

Arne Leitert

# **Tree-Breadth of Graphs with Variants and Applications**

---

Dissertation written by Arne Leitert and submitted to Kent State University in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

August 2017.

**Dissertation Committee**

Dr. Feodor F. Dragan (Advisor),

Dr. Ruoming Jin,

Dr. Ye Zhao,

Dr. Artem Zvavitch, and

Dr. Lothar Reichel.

**Accepted by**

Dr. Javed Khan (Chair, Department of Computer Science) and

Dr. James L. Blank (Dean, College of Arts and Sciences).

---

# Abstract

The *tree-breadth* of a graph is a recently introduced variant of the well known idea of decomposing a graph into a tree of bags. It is a parameter which adds a metric constraint to tree-decompositions limiting the radius of each bag. In this dissertation, we further investigate the tree-breadth of graphs. We present approaches to compute a tree-decomposition with small breadth for a given graph including approximation algorithms for general graphs as well as optimal solutions for special graph classes. Additionally, we introduce a variant of tree-breadth called *strong tree-breadth*. Next, we present various algorithms to approach the (Connected)  $r$ -Domination problem for graphs with bounded tree-breadth. One variant, called *path-breadth*, requires the decomposition to be a path instead of a tree. We use graphs with bounded path-breadth to construct efficient constant-factor approximation algorithms for the bandwidth and line-distortion problems. Motivated by these results, we introduce and investigate the new *Minimum Eccentricity Shortest Path* problem. We analyse the hardness of the problem, show algorithms to compute an optimal solution, and present approximation algorithms differing in quality and runtime.

---

# Contents

<b>Abstract</b>	<b>ii</b>
<b>Contents</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Existing and Recent Results . . . . .	2
1.2 Outline . . . . .	2
<b>2 Preliminaries</b>	<b>4</b>
2.1 General Definitions . . . . .	4
2.2 Tree-Decompositions . . . . .	5
2.3 Layering Partitions . . . . .	6
2.4 Special Graph Classes . . . . .	7
<b>3 Computing Decompositions with Small Breadth</b>	<b>10</b>
3.1 Approximation Algorithms . . . . .	10
3.1.1 Layering Based Approaches . . . . .	11
3.1.2 Neighbourhood Based Approaches . . . . .	13
3.2 Strong Tree-Breadth . . . . .	16
3.2.1 NP-Completeness . . . . .	17
3.2.2 Polynomial Time Cases . . . . .	21
3.3 Computing Decompositions for Special Graph Classes . . . . .	25
<b>4 Connected <math>r</math>-Domination</b>	<b>31</b>
4.1 Using a Layering Partition . . . . .	31
4.2 Using a Tree-Decomposition . . . . .	39
4.2.1 Preprocessing . . . . .	40
4.2.2 $r$ -Domination . . . . .	43
4.2.3 Connected $r$ -Domination . . . . .	45
4.3 Implications for the $p$ -Center Problem . . . . .	51
<b>5 Bandwidth and Line-Distortion</b>	<b>53</b>
5.1 Existing Results . . . . .	54
5.2 $k$ -Dominating Pairs . . . . .	56
5.2.1 Relation to Path-Length and Line-Distortion . . . . .	56

5.2.2	Determining a $k$ -Dominating Pair . . . . .	57
5.3	Bandwidth of Graphs with Bounded Path-Length . . . . .	59
5.4	Path-Length and Line-Distortion . . . . .	61
5.4.1	Bound on Line-Distortion Implies Bound on Path-Length . . . . .	61
5.4.2	Line-Distortion of Graphs with Bounded Path-Length . . . . .	62
5.5	Approximation for AT-Free Graphs . . . . .	65
<b>6</b>	<b>The Minimum Eccentricity Shortest Path Problem</b>	<b>68</b>
6.1	Hardness . . . . .	68
6.2	Computing an Optimal Solution . . . . .	72
6.2.1	General Graphs . . . . .	72
6.2.2	Distance-Hereditary Graphs . . . . .	75
6.2.3	A General Approach for Tree-Structured Graphs . . . . .	77
6.3	Approximation Algorithms . . . . .	84
6.3.1	General Graphs . . . . .	85
6.3.2	Graphs with Bounded Tree-Length and Bounded Hyperbolicity . . . . .	88
6.3.3	Using Layering Partition . . . . .	91
6.4	Relation to Other Parameters . . . . .	93
6.5	Solving $k$ -Domination using a MESP . . . . .	93
6.6	Open Questions . . . . .	96
<b>Bibliography</b>		<b>98</b>
<b>List of Figures</b>		<b>105</b>
<b>List of Tables</b>		<b>106</b>

# Introduction

Decomposing a graph into a tree is an old concept. It is introduced already by HALIN [59]. However, a more popular introduction is given by ROBERTSON and SEYMOUR [79, 80]. The idea is to decompose a graph into multiple induced subgraphs, usually called *bags*, where each vertex can be in multiple bags. These bags are combined to a tree or path in such a way that the following requirements are fulfilled: Each vertex is in at least one bag, each edge is in at least one bag, and, for each vertex, the bags containing it induce a subtree. We give formal definitions in Section 2.2 (page 5).

For a given graph, there can be up-to exponentially many different tree- or path-decompositions. The easiest is to have only one bag containing the whole graph. To make the concept more interesting, it is necessary to add additional restrictions. The most known is called *tree-width*. A decomposition has *width*  $\omega$  if each bag contains at most  $\omega + 1$  vertices. Then, a graph  $G$  has tree-width  $\omega$  (written as  $\text{tw}(G) = \omega$ ) if there is a tree-decomposition for  $G$  which has width  $\omega$  and there is no tree-decomposition with smaller width.

Tree-width is well studied. Determining the minimal width  $\omega$  for a given graph  $G$  is NP-complete [3]. However, if  $\omega$  is fixed, it can be checked in linear time if  $\text{tw}(G) \leq \omega$  [12]. The algorithm in [12] also creates a tree decomposition for  $G$ . For a graph class with bounded tree-width, many NP-complete problems can be solved in polynomial or even linear time.

In the last years, a new perspective on tree-decompositions was invested. Instead of limiting the number of vertices in each bag, the distance between vertices inside a bag is limited. There are two major variants: *breadth* and *length*.

The *breadth* of a decomposition is  $\rho$ , if, for each bag  $B$ , there is a vertex  $v$  such that each vertex in  $B$  has distance at most  $\rho$  to  $v$ . Accordingly, we say the *tree-breadth* of a graph  $G$  is  $\rho$  (written as  $\text{tb}(G) = \rho$ ) if there is a tree-decomposition for  $G$  with breadth  $\rho$  and there is no tree-decomposition with smaller breadth.

The *length* of a decomposition is  $\lambda$ , if the maximal distance of two vertices in each bag is at most  $\lambda$ . Accordingly, we say the *tree-length* of a graph  $G$  is  $\lambda$  (written as  $\text{tl}(G) = \lambda$ ) if there is a tree-decomposition for  $G$  with length  $\lambda$  and there is no tree-decomposition with smaller length.

## 1.1 Existing and Recent Results

The first results on tree-breadth and -length were motivated by tree spanners. A *tree spanner*  $T$  of a graph  $G$  is a spanning tree for  $G$  such that the distance of two vertices in  $T$  approximates their distance in  $G$ . A natural applications for tree spanners is routing in networks. Routing messages directly over a tree can be done very efficiently [90]. Another option is to use the tree as orientation as shown in [44]. Next to routing, tree spanners are used for a protocol locating mobile objects in networks [78]. Other applications can be found for example in biology [5] and in approximating bandwidth [91].

To approximate tree spanners for general graphs, DRAGAN and KÖHLER introduce tree-breadth in [37]. As additional result, they show a simple algorithm for approximating tree-breadth. Later, ABU-ATA and DRAGAN [36] use tree-breadth to construct collective tree spanners for general graphs. In [2], they also analyse the tree-breadth of real world graph showing that many real world graphs have a small tree-breadth. The hardness of computing the tree-breadth of a graph is investigated by DUCCOFFE et al. [48]. They show that it is NP-complete to determine, for a given graph  $G$  and any given integer  $\rho \geq 1$ , if  $\text{tb}(G) \leq \rho$ . Additionally, they present a polynomial-time algorithms to decide if  $\text{tb}(G) = 1$  for the case that  $G$  is a bipartite or a planar graph.

Tree-length is first introduced by DOURISBOURE and GAVOILLE [33]. They investigate the connection to  $k$ -chordal graphs and present first approximation results. Later, DOURISBOURE et al. [32] use tree-length to investigate additive spanners. Additionally, DOURISBOURE [31] shows how to compute efficient routing schemes for graphs with bounded tree-length. An other application of tree-length is the Metric Dimension problem. It asks for the smallest set of vertices such that each vertex in a given graph can be identified by its distances to the vertices in such a set. BELMONTE et al. [7] show that the Metric Dimension problem is Fixed Parameter Tractable for graphs with bounded tree-length. In [72], LOKSHTANOV investigates the hardness of computing the tree-length of a given graph and shows that it is NP-complete to do so, even for length 2. A connection between tree-width and tree-length is presented by COUDERT et al. [26]. They show that, for a given graph  $G$ ,  $\text{tl}(G) \leq \lfloor \ell(G)/2 \rfloor \cdot (\text{tw}(G) - 1)$  where  $\ell(G)$  is the length of a longest isometric cycle in  $G$ .

## 1.2 Outline

In this dissertation, we further investigate tree-breadth. In Chapter 3, we present approaches to compute a tree-decomposition with small breadth for a given graph. This includes approximation algorithms for general graphs as well as optimal solutions for special graph classes. We also introduce a variant of tree-breadth called *strong tree-breadth*. Next, in Chapter 4, we present algorithms to approach the (Connected)  $r$ -Domination problem for graphs with bounded tree-breadth. Our results include (almost) linear time algorithms for the case that no tree-decomposition is given and polynomial time algorithms (with a better solution quality) for the case that a tree-decomposition is given.

In Chapter 5, we use graphs with bounded path-breadth and -length to construct efficient constant-factor approximation algorithms for the bandwidth and line-distortion problems. Motivated by these results, we introduce and investigate the new Minimum Eccentricity Shortest Path problem in Chapter 6. We analyse the hardness of the problem, show algorithms to compute an optimal solution, and present various approximation algorithms differing in quality and runtime. This is done for general graphs as well as for special graph classes.

**Note.** Some of the results in this dissertation were obtained in collaborative work with FEODOR F. DRAGAN<sup>1</sup> and EKKEHARD KÖHLER<sup>2</sup>. Results have been published partially at *SWAT* 2014, Copenhagen, Denmark [38], at *WADS* 2015, Victoria, Canada [41], at *WG* 2015, Munich, Germany [40], at *COCOA* 2016, Hong Kong, China [71], in the *Journal of Graph Algorithms and Applications* [42], and in *Algorithmica* [39].

---

<sup>1</sup>Kent State University, USA

<sup>2</sup>Brandenburgische Technische Universität Cottbus, Germany



# Preliminaries

## 2.1 General Definitions

If not stated or constructed otherwise, all graphs occurring in this dissertation are connected, finite, unweighted, undirected, without loops, and without multiple edges. For a graph  $G = (V, E)$ , we use  $n = |V|$  and  $m = |E|$  to denote the cardinality of the vertex set and the edge set of  $G$ .

The *length* of a path from a vertex  $v$  to a vertex  $u$  is the number of edges in the path. The *distance*  $d_G(u, v)$  in a graph  $G$  of two vertices  $u$  and  $v$  is the length of a shortest path connecting  $u$  and  $v$ . The distance between a vertex  $v$  and a set  $S \subseteq V$  is defined as  $d_G(v, S) = \min_{u \in S} d_G(u, v)$ . Let  $x$  and  $y$  be two vertices in a graph  $G$  such that  $x$  is most distant from some arbitrary vertex and  $y$  is most distant from  $x$ . Such a vertex pair  $x, y$  is called *spread pair* and a shortest path from  $x$  to  $y$  is called *spread path*.

The *eccentricity*  $\text{ecc}_G(v)$  of a vertex  $v$  is  $\max_{u \in V} d_G(u, v)$ . For a set  $S \subseteq V$ , its eccentricity is  $\text{ecc}_G(S) = \max_{u \in V} d_G(u, S)$ . For a vertex pair  $s, t$ , a shortest  $(s, t)$ -path  $P$  has *minimal eccentricity*, if there is no shortest  $(s, t)$ -path  $Q$  with  $\text{ecc}_G(Q) < \text{ecc}_G(P)$ . Two vertices  $x$  and  $y$  are called *mutually furthest* if  $d_G(x, y) = \text{ecc}_G(x) = \text{ecc}_G(y)$ . A vertex  $u$  is *k-dominated* by a vertex  $v$  (by a set  $S \subseteq V$ ), if  $d_G(u, v) \leq k$  ( $d_G(u, S) \leq k$ , respectively).

The *diameter* of a graph  $G$  is  $\text{diam}(G) = \max_{u, v \in V} d_G(u, v)$ . The diameter  $\text{diam}_G(S)$  of a set  $S \subseteq V$  is defined as  $\max_{u, v \in S} d_G(u, v)$ . The *radius* of a set  $S \subseteq V$  is defined as  $\min_{u \in V} \max_{v \in S} d_G(u, v)$ . A pair of vertices  $x, y$  of  $G$  is called a *diametral pair* if  $d_G(x, y) = \text{diam}(G)$ . In this case, every shortest path connecting  $x$  and  $y$  is called a *diametral path*.

For a vertex  $v$  of  $G$ ,  $N_G(v) = \{u \in V \mid uv \in E\}$  is called the *open neighbourhood* of  $v$  and  $N_G[v] = N_G(v) \cup \{v\}$  is called the *closed neighbourhood* of  $v$ . Similarly, for a set  $S \subseteq V$ , we define  $N_G(S) = \{u \in V \mid d_G(u, S) = 1\}$ . The *l-neighbourhood* of a vertex  $v$  in  $G$  is  $N_G^l[v] = \{u \mid d_G(u, v) \leq l\}$ . The *l-neighbourhood* of a vertex  $v$  is also called *l-disk* of  $v$ . Two vertices  $u$  and  $v$  are *true twins* if  $N_G[u] = N_G[v]$  and are *false twins* if they are non-adjacent and  $N_G(u) = N_G(v)$ .

The *degree* of a vertex  $v$  is the number of vertices adjacent to it. If a vertex has degree 1, it is called a *pendant* vertex. A graph  $G$  is called *sparse*, if the sum of all vertex degrees is in  $\mathcal{O}(n)$ .

For some vertex  $v$ ,  $L_i^{(v)} = \{u \in V \mid d_G(u, v) = i\}$  denotes the vertices with distance  $i$  from  $v$ . We will also refer to  $L_i^{(v)}$  as the *i-th layer*. For two vertices  $u$  and  $v$ ,  $I_G(u, v) =$

$\{w \mid d_G(u, v) = d_G(u, w) + d_G(w, v)\}$  is the *interval* between  $u$  and  $v$ . The set  $S_i(s, t) = L_i^{(s)} \cap I_G(u, v)$  is called a *slice* of the interval from  $u$  to  $v$ . For any set  $S \subseteq V$  and a vertex  $v$ ,  $\text{Pr}_G(v, S) = \{u \in S \mid d_G(u, v) = d_G(v, S)\}$  denotes the *projection* of  $v$  on  $S$  in  $G$ .

For a vertex set  $S$ , let  $G[S]$  denote the subgraph of  $G$  induced by  $S$ . With  $G - S$ , we denote the graph  $G[V \setminus S]$ . A vertex set  $S$  is a *separator* for two vertices  $u$  and  $v$  in  $G$  if each path from  $u$  to  $v$  contains a vertex  $s \in S$ ; in this case we say  $S$  *separates*  $u$  from  $v$ . If a separator  $S$  contains only one vertex  $s$ , i. e.,  $S = \{s\}$ , then  $s$  is an *articulation point*. A *block* is a maximal subgraph without articulation points.

A *chord* in a cycle is an edge connecting two non-consecutive vertices of the cycle. A cycle is called *induced* if it has no chords. For each  $k \geq 3$ , an induced cycle of length  $k$  is called as  $C_k$ . A subgraph is called *clique* if all its vertices are pairwise adjacent. A *maximal clique* is a clique that cannot be extended by including any additional vertex.

If clear from context, graph identifying indices of notations may be omitted. For example, we may write  $N[v]$  instead of  $N_G[v]$ .

**One-Sided Binary Search.** Consider a sorted sequence  $\langle x_1, x_2, \dots, x_n \rangle$  in which we search for a value  $x_p$ . We say the value  $x_i$  is at position  $i$ . For a one-sided binary search, instead of starting in the middle at position  $n/2$ , we start at position 1. We then process position 2, then position 4, then position 8, and so on until we reach position  $j = 2^i$  and, next, position  $k = 2^{i+1}$  with  $x_j < x_p \leq x_k$ . Then, we perform a classical binary search on the sequence  $\langle x_{j+1}, \dots, x_k \rangle$ . Note that, because  $x_j < x_p \leq x_k$ ,  $2^i < p \leq 2^{i+1}$  and, hence,  $j < p \leq k < 2p$ . Therefore, a one-sided binary search requires at most  $\mathcal{O}(\log p)$  iterations to find  $x_p$ .

## 2.2 Tree-Decompositions

A *tree-decomposition* for a graph  $G = (V, E)$  is a family  $\mathcal{T} = \{B_1, B_2, \dots\}$  of subsets of  $V$ , called *bags*, such that  $\mathcal{T}$  forms a tree with the bags in  $\mathcal{T}$  as nodes which satisfies the following conditions:

- (i) Each vertex is contained in a bag, i. e.,  $V = \bigcup_{B \in \mathcal{B}} B$ ,
- (ii) for each edge  $uv \in E$ ,  $\mathcal{T}$  contains a bag  $B$  with  $u, v \in B$ , and
- (iii) for each vertex  $v \in V$ , the bags containing  $v$  induce a subtree of  $\mathcal{T}$ .

A *path-decomposition* of graph is a tree-decomposition with the restriction that the bags form a path instead of a tree with multiple branches.

**Lemma 2.1 (Tarjan and Yannakakis [88]).** *Let  $\mathcal{F}$  be a set of bags for some graph  $G$ . Determining if  $\mathcal{F}$  forms a tree-decomposition for  $G$  and, if this is the case, computing the corresponding tree can be done in linear time.*

It follows from Lemma 2.1 that we can consider a tree-decomposition interchangeably either as family of bags or as structured tree of bags with defined edges and neighbourhoods.

**Lemma 2.2 (Diestel [29]).** *Let  $K$  be a maximal clique in some graph  $G$ . Then, each tree-decomposition for  $G$  contains a bag  $B$  such that  $K \subseteq B$ .*

**Lemma 2.3.** *Let  $B$  be a bag of a tree-decomposition  $T$  for a graph  $G$  and let  $C$  be a connected component in  $G - B$ . Then,  $T$  contains a bag  $B_C$  with  $B_C \supseteq N_G(C)$  and  $B_C \cap C \neq \emptyset$ .*

*Proof.* Let  $B_C$  be the bag in  $T$  for which  $B_C \cap C \neq \emptyset$  and the distance between  $B$  and  $B_C$  in  $T$  is minimal. Additionally, let  $B'$  be the bag in  $T$  adjacent to  $B_C$  which is closest to  $B$  and let  $S = B_C \cap B'$ . Note that  $S \cap C = \emptyset$  and, by properties of tree-decompositions,  $S$  separates  $C$  from all vertices in  $B \setminus S$ . Assume that there is a vertex  $u \in N_G(C) \setminus S$ . Because  $u \in N_G(C)$ , there is a vertex  $v \in C$  which is adjacent to  $u$ . This contradicts with  $S$  being a separator for  $u$  and  $v$ . Therefore,  $N_G(C) \subseteq S \subseteq B_C$ .  $\square$

A tree-decomposition  $T$  of  $G$  has *breadth*  $\rho$  if, for each bag  $B$  of  $T$ , there is a vertex  $v$  in  $G$  with  $B \subseteq N_G^\rho[v]$ . The *tree-breadth* of a graph  $G$  is  $\rho$ , written as  $\text{tb}(G) = \rho$ , if  $\rho$  is the minimal breadth of all tree-decomposition for  $G$ . Similarly, *path-breadth* of a graph  $G$  is  $\rho$ , written as  $\text{pb}(G) = \rho$ , if  $\rho$  is the minimal breadth of all path-decomposition for  $G$ .

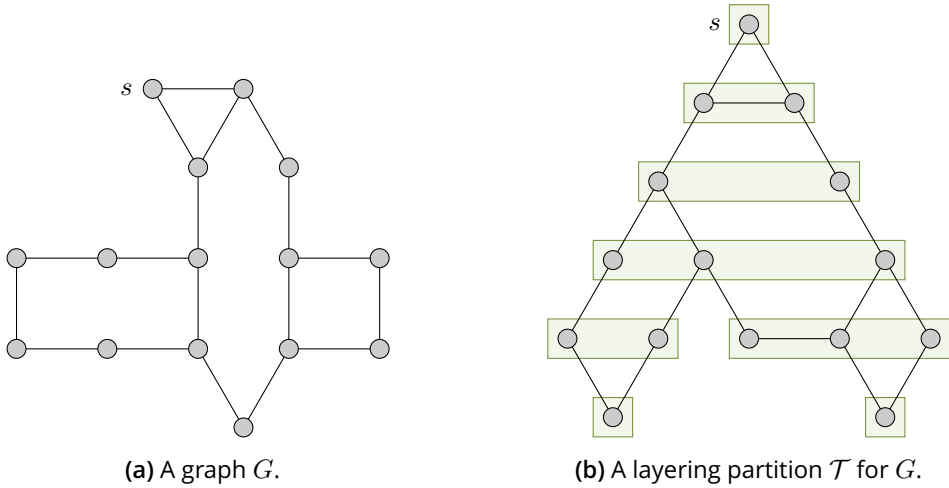
A tree-decomposition  $T$  of  $G$  has *length*  $\lambda$  if, for each bag  $B$  of  $T$ , the diameter of  $B$  is at most  $\lambda$ , i. e., for all vertices  $u, v \in B$ ,  $d_G(u, v) \leq \lambda$ . Accordingly, the *tree-length* of a graph  $G$  is  $\lambda$ , written as  $\text{tl}(G) = \lambda$ , if  $\lambda$  is the minimal breadth of all tree-decomposition for  $G$ . Similarly, *path-length* of a graph  $G$  is  $\lambda$ , written as  $\text{pl}(G) = \lambda$ , if  $\lambda$  is the minimal breadth of all path-decomposition for  $G$ .

Clearly, it follows from their definitions that, for any graph  $G$ ,  $\text{tb}(G) \leq \text{tl}(G) \leq 2\text{tb}(G)$ ,  $\text{pb}(G) \leq \text{pl}(G) \leq 2\text{pb}(G)$ ,  $\text{tb}(G) \leq \text{pb}(G)$ , and  $\text{tl}(G) \leq \text{pl}(G)$ . Additionally, as shown in Lemma 2.4 below, non of these parameter increases when an edge of a graph is contracted.

**Lemma 2.4 (Dragan and Köhler [37]).** *Let  $G$  be a graph and  $H$  be a graph created by contracting an edge of  $G$ . Then, for each  $\phi \in \{\text{tb}, \text{tl}, \text{pb}, \text{pl}\}$ ,  $\phi(H) \leq \phi(G)$ .*

## 2.3 Layering Partitions

BRANDSTÄDT et al. [15] and CHEPOI, DRAGAN [19] introduced a method called *layering partition*. The idea is the following. First, partition the vertices of a given graph  $G$  in distance layers for a given vertex  $s$ . Second, partition each layer  $L_i^{(s)}$  into *clusters* in such a way that two vertices  $u$  and  $v$  are in the same cluster if and only if they are connected by a path only using vertices in the same or upper layers. That is,  $u$  and  $v$  are in the same cluster if and only if, for some  $i$ ,  $\{u, v\} \subseteq L_i^{(s)}$  and there is a path  $P$  from  $u$  to  $v$  in  $G$  such that, for all  $j < i$ ,  $P \cap L_j^{(s)} = \emptyset$ . Note that each cluster  $C$  is a set of vertices of  $G$ , i. e.,  $C \subseteq V$ , and all clusters are pairwise disjoint. The created clusters form a rooted tree  $\mathcal{T}$  with the cluster  $\{s\}$  as root where each cluster is a node of  $\mathcal{T}$  and two clusters  $C$  and  $C'$  are adjacent in  $\mathcal{T}$  if and only if  $G$  contains an edge  $uv$  with  $u \in C$  and  $v \in C'$ . Figure 2.1 gives an example for such a partition.



**Figure 2.1.** Example of a layering partition. A given graph  $G$  (a) and the layering partition of  $G$  generated when starting at vertex  $s$  (b). Example taken from [19].

**Lemma 2.5 (Chepoi and Dragan [19]).** *A layering partition of a given graph can be computed in linear time.*

For a given graph  $G = (V, E)$ , let  $\mathcal{T}$  be a layering partition of  $G$  and let  $\Delta$  be the maximum cluster diameter of  $\mathcal{T}$ . For two vertices  $u$  and  $v$  of  $G$  contained in the clusters  $C_u$  and  $C_v$  of  $\mathcal{T}$ , respectively, we define  $d_{\mathcal{T}}(u, v) := d_{\mathcal{T}}(C_u, C_v)$ .

**Lemma 2.6.** *For all vertices  $u$  and  $v$  of  $G$ ,  $d_{\mathcal{T}}(u, v) \leq d_G(u, v) \leq d_{\mathcal{T}}(u, v) + \Delta$ .*

*Proof.* Clearly, by construction of a layering partition,  $d_{\mathcal{T}}(u, v) \leq d_G(u, v)$  for all vertices  $u$  and  $v$  of  $G$ .

Next, let  $C_u$  and  $C_v$  be the clusters containing  $u$  and  $v$ , respectively. Note that  $\mathcal{T}$  is a rooted tree. Let  $C'$  be the lowest common ancestor of  $C_u$  and  $C_v$ . Therefore,  $d_{\mathcal{T}}(u, v) = d_{\mathcal{T}}(u, C') + d_{\mathcal{T}}(C', v)$ . By construction of a layering partition,  $C'$  contains a vertex  $u'$  and vertex  $v'$  such that  $d_G(u, u') = d_{\mathcal{T}}(u, u')$  and  $d_G(v, v') = d_{\mathcal{T}}(v, v')$ . Since the diameter of each cluster is at most  $\Delta$ ,  $d_G(u, v) \leq d_{\mathcal{T}}(u, u') + \Delta + d_{\mathcal{T}}(v, v') = d_{\mathcal{T}}(u, v) + \Delta$ .  $\square$

## 2.4 Special Graph Classes

**Chordal Graphs.** A graph is *chordal* if every cycle with at least four vertices has a chord. The class of chordal graphs is a well known class which can be recognised in linear time [88]. Due to the strong tree structure of chordal graphs, they have the following property known as  $m$ -convexity:

**Lemma 2.7 (Faber and Jamison [49]).** *Let  $G$  be a chordal graph. If, for two distinct vertices  $u, v$  in a disk  $N^r[x]$ , there is a path  $P$  connecting them with  $P \cap N^r[x] = \{u, v\}$ , then  $u$  and  $v$  are adjacent.*

It is well known [55] that a graph  $G$  is chordal if it admits a so-called *clique tree*. That is,  $G$  admits a tree-decomposition  $T$  such that each bag of  $T$  induces a clique. Such a tree-decomposition can be found in linear time [86].

