections among them.

We now show that the treewidth of a permutation graph is at most twice its biclique number. For a graph G, let $\mathsf{b}(G)$ denote its biclique-number, i.e., the largest integer k such that $K_{k,k}$ is contained in G as a subgraph. By Corollary 5.2, every representation of $K_{k,k}$ requires at least k horizontal intersections. Consequently, the right neighborhood of any vertex in a grounded k-horizontal-intersection ⅂-graph has treewidth at most $2k$. Since every vertex has at most k left neighbors, it follows that the neighborhood of any vertex (including the vertex itself) has treewidth at most $3k + 1$.

Our proof that ties the treewidth of permutation graphs to their biclique number, is inspired by the proof of Bodlaender, Kloks, and Kratsch [BKK95], who showed that treewidth and pathwidth coincide for permutation graphs. Instead of their "scanline" argument, however, we use a "scan-⅂" to construct a path decomposition of a given permutation graph. More

precisely, given a grounded ⌐-representation of a permutation graph in which the top endpoints of the ⌐-shapes are on a common vertical line, we place a short and narrow ⌐-shape L in the bottom left corner. We then alternately increase the height and width of L, and after every such step, we create a new node of the path decomposition whose bag contains all vertices whose ⌐-shapes intersect L. The key observation is that this process keeps the number of horizontal and vertical intersections of L balanced at all times. We observe that k horizontal and k vertical intersections of L at the same time imply the existence of a $K_{k,k}$-subgraph. Therefore, the width of the path decomposition is at most twice the biclique number.

Theorem 5.7. *For every permutation graph G it holds that* $\mathsf{b}(G) \leq \mathsf{tw}(G) = \mathsf{pw}(G) \leq 2\mathsf{b}(G)$.

Proof. The lower bound follows immediately since the treewidth of the complete bipartite graph $K_{k,k}$ is exactly k. Furthermore, Bodlaender, Kloks, and Kratsch [BKK95] showed that treewidth and pathwidth are the same for permutation graphs.

For the upper bound, suppose G has biclique number at most k for some $k \geq 1$. In the following, we construct a path decomposition $\mathcal{P}$ of G of width at most $2k$. Let $\mathcal{L}$ be a grounded ⌐-representation of G, where all top endpoints of the ⌐-shapes share the same x-coordinate. We remark that the ⌐-shapes in such a representation are uniquely determined by their height and width, or equivalently, by the y-coordinate of their top endpoint and the x-coordinate of their bottom endpoint. We further observe that two ⌐-shapes in such a representation intersect if and only if one of them is taller but narrower than the other. In particular, the set of ⌐-shapes in $\mathcal{L}$ that cross a fixed ⌐-shape L is uniquely determined by their relative height and width, so we do not need to consider exact values for the height or width of the ⌐-shapes in the following.

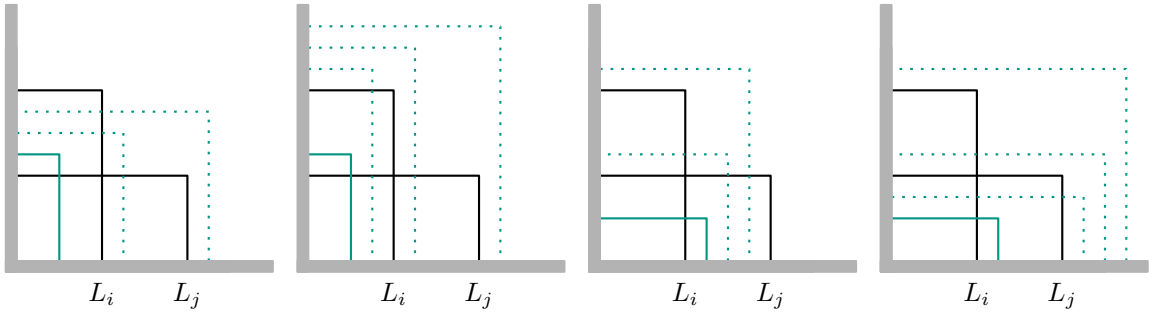
We add another such ⌐-shape L whose width and height are initially smaller than those of every ⌐-shape in $\mathcal{L}$. We use L similar to the scanline in the proof of Bodlaender, Kloks, and Kratsch [BKK95]. Specifically, we alternately increase the height and width of L so that in each step it becomes either taller or wider than the next shortest or next narrowest ⌐-shape in $\mathcal{L}$, respectively, until L is taller and wider than all ⌐-shapes in $\mathcal{L}$.

For each intermediate size of L, we add a node to the path decomposition $\mathcal{P}$ whose bag contains exactly those vertices whose ⌐-shapes intersect L. We now argue that this yields a valid path decomposition of G. Afterward, we show that in each step, the number of ⌐-shapes that cross L is at most $2k + 1$, which implies that $\mathcal{P}$ has width at most $2k$.