**Theorem 2.1 (Gavril [55]).** *A graph  $G$  is chordal if and only if  $\text{tl}(G) \leq 1$ .*

A subclass of chordal graphs are *interval graphs*. A graph is an interval graph if it is the intersection graphs of intervals on a line. It is known [56] that a graph  $G$  is an interval graph if and only if  $G$  admits a path-decomposition  $\mathcal{P}$  such that each bag of  $\mathcal{P}$  induces a clique. Such a decomposition  $\mathcal{P}$  can be computed in linear time.

**Theorem 2.2 (Gilmore and Hoffman [56]).** *A graph  $G$  is an interval graph if and only if  $\text{pl}(G) \leq 1$ .*

**Dually Chordal Graphs.** A graph is *dually chordal* if it is the intersection graph of maximal cliques of a chordal graph. In [16], BRANDSTÄDT et al. introduce dually chordal graphs and show that they can be recognised in linear time. It follows from Theorem 2.3 below that dually chordal graphs have tree-breadth 1.

**Theorem 2.3 (Brandstädt et al. [16]).** *A graph  $G = (V, E)$  is dually chordal if and only if the set  $\{N[v] \mid v \in V\}$  forms a valid tree-decomposition for  $G$ .*

**Distance-Hereditary Graphs.** A graph  $G$  is *distance-hereditary* if the distances in any connected induced subgraph of  $G$  are the same as in  $G$ .

**Lemma 2.8 (Bandelt and Mulder [6]).** *Let  $G$  be a distance-hereditary graph, let  $x$  be an arbitrary vertex of  $G$ , and let  $u, v \in L_k^{(x)}$  be in the same connected component of the graph  $G - L_{k-1}^{(x)}$ . Then,  $N(v) \cap L_{k-1}^{(x)} = N(u) \cap L_{k-1}^{(x)}$ .*

Let  $\sigma = \langle v_1, v_2, \dots, v_n \rangle$  be an ordering for the vertices of some graph  $G$ , let  $V_i = \{v_1, v_2, \dots, v_i\}$ , and let  $G_i$  denote the graph  $G[V_i]$ . An ordering  $\sigma$  is called a *pruning sequence* for  $G$  if, for  $1 \leq j < i \leq n$ , each  $v_i$  satisfies one of the following conditions in  $G_i$ :

- (i)  $v_i$  is a pendant vertex,
- (ii)  $v_i$  is a true twin of some vertex  $v_j$ , or
- (iii)  $v_i$  is a false twin of some vertex  $v_j$ .

**Lemma 2.9 (Bandelt and Mulder [6]).** *A graph  $G$  is distance-hereditary if and only if there is a pruning sequence for  $G$ .*

**$\delta$ -Hyperbolic Graphs.** A graph has *hyperbolicity*  $\delta$  if, for any four vertices  $u, v, w$ , and  $x$ , the two larger of the sums  $d(u, v) + d(w, x)$ ,  $d(u, w) + d(v, x)$ , and  $d(u, x) + d(v, w)$  differ by at most  $2\delta$ .

**Lemma 2.10 (Chepoi et al. [20]).** *Let  $u, v, w$ , and  $x$  be four vertices in a  $\delta$ -hyperbolic graph. If  $d(u, w) > \max\{d(u, v), d(v, w)\} + 2\delta$ , then  $d(v, x) < \max\{d(x, u), d(x, w)\}$ .*

**Theorem 2.4 (Chepoi et al. [20]).** *If a graph  $G$  has hyperbolicity  $\delta$ , then  $\delta \leq \text{tl}(G)$  and  $\text{tl}(G) \in \mathcal{O}(\delta \log n)$ .*

**AT-Free Graphs.** An *asteroidal triple* (AT for short) in a graph is a set of three vertices where every two of them are connected by a path avoiding the neighbourhood of the third. A graph is called *AT-free* if it does not contain an asteroidal triple.

A *Lexicographic Breadth-First Search* (LexBFS for short) is a refinement of a standard BFS which produces a vertex ordering  $\sigma: V \rightarrow \{1, \dots, n\}$ . A LexBFS starts at some start vertex  $s$ , orders the vertices of a graph  $G$  by assigning numbers from  $n$  to 1 to the vertices in the order as they are discovered by the following search process. Each vertex  $v$  has a label consisting of a (reverse) ordered list of the numbers of those neighbours of  $v$  that were already visited by the LexBFS; initially this label is empty. The LexBFS starts with some vertex  $s$ , assigns the number  $n$  to  $s$ , and adds this number to the end of the label of all unnumbered neighbours of  $s$ . Then, in each step, the LexBFS selects the unnumbered vertex  $v$  with the lexicographically largest label, assigns the next available number  $k$  to  $v$ , and adds this number to the end of the labels of all unnumbered neighbours of  $v$ . An ordering  $\sigma$  of the vertex set of a graph generated by a LexBFS is called *LexBFS-ordering*. Note that the closer a vertex is to the start vertex  $s$  in  $G$ , the larger its number is in  $\sigma$ . It is known that a LexBFS-ordering of an arbitrary graph can be generated in linear time [81].

For the two lemmas below, assume that we are given an AT-free graph  $G = (V, E)$ , let  $s$  be an arbitrary vertex of  $G$ , let  $x$  be the vertex last visited (numbered 1) by a LexBFS starting at  $s$ , and let  $\sigma$  be the ordering obtained by a LexBFS starting at  $x$ . Additionally, for some vertex  $v \in L_i^{(x)}$ , let  $N_G^\downarrow(v) = N_G(v) \cap L_{i-1}^{(x)}$ .

**Lemma 2.11 (Corneil et al. [25]).** *Let  $y$  be a vertex of  $G$  and let  $P$  be a shortest path from  $x$  to  $y$ . For each vertex  $z$  with  $\sigma(y) \leq \sigma(z) \leq \sigma(x) = n$ ,  $d_G(z, P) \leq 1$ .*

**Lemma 2.12.** *For every integer  $i \geq 1$  and every two non-adjacent vertices  $u, v \in L_i^{(x)}$  of  $G$ ,  $\sigma(v) < \sigma(u)$  implies  $N_G^\downarrow(v) \subseteq N_G^\downarrow(u)$ . In particular,  $d_G(v, u) \leq 2$  holds for every  $u, v \in L_i^{(x)}$  and every  $i$ .*

*Proof.* Consider an arbitrary neighbour  $w \in L_{i-1}^{(x)}$  of  $v$  and a shortest path  $P$  from  $v$  to  $x$  in  $G$  containing  $w$ . Since  $\sigma(v) < \sigma(u)$ , by Lemma 2.11, path  $P$  must dominate vertex  $u$ . Since  $u$  and  $v$  are not adjacent,  $u$  is in  $L_i$ , and all vertices of  $P \setminus \{v, w\}$  belong to layers  $L_j^{(x)}$  with  $j < i - 1$ , vertex  $u$  must be adjacent to  $w$ . Otherwise,  $u, v$ , and  $x$  form an asteroidal triple.  $\square$

# Computing Decompositions with Small Breadth<sup>\*</sup>

For each graph class  $\mathcal{C}$ , a central problem is its recognition. That is, given a graph  $G$ , is  $G$  a member of  $\mathcal{C}$ ? In case of tree- and path-decompositions, we can define recognition as decision problem or as optimization problem. Using tree-breadth as example, the decision problem asks, for a given graph  $G$  and a given integer  $\rho \geq 1$ , if  $\text{tb}(G) \leq \rho$ . The corresponding optimisation problem asks for a decomposition of  $G$  such that the breadth of this decomposition is minimal. In this chapter, we show approaches to determine a tree- or path-decomposition with small breadth (or length) for a given graph. We show results for general graphs as well as for special graph classes.

We know from LOKSHTANOV [72] and DUCCOFFE et al. [48] that it is NP-hard to determine any of the parameters tree-breadth, tree-length, path-breadth, and path-length for a given graph.

**Theorem 3.1 (Lokshtanov [72]).** *For a given graph  $G$  and a fixed  $\lambda \geq 2$ , it is NP-complete to decide if  $\text{tl}(G) \leq \lambda$ .*

**Theorem 3.2 (Ducoffe et al. [48]).** *Given a graph  $G$  and an integer  $k$ , it is NP-complete to decide any of the following: Is  $\text{tb}(G) \leq k$ ? Is  $\text{pb}(G) \leq k$ ? Is  $\text{pl}(G) \leq k$ ?*

## 3.1 Approximation Algorithms

In this section, we present approaches to compute a decomposition for a given graph which approximates the graph's tree-breadth, tree-length, path-breadth, or path-length. Additionally to showing that it is hard to compute any of these parameters optimally, LOKSHTANOV [72] and DUCCOFFE et al. [48] show that a certain approximation quality is hard, too.

**Theorem 3.3 (Lokshtanov [72]).** *If  $P \neq NP$ , then there is no polynomial time algorithm to calculate a tree-decomposition, for a given graph  $G$ , of length smaller than  $\frac{3}{2} \text{tl}(G)$ .*

**Theorem 3.4 (Ducoffe et al. [48]).** *For any  $\epsilon > 0$ , it is NP-hard to approximate the tree-breadth of a given graph by a factor of  $(2 - \epsilon)$ .*

---

<sup>\*</sup> Results from this chapter have been published partially at SWAT 2014, Copenhagen, Denmark [38], at COCOA 2016, Hong Kong, China [71], and in *Algorithmica* [39].

### 3.1.1 Layering Based Approaches

One approach for computing a tree-decomposition for a given graph is to use a layering partition. Lemma 3.1 shows that the radius and diameter of the clusters of a layering partition are bounded by the tree-breadth and tree-length of the underlying graph.

**Lemma 3.1 (Dourisboure et al. [32], Dragan and Köhler [37]).** *In a graph  $G$  with  $\text{tb}(G) = \rho$  and  $\text{tl}(G) = \lambda$ , let  $C$  be a cluster of an arbitrary layering partition for  $G$ . There is a vertex  $w$  with  $d_G(w, u) \leq 3\rho$  for each  $u \in C$ . Also,  $d_G(u, v) \leq 3\lambda$  for all  $u, v \in C$ .*

Recall that the clusters created by a layering partition form a rooted tree. This tree can be transformed into a tree-decomposition by expanding its clusters as follows. For a cluster  $C$ , add all vertices from the parent of  $C$  which are adjacent to a vertex in  $C$ . That is, for each cluster  $C \subseteq L_i^{(s)}$ , create a bag  $B_C = C \cup (N_G(C) \cap L_{i-1}^{(s)})$ . As shown in Lemma 3.1, the radius and diameter of a cluster are at most three times larger than the tree-breadth and -length of  $G$ , respectively. Therefore, the created tree-decomposition has length at most  $3\lambda + 1$  and breadth at most  $3\rho + 1$ . ABU-ATA and DRAGAN [2] slightly improve this observation and show that the breadth of such a tree-decomposition is indeed at most  $3\rho$ .

**Corollary 3.1 (Dourisboure et al. [32], Abu-Ata and Dragan [2]).** *For a given graph  $G$ , a tree-decomposition with breadth  $3\text{tb}(G)$  and length  $3\text{tl}(G) + 1$  can be computed in linear time.*

A similar strategy as above can be used to approximate the path-breadth and -length of a graph. However, there are two major differences. First, a layering partition creates a tree and, second, can start at any vertex. The first difference can be addressed by using a simple BFS-layering. For the second, however, we need to find the right start vertex.

Consider a given start vertex  $s$  and the layers  $L_i^{(s)}$ . We define an *extended layer*  $\mathbb{L}_i^{(s)}$  as follows:

$$\mathbb{L}_i^{(s)} = L_i^{(s)} \cup \left\{ u \mid uv \in E, u \in L_{i-1}^{(s)}, v \in L_i^{(s)} \right\}$$

**Lemma 3.2.** *Let  $\mathcal{P} = \{X_1, X_2, \dots, X_p\}$  be a path-decomposition for  $G$  with length  $\lambda$ , breadth  $\rho$ , and  $s \in X_1$ . Then each extended layer  $\mathbb{L}_i^{(s)}$  has diameter at most  $2\lambda$  and radius at most  $3\rho$ .*

*Proof.* Let  $x$  and  $y$  be two arbitrary vertices in  $L_i^{(s)}$ . Also let  $x'$  and  $y'$  be arbitrary vertices in  $L_{i-1}^{(s)}$  with  $xx', yy' \in E$ .

First, we show that  $\max\{d_G(x, y), d_G(x, y'), d_G(x', y)\} \leq 2\lambda$ . By induction on  $i$ , we may assume that  $d_G(y', x') \leq 2\lambda$  as  $x', y' \in L_{i-1}^{(s)}$ . If there is a bag in  $\mathcal{P}$  containing both vertices  $x$  and  $y$ , then  $d_G(x, y) \leq \lambda$  and, therefore,  $d_G(x, y') \leq \lambda + 1 \leq 2\lambda$  and  $d_G(y, x') \leq \lambda + 1 \leq 2\lambda$ . Assume now that all bags containing  $x$  are earlier in  $\mathcal{P} =$



$\{X_1, X_2, \dots, X_p\}$  than the bags containing  $y$ . Let  $B$  be a bag of  $\mathcal{P}$  containing both ends of edge  $xx'$ . By the position of this bag  $B$  in  $\mathcal{P}$  and the fact that  $s \in X_1$ , any shortest path connecting  $s$  with  $y$  must have a vertex in  $B$ . Let  $w$  be a vertex of  $B$  that is on a shortest path of  $G$  connecting vertices  $s$  and  $y$  and containing edge  $yy'$ . Such a shortest path must exist because of the structure of the layering that starts at  $s$  and puts  $y'$  and  $y$  in consecutive layers. Since  $x, x', w \in B$ , we have  $\max\{d_G(x, w), d_G(x', w)\} \leq \lambda$ . If  $w = y'$ , we are done;  $\max\{d_G(x, y), d_G(x, y'), d_G(x', y)\} \leq \lambda + 1 \leq 2\lambda$ . So, assume that  $w \neq y'$ . Since  $d_G(x, s) = d_G(s, y) = i$  (by the layering) and  $d_G(x, w) \leq \lambda$ , we must have  $d_G(w, y') + 1 = d_G(w, y) = d_G(s, y) - d_G(s, w) = d_G(s, x) - d_G(s, w) \leq d_G(w, x) \leq \lambda$ . Hence,  $d_G(y, x) \leq d_G(y, w) + d_G(w, x) \leq 2\lambda$ ,  $d_G(y, x') \leq d_G(y, w) + d_G(w, x') \leq 2\lambda$ , and  $d_G(y', x) \leq d_G(y', w) + d_G(w, x) \leq 2\lambda - 1$ . Therefore, we conclude that the distance between any two vertices in  $\mathbb{L}_i^{(s)}$  is at most  $2\lambda$ .

As shown above, the radius of each extended layer  $\mathbb{L}_i^{(s)}$  is at most  $4\rho$  (because  $\lambda \leq 2\rho$ ). Consider an extended layer  $\mathbb{L}_i^{(s)}$  and a family  $\mathcal{F} = \{N_G^{2\rho}[u] \mid u \in \mathbb{L}_i^{(s)}\}$  of disks of  $G$ . Since  $d_G(u, v) \leq 4\rho$  for every pair  $u, v \in \mathbb{L}_i^{(s)}$ , the disks of  $\mathcal{F}$  pairwise intersect. Note that each disk  $N_G^{2\rho}[u] \in \mathcal{F}$  induces a subpath of  $\mathcal{P}$ . If subtrees of a tree  $T$  pairwise intersect, they have a common node in  $T$  [9]. Therefore, there is a bag  $X_j \in \mathcal{P}$  such that each disk in  $\mathcal{F}$  intersects  $X_j$ . Let  $w$  be the center of  $X_j$ , i.e.  $X_j \subseteq N_G^\rho[w]$ . Hence, for each vertex  $u$  with  $N_G^{2\rho}[u] \in \mathcal{F}$ ,  $d_G(w, u) \leq 3\rho$ . Thus, for each  $i \in \{1, \dots, \text{ecc}(s)\}$  there is a vertex  $w_i$  with  $\mathbb{L}_i^{(s)} \subseteq N_G^{3\rho}[w_i]$ .  $\square$

Using Lemma 3.2, Algorithm 3.1 below computes a 3-approximation for the path-breadth and a 2-approximation for the path-length of a given graph.

---

**Algorithm 3.1:** A 2-approximation algorithm for computing the path-length of a graph (respectively, 3-approximation for path-breadth).

---

**Input:** A graph  $G = (V, E)$ .

**Output:** A path-decomposition for  $G$ .

- 1 Compute the pairwise distances of all vertices in  $G$ .
  - 2 **For Each**  $s \in V$
  - 3     Calculate a decomposition  $\mathcal{L}(s) = \{\mathbb{L}_i^{(s)} \mid 1 \leq i \leq \text{ecc}(s)\}$ .
  - 4     Determine the length  $l(s)$  of  $\mathcal{L}(s)$  (breadth  $b(s)$ , respectively).
  - 5 Output a decomposition  $\mathcal{L}(s)$  for which  $l(s)$  ( $b(s)$ , respectively) is minimal.
- 

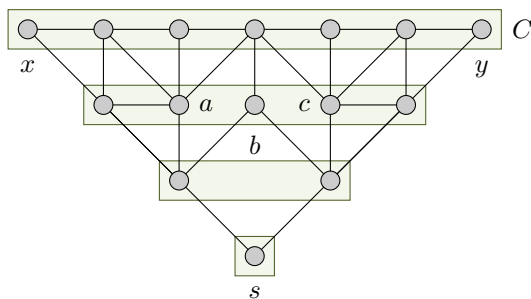
**Theorem 3.5.** *Algorithm 3.1 computes a path-decomposition with length  $2\text{pl}(G)$  (with breadth  $3\text{pb}(G)$ , respectively) of a given graph  $G$  in  $\mathcal{O}(n^3)$  time.*

*Proof.* Let  $\text{pb}(G) = \rho$  and  $\text{pl}(G) = \lambda$ . By Lemma 3.2, there is a vertex  $s$  for which each extended layer  $\mathbb{L}_i^{(s)}$  has diameter at most  $2\lambda$  (radius at most  $3\rho$ , respectively). Thus, the algorithm creates and outputs a decomposition  $\mathcal{L}(s)$  with length at most  $2\lambda$  (breadth at most  $3\rho$ , respectively).

Determining pairwise distances and creating the decomposition  $\mathcal{L}(s)$  for each vertex  $s$  can be done in  $\mathcal{O}(nm)$  total time. Then, for a single vertex  $s$ , the length and breadth of the decomposition  $\mathcal{L}(s)$  can be calculated in  $\mathcal{O}(n^2)$  time. Thus, Algorithm 3.1 runs in  $\mathcal{O}(n^3)$  total time.  $\square$

### 3.1.2 Neighbourhood Based Approaches

In the previous subsection, we use distance layerings to construct a decomposition for a given graph. While this approach is simple, it has limitations. Consider the graph  $G$  in Figure 3.1.  $G$  has tree-breadth 1 and -length 2. However, a layering partition starting at  $s$  creates a cluster  $C$  with radius 3 and diameter 6. Now, one can create a graph  $H$  containing two copies of  $G$  such that the  $s$ -vertices are connected by a path. Therefore, any layering partition of  $H$  creates such a cluster  $C$ .



**Figure 3.1.** A graph  $G$  where a layering partition creates a cluster  $C$  with radius  $3\text{tb}(G)$  and diameter  $3\text{tl}(G)$ . The family of disks  $\{N[a], N[b], N[c], N[x], N[y]\}$  gives a tree-decomposition with breadth 1 and length 2. The cluster  $C$  has radius 3 (for all  $v \in C$ ,  $d(s, v) = 3$ ) and diameter 6 ( $d(x, y) = 6$ ).

An alternative approach to distance layers is to carefully create a bag from the neighbourhood of a vertex. Then add the bag to the tree-decomposition, pick a new vertex, and repeat the process. This approach is clearly more complicated. However, it has a better chance to determine a good decomposition for a given graph.

In [33], DOURISBOURE and GAVOILLE present such an algorithm to compute a tree-decomposition which approximates the tree-length of a given graph. They show that, for a given graph  $G$ , their algorithm successfully computes a tree-decomposition with length at most  $6\text{tl}(G) - 4$ . Additionally, they conjecture that their algorithm can also find a tree-decomposition with length at most  $2\text{tl}(G)$ . However, YANCEY [93] is able to construct a counter example for this conjecture.

In what follows, we present an algorithm which computes a tree-decomposition with breadth at most  $3\text{tb}(G) - 1$  for a given graph  $G$ . To do so, assume for the remainder of this subsection that we are given a graph  $G$  with  $\text{tb}(G) = \rho$ .

For some vertex  $u$  and some positive integer  $\phi$ , let  $\mathcal{C}_G^\phi[u]$  denote the set of connected components in  $G - N_G^\phi[u]$ . We say that a vertex  $v$  is a  $\phi$ -partner of  $u$  for some  $C \in \mathcal{C}_G^\phi[u]$  if  $N_G^\phi[v] \supseteq N_G(C)$  and  $N_G^\phi[v] \cap C \neq \emptyset$ .

**Lemma 3.3.** *Let  $u$  be an arbitrary vertex of  $G$ . If  $\phi \geq 3\rho - 1$ , then  $G$  contains, for each connected component  $C \in \mathcal{C}_G^\phi[u]$ , a vertex  $v$  such that  $v$  is a  $\phi$ -partner of  $u$  for  $C$ .*

*Proof.* Let  $T$  be a tree-decomposition for  $G$  with breadth  $\rho$  and let  $B_u$  be a bag of  $T$  containing  $u$ . Without loss of generality, let  $T$  be rooted in  $B_u$ . Now, let  $B_C$  be the bag of  $T$  for which  $B_C \cap C \neq \emptyset$  and which is closest to  $B_u$  in  $T$ . Additionally, let  $B'_C$  be the parent of  $B_C$  and let  $S = B_C \cap B'_C$ . Note that, by definition of  $B_C$ ,  $S \subseteq N_G^\phi[u]$  and  $B_C$  contains a vertex  $x$  with  $d_G(u, x) > \phi$ . Recall that the distance between two vertices in a bag is at most  $2\rho$ . Thus,  $d_G(u, s) < \rho$  implies that  $d_G(u, x) < 3\rho \leq \phi$ . Therefore, for all  $s \in S$ ,  $d_G(u, s) \geq \rho$ .

Let  $v$  be the center of  $B_C$ . Because  $B_C$  intersects  $C$  and  $T$  has breadth  $\rho$ , clearly,  $d_G(v, C) \leq \rho < \phi$  and, hence,  $C \cap N_G^\phi[v] \neq \emptyset$ . Let  $x$  be a vertex of  $G$  in  $N_G(C)$ . By properties of tree-decompositions, each path from  $u$  to  $x$  intersect  $S$ . Thus, there is a vertex  $s \in S$  with  $d_G(u, x) = d_G(u, s) + d_G(s, x)$ . Because  $d_G(u, s) \geq \rho \geq d_G(v, s)$ , it follows that  $d_G(v, x) \leq d_G(u, x) \leq \phi$  and, thus,  $N(C) \subseteq N_G^\phi[v]$ . Therefore,  $v$  is a  $\phi$ -partner of  $u$  for  $C$ .  $\square$

**Lemma 3.4.** *Let  $C$  be a connected component in  $G - B_u$  for some  $B_u \subseteq N_G^\phi[u]$  and some positive  $\phi$ . Also, let  $C \in \mathcal{C}_G^\phi[u]$  and let  $v$  be a  $\phi$ -partner of  $u$  for  $C$ . Then, for all connected components  $C_v$  in  $G[C] - N_G^\phi[v]$ ,  $C_v \in \mathcal{C}_G^\phi[v]$ .*

*Proof.* Consider a connected component  $C_v$  in  $G[C] - N_G^\phi[v]$ . Clearly,  $C_v \subseteq C$  and there is a connected component  $C' \in \mathcal{C}_G^\phi[v]$  such that  $C' \supseteq C_v$ .

Let  $x$  be an arbitrary vertex in  $C'$ . Then, there is a path  $P \subseteq C'$  from  $x$  to  $C_v$ . Because  $N_G(C) \subseteq B_u$  and  $v$  is a  $\phi$ -partner of  $u$  for  $C$ ,  $N_G(C) \subseteq B_u \cap N_G^\phi[v]$ . Also,  $N_G(C)$  separates all vertices in  $C$  from all other vertices in  $G$ . Therefore,  $x \in C$  and  $C' \subseteq C$ ; otherwise,  $P$  would intersect  $N_G^\phi[v]$ . It follows that each vertex in  $P$  is in the same connected component of  $G[C] - N_G^\phi[v]$  and, thus,  $C_v = C'$ .  $\square$

Based on Lemma 3.3 and Lemma 3.4, we can compute, for a given  $\phi \geq 3\rho - 1$ , a tree-decomposition with breadth  $\phi$  as follows. Pick a vertex  $u$  and make it center of a bag  $B_u = N_G^\phi$ . By Lemma 3.3,  $u$  has a  $\phi$ -partner  $v$  for each connected component  $C_u \in \mathcal{C}_G^\phi[u]$ . Then,  $N_G^\phi[v]$  splits  $C_u$  in more connected components  $C_v$ . Due to Lemma 3.4, any of these components  $C_v$  is in  $\mathcal{C}_G^\phi[v]$ . Thus,  $v$  has a  $\phi$ -partner  $w$  for  $C_v$ . Hence, create a bag  $B_v = N_G^\phi[v] \cap (B_u \cup C_u)$  and continue this until the whole graph is covered. Algorithm 3.2 below implements this approach.

**Theorem 3.6.** *Algorithm 3.2 successfully constructs, for a given graph  $G$  and a positive integer  $\phi$ , a tree-decomposition  $T$  with breadth  $\phi$  in  $\mathcal{O}(n^3)$  time if  $\phi \geq 3\text{tb}(G) - 1$ .*

*Proof (Correctness).* To show the correctness of the algorithm, we have to show that the created tree-decomposition  $T$  is a valid tree-decomposition for  $G$  if  $\phi \geq 3\text{tb}(G) - 1$ . To do so, we show the following invariant for the loop starting in line 4: (i)  $T$  is a valid

---

**Algorithm 3.2:** Constructs, for a given graph  $G = (V, E)$  and a given positive integer  $\phi$ , a tree-decomposition  $T$  with breadth  $\phi$ .

---

- Input:** A graph  $G = (V, E)$  and a positive integer  $\phi$ .  
**Output:** A tree-decomposition  $T$  for  $G$  if  $\phi \geq 3 \text{tb}(G) - 1$ .
- 1 Determine the pairwise distances of all vertices and create an empty tree-decomposition  $T$ .
  - 2 Initialise an empty queue  $Q$  of triples  $(v, B, C)$  where  $v$  is a vertex of  $G$  and  $B$  is a bag and  $C$  is a connected component, i. e.,  $B$  and  $C$  are vertex sets.
  - 3 Pick an arbitrary vertex  $u$  and insert  $(u, \emptyset, V)$  into  $Q$ .
  - 4 **While**  $Q$  is non-empty.
    - 5 Remove a triple  $(v, B_u, C_u)$  from  $Q$ .
    - 6 Create a bag  $B_v := N_G^\phi[v] \cap (B_u \cup C_u)$  and add  $B_v$  into  $T$ .
    - 7 **For Each** connected component  $C_v$  in  $G[C_u] - B_v$ 
      - 8 Find a  $\phi$ -partner  $w$  of  $v$  for  $C_v$ .
      - 9 If no such  $w$  exists, **Stop**.  $\phi < 3 \text{tb}(G) - 1$ .
      - 10 Insert  $(w, B_v, C_v)$  into  $Q$ .
  - 11 Output  $T$ .
- 

tree-decomposition with breadth  $\phi$  for the subgraph covered by  $T$ , (ii) for each connected component  $C$  in  $G - T$ ,  $Q$  contains a triple  $(v, B, C)$ , and (iii) for each triple  $(v, B_u, C_u)$  in  $Q$ ,  $B_u$  is in  $T$ ,  $N_G(C_u) \subseteq B_u$ ,  $C_u \in \mathcal{C}_G[u]$ , and  $v$  is a  $\phi$ -partner of  $u$  for  $C_u$ .

Let  $u$  be the vertex of  $G$  selected in line 3. In the first iteration of the loop, the triple  $(u, \emptyset, V)$  is removed from  $Q$ . Then, the algorithm creates the bag  $B_u = N_G^\phi[u] \cap (\emptyset \cup V) = N_G^\phi[u]$  (line 6) and adds it into  $T$ . Clearly, since  $B_u$  is the only bag of  $T$  at this point, condition (i) is satisfied. Additionally, each connected component  $C$  in  $G - T$  is also in  $\mathcal{C}[u]$ . Next, the algorithm determines a  $\phi$ -partner  $v$  of  $u$  for each  $C_u \in \mathcal{C}[u]$  (line 8). Due to Lemma 3.3, such a  $v$  can be found for each such component  $C_u$ . Therefore, after inserting the triples  $(v, B_u, C_u)$  into  $Q$  (line 10), condition (ii) and condition (iii) are satisfied after the first iteration of the loop.

Now, assume by induction that the invariant holds each time line 4 is checked. If  $Q$  is empty, it follows from condition (ii) that  $T$  covers the whole graph and, thus, by condition (i),  $T$  is a valid tree-decomposition for  $G$ . If  $Q$  is not empty, it contains a triple  $(v, B_u, C_u)$  which is then removed by the algorithm. Because of condition (iii), it follows that  $B_v$  as created in line 6 contains  $N_G(C_u)$  and, thus,  $N_G(C_u) \subseteq B_u \cap B_v$ . Therefore, because  $C_u$  is not covered by  $T$ ,  $T$  remains a valid tree-decomposition after adding  $B_v$ , i. e., condition (i) is satisfied. The bag  $B_v$  splits  $C_u$  in a set  $\mathcal{C}'_v$  of connected components such that, for each  $C' \in \mathcal{C}'_v$ ,  $N_G(C') \subseteq B_v$  and, by Lemma 3.4,  $C' \in \mathcal{C}[v]$ . Therefore, due to Lemma 3.3,  $v$  has a  $\phi$ -partner  $w$  for each  $C' \in \mathcal{C}'_v$ . Thus, finding such a  $\phi$ -partner (line 8) is successful and, after inserting the triples  $(w, B_v, C_v)$  into  $Q$  (line 10), condition (ii) and condition (iii) are satisfied.  $\square$

---

*Proof (Complexity).* Determining the pairwise distances of all vertices, initialising  $Q$  and  $T$ , as well as inserting the first triple  $(u, \emptyset, V)$  into  $Q$  (line 1 to line 3) can be clearly done in  $\mathcal{O}(nm)$  time.

In each iteration of the loop starting in line 4, the algorithm splits a connected component  $C_u$  into a set of connected components  $C_v$ . Because  $v$  is a  $\phi$ -partner of  $u$ , it follows that  $B_v \cap C_v \neq \emptyset$  and, hence,  $C_v \subset C_u$ . Note that the created components  $C_v$  are pairwise disjoint. Therefore, the algorithm creates at most  $n$  connected components  $C_v$ , the set of vertices covered by  $T$  strictly grows, and, thus, there are at most  $n$  iterations of the loop.

Because the pairwise distances are known, it takes at most  $\mathcal{O}(n)$  time to create the bag  $B_v$  and to add it into  $T$  (line 6). Determining the connected components  $C_v$  of  $G[C_u] - B_v$  and determining  $N_G(C_v)$  for each such  $C_v$  can be easily done in  $\mathcal{O}(m)$  time. Therefore, when excluding line 8, a single iteration of the loop starting in line 4 takes at most  $\mathcal{O}(m)$  time.

Let  $\mathbb{C}$  be the set of all connected components created by the algorithm. Clearly, for each connected component  $C \in \mathbb{C}$ ,  $|N_G(C)| \leq n$ . Therefore, because at most  $n$  connected components  $C_v$  are created,  $\sum_{C \in \mathbb{C}} |N_G(C)| \leq n^2$ . Because distances can be checked in constant time, it follows that, for a single vertex  $w$ , it takes at most  $\mathcal{O}(n^2)$  total time to determine, for all  $C \in \mathbb{C}$ , if  $w$  is a  $\phi$ -partner for  $C$ . Therefore, line 8 requires at most  $\mathcal{O}(n^3)$  total time over all iterations.

Hence, Algorithm 3.2 runs in total  $\mathcal{O}(n^3)$  time.  $\square$

If we want to use Algorithm 3.2 to find a decomposition with small breadth for a given graph  $G$ , we have to try different values of  $\phi$ . One way is to perform a one-sided binary search over  $\phi$ : If Algorithm 3.2 creates a valid tree-decomposition, decrease  $\phi$ . Otherwise, increase  $\phi$ . Therefore:

**Corollary 3.2.** *For a given graph  $G$  with tree-breadth  $\rho$ , one can compute a tree-decomposition with breadth  $3\rho - 1$  in  $\mathcal{O}(n^3 \log \rho)$  time.*

## 3.2 Strong Tree-Breadth

By definition, a tree-decomposition has breadth  $\rho$  if each bag  $B$  is the subset of the  $\rho$ -neighbourhood of some vertex  $v$ , i. e., the set of bags is the set of *subsets* of the  $\rho$ -neighbourhoods of *some* vertices. Recall that tree-breadth 1 graphs contain the class of dually chordal graphs which can be defined as follows: A graph  $G$  is dually chordal if it admits a tree-decomposition  $T$  such that, for each vertex  $v$  in  $G$ ,  $T$  contains a bag  $B = N_G[v]$  (see Theorem 2.3, page 8). That is, the set of bags in  $T$  is the set of *complete* neighbourhoods of *all* vertices.

In this chapter, we investigate the case which lays between dually chordal graphs and general tree-breadth  $\rho$  graphs. In particular, tree-decompositions are considered where the set of bags are the *complete*  $\rho$ -neighbourhoods of *some* vertices. We call this *strong*

*tree-breadth*. The *strong breadth* of a tree-decomposition is  $\rho$ , if, for each bag  $B$ , there is a vertex  $v$  such that  $B = N_G^\rho[v]$ . Accordingly, a graph  $G$  has strong tree-breadth smaller than or equal to  $\rho$  (written as  $\text{stb}(G) \leq \rho$ ) if there is a tree-decomposition for  $G$  with strong breadth at most  $\rho$ .

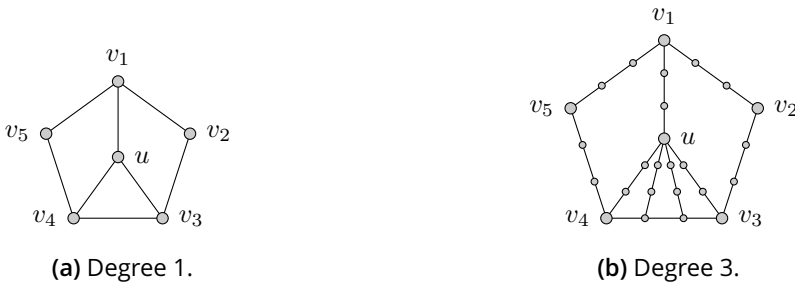
### 3.2.1 NP-Completeness

In this section, we show that it is NP-complete to determine if a given graph has strong tree-breadth  $\rho$  even if  $\rho = 1$ . To do so, we show first that, for some small graphs, the choice of possible centers is restricted. Then, we use these small graphs to construct a reduction.

**Lemma 3.5.** *Let  $C = \{v_1, v_2, v_3, v_4\}$  be an induced  $C_4$  in a graph  $G$  with the edge set  $\{v_1v_2, v_2v_3, v_3v_4, v_4v_1\}$ . If there is no vertex  $w \notin C$  with  $N_G[w] \supseteq C$ , then  $N_G[v_1]$  and  $N_G[v_2]$  cannot both be bags in the same tree-decomposition with strong breadth 1.*

*Proof.* Assume that there is a decomposition  $T$  with strong breadth 1 containing the bags  $B_1 = N_G[v_1]$  and  $B_2 = N_G[v_2]$ . Because  $v_3$  and  $v_4$  are adjacent, there is a bag  $B_3 \supseteq \{v_3, v_4\}$ . Consider the subtrees  $T_1, T_2, T_3$ , and  $T_4$  of  $T$  induced by  $v_1, v_2, v_3$ , and  $v_4$ , respectively. These subtrees pairwise intersect in the bags  $B_1, B_2$ , and  $B_3$ . Because pairwise intersecting subtrees of a tree have a common vertex,  $T$  contains a bag  $N_G[w] \supseteq C$ . Note that there is no  $v_i \in C$  with  $N_G[v_i] \supseteq C$ . Thus,  $w \notin C$ . This contradicts with the condition that there is no vertex  $w \notin C$  with  $N_G[w] \supseteq C$ .  $\square$

Let  $C = \{v_1, \dots, v_5\}$  be a  $C_5$  with the edges  $E_5 = \{v_1v_2, v_2v_3, \dots, v_5v_1\}$ . We call the graph  $H = (C \cup \{u\}, E_5 \cup \{uv_1, uv_3, uv_4\})$ , with  $u \notin C$ , an *extended  $C_5$  of degree 1* and refer to the vertices  $u, v_1, v_2$ , and  $v_5$  as *middle, top, right, and left vertex* of  $H$ , respectively. Based on  $H = (V_H, E_H)$ , we construct an *extended  $C_5$  of degree  $\rho$*  (with  $\rho > 1$ ) as follows. First, replace each edge  $xy \in E_H$  by a path of length  $\rho$ . Second, for each vertex  $w$  on the shortest path from  $v_3$  to  $v_4$ , connect  $u$  with  $w$  using a path of length  $\rho$ . Figure 3.2 gives an illustration.



**Figure 3.2.** Two *extended  $C_5$*  of (a) degree 1 and (b) degree 3. We refer to the vertices  $u, v_1, v_2$ , and  $v_5$  as *middle, top, right, and left vertex*, respectively.

**Lemma 3.6.** *Let  $H$  be an extended  $C_5$  of degree  $\rho$  in a graph  $G$  as defined above. Additionally, let  $H$  be a block of  $G$  and its top vertex  $v_1$  be the only articulation point of  $G$  in  $H$ . Then, there is no vertex  $w$  in  $G$  with  $d_G(w, v_1) < \rho$  which is the center of a bag in a tree-decomposition for  $G$  with strong breadth  $\rho$ .*

*Proof.* Let  $T$  be a tree-decomposition for  $G$  with strong breadth  $\rho$ . Assume that  $T$  contains a bag  $B_w = N_G^\rho[w]$  with  $d_G(w, v_1) < \rho$ . Note that the distance from  $v_1$  to any vertex on the shortest path from  $v_3$  to  $v_4$  is  $2\rho$ . Hence,  $G - B_w$  has a connected component  $C$  containing the vertices  $v_3$  and  $v_4$ . Then, by Lemma 2.3 (page 6), there has to be a vertex  $w' \neq w$  in  $G$  and a bag  $B'_w = N_G^\rho[w']$  in  $T$  such that (i)  $B'_w \supseteq N_G(C)$  and (ii)  $B'_w \cap C \neq \emptyset$ . Thus, if we can show, for a given  $w$ , that there is no such  $w'$ , then  $w$  cannot be center of a bag.

First, consider the case that  $w$  is in  $H$ . We construct a set  $X = \{x, y\} \subseteq N_G(C)$  such that there is a unique shortest path from  $x$  to  $y$  in  $G$  passing  $w$ . If  $w = v_1$ , let  $x = v_2$  and  $y = v_5$ . If  $w$  is on the shortest path from  $v_1$  to  $u$ , let  $x$  and  $y$  be on the shortest path from  $v_1$  to  $v_2$  and from  $v_4$  to  $u$ , respectively. If  $w$  is on the shortest path from  $v_1$  to  $v_2$ , let  $x$  and  $y$  be on the shortest path from  $v_1$  to  $v_5$  and from  $v_2$  to  $v_3$ , respectively. In each case, there is a unique shortest path from  $x$  to  $y$  passing  $w$ . Note that, for all three cases,  $d_G(v_1, y) \geq \rho$ . Thus, each  $w'$  with  $d_G(w', y) \leq \rho$  is in  $H$ . Therefore,  $w$  is the only vertex in  $G$  with  $X \subseteq N_G^\rho[w]$ , i. e., there is no vertex  $w' \neq w$  satisfying condition (i). This implies that  $w$  cannot be center of a bag in  $T$ .

Next, consider the case that  $w$  is not in  $H$ . Without loss of generality, let  $w$  be a center for which  $d_G(v_1, w)$  is minimal. As shown above, there is no vertex  $w'$  in  $H$  with  $d_G(v_1, w') < \rho$  which is center of a bag. Hence,  $w'$  is not in  $H$  either. However, because  $v_1$  is an articulation point,  $w'$  has to be closer to  $v_1$  than  $w$  to satisfy condition (ii). This contradicts with  $d_G(v_1, w)$  being minimal. Therefore, there is no vertex  $w'$  satisfying condition (ii) and  $w$  cannot be center of a bag in  $T$ .  $\square$

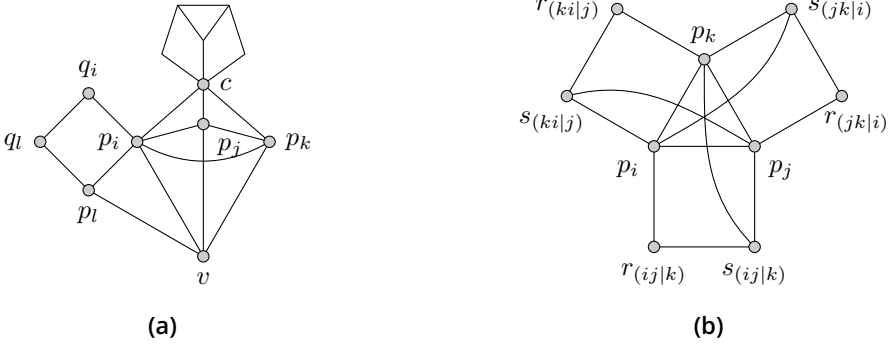
**Theorem 3.7.** *It is NP-complete to decide, for a given graph  $G$ , if  $\text{stb}(G) = 1$ .*

*Proof.* Clearly, the problem is in NP: Select non-deterministically a set  $S$  of vertices such that their neighbourhoods cover each vertex and each edge. Then, check deterministically if the neighbourhoods of the vertices in  $S$  give a valid tree-decomposition. This can be done in linear time (see Lemma 2.1, page 2.1).

To show that the problem is NP-hard, we make a reduction from 1-in-3-SAT [83]. That is, you are given a boolean formula in CNF with at most three literals per clause; find a satisfying assignment such that, in each clause, only one literal becomes true.

Let  $\mathcal{I}$  be an instance of 1-in-3-SAT with the literals  $\mathcal{L} = \{p_1, \dots, p_n\}$ , the clauses  $\mathcal{C} = \{c_1, \dots, c_m\}$ , and, for each  $c \in \mathcal{C}$ ,  $c \subseteq \mathcal{L}$ . We create a graph  $G = (V, E)$  as follows. Create a vertex for each literal  $p \in \mathcal{L}$  and, for all literals  $p_i$  and  $p_j$  with  $p_i \equiv \neg p_j$ , create an induced  $C_4 = \{p_i, p_j, q_i, q_j\}$  with the edges  $p_i p_j$ ,  $q_i q_j$ ,  $p_i q_i$ , and  $p_j q_j$ . For each clause  $c \in \mathcal{C}$  with  $c = \{p_i, p_j, p_k\}$ , create an extended  $C_5$  with  $c$  as top vertex, connect  $c$  with an edge

to all literals it contains, and make all literals in  $c$  pairwise adjacent, i. e., the vertex set  $\{c, p_i, p_j, p_k\}$  induces a maximal clique in  $G$ . Additionally, create a vertex  $v$  and make  $v$  adjacent to all literals. Figure 3.3a gives an illustration for the construction so far.



**Figure 3.3.** Illustration to the proof of Theorem 3.7. The graphs shown are subgraphs of  $G$  as created by a clause  $c = \{p_i, p_j, p_k\}$  and a literal  $p_l$  with  $p_l \equiv \neg p_i$ .

Next, for each clause  $\{p_i, p_j, p_k\}$  and for each  $(xy|z) \in \{(ij|k), (jk|i), (ki|j)\}$ , create the vertices  $r_{(xy|z)}$  and  $s_{(xy|z)}$ , make  $r_{(xy|z)}$  adjacent to  $s_{(xy|z)}$  and  $p_x$ , and make  $s_{(xy|z)}$  adjacent to  $p_y$  and  $p_z$ . See Figure 3.3b for an illustration. Note that  $r_{(ij|k)}$  and  $s_{(ij|k)}$  are specific for the clause  $\{p_i, p_j, p_k\}$ . Thus, if  $p_i$  and  $p_j$  are additionally in a clause with  $p_l$ , then we also create the vertices  $r_{(ij|l)}$  and  $s_{(ij|l)}$ . For the case that a clause only contains two literals  $p_i$  and  $p_j$ , create the vertices  $r_{(ij)}$  and  $s_{(ij)}$ , make  $r_{(ij)}$  adjacent to  $p_i$  and  $s_{(ij)}$ , and make  $s_{(ij)}$  adjacent to  $p_j$ , i. e.,  $\{p_i, p_j, r_{(ij)}, s_{(ij)}\}$  induces a  $C_4$  in  $G$ .

For the reduction, first, consider the case that  $\mathcal{I}$  is a *yes*-instance for 1-in-3-SAT. Let  $f: \mathcal{P} \rightarrow \{T, F\}$  be a satisfying assignment such that each clause contains only one literal  $p_i$  with  $f(p_i) = T$ . Select the following vertices as centers of bags:  $v$ , the middle, left and right vertex of each extended  $C_5$ ,  $p_i$  if  $f(p_i) = T$ , and  $q_j$  if  $f(p_j) = F$ . Additionally, for each clause  $\{p_i, p_j, p_k\}$  with  $f(p_i) = T$ , select the vertices  $s_{(ij|k)}$ ,  $r_{(jk|i)}$ , and  $r_{(ki|j)}$ . The neighbourhoods of the selected vertices give a valid tree-decomposition for  $G$ . Therefore,  $\text{stb}(G) = 1$ .

Next, assume that  $\text{stb}(G) = 1$ . Recall that, for a clause  $c = \{p_i, p_j, p_k\}$ , the vertex set  $\{c, p_i, p_j, p_k\}$  induces a maximal clique  $K_c$  in  $G$ . Because each maximal clique is contained completely in some bag (Lemma 2.2, page 6), some vertex in  $K_c$  is center of such a bag. By Lemma 3.6,  $c$  cannot be center of a bag because it is top of an extended  $C_5$ . Therefore, at least one vertex in  $\{p_i, p_j, p_k\}$  must be center of a bag. Without loss of generality, let  $p_i$  be a center of a bag. By construction,  $p_i$  is adjacent to all  $p \in \{p_j, p_k, p_l\}$ , where  $p_l \equiv \neg p_i$ . Additionally,  $p$  and  $p_i$  are vertices in an induced  $C_4$ , say  $C$ , and there is no vertex  $w$  in  $G$  with  $N_G[w] \supseteq C$ . Thus, by Lemma 3.5, at most one vertex in  $\{p_i, p_j, p_k\}$  can be center of a bag. Therefore, the function  $f: \mathcal{L} \rightarrow \{T, F\}$  defined as

$$f(p_i) = \begin{cases} T & \text{if } p_i \text{ is center of a bag,} \\ F & \text{else} \end{cases}$$



is a satisfying assignment for  $\mathcal{I}$ . □

In [48], DUCCOFFE et al. show how to construct a graph  $G'_\rho$  based on a given graph  $G$  such that  $\text{tb}(G'_\rho) = 1$  if and only if  $\text{tb}(G) \leq \rho$ . We slightly extend their construction to achieve a similar result for strong tree-breadth.

Consider a given graph  $G = (V, E)$  with  $\text{stb}(G) = \rho$ . We construct  $G'_\rho$  as follows. Let  $V = \{v_1, v_2, \dots, v_n\}$ . Add the vertices  $U = \{u_1, u_2, \dots, u_n\}$  and make them pairwise adjacent. Additionally, make each vertex  $u_i$ , with  $1 \leq i \leq n$ , adjacent to all vertices in  $N_G^\rho[v_i]$ . Last, for each  $v_i \in V$ , add an extended  $C_5$  of degree 1 with  $v_i$  as top vertex.

**Lemma 3.7.**  $\text{stb}(G) \leq \rho$  if and only if  $\text{stb}(G'_\rho) = 1$ .

*Proof.* First, consider a tree-decomposition  $T$  for  $G$  with strong breadth  $\rho$ . Let  $T'_\rho$  be a tree-decomposition for  $G'_\rho$  created from  $T$  by adding all vertices in  $U$  into each bag of  $T$  and by making the center, left, and right vertices of each extended  $C_5$  centers of bags. Because the set  $U$  induces a clique in  $G'_\rho$  and  $N_G^\rho[v_i] = N_{G'_\rho}[u_i] \cap V$ , each bag of  $T'_\rho$  is the complete neighbourhood of some vertex.

Next, consider a tree-decomposition  $T'_\rho$  for  $G'_\rho$  with strong breadth 1. Note that each vertex  $v_i$  is top vertex of some extended  $C_5$ . Thus,  $v_i$  cannot be center of a bag. Therefore, each edge  $v_i v_j$  is in a bag  $B_k = N_{G'_\rho}[u_k]$ . By construction of  $G'_\rho$ ,  $B_k \cap V = N_G^\rho[v_k]$ . Thus, we can construct a tree-decomposition  $T$  for  $G$  with strong breadth  $\rho$  by creating a bag  $B_i = N_G^\rho[v_i]$  for each bag  $N_{G'_\rho}[u_i]$  of  $T'_\rho$ . □

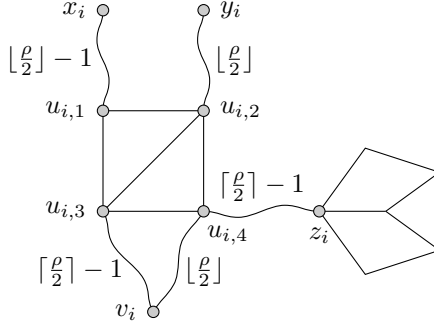
Next, consider a given graph  $G = (V, E)$  with  $V = \{v_1, v_2, \dots, v_n\}$  and  $\text{stb}(G) = 1$ . For a given  $\rho > 1$ , we obtain the graph  $G_\rho^+$  by doing the following for each  $v_i \in V$ :

- Add the vertices  $u_{i,1}, \dots, u_{i,5}, x_i$ , and  $y_i$ .
- Add an extended  $C_5$  of degree  $\rho$  with the top vertex  $z_i$ .
- Connect
  - $u_{i,1}$  and  $x_i$  with a path of length  $\lfloor \rho/2 \rfloor - 1$ ,
  - $u_{i,2}$  and  $y_i$  with a path of length  $\lfloor \rho/2 \rfloor$ ,
  - $u_{i,3}$  and  $v_i$  with a path of length  $\lceil \rho/2 \rceil - 1$ ,
  - $u_{i,4}$  and  $v_i$  with a path of length  $\lfloor \rho/2 \rfloor$ , and
  - $u_{i,4}$  and  $z_i$  with a path of length  $\lceil \rho/2 \rceil - 1$ .
- Add the edges  $u_{i,1}u_{i,2}, u_{i,1}u_{i,3}, u_{i,2}u_{i,3}, u_{i,2}u_{i,4}$ , and  $u_{i,3}u_{i,4}$ .

Note that, for small  $\rho$ , it can happen that  $v_i = u_{i,4}$ ,  $x_i = u_{i,1}$ ,  $y_i = u_{i,2}$ , or  $z_i = u_{i,5}$ . Figure 3.4 gives an illustration.

**Lemma 3.8.**  $\text{stb}(G) = 1$  if and only if  $\text{stb}(G_\rho^+) = \rho$ .

*Proof.* First, assume that  $\text{stb}(G) = 1$ . Then, there is a tree-decomposition  $T$  for  $G$  with strong breadth 1. We construct a tree-decomposition  $T_\rho^+$  for  $G_\rho^+$  with strong breadth  $\rho$ . Make the middle, left, and right vertex of each extended  $C_5$  center of a bag. For each



**Figure 3.4.** Illustration for the graph  $G_\rho^+$ . The graph shown is a subgraph of  $G_\rho^+$  as constructed for each  $v_i$  in  $G$ .

$v_i \in V$ , if  $v_i$  is center of a bag of  $T$ , make  $x_i$  a center of a bag of  $T_\rho^+$ . Otherwise, make  $y_i$  center of a bag of  $T_\rho^+$ . The distance in  $G_\rho^+$  from  $v_i$  to  $x_i$  is  $\rho - 1$ . The distances from  $v_i$  to  $y_i$ , from  $x_i$  to  $z_i$ , and from  $y_i$  to  $z_i$  are  $\rho$ . Thus,  $N_{G_\rho^+}^\rho[x_i] \cap V = N_G[v_i]$ ,  $N_{G_\rho^+}^\rho[y_i] \cap V = \{v_i\}$ , and there is no conflict with Lemma 3.6. Therefore, the constructed  $T_\rho^+$  is a valid tree-decomposition with strong breadth  $\rho$  for  $G_\rho^+$ .