First, we observe that for each ⌐-shape $L_i \in \mathcal{L}$, there is a step in the process, where the height of L is increased from below L_i to above L_i, and analogously, there is a step, where the width of L is increased from narrower than L_i to wider than L_i. We notice that, independent of the order in which these two steps happen, L_i crosses L exactly in between these two steps. It follows that each ⌐-shape $L_i \in \mathcal{L}$ is contained in some bag of the path decomposition $\mathcal{P}$, and moreover, that the bags that contain L_i induce a connected subpath of $\mathcal{P}$.

Next, we consider an edge corresponding to two intersecting ⌐-shapes L_i and L_j in $\mathcal{L}$. Without loss of generality, we assume that L_i is taller and narrower than L_j. Thus, the height of L first exceeds the height of L_j before it exceeds the height of L_i, whereas the width of L first exceeds the width of L_i before it exceeds the width of L_j.

We distinguish two cases. First, suppose that the height of L exceeds the height of L_j before its width exceeds the width of L_i (see Figures 5.7(a) and 5.7(b)). From this step on, L intersects L_j until L is wider than L_j. However, before L becomes wider than L_j,



(a) First, L intersects only L_j. Then, it starts intersecting L_i before it stops intersecting L_j.

(b) First, L intersects only L_j. Then, it starts and stops intersecting L_i before it stops intersecting L_j.

(c) First, L intersects only L_i. Then, it starts intersecting L_j before it stops intersecting L_i.

(d) First, L intersects only L_i. Then, it starts and stops intersecting L_j before it stops intersecting L_i.

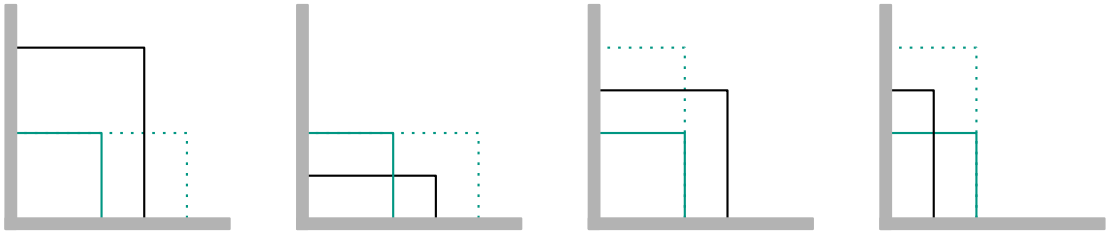
Figure 5.7: If L_i and L_j intersect each other, then there is a step in which L intersects both L_i and L_j.

it becomes wider than L_i. Hence, L starts (and possibly stops) intersecting L_i before it stops intersecting L_j, and in particular, there is at least one step in which L intersects both L_i and L_j. The second case, in which the width of L exceeds the width of L_i before the height of L exceeds the height of L_j, is symmetric (see Figures 5.7(c) and 5.7(d)).

Finally, we show that in every step L is intersected by at most $2k+1$ $\urcorner$-shapes, and thus, $\mathcal{P}$ has width at most $2k$. For a fixed step, let H and V be the sets of vertices of G whose $\urcorner$-shapes intersect the horizontal and vertical segment of L, respectively. Now, suppose the sizes of H and V differ by at most one. We observe that every $\urcorner$-shape intersecting the horizontal segment of L also intersects every $\urcorner$-shape that intersects the vertical segment of L. Hence, every vertex in H is adjacent to every vertex in V, implying that the subgraph $G[H \cup V]$ of G induced by the set of vertices intersecting L contains a biclique of size $\min\{|H|, |V|\}$. Suppose now that L is intersected by more than $2k+1$ $\urcorner$-shapes. Since $|H| + |V| > 2k+1$ and $||H| - |V|| \leq 1$, we have that $\min\{|H|, |V|\} \geq k+1$. Hence, G contains a biclique of size $k+1$, a contradiction. Therefore, it only remains to prove that H and V are indeed balanced in every step, i.e., that $|H|$ and $|V|$ differ by at most one.

To that end, we observe that increasing the width of L, either increases its number of horizontal intersections or decreases its number of vertical intersection by exactly one (see Figures 5.8(a) and 5.8(b)). In particular, increasing the width of L, increases the difference $|H| - |V|$ by one. Conversely, increasing the height of L either increases the number of vertical intersections or decreases its number of horizontal intersections (see Figures 5.8(c) and 5.8(d)), and thus, it decreases the difference $|H| - |V|$ by one. Since we start with $|H| = |V| = 0$, and then alternately increase the height or width of L, we alternately increase and decrease $|H| - |V|$ by one in every step. Therefore, $|H|$ and $|V|$ indeed always differ by at most one. □

As observed before, since the right neighborhood of every vertex in a grounded $\urcorner$-representation is a permutation graph, the treewidth in the right neighborhood is bounded by twice its biclique number. By Corollary 5.2, the complete bipartite graph $K_{k,k}$ requires an $\urcorner$-shape with at least k horizontal intersections in every representation. Therefore, the treewidth in the right neighborhood of any vertex in a grounded k-horizontal-intersection $\urcorner$-graph is at most $2k$, and since every vertex has at most k left neighbors, the treewidth in the entire neighborhood of any vertex (including the vertex itself) is at most $3k+1$.