Next, assume that  $\text{stb}(G_\rho^+) = \rho$  and there is a tree-decomposition  $T_\rho^+$  with strong breadth  $\rho$  for  $G_\rho^+$ . By Lemma 3.6, no vertex in distance less than  $\rho$  to any  $z_i$  can be a center of a bag in  $T_\rho^+$ . Therefore, because the distance from  $v_i$  to  $z_i$  in  $G_\rho^+$  is  $\rho - 1$ , no  $v_i \in V$  can be a center of a bag in  $T_\rho^+$ . The only vertices with a large enough distance to  $z_i$  to be a center of a bag are  $x_i$  and  $y_i$ . Therefore, either  $x_i$  or  $y_i$  is selected as center. To construct a tree-decomposition  $T$  with strong breadth 1 for  $G$ , select  $v_i$  as center if and only if  $x_i$  is a center of a bag in  $T_\rho^+$ . Because  $N_{G_\rho^+}^\rho[x_i] \cap V = N_G[v_i]$  and  $N_{G_\rho^+}^\rho[y_i] \cap V = \{v_i\}$ , the constructed  $T$  is a valid tree-decomposition with strong breadth 1 for  $G$ .  $\square$

Constructing  $G_\rho'$  can be done in  $\mathcal{O}(n^2)$  time and constructing  $G_\rho^+$  can be done in  $\mathcal{O}(\rho \cdot n + m)$  time. Thus, combining Lemma 3.7 and Lemma 3.8 allows us, for a given graph  $G$ , some given  $\rho$ , and some given  $\rho'$ , to construct a graph  $H$  in  $\mathcal{O}(\rho \cdot n^2)$  time such that  $\text{stb}(G) \leq \rho$  if and only if  $\text{stb}(H) \leq \rho'$ . Additionally, by combining Theorem 3.7 and Lemma 3.4, we get:

**Theorem 3.8.** *It is NP-complete to decide, for a graph  $G$  and a given  $\rho$ , if  $\text{stb}(G) = \rho$ .*

### 3.2.2 Polynomial Time Cases

In the previous section, we have shown that, in general, it is NP-complete to determine the strong tree-breadth of a graph. In this section, we investigate cases for which a decomposition can be found in polynomial time.

Let  $G$  be a graph with strong tree-breadth  $\rho$  and let  $T$  be a corresponding tree-decomposition. For a given vertex  $u$  in  $G$ , we denote the set of connected components in

$G - N_G^\rho[u]$  as  $\mathcal{C}_G[u]$ . We say that a vertex  $v$  is a *potential partner* of  $u$  for some  $C \in \mathcal{C}_G[u]$  if  $N_G^\rho[v] \supseteq N_G(C)$  and  $N_G^\rho[v] \cap C \neq \emptyset$ .

**Lemma 3.9.** *Let  $C$  be a connected component in  $G - B_u$  for some  $B_u \subseteq N_G^\rho[u]$ . Also, let  $C \in \mathcal{C}_G[u]$  and  $v$  be a potential partner of  $u$  for  $C$ . Then, for all connected components  $C_v$  in  $G[C] - N_G^\rho[v]$ ,  $C_v \in \mathcal{C}_G[v]$ .*

*Proof.* Consider a connected component  $C_v$  in  $G[C] - N_G^\rho[v]$ . Clearly,  $C_v \subseteq C$  and there is a connected component  $C' \in \mathcal{C}_G[v]$  such that  $C' \supseteq C_v$ .

Let  $x$  be an arbitrary vertex in  $C'$ . Then, there is a path  $P \subseteq C'$  from  $x$  to  $C_v$ . Because  $N_G(C) \subseteq B_u$  and  $v$  is a potential partner of  $u$  for  $C$ ,  $N_G(C) \subseteq B_u \cap N_G^\rho[v]$ . Also,  $N_G(C)$  separates all vertices in  $C$  from all other vertices in  $G$ . Therefore,  $x \in C$  and  $C' \subseteq C$ ; otherwise,  $P$  would intersect  $N_G^\rho[v]$ . It follows that each vertex in  $P$  is in the same connected component of  $G[C] - N_G^\rho[v]$  and, thus,  $C_v = C'$ .  $\square$

From Lemma 2.3 (page 6), it directly follows:

**Corollary 3.3.** *If  $N_G^\rho[u]$  is a bag in  $T$ , then  $T$  contains a bag  $N_G^\rho[v]$  for each  $C \in \mathcal{C}_G[u]$  such that  $v$  is a potential partner of  $u$  for  $C$ .*

Because of Corollary 3.3, there is a vertex set  $U$  such that each  $u \in U$  has a potential partner  $v \in U$  for each connected component  $C \in \mathcal{C}_G[u]$ . With such a set, we can construct a tree-decomposition for  $G$  with the following approach: Pick a vertex  $u \in U$  and make it center of a bag  $B_u$ . For each connected component  $C \in \mathcal{C}_G[u]$ ,  $u$  has a potential partner  $v$ .  $N_G^\rho[v]$  splits  $C$  in more connected components and, because  $v \in U$ ,  $v$  has a potential partner  $w \in U$  for each of these components. Hence, create a bag  $B_v = N_G^\rho[v] \cap (B_u \cup C)$  and continue this until the whole graph is covered. Algorithm 3.3 below determines such a set of vertices with their potential partners (represented as a graph  $H$ ) and then constructs a decomposition as described above.

**Theorem 3.9.** *Algorithm 3.3 constructs, for a given graph  $G$  with strong tree-breadth  $\rho$ , a tree-decomposition  $T$  with breadth  $\rho$  in  $\mathcal{O}(n^2m)$  time.*

*Proof (Correctness).* Algorithm 3.3 works in two parts. First, it creates a graph  $H$  with potential centers (line 1 to line 6). Second, it uses  $H$  to create a tree-decomposition for  $G$  (line 9 to line 15). To show the correctness of the algorithm, we show first that the centers of a tree-decomposition for  $G$  are vertices in  $H$  and, then, show that a tree-decomposition created based on  $H$  is a valid tree-decomposition for  $G$ .

A vertex  $u$  is added to  $H$  (line 4) if, for *at least one* connected component  $C \in \mathcal{C}_G[u]$ ,  $u$  has a potential partner  $v$ . Later,  $u$  is kept in  $H$  (line 5 and line 6) if it has a potential partner  $v$  for *all* connected components in  $C \in \mathcal{C}_G[u]$ . By Corollary 3.3, each center of a bag in a tree-decomposition  $T$  with strong breadth  $\rho$  satisfies these conditions. Therefore, after line 6,  $H$  contains all centers of bags in  $T$ , i. e., if  $G$  has strong tree-breadth  $\rho$ ,  $H$  is non-empty.

---

**Algorithm 3.3:** Constructs, for a given graph  $G = (V, E)$  with strong tree-breadth  $\rho$ , a tree-decomposition  $T$  with breadth  $\rho$ .

---

- 1 Create an empty directed graph  $H = (V_H, E_H)$ . Let  $\phi$  be a function that maps each edge  $(u, v) \in E_H$  to a connected component  $C \in \mathcal{C}_G[u]$ .
  - 2 **For Each**  $u, v \in V$  and all  $C \in \mathcal{C}_G[u]$
  - 3     **If**  $v$  is a potential partner of  $u$  for  $C$  **Then**
  - 4     |     Add the directed edge  $(u, v)$  to  $H$  and set  $\phi(u, v) := C$ . (Add  $u$  and  $v$  to  $H$  if necessary.)
  - 5 **While** there is a vertex  $u \in V_H$  and some  $C \in \mathcal{C}_G[u]$  such that there is no  $(u, v) \in E_H$  with  $\phi(u, v) = C$
  - 6     | Remove  $u$  from  $H$ .
  - 7 **If**  $H$  is empty **Then**
  - 8     | **Stop.**  $\text{stb}(G) > \rho$ .
  - 9 Create an empty tree-decomposition  $T$ .
  - 10 Let  $G - T$  be the subgraph of  $G$  that is not covered by  $T$  and let  $\psi$  be a function that maps each connected component in  $G - T$  to a bag  $B_u \subseteq N_G^\rho[u]$ .
  - 11 Pick an arbitrary vertex  $u \in V_H$ , add  $B_u = N_G^\rho[u]$  as bag to  $T$ , and set  $\psi(C) := B_u$  for each connected component  $C$  in  $G - T$ .
  - 12 **While**  $G - T$  is non-empty
  - 13     | Pick a connected component  $C$  in  $G - T$ , determine the bag  $B_v := \psi(C)$  and find an edge  $(v, w) \in E_H$  with  $\phi(v, w) = C$ .
  - 14     | Add  $B_w = N_G^\rho[w] \cap (B_v \cup C)$  to  $T$ , and make  $B_v$  and  $B_w$  adjacent in  $T$ .
  - 15     | For each new connected component  $C'$  in  $G - T$  with  $C' \subseteq C$ , set  $\psi(C_w) := B_w$ .
  - 16 Output  $T$ .
- 

Next, we show that  $T$  created in the second part of the algorithm (line 9 to line 15) is a valid tree-decomposition for  $G$  with breadth  $\rho$ . To do so, we show the following invariant for the loop starting in line 12: (i)  $T$  is a valid tree-decomposition with breadth  $\rho$  for the subgraph covered by  $T$  and (ii) for each connected component  $C$  in  $G - T$ , the bag  $B_v = \psi(C)$  is in  $T$ ,  $N_G(C) \subseteq B_v$ , and  $C \in \mathcal{C}_G[v]$ . After line 11, the invariant clearly holds. Assume by induction that the invariant holds each time line 12 is checked. If  $T$  covers the whole graph, the check fails and the algorithm outputs  $T$ . If  $T$  does not cover  $G$  completely, there is a connected component  $C$  in  $G - T$ . By condition (ii), the bag  $B_v = \psi(C)$  is in  $T$ ,  $N_G(C) \subseteq B_v$ , and  $C \in \mathcal{C}_G[v]$ . Because of the way  $H$  is constructed and  $C \in \mathcal{C}_G[v]$ , there is an edge  $(v, w) \in E_H$  with  $\phi(v, w) = C$ , i. e.,  $w$  is a potential partner of  $v$  for  $C$ . Thus, line 13 is successful and the algorithm adds a new bag  $B_w = N_G^\rho[w] \cap (B_v \cup C)$  (line 14). Because  $w$  is a potential partner of  $v$  for  $C$ , i. e.,  $N_G(C) \subseteq N_G^\rho[w]$ , and  $N_G(C) \subseteq B_v$ ,  $B_w \supseteq N_G(C)$ . Therefore, after adding  $B_w$

to  $T$ ,  $T$  still satisfies condition (i). Additionally,  $B_w$  splits  $C$  in a set  $\mathcal{C}'$  of connected components such that, for each  $C' \in \mathcal{C}'$ ,  $N_G(C') \subseteq B_w$  and, by Lemma 3.9,  $C' \in \mathcal{C}_G[w]$ . Thus, condition (ii) is also satisfied.  $\square$

*Proof (Complexity).* First, determine the pairwise distance of all vertices. This can be done in  $\mathcal{O}(nm)$  time and allows to check the distance between vertices in constant time.

For a vertex  $u$ , let  $\mathcal{N}[u] = \{N_G(C) \mid C \in \mathcal{C}_G[u]\}$ . Note that, for some  $C \in \mathcal{C}_G[u]$  and each vertex  $x \in N_G(C)$ , there is an edge  $xy$  with  $y \in C$ . Therefore,  $|\mathcal{N}[u]| := \sum_{C \in \mathcal{C}_G[u]} |N_G(C)| \leq m$ . To determine, for some vertex  $u$ , all its potential partners  $v$ , first, compute  $\mathcal{N}[u]$ . This can be done in  $\mathcal{O}(m)$  time. Then, check, for each vertex  $v$  and each  $N_G(C) \in \mathcal{N}[u]$ , if  $N_G(C) \subseteq N_G^\rho[v]$  and add the edge  $(u, v)$  to  $H$  if successful. For a single vertex  $v$  this requires  $\mathcal{O}(m)$  time because  $|\mathcal{N}[u]| \leq m$  and distances can be determined in constant time. Therefore, the total runtime for creating  $H$  (line 1 to line 4) is  $\mathcal{O}(n(m + nm)) = \mathcal{O}(n^2m)$ .

Assume that, for each  $\phi(u, v) = C$ ,  $C$  is represented buy two values: (i) a characteristic vertex  $x \in C$  (for example the vertex with the lowest index) and (ii) the index of  $C$  in  $\mathcal{C}_G[u]$ . While creating  $H$ , count and store, for each vertex  $u$  and each connected component  $C \in \mathcal{C}_G[u]$ , the number of edges  $(u, v) \in E_H$  with  $\phi(u, v) = C$ . Note that there is a different counter for each  $C \in \mathcal{C}_G[u]$ . With this information, we can implement line 5 and line 6 as follows. First check, for every vertex  $v$  in  $H$ , if one of its counters is 0. In this case, remove  $v$  from  $H$  and update the counters for all vertices  $u$  with  $(u, v) \in E_H$  using value (ii) of  $\phi(u, v)$ . If this sets a counter for  $u$  to 0, add  $u$  to a queue  $Q$  of vertices to process. Continue this until each vertex is checked. Then, for each vertex  $u$  in  $Q$ , remove  $u$  form  $H$  and add its neighbours into  $Q$  if necessary until  $Q$  is empty. This way, a vertex is processed at most twice. A single iteration runs in at most  $\mathcal{O}(n)$  time. Therefore, line 5 and line 6 can be implemented in  $\mathcal{O}(n^2)$  time.

Assume that  $\psi$  uses the characteristic vertex  $x$  to represent a connected component, i. e., value (i) of  $\phi$ . Then, finding an edge  $(v, w) \in E_H$  (line 13) can be done in  $\mathcal{O}(m)$  time. Creating  $B_w$  (line 14), splitting  $C$  into new connected components  $C'$ , finding their characteristic vertex, and setting  $\psi(C')$  (line 15) takes  $\mathcal{O}(m)$  time, too. In each iteration, at least one more vertex of  $G$  is covered by  $T$ . Hence, there are at most  $n$  iterations and, thus, the loop starting in line 12 runs in  $\mathcal{O}(mn)$  time.

Therefore, Algorithm 3.3 runs in total  $\mathcal{O}(n^2m)$  time.  $\square$

Algorithm 3.3 creates, for each graph  $G$  with  $\text{stb}(G) \leq \rho$ , a tree-decomposition  $T$  with breadth  $\rho$ . Next, we invest a case where we can construct a tree-decomposition for  $G$  with strong breadth  $\rho$ .

We say that two vertices  $u$  and  $v$  are *perfect partners* if (i)  $u$  and  $v$  are potential partners of each other for some  $C_u \in \mathcal{C}_G[u]$  and some  $C_v \in \mathcal{C}_G[v]$ , (ii)  $C_u$  is the only connected component in  $\mathcal{C}_G[u]$  which is intersected by  $N_G^\rho[v]$ , and (iii)  $C_v$  is the only connected component in  $\mathcal{C}_G[v]$  which is intersected by  $N_G^\rho[u]$ . Accordingly, we say that a tree-decomposition  $T$  has *perfect strong breadth*  $\rho$  if it has strong breadth  $\rho$  and, for each

center  $u$  of some bag and each connected component  $C \in \mathcal{C}_G[u]$ , there is a center  $v$  such that  $v$  is a perfect partner of  $u$  for  $C$ .

**Theorem 3.10.** *A tree-decomposition with perfect strong breadth  $\rho$  can be constructed in polynomial time.*

*Proof.* To construct such a tree-decomposition, we can modify Algorithm 3.3. Instead of checking if  $u$  has a potential partner  $v$  (line 3), check if  $u$  and  $v$  are perfect partners.

Assume by induction that, for each bag  $B_v$  in  $T$ ,  $B_v = N_G^\rho[v]$ . By definition of perfect partners  $v$  and  $w$ ,  $N_G^\rho[w]$  intersects only one  $C \in \mathcal{C}_G[v]$ , i. e.,  $N_G^\rho[w] \subseteq N_G^\rho[v] \cup C$ . Thus, when creating the bag  $B_w$  (line 14),  $B_w = N_G^\rho[w] \cap (B_v \cup C) = N_G^\rho[w] \cap (N_G^\rho[v] \cup C) = N_G^\rho[w]$ . Therefore, the created tree-decomposition  $T$  has perfect strong tree-breadth  $\rho$ .  $\square$

We conjecture that there are weaker cases than perfect strong breadth which allow to construct a tree-decomposition with strong-breadth  $\rho$ . For example, if the centers of two adjacent bags are perfect partners, but a center  $u$  does not need to have a perfect partner for each  $C \in \mathcal{C}_G[u]$ . However, when using a similar approach as in Algorithm 3.3, this would require a more complex way of constructing  $H$ .

### 3.3 Computing Decompositions for Special Graph Classes

In this section, we show how to find good decompositions for some special graph classes.

**Theorem 3.11.** *Chordal graphs have strong tree-breadth 1. An according decomposition can be computed in linear time.*

*Proof.* Let  $\sigma = \langle v_1, v_2, \dots, v_n \rangle$  be an ordering for the vertices of a graph  $G$ ,  $V_i = \{v_1, v_2, \dots, v_i\}$ ,  $G_i$  denote the graph  $G[V_i]$ ,  $N_i[v] = N_G[v] \cap V_i$ , and  $N_i(v) = N_G(v) \cap V_i$ . The reverse of such an ordering  $\sigma$  is called a *perfect elimination ordering* for  $G$  if, for each  $i$ ,  $N_i[v_i]$  induces a clique. It is well known that a graph is chordal if and only if it admits a perfect elimination ordering [30].

Assume that we are given such an ordering  $\sigma$ , i. e., the reverse of a perfect elimination ordering. Additionally, assume by induction over  $i$  that  $G_{i-1}$  admits a tree-decomposition  $T_{i-1}$  with strong breadth 1 such that centers of bags are pairwise non-adjacent. This is clearly the case for  $G_1$ . We now show how to construct  $T_i$  from  $T_{i-1}$ .

First, consider the case that  $v_i$  has a neighbour  $u$  in  $G_i$  which is center of some bag  $B$  in  $T_{i-1}$ . Because  $u \in N_i[v_i]$  and  $N_i[v_i]$  induces a clique,  $N_i[v_i] \subseteq N_i[u]$  and, hence,  $N_i[v_i]$  does not contain a center of any bag. Thus, adding  $v_i$  into  $B$  creates a valid tree-decomposition  $T_i$  for  $G_i$  with strong breadth 1 and pairwise non-adjacent centers.

Next, consider the case that  $v_i$  has no neighbour  $u$  in  $G_i$  that is center of some bag in  $T_{i-1}$ . Note that  $N_i(v_i)$  induces a clique in  $G_{i-1}$ . It is well known that, for each tree-decomposition  $T$  and for each clique  $K$  of graph,  $T$  contains a bag  $B$  with  $K \subseteq B$ . Thus, there is a bag  $B$  in  $T_{i-1}$  with  $N_i(v_i) \subseteq B$ . Therefore, we can create  $T_i$  by adding

the bag  $B' = N_i[v_i]$  to  $T_{i-1}$  and making  $B'$  adjacent to  $B$ . This creates a valid tree-decomposition with strong breadth 1 and pairwise non-adjacent centers.

Based on the approach described above, we can construct a tree-decomposition with strong breadth 1 for a given chordal graph  $G$  as follows. First, compute the reverse of a perfect elimination ordering of  $G$ . Such an ordering can be computed in linear time [81]. Observe that we can simplify the approach above with the following rule: If  $v_i$  has no neighbour in  $G_i$  which is center of a bag, make  $v_i$  center of a bag. Otherwise, proceed with  $v_{i+1}$ . Therefore, by using a simple binary flag for each vertex, one can compute a tree-decomposition with strong breadth 1 for a given chordal graph  $G$  in linear time.  $\square$

**Theorem 3.12.** *Distance-hereditary graphs have strong tree-breadth 1. An according decomposition can be computed in linear time.*

*Proof.* Recall that a graph is distance-hereditary if and only if it admits a pruning sequence  $\sigma$  (see Lemma 2.9, page 8). That is, a sequence  $\sigma = \langle v_1, v_2, \dots, v_n \rangle$  such that, for each vertex  $v_i$  and some  $j < i$ ,  $v_i$  is a pendant vertex,  $v_i$  is a true twin of  $v_j$ , or  $v_i$  is a false twin of  $v_j$ .

Assume that we are given such a pruning sequence. Additionally, assume by induction over  $i$  that  $G_i$  has a tree-decomposition  $T_i$  with strong breadth 1 where the centers of bags are pairwise non-adjacent. Then, there are three cases:

- (i)  $v_{i+1}$  is a pendant vertex in  $G_{i+1}$ . If the neighbour  $u$  of  $v_{i+1}$  is a center of a bag  $B_u$ , add  $v_{i+1}$  to  $B_u$ . Thus,  $T_{i+1}$  is a valid decomposition for  $G_{i+1}$ . Otherwise, if  $u$  is not a center, make  $v_{i+1}$  center of a bag. Because  $u$  is an articulation point,  $T_{i+1} = T_i + N_G[v]$  is a valid decomposition for  $G_{i+1}$ .
- (ii)  $v_{i+1}$  is a true twin of a vertex  $u$  in  $G_{i+1}$ . Simply add  $v_{i+1}$  into any bag containing  $u$ . The resulting decomposition is a valid decomposition for  $G_{i+1}$ .
- (iii)  $v_{i+1}$  is a false twin of a vertex  $u$  in  $G_{i+1}$ . If  $u$  is not center of a bag, add  $v_{i+1}$  into any bag  $u$  is in. Otherwise, make a new bag  $B_{i+1} = N_G[v_{i+1}]$  and make it adjacent to the bag  $N_G[u]$ . Because no vertex in  $N_G(u)$  is center of a bag, the resulting decomposition is a valid decomposition for  $G_{i+1}$ .

Therefore, distance-hereditary graphs have strong tree-breadth 1.

Next, we show how to compute an according tree-decomposition in linear time. The argument above already gives an algorithmic approach. First, we compute a pruning sequence for  $G$ . This can be done in linear time with an algorithm by DAMIAND et al. [27]. Then, we determine which vertex becomes a center of a bag. Note that we can simplify the three cases above with the following rule: If  $v_i$  has no neighbour in  $G_i$  which is center of a bag, make  $v_i$  center of a bag. Otherwise, proceed with  $v_{i+1}$ . This can be easily implemented in linear time with a binary flag for each vertex.  $\square$

Algorithm 3.4 formalizes the method described in the proof of Theorem 3.12.

**Theorem 3.13.** *If  $G$  is an AT-free graph, then  $\text{pl}(G) \leq 2$ . Furthermore, a path-decomposition of  $G$  with length at most 2 can be computed in  $\mathcal{O}(n^2)$  time.*

---

**Algorithm 3.4:** Computes, for a given distance-hereditary graph  $G$ , a tree-decomposition  $T$  with strong breadth 1.

---

- 1 Compute a pruning sequence  $\langle v_1, v_2, \dots, v_n \rangle$  (see [27]).
  - 2 Create a set  $C := \emptyset$ .
  - 3 **For**  $i := 1$  **To**  $n$
  - 4     **If**  $N_G[v_i] \cap V_i \cap C = \emptyset$  **Then**
  - 5         Add  $v_i$  to  $C$ .
  - 6 Create a tree-decomposition  $T$  with the vertices in  $C$  as centers of its bags.
- 

*Proof.* Let  $s$  be an arbitrary vertex of  $G$ , let  $x$  be the vertex last visited (numbered 1) by a LexBFS starting at  $s$ , and let  $\sigma$  be the ordering obtained by a LexBFS starting at  $x$ . Clearly,  $\sigma$  can be generated in linear time. Note that the LexBFS which computes  $\sigma$  also computes all distance layers  $L_i^{(x)}$  of  $G$  with  $0 \leq i \leq \text{ecc}(x)$ . For some vertex  $v \in L_i^{(x)}$ , let  $N_G^\downarrow(v) = N_G(v) \cap L_{i-1}^{(x)}$ .

We can transform an AT-free graph  $G = (V, E)$  into an interval graph  $G^+ = (V, E^+)$  by applying the following two operations:

- (1) *Make layers complete graphs.* In each layer  $L_i^{(x)}$ , make every two vertices  $u, v \in L_i^{(x)}$  adjacent to each other in  $G^+$ .
- (2) *Make down-neighbourhoods of adjacent vertices of a layer comparable.* For each  $i$  and every edge  $uv$  of  $G$  with  $u, v \in L_i^{(x)}$  and  $\sigma(v) < \sigma(u)$ , make every  $w \in N_G^\downarrow(v)$  adjacent to  $u$  in  $G^+$ .

*Claim 1.*  $G^+$  is a subgraph of  $G^2$ .

*Proof (Claim).* Clearly, for every edge  $uv$  of  $G^+$  added by operation (2),  $d_G(u, w) \leq 2$  holds. Also, for every edge  $uv$  of  $G^+$  added by operation (1),  $d_G(u, v) \leq 2$  holds by Lemma 2.12 (page 9).  $\diamond$

*Claim 2.*  $G^+$  is an interval graph.

*Proof (Claim).* It is known [77] that a graph is an interval graph if and only if its vertices admit an *interval ordering*. That is, an ordering  $\tau: V \rightarrow \{1, \dots, n\}$  such that, for any three vertices  $a, b$ , and  $c$  with  $\tau(a) < \tau(b) < \tau(c)$ ,  $ac \in E$  implies  $bc \in E$ . We show here that the LexBFS-ordering  $\sigma$  of  $G$  is an interval ordering of  $G^+$ . Recall that, for each  $v \in L_i^{(x)}$  and every  $u \in L_j^{(x)}$  with  $i > j$ , it holds that  $\sigma(v) < \sigma(u)$  since  $\sigma$  is a LexBFS-ordering. Consider three arbitrary vertices  $a, b$ , and  $c$  of  $G$  and assume that  $\sigma(a) < \sigma(b) < \sigma(c)$  and  $ac \in E^+$ . Assume also that  $a \in L_i^{(x)}$  for some  $i$ . If  $c$  belongs to  $L_i^{(x)}$ , then  $b$  must be in  $L_i^{(x)}$  as well. Hence,  $bc \in E^+$  due to operation (1). If both  $b$  and  $c$  are in  $L_{i-1}^{(x)}$ , then again  $bc \in E^+$  due to operation (1). Consider now the remaining case that  $a, b \in L_i^{(x)}$  and  $c \in L_{i-1}^{(x)}$ . If  $ac \in E$ , then  $bc \in E^+$  because either  $ab \in E$  and, thus, operation (2) applies, or  $ab \notin E$  and, thus, Lemma 2.12 (page 9) implies  $bc \in E^+$ .