(a) If the next narrowest ⅂-shape (black) is taller than L (green), then increasing the width of L creates a new horizontal intersection.

(b) If the next narrowest ⅂-shape (black) is shorter than L (green), then increasing the width of L destroys a vertical intersection.

(c) If the next shortest ⅂-shape (black) is wider than L (green), then increasing the height of L creates a new vertical intersection.

(d) If the next shortest ⅂-shape (black) is narrower than L (green), then increasing the width of L destroys a horizontal intersection.

Figure 5.8: Increasing the width of L (green) either (a) increases its number of horizontal intersections or (b) decreases its number of vertical intersections. Increasing the height of L either (c) increases its number of vertical intersections or (d) decreases its number of horizontal intersections.

Corollary 5.8. *For every grounded k-horizontal-intersection ⅂-graph G and every vertex $v \in V(G)$, we have* $\mathsf{tw}(G[N[v]]) \leq 3k + 1$.

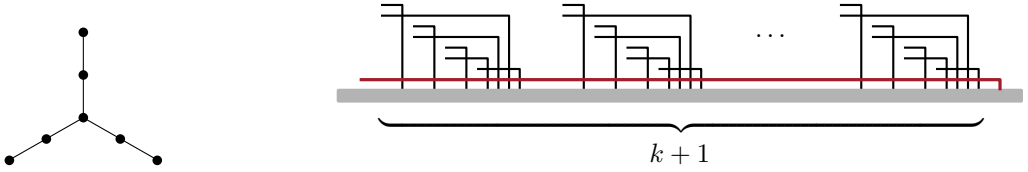
In particular, there are no grounded k-horizontal-intersection ⅂-graphs with unbounded treewidth and a universal vertex. Hence, if we add a universal vertex to G_L, then the neighborhood of this vertex has treewidth $L - 1$, implying that any grounded ⅂-representation contains at least one ⅂-shape that is intersected at least $k \geq (L - 2)/3$ times.

Corollary 5.9. *For every $L \geq 3$, the graph obtained from G_L by adding a universal vertex is not a grounded k-horizontal-intersection ⅂-graph for any $k < (L - 2)/3$.*

This complements the first part of Corollary 5.5 saying that G_L is not a grounded k-vertical-intersection ⅂-graph for any $k < (L-3)/2$, even without the universal vertex. In particular, all three known examples of grounded ⅂-graphs without product structure, namely cliques, bicliques, and G_L with a universal vertex, require both many horizontal and many vertical intersections. Moreover, the standard technique of disproving linear local treewidth, and thus, product structure by adding a universal vertex to a graph with unbounded treewidth does not work for grounded k-horizontal-intersection ⅂-graphs. This makes our question whether these graph classes admit product structure even more interesting.

Incidentally, we proved that biclique-free permutation graphs have bounded treewidth and pathwidth, and thus, linear local treewidth and product structure. Since linear local treewidth in turn implies a constant biclique number, it follows that these five properties – constant biclique number, constant pathwidth, constant treewidth, product structure, and linear local treewidth – are equivalent for all families of permutation graphs. Therefore, permutation graphs are a graph class, and even a subclass of grounded ⅂-graphs, with a positive answer to our main question.

Corollary 5.10. *For every family of permutation graphs, linear local treewidth and product structure are equivalent.*



(a) The subdivided claw.

(b) A grounded 2-vertical-intersection ⅂-representation of the graph consisting of $k+1$ copies of the subdivided claw and a universal vertex. This graph is not a grounded k-horizontal-intersection ⅂-graph.

Figure 5.9: An example of a grounded 2-vertical-intersection ⅂-graph that is not a grounded k-horizontal-intersection ⅂-graph.

In the following, we also complement the second part of Corollary 5.5 by showing that there are grounded 2-vertical-intersection ⅂-graphs that require an unbounded number of horizontal intersections in any ⅂-representation. This is in stark contrast to Lemma 5.3, where we showed that grounded 1-horizontal-intersection and grounded 1-vertical-intersection ⅂-graphs are the same. The proof is based on the structural observation that the right neighborhood of any vertex in a grounded ⅂-graph induces a permutation graph, while the same restriction does not apply to left neighborhoods.