If  $ac \in E^+ \setminus E$  then, according to operation (2), edge  $ac$  was created in  $G^+$  because some vertex  $a' \in L_i^{(x)}$  existed such that  $\sigma(a') < \sigma(a)$  and  $a'a, a'c \in E$ . Since  $a'c \in E$  and  $\sigma(a') < \sigma(b) < \sigma(c)$ , as before,  $bc \in E^+$  must hold.  $\diamond$

To complete the proof, we recall that a graph is an interval graph if and only if it has a path-decomposition with each bag being a maximal clique (see Theorem 2.2, page 8). Furthermore, such a path-decomposition of an interval graph can easily be computed in linear time. Let  $\mathcal{P}^+ = \{X_1, \dots, X_q\}$  be a path-decomposition of our interval graph  $G^+$ . Then,  $\mathcal{P} := \mathcal{P}^+ = \{X_1, \dots, X_q\}$  is a path-decomposition of  $G$  with length at most 2 since, for every edge  $uv$  of  $G^+$ , the distance in  $G$  between  $u$  and  $v$  is at most 2, as shown in Claim 1.  $\square$

Algorithm 3.5 formalizes the steps described in the previous proof.

---

**Algorithm 3.5:** Computes a path-decomposition of length at most 2 for a given AT-free graph.

---

**Input:** An AT-free graph  $G = (V, E)$ .

**Output:** A path-decomposition of  $G$ .

- 1 Calculate a LexBFS-ordering  $\sigma$  of  $G$  with an arbitrary start vertex  $s \in V$ . Let  $x$  be the last visited vertex, i. e.,  $\sigma(x) = 1$ .
  - 2 Calculate a LexBFS-ordering  $\sigma'$  of  $G$  starting at  $x$ .
  - 3 Set  $E^+ := E$ .
  - 4 **For Each** vertex pair  $u, v$  with  $d_G(x, u) = d_G(x, v)$  and  $\sigma'(u) < \sigma'(v)$
  - 5     Add  $uv$  to  $E^+$ .
  - 6     For each  $w \in N_G(u)$  with  $\sigma'(v) < \sigma'(w)$ , add  $vw$  to  $E^+$ .
  - 7 Calculate a path-decomposition  $\mathcal{P}$  of the interval graph  $G^+ = (V, E^+)$  by determining the maximal cliques of  $G^+$ .
  - 8 Output  $\mathcal{P}$ .
- 

Because the class of cocomparability graphs is a proper subclass of AT-free graphs, we obtain the following corollary.

**Corollary 3.4.** *If  $G$  is a cocomparability graph, then  $\text{pl}(G) \leq 2$ . Furthermore, a path-decomposition of  $G$  with length at most 2 can be computed in  $\mathcal{O}(n^2)$  time.*

Note that the complement of an induced cycle on six vertices has path-breadth 2 (see [39] for details). Thus, the bound 2 on the path-breadth of cocomparability graphs (and therefore, of AT-free graphs) is sharp.

Consider two parallel lines (upper and lower) in the plane. Assume that each line contains  $n$  points, labelled 1 to  $n$ . Each two points with the same label define a segment with that label. The intersection graph of such a set of segments between two parallel lines is called a *permutation graph*.

Assume now that each of the two parallel lines contains  $n$  intervals, labelled 1 to  $n$ , and each two intervals with the same label define a trapezoid with that label (a trapezoid can degenerate to a triangle or to a segment). The intersection graph of such a set of trapezoids between two parallel lines is called a *trapezoid graph*. Clearly, every permutation graph is a trapezoid graph, but not vice versa.

**Theorem 3.14.** *If  $G$  is a permutation graph, then  $\text{spb}(G) = 1$ . Furthermore, a path-decomposition of  $G$  with optimal breadth can be computed in linear time.*

*Proof.* We assume that a permutation model of  $G$  is given in advance (if not, we can compute one for  $G$  in linear time [74]). That is, each vertex  $v$  of  $G$  is associated with a segment  $s(v)$  such that  $uv \in E$  if and only if segments  $s(v)$  and  $s(u)$  intersect. In what follows, “u. p.” and “l. p.” refer to a vertex’s point on the upper and lower, respectively, line of the permutation model.

First, we compute an (inclusion) maximal independent set  $M$  of  $G$  in linear time as follows. Put in  $M$  (which is initially empty) a vertex  $x_1$  whose u. p. is leftmost. For each  $i \geq 2$ , select a vertex  $x_i$  whose u. p. is leftmost among all vertices whose segments do not intersect  $s(x_1), \dots, s(x_{i-1})$ . In fact, it is enough to check intersection with  $s(x_{i-1})$  only. If such a vertex exists, put it in  $M$  and continue. If no such vertex exists,  $M = \{x_1, \dots, x_k\}$  has been constructed.

Now, we claim that  $\{N_G[x_1], \dots, N_G[x_k]\}$  is a path-decomposition of  $G$  with strong breadth 1 and, hence, with length at most 2. Clearly, each vertex of  $G$  is in some bag since every vertex not in  $M$  is adjacent to a vertex in  $M$ , by the maximality of  $M$ . Consider an arbitrary edge  $uv$  of  $G$ . Assume that neither  $u$  nor  $v$  is in  $M$  and that the u. p. of  $u$  is to the left of the u. p. of  $v$ . Necessarily, the l. p. of  $v$  is to the left of the l. p. of  $u$ , since the segments  $s(v)$  and  $s(u)$  intersect. Assume that the u. p. of  $u$  is between the u. p.s of  $x_i$  and  $x_{i+1}$ . From the construction of  $M$ ,  $s(u)$  and  $s(x_i)$  must intersect, i. e., the l. p. of  $u$  is to the left of the l. p. of  $x_i$ . But then, since the l. p. of  $v$  is to the left of the l. p. of  $x_i$ , segments  $s(v)$  and  $s(x_i)$  must intersect, too. Thus, edge  $uv$  is in the bag  $N_G[x_i]$ .

To show that all bags containing any particular vertex form a contiguous subsequence of the sequence  $\langle N_G[x_1], \dots, N_G[x_k] \rangle$ , consider an arbitrary vertex  $v$  of  $G$  and let  $v \in N_G[x_i] \cap N_G[x_j]$  for  $i < j$ . Consider an arbitrary bag  $N_G[x_l]$  with  $i < l < j$ . We know that the vertices  $x_i, x_l, x_j \in M$  are pairwise non-adjacent. Furthermore, the segment  $s(v)$  intersects the segments  $s(x_i)$  and  $s(x_j)$ . As segment  $s(x_l)$  is between  $s(x_i)$  and  $s(x_j)$ , necessarily,  $s(v)$  intersects  $s(x_l)$  as well.  $\square$

**Theorem 3.15.** *If  $G$  is a trapezoid graph, then  $\text{pb}(G) = 1$ . Furthermore, a path-decomposition of  $G$  with breadth 1 can be computed in  $\mathcal{O}(n^2)$  time.*

*Proof.* We show that every trapezoid graph  $G$  is a minor of a permutation graph.

First, we compute in  $\mathcal{O}(n^2)$  time a trapezoid model for  $G$  [73]. Then, we replace each trapezoid  $\mathcal{T}_i$  in this model with its two diagonals obtaining a permutation model with  $2n$  vertices. Let  $H$  be the permutation graph of this permutation model. It is easy to see

that two trapezoids  $\mathcal{T}_1$  and  $\mathcal{T}_2$  intersect if and only if a diagonal of  $\mathcal{T}_1$  and a diagonal of  $\mathcal{T}_2$  intersect.

Now,  $G$  can be obtained back from  $H$  by a series of  $n$  edge contractions. For each trapezoid  $\mathcal{T}_i$ , contract the edge of  $H$  that corresponds to two diagonals of  $\mathcal{T}_i$ .

Since contracting edges does not increase the path-breadth (Lemma 2.3, page 6), we get  $\text{pb}(G) = \text{pb}(H) = 1$  by Theorem 3.14. Any path-decomposition of  $H$  with breadth 1 is a path-decomposition of  $G$  with breadth 1.  $\square$

A bipartite graph is *chordal bipartite* if each cycle of length at least 6 has a chord. To the best of our knowledge, there is no linear time algorithm known to recognise chordal bipartite graphs. However, in [43], DRAGAN and LOMONOSOV show that any chordal bipartite graph  $G = (X, Y, E)$  admits a tree-decomposition with the set of bags  $\mathcal{B} = \{B_1, B_2, \dots, B_{|X|}\}$ , where  $B_i = N_G[x_i]$ ,  $x_i \in X$ . Therefore, we can still compute a tree-decomposition in linear time with three steps. First, compute a 2-colouring. Second, select a colour and make the neighbourhood of all vertices with this colour bags. Third, use the algorithm in [88] to check if the selected bags give a valid tree-decomposition. Thus, it follows:

**Theorem 3.16 (Dragan and Lomonosov [43]).** *Each chordal bipartite graph has strong tree-breadth 1. An according tree-decomposition can be found in linear time.*

# Connected $r$ -Domination

The Domination problem is a classical problem in computer science. It is a variant of the Set Covering problem, one of KARP's 21 NP-complete problems [65]. A generalisation of it is the  $r$ -Domination problem. Assume that we are given a graph  $G = (V, E)$  and a function  $r: V \rightarrow \mathbb{N}$ . Then, we say a vertex set  $D$  is an  $r$ -dominating set for  $G$  if, for each vertex  $u \in V$ ,  $d(u, D) \leq r(u)$ . The  $r$ -Domination problem asks for the smallest  $r$ -dominating set  $D$ .

Clearly, the problem is NP-complete in general. Although it can be solved for dually chordal graphs in linear time [14], it remains NP-complete for chordal graphs [13]. Even more, under reasonable assumptions, the  $r$ -Domination problem cannot be approximated within a factor of  $(1 - \varepsilon) \ln n$  in polynomial time for chordal graphs [22] and, thus, for graphs with bounded tree-breadth.

In this chapter, we use an approach presented by CHEPOI and ESTELLON [21]. Assume that  $D_r$  is a minimum  $r$ -dominating set for some graph  $G$ . Instead of finding an  $r$ -dominating set which is slightly larger than  $D_r$ , we investigate how to determine a set  $D$  such that, for some factor  $\phi \in \mathcal{O}(\text{tb}(G))$ ,  $D$  is an  $(r + \phi)$ -dominating set for  $G$  with  $|D| \leq |D_r|$ . That is, for each vertex  $v$  of  $G$ ,  $d(v, D) \leq r(v) + \phi$ .

CHEPOI and ESTELLON [21] present a polynomial time algorithm to compute an  $(r + 2\delta)$ -dominating set for  $\delta$ -hyperbolic graphs. Due to Theorem 2.4 (page 2.4), their algorithm gives an  $(r + 2\lambda)$ -dominating set for graphs with tree-length  $\lambda$ . We slightly improve their result by constructing an  $(r + \rho)$ -dominating set for a graph  $G$  under the assumption that a tree-decomposition for  $G$  with breadth  $\rho$  is given. Additionally, we present algorithms to compute connected  $(r + \phi)$ -dominating sets for different values of  $\phi$  and with different runtimes.

For this chapter, let  $|T|$  denote the number of nodes and  $\Lambda(T)$  denote the number of leaves of a tree  $T$ . If  $T$  contains only one node, let  $\Lambda(T) := 0$ . With  $\alpha$ , we denote the inverse ACKERMANN function (see, e. g., [23]).

## 4.1 Using a Layering Partition

For the remainder of this section, assume that we are given a graph  $G = (V, E)$  and a layering partition  $\mathcal{T}$  of  $G$  for an arbitrary start vertex. We denote the largest diameter of all clusters of  $\mathcal{T}$  as  $\Delta$ , i. e.,  $\Delta := \max \{ d_G(x, y) \mid x, y \text{ are in a cluster } C \text{ of } \mathcal{T} \}$ .

Theorem 4.1 below shows that we can use the layering partition  $\mathcal{T}$  to compute an  $(r + \Delta)$ -dominating set for  $G$  in linear time which is not larger than a minimum  $r$ -

dominating set for  $G$ . This is done by finding a minimum  $r$ -dominating set of  $\mathcal{T}$  where, for each cluster  $C$  of  $\mathcal{T}$ ,  $r(C)$  is defined as  $\min_{v \in C} r(v)$ .

**Theorem 4.1.** *Let  $D$  be a minimum  $r$ -dominating set for a given graph  $G$ . An  $(r + \Delta)$ -dominating set  $D'$  for  $G$  with  $|D'| \leq |D|$  can be computed in linear time.*

*Proof.* First, create a layering partition  $\mathcal{T}$  of  $G$  and, for each cluster  $C$  of  $\mathcal{T}$ , set  $r(C) := \min_{v \in C} r(v)$ . Second, find a minimum  $r$ -dominating set  $\mathcal{S}$  for  $\mathcal{T}$ , i. e., a set  $\mathcal{S}$  of clusters such that, for each cluster  $C$  of  $\mathcal{T}$ ,  $d_{\mathcal{T}}(C, \mathcal{S}) \leq r(C)$ . Third, create a set  $D'$  by picking an arbitrary vertex of  $G$  from each cluster in  $\mathcal{S}$ . All three steps can be performed in linear time, including the computation of  $\mathcal{S}$  (see [14]).

Next, we show that  $D'$  is an  $(r + \Delta)$ -dominating set for  $G$ . By construction of  $\mathcal{S}$ , each cluster  $C$  of  $\mathcal{T}$  has distance at most  $r(C)$  to  $\mathcal{S}$  in  $\mathcal{T}$ . Thus, for each vertex  $u$  of  $G$ ,  $\mathcal{S}$  contains a cluster  $C_{\mathcal{S}}$  with  $d_{\mathcal{T}}(u, C_{\mathcal{S}}) \leq r(u)$ . Additionally, by Lemma 2.6 (page 7),  $d_G(u, v) \leq r(u) + \Delta$  for any vertex  $v \in C_{\mathcal{S}}$ . Therefore, for any vertex  $u$ ,  $d_G(u, D') \leq r(u) + \Delta$ , i. e.,  $D'$  is an  $(r + \Delta)$ -dominating set for  $G$ .

It remains to show that  $|D'| \leq |D|$ . Let  $\mathcal{D}$  be the set of clusters of  $\mathcal{T}$  that contain a vertex of  $D$ . Because  $D$  is an  $r$ -dominating set for  $G$ , it follows from Lemma 2.6 (page 7) that  $\mathcal{D}$  is an  $r$ -dominating set for  $\mathcal{T}$ . Clearly, since clusters are pairwise disjoint,  $|\mathcal{D}| \leq |D|$ . By minimality of  $\mathcal{S}$ ,  $|\mathcal{S}| \leq |\mathcal{D}|$  and, by construction of  $D'$ ,  $|D'| = |\mathcal{S}|$ . Therefore,  $|D'| \leq |D|$ .  $\square$

We now show how to construct a connected  $(r + 2\Delta)$ -dominating set for  $G$  using  $\mathcal{T}$  in such a way that the set created is not larger than a minimum connected  $r$ -dominating set for  $G$ . For the remainder of this section, let  $D_r$  be a minimum connected  $r$ -dominating set of  $G$  and let, for each cluster  $C$  of  $\mathcal{T}$ ,  $r(C)$  be defined as above. Additionally, we say that a subtree  $T'$  of some tree  $T$  is an  $r$ -dominating subtree of  $T$  if the nodes (clusters in case of a layering partition) of  $T'$  form a connected  $r$ -dominating set for  $T$ .

The first step of our approach is to construct a minimum  $r$ -dominating subtree  $T_r$  of  $\mathcal{T}$ . Such a subtree  $T_r$  can be computed in linear time [35]. Lemma 4.1 below shows that  $T_r$  gives a lower bound for the cardinality of  $D_r$ .

**Lemma 4.1.** *If  $T_r$  contains more than one cluster, each connected  $r$ -dominating set of  $G$  intersects all clusters of  $T_r$ . Therefore,  $|T_r| \leq |D_r|$ .*

*Proof.* Let  $D$  be an arbitrary connected  $r$ -dominating set of  $G$ . Assume that  $T_r$  has a cluster  $C$  such that  $C \cap D = \emptyset$ . Because  $D$  is connected, the subtree of  $\mathcal{T}$  induced by the clusters intersecting  $D$  is connected, too. Thus, if  $D$  intersects all leaves of  $T_r$ , then it intersects all clusters of  $T_r$ . Hence, we can assume, without loss of generality, that  $C$  is a leaf of  $T_r$ . Because  $T_r$  has at least two clusters and by minimality of  $T_r$ ,  $\mathcal{T}$  contains a cluster  $C'$  such that  $d_{\mathcal{T}}(C', C) = d_{\mathcal{T}}(C', T_r) = r(C')$ . Note that each path in  $G$  from a vertex in  $C'$  to a vertex in  $D$  intersects  $C$ . Therefore, by Lemma 2.6 (page 7), there is

a vertex  $u \in C'$  with  $r(u) = d_{\mathcal{T}}(u, C) < d_{\mathcal{T}}(u, D) \leq d_G(u, D)$ . That contradicts with  $D$  being an  $r$ -dominating set.

Because any  $r$ -dominating set of  $G$  intersects each cluster of  $T_r$  and because these clusters are pairwise disjoint, it follows that  $|T_r| \leq |D_r|$ .  $\square$

As we show later in Corollary 4.1, each connected vertex set  $S \subseteq V$  that intersects each cluster of  $T_r$  gives an  $(r + \Delta)$ -dominating set for  $G$ . It follows from Lemma 4.1 that, if such a set  $S$  has minimum cardinality,  $|S| \leq |D_r|$ . However, finding a minimum cardinality connected set intersecting each cluster of a layering partition (or of a subtree of it) is as hard as finding a minimum Steiner tree.

The main idea of our approach is to construct a minimum  $(r + \delta)$ -dominating subtree  $T_\delta$  of  $\mathcal{T}$  for some integer  $\delta$ . We then compute a small enough connected set  $S_\delta$  that intersects all cluster of  $T_\delta$ . By trying different values of  $\delta$ , we eventually construct a connected set  $S_\delta$  such that  $|S_\delta| \leq |T_r|$  and, thus,  $|S_\delta| \leq |D_r|$ . Additionally, we show that  $S_\delta$  is a connected  $(r + 2\Delta)$ -dominating set of  $G$ .

For some non-negative integer  $\delta$ , let  $T_\delta$  be a minimum  $(r + \delta)$ -dominating subtree of  $\mathcal{T}$ . Clearly,  $T_0 = T_r$ . The following two lemmas set an upper bound for the maximum distance of a vertex of  $G$  to a vertex in a cluster of  $T_\delta$  and for the size of  $T_\delta$  compared to the size of  $T_r$ .

**Lemma 4.2.** *For each vertex  $v$  of  $G$ ,  $d_{\mathcal{T}}(v, T_\delta) \leq r(v) + \delta$ .*

*Proof.* Let  $C_v$  be the cluster of  $\mathcal{T}$  containing  $v$  and let  $C$  be the cluster of  $T_\delta$  closest to  $C_v$  in  $\mathcal{T}$ . By construction of  $T_\delta$ ,  $d_{\mathcal{T}}(v, C) = d_{\mathcal{T}}(C_v, C) \leq r(C_v) + \delta \leq r(v) + \delta$ .  $\square$

Because the diameter of each cluster is at most  $\Delta$ , Lemma 2.6 (page 7) and Lemma 4.2 imply the following.

**Corollary 4.1.** *If a vertex set intersects all clusters of  $T_\delta$ , it is an  $(r + (\delta + \Delta))$ -dominating set of  $G$ .*

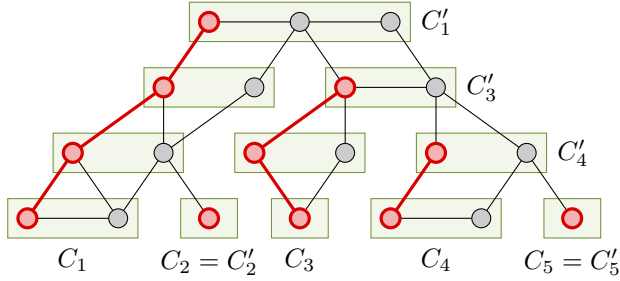
**Lemma 4.3.**  $|T_\delta| \leq |T_r| - \delta \cdot \Lambda(T_\delta)$ .

*Proof.* First, consider the case when  $T_\delta$  contains only one cluster, i. e.,  $|T_\delta| = 1$ . Then,  $\Lambda(T_\delta) = 0$  and, thus, the statement clearly holds. Next, let  $T_\delta$  contain more than one cluster, let  $C_u$  be an arbitrary leaf of  $T_\delta$ , and let  $C_v$  be a cluster of  $T_r$  with maximum distance to  $C_u$  such that  $C_u$  is the only cluster on the shortest path from  $C_u$  to  $C_v$  in  $T_r$ , i. e.,  $C_v$  is not in  $T_\delta$ . Due to the minimality of  $T_\delta$ ,  $d_{T_r}(C_u, C_v) = \delta$ . Thus, the shortest path from  $C_u$  to  $C_v$  in  $T_r$  contains  $\delta$  clusters (including  $C_v$ ) which are not in  $T_\delta$ . Therefore,  $|T_\delta| \leq |T_r| - \delta \cdot \Lambda(T_\delta)$ .  $\square$

Now that we have constructed and analysed  $T_\delta$ , we show how to construct  $S_\delta$ . First, we construct a set of shortest paths such that each cluster of  $T_\delta$  is intersected by exactly

one path. Second, we connect these paths with each other to form a connected set using an approach which is similar to KRUSKAL's algorithm for minimum spanning trees.

Let  $\mathcal{L} = \{C_1, C_2, \dots, C_\lambda\}$  be the leaf clusters of  $T_\delta$  (excluding the root) with either  $\lambda = \Lambda(T_\delta) - 1$  if the root of  $T_\delta$  is a leaf, or with  $\lambda = \Lambda(T_\delta)$  otherwise. We construct a set  $\mathcal{P} = \{P_1, P_2, \dots, P_\lambda\}$  of paths as follows. Initially,  $\mathcal{P}$  is empty. For each cluster  $C_i \in \mathcal{L}$ , in turn, find the ancestor  $C'_i$  of  $C_i$  which is closest to the root of  $T_\delta$  and does not intersect any path in  $\mathcal{P}$  yet. If we assume that the indices of the clusters in  $\mathcal{L}$  represent the order in which they are processed, then  $C'_1$  is the root of  $T_\delta$ . Then, select an arbitrary vertex  $v$  in  $C_i$  and find a shortest path  $P_i$  in  $G$  from  $v$  to  $C'_i$ . Add  $P_i$  to  $\mathcal{P}$  and continue with the next cluster in  $\mathcal{L}$ . Figure 4.1 gives an example.



**Figure 4.1.** Example for the set  $\mathcal{P}$  for a subtree of a layering partition. Paths are shown in red. Each path  $P_i$ , with  $1 \leq i \leq 5$ , starts in the leaf  $C_i$  and ends in the cluster  $C'_i$ . For  $i = 2$  and  $i = 5$ ,  $P_i$  contains only one vertex.

**Lemma 4.4.** *For each cluster  $C$  of  $T_\delta$ , there is exactly one path  $P_i \in \mathcal{P}$  intersecting  $C$ . Additionally,  $C$  and  $P_i$  share exactly one vertex, i. e.,  $|C \cap P_i| = 1$ .*

*Proof.* Observe that, by construction of a layering partition, each vertex in a cluster  $C$  is adjacent to some vertex in the parent cluster of  $C$ . Therefore, a shortest path  $P$  in  $G$  from  $C$  to any of its ancestors  $C'$  only intersects clusters on the path from  $C$  to  $C'$  in  $\mathcal{T}$  and each cluster shares only one vertex with  $P$ . It remains to show that each cluster intersects exactly one path.

Without loss of generality, assume that the indices of clusters in  $\mathcal{L}$  and paths in  $\mathcal{P}$  represent the order in which they are processed and created, i. e., assume that the algorithm first creates  $P_1$  which starts in  $C_1$ , then  $P_2$  which starts in  $C_2$ , and so on. Additionally, let  $\mathcal{L}_i = \{C_1, C_2, \dots, C_i\}$  and  $\mathcal{P}_i = \{P_1, P_2, \dots, P_i\}$ .

To prove that each cluster intersects exactly one path, we show by induction over  $i$  that, if a cluster  $C_i$  of  $T_\delta$  satisfies the statement, then all ancestors of  $C_i$  satisfy it, too. Thus, if  $C_\lambda$  satisfies the statement, each cluster satisfies it.

First, consider  $i = 1$ . Clearly, since  $P_1$  is the first path,  $P_1$  connects the leaf  $C_1$  with the root of  $T_\delta$  and no cluster intersects more than one path at this point. Therefore, the statement is true for  $C_1$  and each of its ancestors.

Next, assume that  $i > 1$  and that the statement is true for each cluster in  $\mathcal{L}_{i-1}$  and their respective ancestors. Then, the algorithm creates  $P_i$  which connects the leaf  $C_i$

with the cluster  $C'_i$ . Assume that there is a cluster  $C$  on the path from  $C_i$  to  $C'_i$  in  $\mathcal{T}$  such that  $C$  intersects a path  $P_j$  with  $j < i$ . Clearly,  $C'_i$  is an ancestor of  $C$ . Thus, by induction hypothesis,  $C'_i$  is also intersected by some path  $P \neq P_i$ . This contradicts with the way  $C'_i$  is selected by the algorithm. Therefore, each cluster on the path from  $C_i$  to  $C'_i$  in  $\mathcal{T}$  only intersects  $P_i$  and  $P_i$  does not intersect any other clusters.

Because  $i > 1$ ,  $C'_i$  has a parent cluster  $C''$  in  $T_\delta$  that is intersected by a path  $P_j$  with  $j < i$ . By induction hypothesis, each ancestor of  $C''$  is intersected by a path in  $\mathcal{P}_{i-1}$ . Therefore, each ancestor of  $C_i$  is intersected by exactly one path in  $\mathcal{P}_i$ .  $\square$

Next, we use the paths in  $\mathcal{P}$  to create the set  $S_\delta$ . As first step, let  $S_\delta := \bigcup_{P_i \in \mathcal{P}} P_i$ . Later, we add more vertices into  $S_\delta$  to ensure it is a connected set.

Now, create a partition  $\mathcal{V} = \{V_1, V_2, \dots, V_\lambda\}$  of  $V$  such that, for each  $i$ ,  $P_i \subseteq V_i$ ,  $V_i$  is connected, and  $d_G(v, P_i) = \min_{P \in \mathcal{P}} d_G(v, P)$  for each vertex  $v \in V_i$ . That is,  $V_i$  contains the vertices of  $G$  which are not more distant to  $P_i$  in  $G$  than to any other path in  $\mathcal{P}$ . Additionally, for each vertex  $v \in V$ , set  $P(v) := P_i$  if and only if  $v \in V_i$  (i. e.,  $P(v)$  is the path in  $\mathcal{P}$  which is closest to  $v$ ) and set  $d(v) := d_G(v, P(v))$ . Such a partition as well as  $P(v)$  and  $d(v)$  can be computed by performing a BFS on  $G$  starting at all paths  $P_i \in \mathcal{P}$  simultaneously. Later, the BFS also allows us to easily determine the shortest path from  $v$  to  $P(v)$  for each vertex  $v$ .