Lemma 5.11. *For every $k \geq 0$, there exists a grounded 2-vertical-intersection ⅂-graph that is not a grounded k-horizontal-intersection ⅂-graph.*

Proof. Let T be the *subdivided claw*, that is, the tree on 7 vertices that is obtained from $K_{1,3}$ by subdividing each edge exactly once (see Figure 5.9(a)). Let G be the disjoint union of $k+1$ copies of T plus a universal vertex. As illustrated in Figure 5.9, G is a grounded 2-vertical-intersection ⅂-graph. Indeed, by Lemma 5.3 we can construct a grounded 1-vertical-intersection ⅂-representation for the forest consisting of $k+1$ disjoint copies of T. Then, adding an ⅂-shape corresponding to the universal vertex as a very short and wide ⅂-shape, increases the number of vertical intersections of each ⅂-shape by exactly one (see Figure 5.9(b)).

Now, suppose for the sake of contradiction that G admits a grounded k-horizontal-intersection ⅂-representation $\mathcal{L}$, and let $L \in \mathcal{L}$ be the ⅂-shape that corresponds to the universal vertex. Since L has at most k horizontal intersections, there exists a copy of T such that none of its corresponding ⅂-shapes intersects the horizontal segment of L. It follows that every one of these ⅂-shapes intersects the vertical segment of L. As observed before, the set of vertices whose ⅂-shapes intersect the vertical segment of L induces a permutation graph. Therefore, T itself must be a permutation graph.

However, Acan and Hitczenko [AH16] showed that every tree that is a permutation graph is a *caterpillar*, i.e., a tree in which all non-leaf vertices lie on a single path. It is easy to see that T is not a caterpillar since removing its three leaves leaves a $K_{1,3}$, which is not a path. This contradicts the conclusion that T is a permutation graph. Hence, G is not a grounded k-horizontal-intersection ⅂-graph. □

6. Conclusion

The main goal of this thesis was to identify candidates of graph classes, beyond minor-closed graph classes, for which linear local treewidth and product structure are equivalent.

Question 6.1. *For which graph classes are linear local treewidth and product structure equivalent?*

In particular, we considered string graphs, and found a promising candidate with grounded $\urcorner$-graphs.

Question 6.2. *Are linear local treewidth and product structure equivalent for (an interesting subclass of) string graphs, for example, for grounded $\urcorner$-graphs?*

Although we were not able to answer Question 6.2 for the entire class of grounded $\urcorner$-graphs, we introduced two new subclasses of grounded $\urcorner$-graphs, which we call grounded k-horizontal-intersection and grounded k-vertical-intersection $\urcorner$-graphs. Motivated by the observation that all known examples of grounded $\urcorner$-graphs without product structure require an unbounded number of both horizontal and vertical intersections per $\urcorner$-shape in every representation, we ask the following question.

Question 6.3. *Is it true that a family $\mathcal{G}$ of grounded $\urcorner$-graphs admits product structure if and only if it admits a representation in which the number of horizontal or vertical intersections per $\urcorner$-shape is bounded?*

More precisely, we are asking whether there are functions f and g such that for every grounded $\urcorner$-graph G it holds that $f(k) \leq \mathsf{rtw}(G) \leq g(k)$, where k is the minimum integer such that G is either a grounded k-horizontal-intersection or a grounded k-vertical-intersection $\urcorner$-graph. Naturally, the upper bound is particularly interesting as it would provide a direct way to prove product structure for families of grounded $\urcorner$-graphs. For grounded k-vertical-intersection $\urcorner$-graphs, we prove not only bounded row treewidth, but even bounded treewidth. Thus, in order to obtain the upper bound in Question 6.3, it would suffice to show that grounded k-horizontal-intersection $\urcorner$-graphs admit product structure, or more precisely, that their row treewidth is bounded by a function of k.

Question 6.4. *Do grounded k-horizontal-intersection $\urcorner$-graphs admit product structure?*

For the other direction, it remains to rule out the existence of a family of grounded ⅂-graphs with product structure that requires an unbounded number of both horizontal and vertical intersections per ⅂-shape in every representation. Equivalently, it remains to show that for every family $\mathcal{G}$ of grounded ⅂-graphs that admits product structure there exists a constant k such that every graph $G \in \mathcal{G}$ is a grounded k-horizontal-intersection or a grounded k-vertical-intersection ⅂-graph. An even stronger result would be to show that the same holds for every family of grounded ⅂-graphs with linear local treewidth. Together with the upper bound, this would imply that product structure and linear local treewidth are equivalent for every family of grounded ⅂-graphs.

Question 6.5. *Is it true that for every family $\mathcal{G}$ of grounded ⅂-graphs with linear local treewidth there exists a constant k such that every graph $G \in \mathcal{G}$ is a grounded k-horizontal-intersection or a grounded k-vertical-intersection ⅂-graph?*

Beyond grounded ⅂-graphs, another interesting subclass of string graphs are d-directional grounded segment graphs. Once Question 6.3 is resolved, these graphs appear to be a promising next direction, starting with the case $d = 2$. Recall that 2-directional grounded segment graphs are bipartite. As shown by An, Oh, and Xue [AOX24], there exist biclique-free 2-directional grounded segment graphs with unbounded treewidth, obtained via a construction similar to the one from Chapter 4. Due to this similarity, we conjecture that their construction also admits product structure. However, in contrast to our construction, one cannot add a universal vertex due to their bipartiteness. Moreover, there seems to be no straightforward way to add a vertex adjacent to many other vertices of the construction, e.g., to all vertices on one side of the bipartition, without introducing segments of additional directions. Hence, to the best of our knowledge, bicliques are currently the only known examples of 2-directional grounded segment graphs without product structure.

Question 6.6. *Do biclique-free 2-directional grounded segment graphs admit product structure?*

For $d \geq 3$, adding a universal vertex to the construction of An, Oh, and Xue [AOX24] becomes straightforward, immediately yielding a biclique-free example without product structure. Hence, unlike for $d = 2$, the absence of bicliques is not even a candidate for characterizing product structure. Consequently, one would need to identify an entirely new structural condition for this purpose.

Bibliography

[ABD+26] Tara Abrishami, Marcin Briański, James Davies, Xiying Du, Jana Masaříková, Paweł Rzążewski, and Bartosz Walczak. "Burling Graphs in Graphs with Large Chromatic Number". In: *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Proceedings. Society for Industrial and Applied Mathematics, Jan. 2026, pp. 3978–3998. DOI: 10.1137/1.9781611978971.146.

[AH16] Hüseyin Acan and Paweł Hitczenko. "On random trees obtained from permutation graphs". In: *Discrete Mathematics* 339.12 (Dec. 2016), pp. 2871–2883. ISSN: 0012-365X. DOI: 10.1016/j.disc.2016.05.028.

[AIK+25] Shinwoo An, Seonghyuk Im, Seokbeom Kim, and Myounghwan Lee. "An efficient algorithm for $\mathcal{F}$-subgraph-free Edge Deletion on graphs having a product structure". In: *arXiv* (2025). DOI: 10.48550/arXiv.2510.14674.

[ALD+17] Abu Reyan Ahmed, Felice De Luca, Sabin Devkota, Alon Efrat, Md Iqbal Hossain, Stephen Kobourov, Jixian Li, Sammi Abida Salma, and Eric Welch. "L-Graphs and Monotone L-Graphs". In: *arXiv* (Mar. 2017). DOI: 10.48550/arXiv.1703.01544.

[AOX24] Shinwoo An, Eunjin Oh, and Jie Xue. "Sparse Outerstring Graphs Have Logarithmic Treewidth". In: *32nd Annual European Symposium on Algorithms (ESA 2024)*. Ed. by Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman. Vol. 308. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 10:1–10:18. ISBN: 978-3-95977-338-6. DOI: 10.4230/LIPIcs.ESA.2024.10.

[BBD18] Therese Biedl, Ahmad Biniaz, and Martin Derka. "On the Size of Outer-String Representations". In: *LIPIcs, Volume 101, SWAT 2018* 101 (2018). Ed. by David Eppstein, 10:1–10:14. ISSN: 1868-8969. DOI: 10.4230/LIPICS.SWAT.2018.10.

[BDH+26] Thomas Bläsius, Emil Dohse, Deborah Haun, and Laura Merker. "Product Structure and Treewidth of Hyperbolic Uniform Disk Graphs". In: *42nd International Symposium on Computational Geometry (SoCG 2026)*. Ed. by Hee-Kap Ahn, Michael Hoffmann, and Amir Nayyeri. Vol. 367. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026, 18:1–18:17. ISBN: 978-3-95977-418-5. DOI: 10.4230/LIPIcs.SoCG.2026.18.

[BDJ+22a] Prosenjit Bose, Vida Dujmović, Mehrnoosh Javarsineh, and Pat Morin. "Asymptotically Optimal Vertex Ranking of Planar Graphs". In: *arXiv* (Aug. 2022). DOI: 10.48550/arXiv.2007.06455.

[BDJ+22b] Prosenjit Bose, Vida Dujmović, Mehrnoosh Javarsineh, Pat Morin, and David R. Wood. "Separating layered treewidth and row treewidth". In: *Discrete Mathematics & Theoretical Computer Science* vol. 24:1, 18 (May 2022): *Graph Theory*. ISSN: 1365-8050. DOI: 10.46298/dmtcs.7458.

[BGP22] Marthe Bonamy, Cyril Gavoille, and Michał Pilipczuk. "Shorter Labeling Schemes for Planar Graphs". In: *SIAM Journal on Discrete Mathematics* 36.3 (2022), pp. 2082–2099. ISSN: 0895-4801. DOI: 10.1137/20M1330464.