To manage the subsets of  $\mathcal{V}$ , we use a Union-Find data structure such that, for two vertices  $u$  and  $v$ ,  $\text{Find}(u) = \text{Find}(v)$  if and only if  $u$  and  $v$  are in the same set of  $\mathcal{V}$ . A Union-Find data structure additionally allows us to easily join two set of  $\mathcal{V}$  into one by performing a single Union operation. Note that, whenever we join two sets of  $\mathcal{V}$  into one,  $P(v)$  and  $d(v)$  remain unchanged for each vertex  $v$ .

Next, create an edge set  $E' = \{uv \mid \text{Find}(u) \neq \text{Find}(v)\}$ , i. e., the set of edges  $uv$  such that  $u$  and  $v$  are in different sets of  $\mathcal{V}$ . Sort  $E'$  in such a way that an edge  $uv$  precedes an edge  $xy$  only if  $d(u) + d(v) \leq d(x) + d(y)$ .

The last step to create  $S_\delta$  is similar to KRUSKAL's minimum spanning tree algorithm. Iterate over the edges in  $E'$  in increasing order. If, for an edge  $uv$ ,  $\text{Find}(u) \neq \text{Find}(v)$ , i. e., if  $u$  and  $v$  are in different sets of  $\mathcal{V}$ , then join these sets into one by performing  $\text{Union}(u, v)$ , add the vertices on the shortest path from  $u$  to  $P(u)$  to  $S_\delta$ , and add the vertices on the shortest path from  $v$  to  $P(v)$  to  $S_\delta$ . Repeat this, until  $\mathcal{V}$  contains only one set, i. e., until  $\mathcal{V} = \{V\}$ .

Algorithm 4.1 below summarises the steps to create a set  $S_\delta$  for a given subtree of a layering partition subtree  $T_\delta$ .

**Lemma 4.5.** *For a given graph  $G$  and a given subtree  $T_\delta$  of some layering partition of  $G$ , Algorithm 4.1 constructs, in  $\mathcal{O}(m \alpha(n))$  time, a connected set  $S_\delta$  with  $|S_\delta| \leq |T_\delta| + \Delta \cdot \Lambda(T_\delta)$  which intersects each cluster of  $T_\delta$ .*

*Proof (Correctness).* First, we show that  $S_\delta$  is connected at the end of the algorithm. To do so, we show by induction that, at any time,  $S_\delta \cap V'$  is a connected set for each



---

**Algorithm 4.1:** Computes a connected vertex set that intersects each cluster of a given layering partition.

---

**Input:** A graph  $G = (V, E)$  and a subtree  $T_\delta$  of some layering partition of  $G$ .

**Output:** A connected set  $S_\delta \subseteq V$  that intersects each cluster of  $T_\delta$  and contains at most  $|T_\delta| + (\Lambda(T_\delta) - 1) \cdot \Delta$  vertices.

- 1 Let  $\mathcal{L} = \{C_1, C_2, \dots, C_\lambda\}$  be the set of clusters excluding the root that are leaves of  $T_\delta$ .
  - 2 Create an empty set  $\mathcal{P}$ .
  - 3 **For Each** cluster  $C_i \in \mathcal{L}$
  - 4     Select an arbitrary vertex  $v \in C_i$ .
  - 5     Find the highest ancestor  $C'_i$  of  $C_i$  (i. e., the ancestor which is closest to the root of  $T_\delta$ ) that is not flagged.
  - 6     Find a shortest path  $P_i$  from  $v$  to an ancestor of  $v$  in  $C'_i$  (i. e., a shortest path from  $C_i$  to  $C'_i$  in  $G$  that contains exactly one vertex of each cluster of the corresponding path in  $T_\delta$ ).
  - 7     Add  $P_i$  to  $\mathcal{P}$ .
  - 8     Flag each cluster intersected by  $P_i$ .
  - 9 Create a set  $S_\delta := \bigcup_{P_i \in \mathcal{P}} P_i$ .
  - 10 Perform a BFS on  $G$  starting at all paths  $P_i \in \mathcal{P}$  simultaneously. This results in a partition  $\mathcal{V} = \{V_1, V_2, \dots, V_\lambda\}$  of  $V$  with  $P_i \subseteq V_i$  for each  $P_i \in \mathcal{P}$ . For each vertex  $v$ , set  $P(v) := P_i$  if and only if  $v \in V_i$  and let  $d(v) := d_G(v, P(v))$ .
  - 11 Create a Union-Find data structure and add all vertices of  $G$  such that  $\text{Find}(v) = i$  if and only if  $v \in V_i$ .
  - 12 Determine the edge set  $E' = \{uv \mid \text{Find}(u) \neq \text{Find}(v)\}$ .
  - 13 Sort  $E'$  such that  $uv \leq xy$  if and only if  $d(u) + d(v) \leq d(x) + d(y)$ . Let  $\langle e_1, e_2, \dots, e_{|E'|} \rangle$  be the resulting sequence.
  - 14 **For**  $i := 1$  **To**  $|E'|$
  - 15     Let  $uv = e_i$ .
  - 16     **If**  $\text{Find}(u) \neq \text{Find}(v)$  **Then**
  - 17         Add the shortest path from  $u$  to  $P(u)$  to  $S_\delta$ .
  - 18         Add the shortest path from  $v$  to  $P(v)$  to  $S_\delta$ .
  - 19         Union( $u, v$ )
  - 20 Output  $S_\delta$ .
- 

set  $V' \in \mathcal{V}$ . Clearly, when  $\mathcal{V}$  is created, for each set  $V_i \in \mathcal{V}$ ,  $S_\delta \cap V_i = P_i$ . Now, assume that the algorithm joins the set  $V_u$  and  $V_v$  in  $\mathcal{V}$  into one set based on the edge  $uv$  with  $u \in V_u$  and  $v \in V_v$ . Let  $S_u = S_\delta \cap V_u$  and  $S_v = S_\delta \cap V_v$ . Note that  $P(u) \subseteq S_u$  and  $P(v) \subseteq S_v$ . The algorithm now adds all vertices to  $S_\delta$  which are on a path from  $P(u)$  to  $P(v)$ . Therefore,  $S_\delta \cap (V_u \cup V_v)$  is a connected set. Because  $\mathcal{V} = \{V\}$  at the end of the algorithm,  $S_\delta$  is connected eventually. Additionally, since  $P_i \subseteq S_\delta$  for each  $P_i \in \mathcal{P}$ , it

follows that  $S_\delta$  intersects each cluster of  $T_\delta$ .

Next, we show that the cardinality of  $S_\delta$  is at most  $|T_\delta| + \Delta \cdot \Lambda(T_\delta)$ . When first created, the set  $S_\delta$  contains all vertices of all paths in  $\mathcal{P}$ . Therefore, by Lemma 4.4,  $|S_\delta| = \sum_{P_i \in \mathcal{P}} |P_i| = |T_\delta|$ . Then, each time two sets of  $\mathcal{V}$  are joined into one set based on an edge  $uv$ ,  $S_\delta$  is extended by the vertices on the shortest paths from  $u$  to  $P(u)$  and from  $v$  to  $P(v)$ . Therefore, the size of  $S_\delta$  increases by  $d(u) + d(v)$ , i. e.,  $|S_\delta| := |S_\delta| + d(u) + d(v)$ . Let  $X$  denote the set of all edges used to join two sets of  $\mathcal{V}$  into one at some point during the algorithm. Note that  $|X| = |\mathcal{P}| - 1 \leq \Lambda(T_\delta)$ . Therefore, at the end of the algorithm,

$$|S_\delta| = \sum_{P_i \in \mathcal{P}} |P_i| + \sum_{uv \in X} (d(u) + d(v)) \leq |T_\delta| + \Lambda(T_\delta) \cdot \max_{uv \in X} (d(u) + d(v)).$$

*Claim 1.* For each edge  $uv \in X$ ,  $d(u) + d(v) \leq \Delta$ .

*Proof (Claim).* To represent the relations between paths in  $\mathcal{P}$  and vertex sets in  $\mathcal{V}$ , we define a function  $f: \mathcal{P} \rightarrow \mathcal{V}$  such that  $f(P_i) = V_j$  if and only if  $P_i \subseteq V_j$ . Directly after constructing  $\mathcal{V}$ ,  $f$  is a bijection with  $f(P_i) = V_i$ . At the end of the algorithm, after all sets of  $\mathcal{V}$  are joined into one,  $f(P_i) = V$  for all  $P_i \in \mathcal{P}$ .

Recall the construction of  $\mathcal{P}$  and assume that the indices of the paths in  $\mathcal{P}$  represent the order in which they are created. Assume that  $i > 1$ . By construction, the path  $P_i \in \mathcal{P}$  connects the leaf  $C_i$  with the cluster  $C'_i$  in  $T_\delta$ . Because  $i > 1$ ,  $C'_i$  has a parent cluster in  $T_\delta$  that is intersected by a path  $P_j \in \mathcal{P}$  with  $j < i$ . We define  $P_j$  as the *parent* of  $P_i$ . By Lemma 4.4, this parent  $P_j$  is unique for each  $P_i \in \mathcal{P}$  with  $i > 1$ . Based on this relation between paths in  $\mathcal{P}$ , we can construct a rooted tree  $\mathbb{T}$  with the node set  $\{x_i \mid P_i \in \mathcal{P}\}$  such that each node  $x_i$  represents the path  $P_i$  and  $x_j$  is the parent of  $x_i$  if and only if  $P_j$  is the parent of  $P_i$ .

Because each node of  $\mathbb{T}$  represents a path in  $\mathcal{P}$ ,  $f$  defines a colouring for the nodes of  $\mathbb{T}$  such that  $x_i$  and  $x_j$  have different colours if and only if  $f(P_i) \neq f(P_j)$ . As long as  $|\mathcal{V}| > 1$ ,  $\mathbb{T}$  contains two adjacent nodes with different colours. Let  $x_i$  and  $x_j$  be these nodes with  $j < i$  and let  $P_i$  and  $P_j$  be the corresponding paths in  $\mathcal{P}$ . Note that  $x_j$  is the parent of  $x_i$  in  $\mathbb{T}$  and, hence,  $P_j$  is the parent of  $P_i$ . Therefore,  $P_i$  ends in a cluster  $C'_i$  which has a parent cluster  $C$  that intersects  $P_j$ . By properties of layering partitions, it follows that  $d_G(P_i, P_j) \leq \Delta + 1$ . Recall that, by construction,  $d(v) = \min_{P \in \mathcal{P}} d_G(v, P)$  for each vertex  $v$ . Thus, for each edge  $uv$  on a shortest path from  $P_i$  to  $P_j$  in  $G$  (with  $u$  being closer to  $P_i$  than to  $P_j$ ),  $d(u) + d(v) \leq d_G(u, P_i) + d_G(v, P_j) \leq \Delta$ . Therefore, because  $f(P_i) \neq f(P_j)$ , there is an edge  $uv$  on a shortest path from  $P_i$  to  $P_j$  such that  $f(P(u)) \neq f(P(v))$  and  $d(u) + d(v) \leq \Delta$ .  $\diamond$

From the claim above, it follows that, as long as  $\mathcal{V}$  contains multiple sets, there is an edge  $uv \in E'$  such that  $d(u) + d(v) \leq \Delta$  and  $\text{Find}(u) \neq \text{Find}(v)$ . Therefore,  $\max_{uv \in X} (d(u) + d(v)) \leq \Delta$  and, hence,  $|S_\delta| \leq |T_\delta| + \Delta \cdot \Lambda(T_\delta)$ .  $\square$

*Proof (Complexity).* First, the algorithm computes  $\mathcal{P}$  (line 2 to line 8). If the parent of each vertex from the original BFS that was used to construct  $\mathcal{T}$  is still known,  $\mathcal{P}$  can be

constructed in  $\mathcal{O}(n)$  total time. After picking a vertex  $v$  in  $C_i$ , simply follow the parent pointers until a vertex in  $C'_i$  is reached. Computing  $\mathcal{V}$  as well as  $P(v)$  and  $d(v)$  for each vertex  $v$  of  $G$  (line 10) can be done with single BFS and, thus, requires at most  $\mathcal{O}(n + m)$  time.

Recall that, for a Union-Find data structure storing  $n$  elements, each operation requires at most  $\mathcal{O}(\alpha(n))$  amortised time. Therefore, initialising such a data structure to store all vertices (line 11) and computing  $E'$  (line 12) requires at most  $\mathcal{O}(m\alpha(n))$  time. Note that, for each vertex  $v$ ,  $d(v) \leq |V|$ . Thus, sorting  $E'$  (line 13) can be done in linear time using counting sort. When iterating over  $E'$  (line 14 to line 19), for each edge  $uv \in E'$ , the Find-operation is called twice and the Union-operation is called at most once. Thus, the total runtime for all these operations is at most  $\mathcal{O}(m\alpha(n))$ .

Let  $P_u = \{u, \dots, x, y, \dots, p\}$  be the shortest path in  $G$  from a vertex  $u$  to  $P(u)$ . Assume that  $y$  has been added to  $S_\delta$  in a previous iteration. Thus,  $\{y, \dots, p\} \subseteq S_\delta$  and, when adding  $P_u$  to  $S_\delta$ , the algorithm only needs to add  $\{u, \dots, x\}$ . Therefore, by using a simple binary flag to determine if a vertex is contained in  $S_\delta$ , constructing  $S_\delta$  (line 9, line 17, and line 18) requires at most  $\mathcal{O}(n)$  time.

In total, Algorithm 4.1 runs in  $\mathcal{O}(m\alpha(n))$  time.  $\square$

Because, for each integer  $\delta \geq 0$ ,  $|S_\delta| \leq |T_\delta| + \Delta \cdot \Lambda(T_\delta)$  (Lemma 4.5) and  $|T_\delta| \leq |T_r| - \delta \cdot \Lambda(T_\delta)$  (Lemma 4.3), we have the following.

**Corollary 4.2.** *For each  $\delta \geq \Delta$ ,  $|S_\delta| \leq |T_r|$  and, thus,  $|S_\delta| \leq |D_r|$ .*

To the best of our knowledge, there is no algorithm known that computes  $\Delta$  in less than  $\mathcal{O}(nm)$  time. Additionally, under reasonable assumptions, computing the diameter or radius of a general graph requires  $\Omega(n^2)$  time [1]. We conjecture that the runtime for computing  $\Delta$  for a given graph has a similar lower bound.

To avoid the runtime required for computing  $\Delta$ , we use the following approach shown in Algorithm 4.2 below. First, compute a layering partition  $\mathcal{T}$  and the subtree  $T_r$ . Second, for a certain value of  $\delta$ , compute  $T_\delta$  and perform Algorithm 4.1 on it. If the resulting set  $S_\delta$  is larger than  $T_r$  (i. e.,  $|S_\delta| > |T_r|$ ), increase  $\delta$ ; otherwise, if  $|S_\delta| \leq |T_r|$ , decrease  $\delta$ . Repeat the second step with the new value of  $\delta$ .

One strategy to select values for  $\delta$  is a classical binary search over the number of vertices of  $G$ . In this case, Algorithm 4.1 is called up-to  $\mathcal{O}(\log n)$  times. Empirical analysis [2], however, have shown that  $\Delta$  is usually very small. Therefore, we use a one-sided binary search instead.

Because of Corollary 4.2, using a one-sided binary search allows us to find a value  $\delta \leq \Delta$  for which  $|S_\delta| \leq |T_r|$  by calling Algorithm 4.1 at most  $\mathcal{O}(\log \Delta)$  times. Algorithm 4.2 below implements this approach.

**Theorem 4.2.** *For a given graph  $G$ , Algorithm 4.2 computes a connected  $(r + 2\Delta)$ -dominating set  $D$  with  $|D| \leq |D_r|$  in  $\mathcal{O}(m\alpha(n)\log \Delta)$  time.*

---

**Algorithm 4.2:** Computes a connected  $(r+2\Delta)$ -dominating set for a given graph  $G$ .

---

**Input:** A graph  $G = (V, E)$  and a function  $r: V \rightarrow \mathbb{N}$ .

**Output:** A connected  $(r+2\Delta)$ -dominating set  $D$  for  $G$  with  $|D| \leq |D_r|$ .

- 1 Create a layering partition  $\mathcal{T}$  of  $G$ .
  - 2 For each cluster  $C$  of  $\mathcal{T}$ , set  $r(C) := \min_{v \in C} r(v)$ .
  - 3 Compute a minimum  $r$ -dominating subtree  $T_r$  for  $\mathcal{T}$  (see [35]).
  - 4 **One-Sided Binary Search** over  $\delta$ , starting with  $\delta = 0$
  - 5     Create a minimum  $\delta$ -dominating subtree  $T_\delta$  of  $T_r$  (i. e.,  $T_\delta$  is a minimum  $(r + \delta)$ -dominating subtree for  $\mathcal{T}$ ).
  - 6     Run Algorithm 4.1 on  $T_\delta$  and let the set  $S_\delta$  be the corresponding output.
  - 7     **if**  $|S_\delta| \leq |T_r|$  **Then**
  - 8         Decrease  $\delta$ .
  - 9     **Else**
  - 10         Increase  $\delta$ .
  - 11 Output  $S_\delta$  with the smallest  $\delta$  for which  $|S_\delta| \leq |T_r|$ .
- 

*Proof.* Clearly, the set  $D$  is connected because  $D = S_\delta$  for some  $\delta$  and, by Lemma 4.5, the set  $S_\delta$  is connected. By Corollary 4.2, for each  $\delta \geq \Delta$ ,  $|S_\delta| \leq |T_r|$ . Thus, for each  $\delta \geq \Delta$ , the binary search decreases  $\delta$  and, eventually, finds some  $\delta$  such that  $\delta \leq \Delta$  and  $|S_\delta| \leq |T_r|$ . Therefore, the algorithm finds a set  $D$  with  $|D| \leq |D_r|$ . Note that, because  $D = S_\delta$  for some  $\delta \leq \Delta$  and because  $S_\delta$  intersects each cluster of  $\mathcal{T}$  (Lemma 4.5), it follows from Lemma 4.2 that, for each vertex  $v$  of  $G$ ,  $d_{\mathcal{T}}(v, D) \leq r(v) + \Delta$  and, by Lemma 2.6,  $d_G(v, D) \leq r(v) + 2\Delta$ . Thus,  $D$  is an  $(r + 2\Delta)$ -dominating set for  $G$ .

Creating a layering partition for a given graph and computing a minimum connected  $r$ -dominating set of a tree can be done in linear time [35]. The one-sided binary search over  $\delta$  has at most  $\mathcal{O}(\log \Delta)$  iterations. Each iteration of the binary search requires at most linear time to compute  $T_\delta$ ,  $\mathcal{O}(m \alpha(n))$  time to compute  $S_\delta$  (Lemma 4.5), and constant time to decide whether to increase or decrease  $\delta$ . Therefore, Algorithm 4.2 runs in  $\mathcal{O}(m \alpha(n) \log \Delta)$  total time.  $\square$

## 4.2 Using a Tree-Decomposition

Theorem 4.1 and Theorem 4.2 respectively show how to compute an  $(r + \Delta)$ -dominating set in linear time and a connected  $(r + 2\Delta)$ -dominating set in  $\mathcal{O}(m \alpha(n) \log \Delta)$  time. It is known that the maximum diameter  $\Delta$  of clusters of any layering partition of a graph approximates the tree-breadth and tree-length of this graph. Indeed, as shown in Lemma 3.1 (page 11), for a graph  $G$  with  $\text{tl}(G) = \lambda$ ,  $\Delta \leq 3\lambda$ .

**Corollary 4.3.** *Let  $D$  be a minimum  $r$ -dominating set for a given graph  $G$  with  $\text{tl}(G) = \lambda$ . An  $(r + 3\lambda)$ -dominating set  $D'$  for  $G$  with  $|D'| \leq |D|$  can be computed in linear time.*

---

**Corollary 4.4.** *Let  $D$  be a minimum connected  $r$ -dominating set for a given graph  $G$  with  $\text{tl}(G) = \lambda$ . A connected  $(r + 6\lambda)$ -dominating set  $D'$  for  $G$  with  $|D'| \leq |D|$  can be computed in  $\mathcal{O}(m \alpha(n) \log \lambda)$  time.*

In this section, we consider the case when we are given a tree-decomposition with breadth  $\rho$  and length  $\lambda$ . We present algorithms to compute an  $(r + \rho)$ -dominating set as well as a connected  $(r + \min(3\lambda, 5\rho))$ -dominating set in  $\mathcal{O}(nm)$  time.

For the remainder of this section, assume that we are given a graph  $G = (V, E)$  and a tree-decomposition  $\mathcal{T}$  of  $G$  with breadth  $\rho$  and length  $\lambda$ . We assume that  $\rho$  and  $\lambda$  are known and that, for each bag  $B$  of  $\mathcal{T}$ , we know a vertex  $c(B)$  with  $B \subseteq N_G^\rho[c(B)]$ . Let  $\mathcal{T}$  be minimal, i. e.,  $B \not\subseteq B'$  for any two bags  $B$  and  $B'$ . Thus, the number of bags is not exceeding the number vertices of  $G$ . Additionally, let each vertex of  $G$  store a list of bags containing it and let each bag of  $\mathcal{T}$  store a list of vertices it contains. One can see this as a bipartite graph where one subset of vertices are the vertices of  $G$  and the other subset are the bags of  $\mathcal{T}$ . Therefore, the total input size is in  $\mathcal{O}(n + m + M)$  where  $M \leq n^2$  is the sum of the cardinality of all bags of  $\mathcal{T}$ .

### 4.2.1 Preprocessing

Before approaching the (Connected)  $r$ -Domination problem, we compute a subtree  $T_r$  of  $\mathcal{T}$  such that, for each vertex  $v$  of  $G$ ,  $T_r$  contains a bag  $B$  with  $d_G(v, B) \leq r(v)$ . We call such a (not necessarily minimal) subtree an  $r$ -covering subtree of  $\mathcal{T}$ .

We do not know how to compute a minimum  $r$ -covering subtree  $T_r$  directly. However, if we are given a bag  $B$  of  $\mathcal{T}$ , we can compute the smallest  $r$ -covering subtree  $T_B$  which contains  $B$ . Then, we can identify a bag  $B'$  in  $T_B$  for which we know it is a bag of  $T_r$ . Thus, we can compute  $T_r$  by computing the smallest  $r$ -covering subtree which contains  $B'$ .

The idea for computing  $T_B$  is to determine, for each vertex  $v$  of  $G$ , the bag  $B_v$  of  $\mathcal{T}$  for which  $d_G(v, B_v) \leq r(v)$  and which is closet to  $B$ . Then, let  $T_B$  be the smallest tree that contains all these bags  $B_v$ . Algorithm 4.3 below implements this approach.

Additionally to computing the tree  $T_B$ , we make it a rooted tree with  $B$  as the root, give each vertex  $v$  a pointer  $\beta(v)$  to a bag of  $T_B$ , and give each bag  $B'$  a counter  $\sigma(B')$ . The pointer  $\beta(v)$  identifies the bag  $B_v$  which is closest to  $B$  in  $T_B$  and intersects the  $r$ -neighbourhood of  $v$ . The counter  $\sigma(B')$  states the number of vertices  $v$  with  $\beta(v) = B'$ . Even though setting  $\beta$  and  $\sigma$  as well as rooting the tree are not necessary for computing  $T_B$ , we use it when computing an  $(r + \rho)$ -dominating set later.

**Lemma 4.6.** *For a given tree-decomposition  $\mathcal{T}$  and a given bag  $B$  of  $\mathcal{T}$ , Algorithm 4.3 computes an  $r$ -covering subtree  $T_B$  in  $\mathcal{O}(nm)$  time such that  $T_B$  contains  $B$  and has a minimal number of bags.*

*Proof (Correctness).* Note that, by construction of the set  $\mathcal{B}$  (line 5 to line 7),  $\mathcal{B}$  contains a bag  $B_u$  for each vertex  $u$  of  $G$  such that  $d_G(u, B_u) \leq r(u)$ . Thus, each subtree of  $\mathcal{T}$  which contains all bags of  $\mathcal{B}$  is an  $r$ -covering subtree. To show the correctness of the

---

**Algorithm 4.3:** Computes the smallest  $r$ -covering subtree  $T_B$  of a given tree-decomposition  $\mathcal{T}$  that contains a given bag  $B$  of  $\mathcal{T}$ .

---

- 1 Make  $\mathcal{T}$  a rooted tree with the bag  $B$  as the root.
  - 2 Create a set  $\mathcal{B}$  of bags and initialise it with  $\mathcal{B} := \{B\}$ .
  - 3 For each bag  $B'$  of  $\mathcal{T}$ , set  $\sigma(B') := 0$  and determine  $d_{\mathcal{T}}(B', B)$ .
  - 4 For each vertex  $u$ , determine the bag  $B(u)$  which contains  $u$  and has minimal distance to  $B$ .
  - 5 **For Each**  $u \in V$
  - 6     Determine a vertex  $v$  such that  $d_G(u, v) \leq r(u)$  and  $d_{\mathcal{T}}(B(v), B)$  is minimal and let  $B_u := B(v)$ .
  - 7     Add  $B_u$  to  $\mathcal{B}$ , set  $\beta(u) := B_u$ , and increase  $\sigma(B_u)$  by 1.
  - 8 Output the smallest subtree  $T_B$  of  $\mathcal{T}$  that contains all bags in  $\mathcal{B}$ .
- 

algorithm, it remains to show that the smallest  $r$ -covering subtree of  $\mathcal{T}$  which contains  $B$  has to contain each bag from the set  $\mathcal{B}$ . Then, the subtree  $T_B$  constructed in line 8 is the desired subtree.

Clearly, by properties of tree-decompositions, the set of bags which intersect the  $r$ -neighbourhood of some vertex  $u$  induces a subtree  $T_u$  of  $\mathcal{T}$ . That is,  $T_u$  contains exactly the bags  $B'$  with  $d_G(u, B') \leq r(u)$ . Note that  $\mathcal{T}$  is a rooted tree with  $B$  as the root. Clearly, the bag  $B_u \in \mathcal{B}$  (determined in line 6) is the root of  $T_u$  since it is the bag closest to  $B$ . Hence, each bag  $B'$  with  $d_G(u, B') \leq r(u)$  is a descendant of  $B_u$ . Therefore, if a subtree of  $\mathcal{T}$  contains  $B$  and does not contain  $B_u$ , then it also cannot contain any descendant of  $B_u$  and, thus, contains no bag intersecting the  $r$ -neighbourhood of  $u$ .  $\square$

*Proof (Complexity).* Recall that  $\mathcal{T}$  has at most  $n$  bags and that the sum of the cardinality of all bags of  $\mathcal{T}$  is  $M \leq n^2$ . Thus, line 3 and line 4 require at most  $\mathcal{O}(M)$  time. Using a BFS, it takes at most  $\mathcal{O}(m)$  time, for a given vertex  $u$ , to determine a vertex  $v$  such that  $d_G(u, v) \leq r(u)$  and  $d_{\mathcal{T}}(B(v), B)$  is minimal (line 6). Therefore, the loop starting in line 5 and, thus, Algorithm 4.3 run in at most  $\mathcal{O}(nm)$  total time.  $\square$

Lemma 4.7 and Lemma 4.8 below show that each leaf  $B' \neq B$  of  $T_B$  is a bag of a minimum  $r$ -covering subtree  $T_r$  of  $\mathcal{T}$ . Note that both lemmas only apply if  $T_B$  has at least two bags. If  $T_B$  contains only one bag, it is clearly a minimum  $r$ -covering subtree.