[BKK95] Hans L. Bodlaender, Ton Kloks, and Dieter Kratsch. "Treewidth and Pathwidth of Permutation Graphs". In: *SIAM Journal on Discrete Mathematics* 8.4 (Nov. 1995), pp. 606–616. ISSN: 0895-4801, 1095-7146. DOI: 10.1137/S089548019223992X.

[BKW25] Édouard Bonnet, O-joung Kwon, and David R. Wood. "Reduced bandwidth: a qualitative strengthening of twin-width in minor-closed classes (and beyond)". In: *arXiv* (2025). DOI: 10.48550/arXiv.2202.11858.

[BMO22] Prosenjit Bose, Pat Morin, and Saeed Odak. "An Optimal Algorithm for Product Structure in Planar Graphs". In: *LIPIcs, Volume 227, SWAT 2022* 227 (2022), 19:1–19:14. ISSN: 1868-8969. DOI: 10.4230/LIPICS.SWAT.2022.19.

[CCT+26] Hsien-Chih Chang, Jonathan Conroy, Zihan Tan, and Da Wei Zheng. "Cutting Planarians: Planar Emulators for String Graphs". In: *Proceedings of the 58th Annual ACM Symposium on Theory of Computing*. STOC '26. New York, NY, USA: Association for Computing Machinery, 2026, pp. 2140–2151. ISBN: 979-8-4007-2536-4. DOI: 10.1145/3798129.3800917.

[CEF25] Maria Chudnovsky, David Eppstein, and David Fischer. "String Graph Obstacles of High Girth and of Bounded Degree". In: *33rd International Symposium on Graph Drawing and Network Visualization (GD 2025)*. Ed. by Vida Dujmović and Fabrizio Montecchiani. Vol. 357. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 24:1–24:18. ISBN: 978-3-95977-403-1. DOI: 10.4230/LIPIcs.GD.2025.24.

[CFK+15] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Cham: Springer International Publishing, 2015. ISBN: 978-3-319-21274-6. DOI: 10.1007/978-3-319-21275-3.

[CFM+18] Jean Cardinal, Stefan Felsner, Tillmann Miltzow, Casey Tompkins, and Birgit Vogtenhuber. "Intersection Graphs of Rays and Grounded Segments". In: *Journal of Graph Algorithms and Applications* 22.2 (Jan. 2018), pp. 273–295. ISSN: 1526-1719. DOI: 10.7155/jgaa.00470.

[Dav25] James Davies. "String graphs are quasi-isometric to planar graphs". In: *arXiv* (Oct. 2025). DOI: 10.48550/arXiv.2510.19602.

[Dee26] DeepL SE. *DeepL Write*. 2026. https://www.deepl.com/de/write.

[DEG+21] Vida Dujmović, Louis Esperet, Cyril Gavoille, Gwenaël Joret, Piotr Micek, and Pat Morin. "Adjacency Labelling for Planar Graphs (and Beyond)". In: *J. ACM* 68.6 (Oct. 2021). ISSN: 0004-5411. DOI: 10.1145/3477542.

[DEJ+20] Vida Dujmović, Louis Esperet, Gwenaël Joret, Bartosz Walczak, and David R. Wood. "Planar graphs have bounded nonrepetitive chromatic number". In: *Advances in Combinatorics* (Mar. 2020). DOI: 10.19086/aic.12100.

[DFM+21] Michał Dębski, Stefan Felsner, Piotr Micek, and Felix Schröder. "Improved Bounds for Centered Colorings". In: *Advances in Combinatorics* (Aug. 2021). DOI: 10.19086/aic.27351.

[DHJ+21] Zdeněk Dvořák, Tony Huynh, Gwenael Joret, Chun-Hung Liu, and David R. Wood. "Notes on Graph Product Structure Theory". In: *2019-20 MATRIX Annals*. Ed. by Jan de Gier, Cheryl E. Praeger, and Terence Tao. Cham: Springer International Publishing, 2021, pp. 513–533. ISBN: 978-3-030-62497-2. DOI: 10.1007/978-3-030-62497-2_32.

[DJM+20] Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. "Planar Graphs Have Bounded Queue-Number". In: *Journal of the ACM (JACM)* 67.4 (Aug. 2020), 22:1–22:38. ISSN: 0004-5411. DOI: 10.1145/3385731.

[DMW23] Vida Dujmović, Pat Morin, and David R. Wood. "Graph product structure for non-minor-closed classes". In: *Journal of Combinatorial Theory, Series B* 162 (2023), pp. 34–67. ISSN: 0095-8956. DOI: 10.1016/j.jctb.2023.03.004.

[Duj26] Vida Dujmović. "Graph Product Structure Theory with Applications". In: *Courses in Discrete and Computational Geometry*. Ed. by János Pach and Géza Tóth. Cham: Springer Nature Switzerland, 2026, pp. 3–44. ISBN: 978-3-032-10503-5. DOI: 10.1007/978-3-032-10503-5_1.

[EET76] G. Ehrlich, S. Even, and R. E. Tarjan. "Intersection graphs of curves in the plane". In: *Journal of Combinatorial Theory, Series B* 21.1 (1976), pp. 8–20. ISSN: 0095-8956. DOI: 10.1016/0095-8956(76)90022-8.

[EJM23] Louis Esperet, Gwenaël Joret, and Pat Morin. "Sparse universal graphs for planarity". In: *Journal of the London Mathematical Society* 108.4 (2023), pp. 1333–1357. ISSN: 1469-7750. DOI: 10.1112/jlms.12781.

[Gav74] Fănică Gavril. "The intersection graphs of subtrees in trees are exactly the chordal graphs". In: *Journal of Combinatorial Theory, Series B* 16.1 (Feb. 1974), pp. 47–56. ISSN: 0095-8956. DOI: 10.1016/0095-8956(74)90094-X.

[GIP18] Daniel Gonçalves, Lucas Isenmann, and Claire Pennarun. "Planar graphs as l-intersection or l-contact graphs". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '18. USA: Society for Industrial and Applied Mathematics, Jan. 2018, pp. 172–184. ISBN: 978-1-61197-503-1. DOI: 10.1137/1.9781611975031.12.

[GL85] A. Gyárfás and J. Lehel. "Covering and coloring problems for relatives of intervals". In: *Discrete Mathematics* 55.2 (1985), pp. 167–180. ISSN: 0012-365X. DOI: 10.1016/0012-365X(85)90045-7.

[HKW25] Kevin Hendrey, Nikolai Karol, and David R. Wood. "Structure of k-Matching-Planar Graphs". In: *arXiv* (July 2025). DOI: 10.48550/arXiv.2507.22395.

[HLR92] Lenwood S. Heath, Frank Thomson Leighton, and Arnold L. Rosenberg. "Comparing Queues and Stacks As Machines for Laying Out Graphs". In: *SIAM Journal on Discrete Mathematics* 5.3 (1992), pp. 398–412. DOI: 10.1137/0405031.

[HMŠ+21] Tony Huynh, Bojan Mohar, Robert Šámal, Carsten Thomassen, and David R. Wood. "Universality in minor-closed graph classes". In: *arXiv* (2021). DOI: 10.48550/arXiv.2109.00327.

[HW24] Robert Hickingbotham and David R. Wood. "Shallow Minors, Graph Products, and Beyond-Planar Graphs". In: *SIAM Journal on Discrete Mathematics* 38.1 (Mar. 2024), pp. 1057–1089. ISSN: 0895-4801. DOI: 10.1137/22M1540296.

[ISW24] Freddie Illingworth, Alex Scott, and David Wood. "Product structure of graphs with an excluded minor". In: *Transactions of the American Mathematical Society, Series B* 11.35 (Nov. 2024), pp. 1233–1248. ISSN: 2330-0000. DOI: 10.1090/btran/192.

[JP22] Hugo Jacob and Marcin Pilipczuk. "Bounding Twin-Width for Bounded-Treewidth Graphs, Planar Graphs, and Bipartite Graphs". In: *Graph-Theoretic Concepts in Computer Science*. Ed. by Michael A. Bekos and Michael Kaufmann. Cham: Springer International Publishing, 2022, pp. 287–299. ISBN: 978-3-031-15914-5. DOI: 10.1007/978-3-031-15914-5_21.

[JT19] Vít Jelínek and Martin Töpfer. "On Grounded L-Graphs and their Relatives". In: *The Electronic Journal of Combinatorics* (2019), P3.17–P3.17. ISSN: 1077-8926. DOI: 10.37236/8096.

[Kar25] Nikolai Karol. "String Graphs: Product Structure and Localised Representations". In: *arXiv* (Nov. 2025). DOI: 10.48550/arXiv.2511.15156.

[KM91] Jan Kratochvíl and Jiří Matoušek. "String graphs requiring exponential representations". In: *Journal of Combinatorial Theory, Series B* 53.1 (1991), pp. 1–4. ISSN: 0095-8956. DOI: 10.1016/0095-8956(91)90050-T.

[Kra91a] Jan Kratochvíl. "String graphs. I. The number of critical nonstring graphs is infinite". In: *Journal of Combinatorial Theory, Series B* 52.1 (May 1991), pp. 53–66. ISSN: 0095-8956. DOI: 10.1016/0095-8956(91)90090-7.

[Kra91b] Jan Kratochvíl. "String graphs. II. recognizing string graphs is NP-hard". In: *Journal of Combinatorial Theory, Series B* 52.1 (May 1991), pp. 67–78. ISSN: 0095-8956. DOI: 10.1016/0095-8956(91)90091-W.

[KW26] Nikolai Karol and David R. Wood. "Sparse String Graphs and Region Intersection Graphs over Minor-Closed Classes have Linear Expansion". In: *arXiv* (Apr. 2026). DOI: 10.48550/arXiv.2604.07903.