**Lemma 4.7.** *For each leaf  $B' \neq B$  of  $T_B$ , there is a vertex  $v$  in  $G$  such that  $B'$  is the only bag of  $T_B$  with  $d_G(v, B') \leq r(v)$ .*

*Proof.* Assume that Lemma 4.7 is false. Then, there is a leaf  $B'$  such that, for each vertex  $v$  with  $d_G(v, B') \leq r(v)$ ,  $T_B$  contains a bag  $B'' \neq B'$  with  $d_G(v, B'') \leq r(v)$ . Thus, for each vertex  $v$ , the  $r$ -neighbourhood of  $v$  is intersected by a bag of the tree-decomposition  $T_B - B'$ . This contradicts with the minimality of  $T_B$ .  $\square$

**Lemma 4.8.** *For each leaf  $B' \neq B$  of  $T_B$ , there is a minimum  $r$ -covering subtree  $T_r$  of  $\mathcal{T}$  which contains  $B'$ .*

*Proof.* Assume that  $T_r$  is a minimum  $r$ -covering subtree which does not contain  $B'$ . Because of Lemma 4.7, there is a vertex  $v$  of  $G$  such that  $B'$  is the only bag of  $T_B$  which intersects the  $r$ -neighbourhood of  $v$ . Therefore,  $T_r$  contains only bags which are descendants of  $B'$ . Partition the vertices of  $G$  into the sets  $V^\uparrow$  and  $V^\downarrow$  such that  $V^\downarrow$  contains the vertices of  $G$  which are contained in  $B'$  or in a descendant of  $B'$ . Because  $T_r$  is an  $r$ -covering subtree and because  $T_r$  only contains descendants of  $B'$ , it follows from properties of tree-decompositions that, for each vertex  $v \in V^\uparrow$ , there is a path of length at most  $r(v)$  from  $v$  to a bag of  $T_r$  passing through  $B'$  and, thus,  $d_G(v, B') \leq r(v)$ . Similarly, since  $T_B$  is an  $r$ -covering subtree, it follows that, for each vertex  $v \in V^\downarrow$ ,  $d_G(v, B') \leq r(v)$ . Therefore, for each vertex  $v$  of  $G$ ,  $d_G(v, B') \leq r(v)$  and, thus,  $B'$  induces an  $r$ -covering subtree  $T_r$  of  $\mathcal{T}$  with  $|T_r| = 1$ .  $\square$

Algorithm 4.4 below uses Lemma 4.8 to compute a minimum  $r$ -covering subtree  $T_r$  of  $\mathcal{T}$ .

---

**Algorithm 4.4:** Computes a minimum  $r$ -covering subtree  $T_r$  of a given tree-decomposition  $\mathcal{T}$ .

---

- 1 Pick an arbitrary bag  $B$  of  $\mathcal{T}$ .
  - 2 Determine the subtree  $T_B$  of  $\mathcal{T}$  using Algorithm 4.3.
  - 3 **if**  $|T_B| = 1$  **Then**
  - 4      $\lfloor$  Output  $T_r := T_B$ .
  - 5 **Else**
  - 6      $\lfloor$  Select an arbitrary leaf  $B' \neq B$  of  $T_B$ .
  - 7      $\lfloor$  Determine the subtree  $T_{B'}$  of  $\mathcal{T}$  using Algorithm 4.3.
  - 8      $\lfloor$  Output  $T_r := T_{B'}$ .
- 

**Lemma 4.9.** *Algorithm 4.4 computes a minimum  $r$ -covering subtree  $T_r$  of  $\mathcal{T}$  in  $\mathcal{O}(nm)$  time.*

*Proof.* Algorithm 4.4 first picks an arbitrary bag  $B$  and then uses Algorithm 4.3 to compute the smallest  $r$ -covering subtree  $T_B$  of  $\mathcal{T}$  which contains  $B$ . By Lemma 4.8, for each leaf  $B'$  of  $T_B$ , there is a minimum  $r$ -covering subtree  $T_r$  which contains  $B'$ . Thus, performing Algorithm 4.3 again with  $B'$  as input creates such a subtree  $T_r$ .

Clearly, with exception of calling Algorithm 4.3, all steps of Algorithm 4.4 require only constant time. Because Algorithm 4.3 requires at most  $\mathcal{O}(nm)$  time (see Lemma 4.6) and is called at most two times, Algorithm 4.4 runs in at most  $\mathcal{O}(nm)$  total time.  $\square$

Algorithm 4.4 computes  $T_r$  by, first, computing  $T_B$  for some bag  $B$  and, second, computing  $T_{B'} = T_r$  for some leaf  $B'$  of  $T_B$ . Note that, because both trees are computed

using Algorithm 4.3, Lemma 4.7 applies to  $T_B$  and  $T_{B'}$ . Therefore, we can slightly generalise Lemma 4.7 as follows.

**Corollary 4.5.** *For each leaf  $B$  of  $T_r$ , there is a vertex  $v$  in  $G$  such that  $B$  is the only bag of  $T_r$  with  $d_G(v, B) \leq r(v)$ .*

### 4.2.2 $r$ -Domination

In this subsection, we use the minimum  $r$ -covering subtree  $T_r$  to determine an  $(r + \rho)$ -dominating set  $S$  in  $\mathcal{O}(nm)$  time using the following approach. First, compute  $T_r$ . Second, pick a leaf  $B$  of  $T_r$ . If there is a vertex  $v$  such that  $v$  is not dominated and  $B$  is the only bag intersecting the  $r$ -neighbourhood of  $v$ , then add the center of  $B$  into  $S$ , flag all vertices  $u$  with  $d_G(u, B) \leq r(u)$  as dominated, and remove  $B$  from  $T_r$ . Repeat the second step until  $T_r$  contains no more bags and each vertex is flagged as dominated. Algorithm 4.5 below implements this approach. Note that, instead of removing bags from  $T_r$ , we use a reversed BFS-order of the bags to ensure the algorithm processes bags in the correct order.

---

**Algorithm 4.5:** Computes an  $(r + \rho)$ -dominating set  $S$  for a given graph  $G$  with a given tree-decomposition  $\mathcal{T}$  with breadth  $\rho$ .

---

- 1 Compute a minimum  $r$ -covering subtree  $T_r$  of  $\mathcal{T}$  using Algorithm 4.4.
  - 2 Give each vertex  $v$  a binary flag indicating if  $v$  is *dominated*. Initially, no vertex is dominated.
  - 3 Create an empty vertex set  $S_0$ .
  - 4 Let  $\langle B_1, B_2, \dots, B_k \rangle$  be the reverse of a BFS-order of  $T_r$  starting at its root.
  - 5 **For**  $i = 1$  **To**  $k$
  - 6     **If**  $\sigma(B_i) > 0$  **Then**
  - 7         Determine all vertices  $u$  such that  $u$  has not been flagged as dominated and that  $d_G(u, B_i) \leq r(u)$ . Add all these vertices into a new set  $X_i$ .
  - 8         Let  $S_i = S_{i-1} \cup \{c(B_i)\}$ .
  - 9         For each vertex  $u \in X_i$ , flag  $u$  as dominated, and decrease  $\sigma(\beta(u))$  by 1.
  - 10     **Else**
  - 11         Let  $S_i = S_{i-1}$ .
  - 12 Output  $S := S_k$ .
- 

**Theorem 4.3.** *Let  $D$  be a minimum  $r$ -dominating set for a given graph  $G$ . Given a tree-decomposition with breadth  $\rho$  for  $G$ , Algorithm 4.5 computes an  $(r + \rho)$ -dominating set  $S$  with  $|S| \leq |D|$  in  $\mathcal{O}(nm)$  time.*

*Proof (Correctness).* First, we show that  $S$  is an  $(r + \rho)$ -dominating set for  $G$ . Note that a vertex  $v$  is flagged as dominated only if  $S_i$  contains a vertex  $c(B_j)$  with  $d_G(v, B_j) \leq r(v)$



(see line 7 to line 9). Thus,  $v$  is flagged as dominated only if  $d_G(v, S_i) \leq d_G(v, c(B_j)) \leq r(v) + \rho$ . Additionally, by construction of  $T_r$  (see Algorithm 4.3), for each vertex  $v$ ,  $T_r$  contains a bag  $B$  with  $\beta(v) = B$ ,  $\sigma(B)$  states the number of vertices  $v$  with  $\beta(v) = B$ , and  $\sigma(B)$  is decreased by 1 only if such a vertex  $v$  is flagged as dominated (see line 9). Therefore, if  $G$  contains a vertex  $v$  with  $d_G(v, S_i) > r(v) + \rho$ , then  $v$  is not flagged as dominated and  $T_r$  contains a bag  $B_i$  with  $\beta(v) = B_i$  and  $\sigma(B_i) > 0$ . Thus, when  $B_i$  is processed by the algorithm,  $c(B_i)$  will be added to  $S_i$  and, hence,  $d_G(v, S_i) \leq r(v) + \rho$ .

Let  $V_i^S = \{u \mid d_G(u, B_j) \leq r(u), c(B_j) \in S_i\}$  be the set of vertices which are flagged as dominated after the algorithm processed  $B_i$ , i. e., each vertex in  $V_i^S$  is  $(r + \rho)$ -dominated by  $S_i$ . Similarly, for some set  $D_i \subseteq D$ , let  $V_i^D = \{u \mid d_G(u, D_i) \leq r(u)\}$  be the set of vertices dominated by  $D_i$ . To show that  $|S| \leq |D|$ , we show by induction over  $i$  that, for each  $i$ , (i) there is a set  $D_i \subseteq D$  such that  $V_i^D \subseteq V_i^S$ , (ii)  $|S_i| = |D_i|$ , and (iii) if, for some vertex  $v$ ,  $\beta(v) = B_j$  with  $j \leq i$ , then  $v \in V_i^S$ .

For the base case, let  $S_0 = D_0 = \emptyset$ . Then,  $V_0^S = V_0^D = \emptyset$  and all three statements are satisfied. For the inductive step, first, consider the case when  $\sigma(B_i) = 0$ . Because  $\sigma(B_i) = 0$ , each vertex  $v$  with  $\beta(v) = B_i$  is flagged as dominated, i. e.,  $v \in V_{i-1}^S$ . Thus, by setting  $S_i = S_{i-1}$  (line 11) and  $D_i = D_{i-1}$ , all three statements are satisfied for  $i$ . Next, consider the case when  $\sigma(B_i) > 0$ . Therefore,  $G$  contains a vertex  $u$  with  $\beta(u) = B_i$  and  $u \notin V_{i-1}^S$ . Then, the algorithm sets  $S_i = S_{i-1} \cup \{c(B_i)\}$  and flags all such  $u$  as dominated (see line 7 to line 9). Thus,  $u \in V_i^S$  and statement (iii) is satisfied. Let  $d_u$  be a vertex in  $D$  with minimal distance to  $u$ . Thus,  $d_G(d_u, u) \leq r(u)$ , i. e.,  $d_u$  is in the  $r$ -neighbourhood of  $u$ . Note that, because  $u \notin V_{i-1}^S$  and  $V_{i-1}^D \subseteq V_{i-1}^S$ ,  $d_u \notin D_{i-1}$ . Therefore, by setting  $D_i = D_{i-1} \cup \{d_u\}$ ,  $|S_i| = |S_{i-1}| + 1 = |D_{i-1}| + 1 = |D_i|$  and statement (ii) is satisfied. Recall that  $\beta(u)$  points to the bag closest to the root of  $T_r$  which intersects the  $r$ -neighbourhood of  $u$ . Thus, because  $\beta(u) = B_i$ , each bag  $B \neq B_i$  with  $d_G(u, B) \leq r(u)$  is a descendant of  $B_i$ . Therefore,  $d_u$  is in  $B_i$  or in a descendant of  $B_i$ . Let  $v$  be an arbitrary vertex of  $G$  such that  $v \notin V_{i-1}^S$  and  $d_G(v, d_u) \leq r(v)$ , i. e.,  $v$  is dominated by  $d_u$  but not by  $S_{i-1}$ . Due to statement (iii) of the induction hypothesis,  $\beta(v) = B_j$  with  $j \geq i$ , i. e.,  $B_j$  cannot be a descendant of  $B_i$ . Partition the vertices of  $G$  into the sets  $V_i^\uparrow$  and  $V_i^\downarrow$  such that  $V_i^\downarrow$  contains the vertices which are contained in  $B_i$  or in a descendant of  $B_i$ . If  $v \in V_i^\downarrow$ , then there is a path of length at most  $r(v)$  from  $v$  to  $B_j$  passing through  $B_i$ . If  $v \in V_i^\uparrow$ , then, because  $d_u \in V_i^\downarrow$ , there is a path of length at most  $r(v)$  from  $v$  to  $d_u$  passing through  $B_i$ . Therefore,  $d_G(v, B_i) \leq r(v)$ . That is, each vertex  $r$ -dominated by  $d_u$ , is  $(r + \rho)$ -dominated by some  $c(B_j) \in S_i$ . Therefore, because  $S_i = S_{i-1} \cup \{c(B_i)\}$  and  $D_i = D_{i-1} \cup \{d_u\}$ ,  $v \in V_i^S \cap V_i^D$  and, thus, statement (i) is satisfied.  $\square$

*Proof (Complexity).* Computing  $T_r$  (line 1) takes at most  $\mathcal{O}(nm)$  time (see Lemma 4.9). Because  $T_r$  has at most  $n$  bags, computing a BFS-order of  $T_r$  (line 4) takes at most  $\mathcal{O}(n)$  time. For some bag  $B_i$ , determining all vertices  $u$  with  $d_G(u, B_i) \leq r(u)$ , flagging  $u$  as dominated, and decreasing  $\sigma(\beta(u))$  (line 7 to line 9) can be done in  $\mathcal{O}(m)$  time by performing a BFS starting at all vertices of  $B_i$  simultaneously. Therefore, because  $T_r$  has

at most  $n$  bags, Algorithm 4.5 requires at most  $\mathcal{O}(nm)$  total time.  $\square$

### 4.2.3 Connected $r$ -Domination

In this subsection, we show how to compute a connected  $(r + 5\rho)$ -dominating set and a connected  $(r + 3\lambda)$ -dominating set for  $G$ . For both results, we use almost the same algorithm. To identify and emphasise the differences, we use the label  $(\heartsuit)$  for parts which are only relevant to determine a connected  $(r + 5\rho)$ -dominating set and use the label  $(\diamond)$  for parts which are only relevant to determine a connected  $(r + 3\lambda)$ -dominating set.

For the remainder of this subsection, let  $D_r$  be a minimum connected  $r$ -dominating set of  $G$ . For  $(\heartsuit)$   $\phi = 3\rho$  or  $(\diamond)$   $\phi = 2\lambda$ , let  $T_\phi$  be a minimum  $(r + \phi)$ -covering subtree of  $\mathcal{T}$  as computed by Algorithm 4.4.

The idea of our algorithm is to, first, compute  $T_\phi$  and, second, compute a small enough connected set  $C_\phi$  such that  $C_\phi$  intersects each bag of  $T_\phi$ . Lemma 4.10 below shows that such a set  $C_\phi$  is an  $(r + (\phi + \lambda))$ -dominating set.

**Lemma 4.10.** *Let  $C_\phi$  be a connected set that contains at least one vertex of each leaf of  $T_\phi$ . Then,  $C_\phi$  is an  $(r + (\phi + \lambda))$ -dominating set.*

*Proof.* Clearly, since  $C_\phi$  is connected and contains a vertex of each leaf of  $T_\phi$ ,  $C_\phi$  contains a vertex of every bag of  $T_\phi$ . By construction of  $T_\phi$ , for each vertex  $v$  of  $G$ ,  $T_\phi$  contains a bag  $B$  such that  $d_G(v, B) \leq r(v) + \phi$ . Therefore,  $d_G(v, C_\phi) \leq r(v) + \phi + \lambda$ , i. e.,  $C_\phi$  is an  $(r + (\phi + \lambda))$ -dominating set.  $\square$

To compute a connected set  $C_\phi$  which intersects all leaves of  $T_\phi$ , we first consider the case when  $T_\phi$  contains only one bag  $B$ . In this case, we can construct  $C_\phi$  by simply picking an arbitrary vertex  $v \in B$  and setting  $C_\phi = \{v\}$ . Similarly, if  $T_\phi$  contains exactly two bags  $B$  and  $B'$ , pick a vertex  $v \in B \cap B'$  and set  $C_\phi = \{v\}$ . In both cases, due to Lemma 4.10,  $C_\phi$  is clearly an  $(r + (\phi + \lambda))$ -dominating set with  $|C_\phi| \leq |D_r|$ .

Now, consider the case when  $T_\phi$  contains at least three bags. Additionally, assume that  $T_\phi$  is a rooted tree such that its root  $R$  is a leaf.

**Notation.** Based on its degree in  $T_\phi$ , we refer to each bag  $B$  of  $T_\phi$  either as *leaf*, as *path bag* if  $B$  has degree 2, or as *branching bag* if  $B$  has a degree larger than 2. Additionally, we call a maximal connected set of path bags a *path segment* of  $T_\phi$ . Let  $\mathbb{L}$  denote the set of leaves,  $\mathbb{P}$  denote the set of path segments, and  $\mathbb{B}$  denote the set of branching bags of  $T_\phi$ . Clearly, for any given tree  $T$ , the sets  $\mathbb{L}$ ,  $\mathbb{P}$ , and  $\mathbb{B}$  are pairwise disjoint and can be computed in linear time.

Let  $B$  and  $B'$  be two adjacent bags of  $T_\phi$  such that  $B$  is the parent of  $B'$ . We call  $S = B \cap B'$  the *up-separator* of  $B'$ , denoted as  $S^\uparrow(B')$ , and a *down-separator* of  $B$ , denoted as  $S^\downarrow(B)$ , i. e.,  $S = S^\uparrow(B') = S^\downarrow(B)$ . Note that a branching bag has multiple down-separators and that (with exception of  $R$ ) each bag has exactly one up-separator. For each branching bag  $B$ , let  $\mathcal{S}^\downarrow(B)$  be the set of down-separators of  $B$ . Accordingly,

for a path segment  $P \in \mathbb{P}$ ,  $S^\uparrow(P)$  is the up-separator of the bag in  $P$  closest to the root and  $S^\downarrow(P)$  is the down separator of the bag in  $P$  furthest from the root. Let  $\nu$  be a function that assigns a vertex of  $G$  to a given separator. Initially,  $\nu(S)$  is undefined for each separator  $S$ .

**Algorithm.** Now, we show how to compute  $C_\phi$ . We, first, split  $T_\phi$  into the sets  $\mathbb{L}$ ,  $\mathbb{P}$ , and  $\mathbb{B}$ . Second, for each  $P \in \mathbb{P}$ , we create a small connected set  $C_P$ , and, third, for each  $B \in \mathbb{B}$ , we create a small connected set  $C_B$ . If this is done properly, the union  $C_\phi$  of all these sets forms a connect set which intersects each bag of  $T_\phi$ .

Note that, due to properties of tree-decompositions, it can be the case that there are two bags  $B$  and  $B'$  which have a common vertex  $v$ , even if  $B$  and  $B'$  are non-adjacent in  $T_\phi$ . In such a case, either  $v \in S^\downarrow(B) \cap S^\uparrow(B')$  if  $B$  is an ancestor of  $B'$ , or  $v \in S^\uparrow(B) \cap S^\downarrow(B')$  if neither is ancestor of the other. To avoid problems caused by this phenomena and to avoid counting vertices multiple times, we consider any vertex in an up-separator as part of the bag above. That is, whenever we process some segment or bag  $X \in \mathbb{L} \cup \mathbb{P} \cup \mathbb{B}$ , even though we add a vertex  $v \in S^\uparrow(X)$  to  $C_\phi$ ,  $v$  is not contained in  $C_X$ .

*Processing Path Segments.* First, after splitting  $T_\phi$ , we create a set  $C_P$  for each path segment  $P \in \mathbb{P}$  as follows. We determine  $S^\uparrow(P)$  and  $S^\downarrow(P)$  and then find a shortest path  $Q_P$  from  $S^\uparrow(P)$  to  $S^\downarrow(P)$ . Note that  $Q_P$  contains exactly one vertex from each separator. Let  $x \in S^\uparrow(P)$  and  $y \in S^\downarrow(P)$  be these vertices. Then, we set  $\nu(S^\uparrow(P)) = x$  and  $\nu(S^\downarrow(P)) = y$ . Last, we add the vertices of  $Q_P$  into  $C_\phi$  and define  $C_P$  as  $Q_P \setminus S^\uparrow(P)$ . Let  $C_\mathbb{P}$  be the union of all sets  $C_P$ , i. e.,  $C_\mathbb{P} = \bigcup_{P \in \mathbb{P}} C_P$ .

**Lemma 4.11.**  $|C_\mathbb{P}| \leq |D_r| - \phi \cdot \Lambda(T_\phi)$ .

*Proof.* Recall that  $T_\phi$  is a minimum  $(r + \phi)$ -covering subtree of  $\mathcal{T}$ . Thus, by Corollary 4.5, for each leaf  $B \in \mathbb{L}$  of  $T_\phi$ , there is a vertex  $v$  in  $G$  such that  $B$  is the only bag of  $T_\phi$  with  $d_G(v, B) \leq r(v) + \phi$ . Because  $D_r$  is a connected  $r$ -dominating set,  $D_r$  intersects the  $r$ -neighbourhood of each of these vertices  $v$ . Thus, by properties of tree-decompositions,  $D_r$  intersects each bag of  $T_\phi$ . Additionally, for each such  $v$ ,  $D_r$  contains a path  $D_v$  with  $|D_v| \geq \phi$  such that  $D_v$  intersects the  $r$ -neighbourhood of  $v$ , intersects the corresponding leaf  $B$  of  $T_\phi$ , and does not intersect  $S^\uparrow(B)$  ( $S^\downarrow(B)$  if  $B = R$ ). Let  $D_\mathbb{L}$  be the union of all such sets  $D_v$ . Therefore,  $|D_\mathbb{L}| \geq \phi \cdot \Lambda(T_\phi)$ .

Because  $D_r$  intersects each bag of  $T_\phi$ ,  $D_r$  also intersects the up- and down-separators of each path segment. For a path segment  $P \in \mathbb{P}$ , let  $x$  and  $y$  be two vertices of  $D_r$  such that  $x \in S^\uparrow(P)$ ,  $y \in S^\downarrow(P)$ , and for which the distance in  $G[D_r]$  is minimal. Let  $D_P$  be the set of vertices on the shortest path in  $G[D_r]$  from  $x$  to  $y$  without  $x$ , i. e.,  $x \notin D_P$ . Note that, by construction, for each  $P \in \mathbb{P}$ ,  $D_P$  contains exactly one vertex in  $S^\downarrow(P)$  and no vertex in  $S^\uparrow(P)$ . Thus, for all  $P, P' \in \mathbb{P}$ ,  $D_P \cap D_{P'} = \emptyset$ . Let  $D_\mathbb{P}$  be the union of all such sets  $D_P$ , i. e.,  $D_\mathbb{P} = \bigcup_{P \in \mathbb{P}} D_P$ . By construction,  $|D_\mathbb{P}| = \sum_{P \in \mathbb{P}} |D_P|$  and  $D_\mathbb{L} \cap D_\mathbb{P} = \emptyset$ .

Therefore,  $|D_r| \geq |D_{\mathbb{P}}| + |D_{\mathbb{L}}|$  and, hence,

$$\sum_{P \in \mathbb{P}} |D_P| \leq |D_r| - |D_{\mathbb{L}}| \leq |D_r| - \phi \cdot \Lambda(T_\phi).$$

Recall that, for each  $P \in \mathbb{P}$ , the sets  $C_P$  and  $D_P$  are constructed based on a path from  $S^\uparrow(P)$  to  $S^\downarrow(P)$ . Since  $C_P$  is based on a shortest path in  $G$ , it follows that  $|C_P| = d_G(S^\uparrow(P), S^\downarrow(P)) \leq |D_P|$ . Therefore,

$$|C_{\mathbb{P}}| \leq \sum_{P \in \mathbb{P}} |C_P| \leq \sum_{P \in \mathbb{P}} |D_P| \leq |D_r| - \phi \cdot \Lambda(T_\phi). \quad \square$$

*Processing Branching Bags.* After processing path segments, we process the branching bags of  $T_\phi$ . Similar to path segments, we have to ensure that all separators are connected. Branching bags, however, have multiple down-separators. To connect all separators of some bag  $B$ , we pick a vertex  $s$  in each separator  $S \in \mathcal{S}^\downarrow(B) \cup \{S^\uparrow(B)\}$ . If  $\nu(S)$  is defined, we set  $s = \nu(S)$ . Otherwise, we pick an arbitrary  $s \in S$  and set  $\nu(S) = s$ . Let  $\mathcal{S}^\downarrow(B) = \{S_1, S_2, \dots\}$ ,  $s_i = \nu(S_i)$ , and  $t = \nu(S^\uparrow(B))$ . We then connect these vertices as follows. (See Figure 4.2 for an illustration.)

( $\heartsuit$ ) Connect each vertex  $s_i$  via a shortest path  $Q_i$  (of length at most  $\rho$ ) with the center  $c(B)$  of  $B$ . Additionally, connect  $c(B)$  via a shortest path  $Q_t$  (of length at most  $\rho$ ) with  $t$ . Add all vertices from the paths  $Q_i$  and from the path  $Q_t$  into  $C_\phi$  and let  $C_B$  be the union of these paths without  $t$ .

( $\diamond$ ) Connect each vertex  $s_i$  via a shortest path  $Q_i$  (of length at most  $\lambda$ ) with  $t$ . Add all vertices from the paths  $Q_i$  into  $C_\phi$  and let  $C_B$  be the union of these paths without  $t$ .

Let  $C_{\mathbb{B}}$  be the union of all created sets  $C_B$ , i. e.,  $C_{\mathbb{B}} = \bigcup_{B \in \mathbb{B}} C_B$ .



Figure 4.2. Construction of the set  $C_B$  for a branching bag  $B$ .

Before analysing the cardinality of  $C_{\mathbb{B}}$  in Lemma 4.13 below, we need an axillary lemma.

**Lemma 4.12.** *For a tree  $T$  which is rooted in one of its leaves, let  $b$  denote the number of branching nodes,  $c$  denote the total number of children of branching nodes, and  $l$  denote the number of leaves. Then,  $c + b \leq 3l - 1$  and  $c \leq 2l - 1$ .*

*Proof.* Assume that we construct  $T$  by starting with only the root and then step by step adding leaves to it. Let  $T_i$  be the subtree of  $T$  with  $i$  nodes during this construction. We define  $b_i$ ,  $c_i$ , and  $l_i$  accordingly. Now, assume by induction over  $i$  that Lemma 4.12 is true for  $T_i$ . Let  $v$  be the leaf we add to construct  $T_{i+1}$  and let  $u$  be its neighbour.