[LT79] Richard J. Lipton and Robert Endre Tarjan. "A Separator Theorem for Planar Graphs". In: *SIAM Journal on Applied Mathematics* 36.2 (Apr. 1979), pp. 177–189. ISSN: 0036-1399. DOI: 10.1137/0136016.

[Mat14] Jiří Matoušek. "Near-Optimal Separators in String Graphs". en. In: *Combinatorics, Probability and Computing* 23.1 (Jan. 2014), pp. 135–139. ISSN: 0963-5483, 1469-2163. DOI: 10.1017/S0963548313000400.

[McG96] Sean McGuinness. "On bounding the chromatic number of L-graphs". In: *Discrete Mathematics* 154.1 (1996), pp. 179–187. ISSN: 0012-365X. DOI: 10.1016/0012-365X(95)00316-O.

[Mor21] Pat Morin. "A Fast Algorithm for the Product Structure of Planar Graphs". In: *Algorithmica* 83.5 (May 2021), pp. 1544–1558. ISSN: 1432-0541. DOI: 10.1007/s00453-020-00793-5.

[MP92] M. Middendorf and F. Pfeiffer. "The max clique problem in classes of string-graphs". In: *Discrete Mathematics* 108.1 (Oct. 1992), pp. 365–372. ISSN: 0012-365X. DOI: 10.1016/0012-365X(92)90688-C.

[MSS+24] Laura Merker, Lena Scherzer, Samuel Schneider, and Torsten Ueckerdt. "Intersection Graphs with and Without Product Structure". In: *32nd International Symposium on Graph Drawing and Network Visualization (GD 2024)*. Ed. by Stefan Felsner and Karsten Klein. Vol. 320. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 23:1–23:19. ISBN: 978-3-95977-343-0. DOI: 10.4230/LIPIcs.GD.2024.23.

[MSS26] Laura Merker, Lena Scherzer, and Samuel Schneider. "Separating Geodesic Structure and Product Structure". In: arXiv:2607.02098 (2026). DOI: 10.48550/arXiv.2607.02098.

[Ope26] OpenAI, Inc. *ChatGPT*. 2026. https://chatgpt.com/.

[PKK+14] Arkadiusz Pawlik, Jakub Kozik, Tomasz Krawczyk, Michał Lasoń, Piotr Micek, William T. Trotter, and Bartosz Walczak. "Triangle-free intersection graphs of line segments with large chromatic number". In: *Journal of Combinatorial Theory, Series B* 105 (Mar. 2014), pp. 6–10. ISSN: 0095-8956. DOI: 10.1016/j.jctb.2013.11.001.

[PS21] Michał Pilipczuk and Sebastian Siebertz. "Polynomial bounds for centered colorings on proper minor-closed graph classes". In: *Journal of Combinatorial Theory, Series B* 151 (2021), pp. 111–147. ISSN: 0095-8956. DOI: 10.1016/j.jctb.2021.06.002.

[Pup20] Sergey Pupyrev. "Improved Bounds for Track Numbers of Planar Graphs". In: *Journal of Graph Algorithms and Applications* 24.3 (Mar. 2020), pp. 323–341. ISSN: 1526-1719. DOI: 10.7155/jgaa.00536.

[RS03] Neil Robertson and P. D. Seymour. "Graph Minors. XVI. Excluding a non-planar graph". In: *Journal of Combinatorial Theory, Series B* 89.1 (2003), pp. 43–76. ISSN: 0095-8956. DOI: 10.1016/S0095-8956(03)00042-X.

[Sch25] Lena Scherzer. "Comparing Variants of Product Structure". MA thesis. 2025. https://i11www.iti.kit.edu/_media/teaching/theses/ma-scherzer-25.pdf.

[SŠ04] Marcus Schaefer and Daniel Štefankovič. "Decidability of string graphs". In: *Journal of Computer and System Sciences*. Special Issue on STOC 2001 68.2 (Mar. 2004), pp. 319–334. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2003.07.002.

Appendix

A Declaration of AI Usage

For this thesis, generative AI was used exclusively as a writing aid and for basic literature research.

For language revision, GPT-5.5 [Ope26] and DeepL Write [Dee26] were used to suggest improvements to grammar, style, and wording. No paragraphs of this thesis were generated by these or any other AI-based tools. All suggested changes were reviewed individually and only incorporated after manual evaluation.

GPT-5.5 [Ope26] was also used to assist in identifying relevant literature. Its use was limited to locating references for statements already known to be correct, identifying publications on specific established results, and obtaining a broad overview of related work. All suggested references were verified before being cited in this thesis.

In particular, AI-based tools were not used to derive any mathematical arguments or proofs. They were also not used to summarize scientific publications, explain technical concepts, or generate figures.