First, consider the case when  $u$  is a leaf of  $T_i$ . Then,  $u$  becomes a path node of  $T_{i+1}$ . Therefore,  $b_{i+1} = b_i$ ,  $c_{i+1} = c_i$ , and  $l_{i+1} = l_i$ . Next, assume that  $u$  is path node of  $T_i$ . Then,  $u$  is a branch node of  $T_{i+1}$ . Thus,  $b_{i+1} = b_i + 1$ ,  $c_{i+1} = c_i + 2$ , and  $l_{i+1} = l_i + 1$ . Therefore,  $c_{i+1} + b_{i+1} = c_i + b_i + 3 \leq 3(l_i + 1) - 1 = 3l_{i+1} - 1$  and  $c_{i+1} = c_i + 2 \leq 2(l_i + 1) - 1 = 2l_{i+1} - 1$ . It remains to check the case when  $u$  is a branch node of  $T_i$ . Then,  $b_{i+1} = b_i$ ,  $c_{i+1} = c_i + 1$ , and  $l_{i+1} = l_i + 1$ . Thus,  $c_{i+1} + b_{i+1} = c_i + b_i + 1 \leq 3l_i - 1 + 1 \leq 3l_{i+1} - 1$  and  $c_{i+1} = c_i + 1 \leq 2l_i - 1 + 1 \leq 2l_{i+1} - 1$ . Therefore, in all three cases, Lemma 4.12 is true for  $T_{i+1}$ .  $\square$

**Lemma 4.13.**  $|C_{\mathbb{B}}| \leq \phi \cdot \Lambda(T_\phi)$ .

*Proof.* For some branching bag  $B \in \mathbb{B}$ , the set  $C_B$  contains ( $\heartsuit$ ) a path of length at most  $\rho$  for each  $S_i \in \mathcal{S}^\downarrow(B)$  and a path of length at most  $\rho$  to  $S^\uparrow(B)$ , or ( $\diamond$ ) a path of length at most  $\lambda$  for each  $S_i \in \mathcal{S}^\downarrow(B)$ . Thus, ( $\heartsuit$ )  $|C_B| \leq \rho \cdot |\mathcal{S}^\downarrow(B)| + \rho$  or ( $\diamond$ )  $|C_B| \leq \lambda \cdot |\mathcal{S}^\downarrow(B)|$ . Recall that  $\mathcal{S}^\downarrow(B)$  contains exactly one down-separator for each child of  $B$  in  $T_\phi$  and that  $C_{\mathbb{B}}$  is the union of all sets  $C_B$ . Therefore, Lemma 4.12 implies the following.

$$\begin{aligned}
 |C_{\mathbb{B}}| &\leq \sum_{B \in \mathbb{B}} |C_B| \\
 (\heartsuit) &\leq \rho \cdot \sum_{B \in \mathbb{B}} |\mathcal{S}^\downarrow(B)| + \rho \cdot |\mathbb{B}| \leq 3\rho \cdot \Lambda(T_\phi) - 1 \\
 (\diamond) &\leq \lambda \cdot \sum_{B \in \mathbb{B}} |\mathcal{S}^\downarrow(B)| \leq 2\lambda \cdot \Lambda(T_\phi) - 1 \\
 &\leq \phi \cdot \Lambda(T_\phi) - 1. \quad \square
 \end{aligned}$$

*Properties of  $C_\phi$ .* We now analyse the created set  $C_\phi$  with the result that  $C_\phi$  is a connected  $(r + \phi)$ -dominating set for  $G$ .

**Lemma 4.14.**  $C_\phi$  contains a vertex in each bag of  $T_\phi$ .

*Proof.* Clearly, by construction,  $C_\phi$  contains a vertex in each path bag and in each branching bag. Now, consider a leaf  $L$  of  $T_\phi$ .  $L$  is adjacent to a path segment or branching bag  $X \in \mathbb{P} \cap \mathbb{B}$ . Whenever such an  $X$  is processed, the algorithm ensures that all separators of  $X$  contain a vertex of  $C_\phi$ . Since one of these separators is also the separator of  $L$ , it follows that each leaf  $L$  and, thus, each bag of  $T_\phi$  contains a vertex of  $C_\phi$ .  $\square$

**Lemma 4.15.**  $|C_\phi| \leq |D_r|$ .

*Proof.* Note that, for each vertex  $u$  we add to  $C_\phi$ , we also add  $u$  to a unique set  $C_X$  for some  $X \in \mathbb{P} \cap \mathbb{B}$ . The exception is the vertex  $v$  in  $S^\downarrow(R)$  which is added to no such set  $C_X$ .

It follows from our construction of the sets  $C_X$  that there is only one such vertex  $v$  and that  $v = \nu(S^\uparrow(R))$ . Thus,  $|C_\phi| = |C_{\mathbb{P}}| + |C_{\mathbb{B}}| + 1$ . Now, it follows from Lemma 4.11 and Lemma 4.13 that

$$|C_\phi| \leq |D_r| - \phi \cdot \Lambda(T_\phi) + \phi \cdot \Lambda(T_\phi) - 1 + 1 \leq |D_r|. \quad \square$$

**Lemma 4.16.**  $C_\phi$  is connected.

*Proof.* First, note that, by maximality, two path segments of  $T_\phi$  cannot share a common separator. Also, note that, when processing a branching bag  $B$ , the algorithm first checks if, for any separator  $S$  of  $B$ ,  $\nu(S)$  is already defined; if this is the case, it will not be overwritten. Therefore, for each separator  $S$  in  $T_\phi$ ,  $\nu(S)$  is defined and never overwritten.

Next, consider a path segment or branching bag  $X \in \mathbb{P} \cup \mathbb{B}$  and let  $S$  and  $S'$  be two separators of  $X$ . Whenever such an  $X$  is processed, our approach ensures that  $C_\phi$  connects  $\nu(S)$  with  $\nu(S')$ . Additionally, observe that, when processing  $X$ , each vertex added to  $C_\phi$  is connected via  $C_\phi$  with  $\nu(S)$  for some separator  $S$  of  $X$ .

Thus, for any two separators  $S$  and  $S'$  in  $T_\phi$ ,  $C_\phi$  connects  $\nu(S)$  with  $\nu(S')$  and, additionally, each vertex  $v \in C_\phi$  is connected via  $C_\phi$  with  $\nu(S)$  for some separator  $S$  in  $T_\phi$ . Therefore,  $C_\phi$  is connected.  $\square$

From Lemma 4.14, Lemma 4.15, Lemma 4.16, and from applying Lemma 4.10 it follows:

**Corollary 4.6.**  $C_\phi$  is a connected  $(r + (\phi + \lambda))$ -dominating set for  $G$  with  $|C_\phi| \leq |D_r|$ .

*Implementation.* Algorithm 4.6 below implements our approach described above. This also includes the case when  $T_\phi$  contains at most two bags.

**Theorem 4.4.** Algorithm 4.6 computes a connected  $(r + (\phi + \lambda))$ -dominating set  $C_\phi$  with  $|C_\phi| \leq |D_r|$  in  $\mathcal{O}(nm)$  time.

*Proof.* Since Algorithm 4.6 constructs a set  $C_\phi$  as described above, its correctness follows from Corollary 4.6. It remains to show that the algorithm runs in  $\mathcal{O}(nm)$  time.

Computing  $T_\phi$  (line 2) can be done in  $\mathcal{O}(nm)$  time (see Lemma 4.9). Picking a vertex  $u$  in the case when  $T_\phi$  contains at most two bags (line 3 to line 6) can be easily done in  $\mathcal{O}(n)$  time. Recall that  $T_\phi$  has at most  $n$  bags. Thus, splitting  $T_\phi$  in the sets  $\mathbb{L}$ ,  $\mathbb{P}$ , and  $\mathbb{B}$  can be done in  $\mathcal{O}(n)$  time.

Determining all up-separators in  $T_\phi$  can be done in  $\mathcal{O}(M)$  time as follows. Process all bags of  $T_\phi$  in an order such that a bag is processed before its descendants, e. g., use a preorder or BFS-order. Whenever a bag  $B$  is processed, determine a set  $S \subseteq B$  of flagged vertices, store  $S$  as up-separator of  $B$ , and, afterwards, flag all vertices in  $B$ . Clearly,  $S$  is empty for the root. Because a bag  $B$  is processed before its descendants, all flagged vertices in  $B$  also belong to its parent. Thus, by properties of tree-decompositions, these vertices are exactly the vertices in  $S^\uparrow(B)$ . Clearly, processing a single bag  $B$  takes at

---

**Algorithm 4.6:** Computes ( $\heartsuit$ ) a connected  $(r + 5\rho)$ -dominating set or ( $\diamond$ ) a connected  $(r + 3\lambda)$ -dominating set for a given graph  $G$  with a given tree-decomposition  $\mathcal{T}$  with breadth  $\rho$  and length  $\lambda$ .

---

- 1 ( $\heartsuit$ ) Set  $\phi := 3\rho$ .
  - ( $\diamond$ ) Set  $\phi := 2\lambda$ .
  - 2 Compute a minimum  $(r + \phi)$ -covering subtree  $T_\phi$  of  $\mathcal{T}$  using Algorithm 4.4.
  - 3 **If**  $T_\phi$  contains only one bag  $B$  **Then**
  - 4   └ Pick an arbitrary vertex  $u \in B$ , output  $C_\phi := \{u\}$ , and **stop**.
  - 5 **If**  $T_\phi$  contains exactly two bags  $B$  and  $B'$  **Then**
  - 6   └ Pick an arbitrary vertex  $u \in B \cap B'$ , output  $C_\phi := \{u\}$ , and **stop**.
  - 7 Pick a leaf of  $T_\phi$  and make it the root of  $T_\phi$ .
  - 8 Split  $T_\phi$  into a set  $\mathbb{L}$  of leaves, a set  $\mathbb{P}$  of path segments, and a set  $\mathbb{B}$  of branching bags.
  - 9 Create an empty set  $C_\phi$ .
  - 10 **For Each**  $P \in \mathbb{P}$
  - 11   └ Find a shortest path  $Q_P$  from  $S^\uparrow(P)$  to  $S^\downarrow(P)$  and add its vertices into  $C_\phi$ .
  - 12   └ Let  $x \in S^\uparrow(P)$  be the start vertex and  $y \in S^\downarrow(P)$  be the end vertex of  $Q_P$ . Set  $\nu(S^\uparrow(P)) := x$  and  $\nu(S^\downarrow(P)) := y$ .
  - 13 **For Each**  $B \in \mathbb{B}$
  - 14   └ If  $\nu(S^\uparrow(B))$  is defined, let  $u := \nu(S^\uparrow(B))$ . Otherwise, let  $u$  be an arbitrary vertex in  $S^\uparrow(B)$  and set  $\nu(S^\uparrow(B)) := u$ .
  - 15   └ ( $\heartsuit$ ) Let  $v := c(B)$  be the center of  $B$ .
  - ( $\diamond$ ) Let  $v := u$ .
  - 16   └ Find a shortest path from  $u$  to  $v$  and add its vertices into  $C_\phi$ .
  - 17   └ **For Each**  $S_i \in S^\downarrow(B)$
  - 18     └ If  $\nu(S_i)$  is defined, let  $w_i := \nu(S_i)$ . Otherwise, let  $w_i$  be an arbitrary vertex in  $S_i$  and set  $\nu(S_i) := w_i$ .
  - 19     └ Find a shortest path from  $w_i$  to  $v$  and add the vertices of this path into  $C_\phi$ .
  - 20 Output  $C_\phi$ .
- 

most  $\mathcal{O}(|B|)$  time. Thus, processing all bags takes at most  $\mathcal{O}(M)$  time. Note that it is not necessary to determine the down-separators of a (branching) bag. They can easily be accessed via the children of a bag.

Processing a single path segment (line 11 and line 12) can be easily done in  $\mathcal{O}(m)$  time. Processing a branching bag  $B$  (line 13 to line 19) can be implemented to run in  $\mathcal{O}(m)$  time by, first, determining  $\nu(S)$  for each separator  $S$  of  $B$  and, second, running a BFS starting at  $v$  (defined in line 15) to connect  $v$  with each vertex  $\nu(S)$ . Because  $T_\phi$  has at most  $n$  bags, it takes at most  $\mathcal{O}(nm)$  time to process all path segments and branching bags of  $T_\phi$ . Therefore, Algorithm 4.6 runs in  $\mathcal{O}(nm)$  total time.  $\square$

### 4.3 Implications for the $p$ -Center Problem

The  $p$ -Center problems asks for a vertex set  $C$  such that  $|C| \leq p$  and the eccentricity of  $C$  is minimal. It is known (see, e. g., [14]) that the  $p$ -Center problem and  $r$ -Domination problem are closely related. Indeed, one can solve each of these problems by solving the other problem a logarithmic number of times. Lemma 4.17 below generalises this observation. Informally, it states that we are able to find a  $+\phi$ -approximation for the  $p$ -Center problem if we can find a good  $(r + \phi)$ -dominating set.

**Lemma 4.17.** *For a given graph  $G$ , let  $D_r$  be an optimal (connected)  $r$ -dominating set and  $C_p$  be an optimal (connected)  $p$ -center. If, for some non-negative integer  $\phi$ , there is an algorithm to compute a (connected)  $(r + \phi)$ -dominating set  $D$  with  $|D| \leq |D_r|$  in  $\mathcal{O}(T(G))$  time, then there is an algorithm to compute a (connected)  $p$ -center  $C$  with  $\text{ecc}(C) \leq \text{ecc}(C_p) + \phi$  in  $\mathcal{O}(T(G) \log n)$  time.*

*Proof.* Let  $\mathcal{A}$  be an algorithm which computes a (connected)  $(r + \phi)$  dominating set  $D = \mathcal{A}(G, r)$  for  $G$  with  $|D| \leq |D_r|$  in  $\mathcal{O}(T(G))$  time. Then, we can compute a (connected)  $p$ -center for  $G$  as follows. Make a binary search over the integers  $i \in [0, n]$ . In each iteration, set  $r_i(u) = i$  for each vertex  $u$  of  $G$  and compute the set  $D_i = \mathcal{A}(G, r_i)$ . Then, increase  $i$  if  $|D_i| > p$  and decrease  $i$  otherwise. Note that, by construction,  $\text{ecc}(D_i) \leq i + \phi$ . Let  $D$  be the resulting set, i. e., out of all computed sets  $D_i$ ,  $D$  is the set with minimal  $i$  for which  $|D_i| \leq p$ . It is easy to see that finding  $D$  requires at most  $\mathcal{O}(T(G) \log n)$  time.

Clearly,  $C_p$  is a (connected)  $r$ -dominating set for  $G$  when setting  $r(u) = \text{ecc}(C_p)$  for each vertex  $u$  of  $G$ . Thus, for each  $i \geq \text{ecc}(C_p)$ ,  $|D_i| \leq |C_p| \leq p$  and, hence, the binary search decreases  $i$  for next iteration. Therefore, there is an  $i \leq \text{ecc}(C_p)$  such that  $D = D_i$ . Hence,  $|D| \leq |C_p|$  and  $\text{ecc}(D) \leq \text{ecc}(C_p) + \phi$ .  $\square$

From Lemma 4.17, the results in Table 4.1 and Table 4.2 follow immediately.

**Table 4.1.** Implications of our results for the  $p$ -Center problem.

Approach	Approx.	Time
Layering Partition	$+\Delta$	$\mathcal{O}(m \log n)$
Tree-Decomposition	$+\rho$	$\mathcal{O}(nm \log n)$

**Table 4.2.** Implications of our results for the Connected  $p$ -Center problem.

Approach	Approx.	Time
Layering Partition	$+2\Delta$	$\mathcal{O}(m \alpha(n) \log \Delta \log n)$
Tree-Decomposition	$+\min(5\rho, 3\lambda)$	$\mathcal{O}(nm \log n)$

Theorem 4.5 below shows that we can slightly improve the result for the  $p$ -Center problem when using a layering partition.



**Theorem 4.5.** *For a given graph  $G$ , a  $+\Delta$ -approximation for the  $p$ -Center problem can be computed in linear time.*

*Proof.* First, create a layering partition  $\mathcal{T}$  of  $G$ . Second, find an optimal  $p$ -center  $\mathcal{S}$  for  $\mathcal{T}$ . Third, create a set  $S$  by picking an arbitrary vertex of  $G$  from each cluster in  $\mathcal{S}$ . All three steps can be performed in linear time, including the computation of  $\mathcal{S}$  (see [54]).

Let  $C$  be an optimal  $p$ -center for  $G$ . Note that, by Lemma 2.6 (page 7),  $C$  also induces a  $p$ -center for  $\mathcal{T}$ . Therefore, because  $S$  induces an optimal  $p$ -center for  $\mathcal{T}$ , Lemma 2.6 (page 7) implies that, for each vertex  $u$  of  $G$ ,

$$d_G(u, C) \leq d_G(u, S) \leq d_{\mathcal{T}}(u, \mathcal{S}) + \Delta \leq d_{\mathcal{T}}(u, C) + \Delta \leq d_G(u, C) + \Delta. \quad \square$$

## Bandwidth and Line-Distortion<sup>\*</sup>

Consider a given graph  $G = (V, E)$ . Then, an injective function  $f: V \rightarrow \mathbb{N}$  is called *layout* of  $G$ . The *bandwidth* of a layout  $f$ , denoted as  $\text{bw}(f)$ , is defined as maximum stretch of any edge, i. e.,  $\text{bw}(f) = \max_{uv \in E} |f(u) - f(v)|$ . Additionally, the *bandwidth* of a graph  $G$ , denoted as  $\text{bw}(G)$ , is defined as the minimum bandwidth of all layouts of  $G$ . Accordingly, the *bandwidth problem* asks, for a given graph  $G = (V, E)$ , to find a layout  $f$  with minimum bandwidth.

A *non-contractive embedding*  $f$  of  $G$  is a layout  $f$  of  $G$  with the additional requirement that, for all vertices  $u$  and  $v$ ,  $d(u, v) \leq |f(u) - f(v)|$ . The *distortion* of such an embedding  $f$  is the smallest integer  $k$  such that  $|f(u) - f(v)| \leq k \cdot d(u, v)$  for all edges  $uv$  of  $G$ . The minimum *line-distortion* of a graph  $G$ , denoted as  $\text{ld}(G)$ , is defined as the minimum distortion of all non-contractive embeddings of  $G$ . Accordingly, the *line-distortion problem* asks, for a given graph  $G$ , to find an embedding  $f$  with minimum distortion.

Both problems may appear to be closely related to each other. The only difference between the two parameters is that a minimum distortion embedding has to be non-contractive whereas there is no such restriction for bandwidth. It is known that  $\text{bw}(G) \leq \text{ld}(G)$  for every connected graph  $G$  [62]. However, the bandwidth and the minimum line-distortion of a graph can be very different. For example, a cycle of length  $n$  has bandwidth 2, whereas its minimum line-distortion is exactly  $n - 1$  [62].

Computing a minimum distortion embedding of a given graph  $G$  into a line  $\ell$  was recently identified as a fundamental algorithmic problem with important applications in various areas of computer science, like computer vision [89], as well as in computational chemistry and biology (see [63, 64]).

In this chapter, we investigate possible connections between the line-distortion, bandwidth, and the path-length of a graph. We show that, for every graph  $G$ ,  $\text{pl}(G) \leq \text{ld}(G)$  and  $\text{pb}(G) \leq \lceil \text{ld}(G)/2 \rceil$  hold. Additionally, we show that, for every class of graphs with path-length bounded by a constant, there is an efficient constant-factor approximation algorithm for the minimum bandwidth problem. Furthermore, we demonstrate that, for graphs with path-length bounded by a constant, there is an efficient constant-factor approximation algorithm for the minimum line-distortion problem. In the last section of this chapter, we give a linear time 6-approximation algorithm for the minimum line-distortion problem and a linear time 4-approximation algorithm for the minimum bandwidth problem for AT-free graphs.

---

<sup>\*</sup> Results from this chapter have been published partially at *SWAT 2014*, Copenhagen, Denmark [38], and in *Algorithmica* [39].

## 5.1 Existing Results

Bandwidth is known to be one of the hardest graph problems; it is NP-hard even for very simple graphs like caterpillars of hair-length at most 3 [75], and it is hard to approximate by a constant factor even for trees [11] and caterpillars with arbitrary hair-lengths [46]. Polynomial-time algorithms for the exact computation of bandwidth are known for very few graph classes, including bipartite permutation graphs [60] and interval graphs [66, 70, 87]. Constant-factor approximation algorithms are known for AT-free graphs [67] and convex bipartite graphs [84]. Recently, GOLOVACH et al. [57] showed also that the bandwidth minimization problem is Fixed Parameter Tractable on AT-free graphs by presenting an  $n2^{\mathcal{O}(k \log k)}$  time algorithm. For general (unweighted) graphs, the minimum bandwidth can be approximated to within a factor of  $\mathcal{O}(\log^{3.5} n)$  [50]. For trees and chordal graphs, the minimum bandwidth can be approximated to within a factor of  $\mathcal{O}(\log^{2.5} n)$  [58]. For caterpillars, the minimum bandwidth can be approximated to within a factor of  $\mathcal{O}(\log n / \log \log n)$  [51].

Table 5.1 and Table 5.2 summarise the results mentioned above.

**Table 5.1.** Existing solutions for calculating bandwidth.

Graph Class	Solution Quality	Run Time	
AT-free	optimal	$n2^{\mathcal{O}(k \log k)}$	[57]
	2-approx.	$\mathcal{O}(nm)$	[67]
	4-approx.	$\mathcal{O}(m + n \log n)$	[67]
convex bipartite	2-approx.	$\mathcal{O}(n \log^2 n)$	[84]
	4-approx.	$\mathcal{O}(n)$	[84]
bipartite permutation	optimal	$\mathcal{O}(n^4 \log n)$	[60]
interval	optimal	$\mathcal{O}(n \log^2 n)$	[87]
chordal	$\mathcal{O}(\log^{2.5} n)$ -approx.	polynomial	[58]
caterpillars	$\mathcal{O}(\log n / \log \log n)$	polynomial	[51]

**Table 5.2.** Existing hardness results for calculating bandwidth.

Graph Class	Result	
caterpillars (hair-length at most 3)	NP-hard	[75]
caterpillars	hard to approximate by a constant factor	[46]
trees	hard to approximate by a constant factor	[11]
convex bipartite	NP-hard	[84]

In [18], BĂDOIU et al. showed that the line-distortion problem is hard to approximate within a constant factor. They gave an exponential-time exact algorithm and a polynomial-

time  $\mathcal{O}(n^{1/2})$ -approximation algorithm for arbitrary unweighted input graphs, along with a polynomial-time  $\mathcal{O}(n^{1/3})$ -approximation algorithm for unweighted trees. In another paper, BĂDOIU et al. [17] showed that the problem is hard to  $\mathcal{O}(n^{1/12})$ -approximate, even for weighted trees. They also gave a better polynomial-time approximation algorithm for general weighted graphs, along with a polynomial-time algorithm that approximates the minimum line-distortion  $k$  embedding of a weighted tree by a factor that is polynomial in  $k$ .

Fast exponential-time exact algorithms for computing the line-distortion of a graph were proposed in [52, 53]. FOMIN et al. [53] showed that a minimum distortion embedding of an unweighted graph into a line can be found in time  $5^{n+o(n)}$ . FELLOWS et al. [52] gave an  $\mathcal{O}(nk^4(2k+1)^{2k})$  time algorithm that for an unweighted graph  $G$  and integer  $k$  either constructs an embedding of  $G$  into a line with distortion at most  $k$ , or concludes that no such embedding exists. They extended their approach also to weighted graphs obtaining an  $\mathcal{O}(nk^{4W}(2k+1)^{2kW})$  time algorithm, where  $W$  is the largest edge weight. Thus, the problem of minimum distortion embedding of a given graph  $G$  into a line  $\ell$  is Fixed Parameter Tractable.

Recently, HEGGERNES et al. [61, 62] initiated the study of minimum distortion embeddings into a line of specific graph classes. In particular, they gave polynomial-time algorithms for the problem on bipartite permutation graphs and on threshold graphs [62]. Furthermore, HEGGERNES et al. [61] showed that the problem of computing a minimum distortion embedding of a given graph into a line remains NP-hard even when the input graph is restricted to a bipartite, cobipartite, or split graph, implying that it is NP-hard also on chordal, cocomparability, and AT-free graphs. They also gave polynomial-time constant-factor approximation algorithms for split and cocomparability graphs.

Table 5.3 and Table 5.4 summarise the results mentioned above.

**Table 5.3.** Existing solutions for calculating the minimum distortion.

Graph Class	Solution Quality	Run Time	
trees (unweighted)	$\mathcal{O}(n^{1/3})$ -approx.	polynomial	[18]
trees (weighted)	$k^{\mathcal{O}(1)}$ -approx.	polynomial	[17]
general (unweighted)	optimal	$5^{n+o(n)}$	[53]
general	optimal	$\mathcal{O}(nk^4(2k+1)^{2k})$	[52]
bipartite permutation	optimal	$\mathcal{O}(n^2)$	[62]
threshold graphs	optimal	linear	[62]
split	6-approx.	linear	[61]
cocomparability	6-approx.	$\mathcal{O}(n \log^2 n + m)$	[61]

**Table 5.4.** Existing hardness results for calculating the minimum distortion.

Graph Class	Result	
general	$\mathcal{O}(1)$ -approximation is NP-hard	[18]
trees (weighted)	Hard to $\mathcal{O}(n^{1/12})$ -approximate	[17]
bipartite	NP-hard	[61]
cobipartite	NP-hard	[61]
split	NP-hard	[61]

## 5.2 $k$ -Dominating Pairs

A pair of vertices  $x$  and  $y$  of a graph  $G$  is called a  $k$ -dominating pair if every path between  $x$  and  $y$  has eccentricity at most  $k$ . It is known that every AT-free graph has a 1-dominating pair [24].

In this section, we investigate the relation of  $k$ -dominating pairs with the path-length and line-distortion of a graph. Additionally, we present approaches to determine if a given graph contains a  $k$ -domination pair.

### 5.2.1 Relation to Path-Length and Line-Distortion

**Lemma 5.1.** *Every graph  $G$  with  $\text{pl}(G) \leq \lambda$  has a  $\lambda$ -dominating pair.*

*Proof.* Consider a path-decomposition  $\mathcal{P} = \{X_1, \dots, X_q\}$  of length  $\text{pl}(G) \leq \lambda$  of  $G$ . Consider any two vertices  $x \in X_1$  and  $y \in X_q$  and a path  $P$  between them in  $G$ . Necessarily, by properties of path decompositions, every path of  $G$  connecting the vertices  $x$  and  $y$  has a vertex in every bag of  $\mathcal{P}$ . Hence, as each vertex  $v$  of  $G$  belongs to some bag  $X_i$  of  $\mathcal{P}$ , there is a vertex  $u \in P$  with  $u \in X_i$  and, thus,  $d(v, u) \leq \lambda$ .  $\square$

Next, we show that the line-distortion of a graph  $G$  gives an upper bound on the minimum  $k$  for which  $G$  contains a  $k$ -dominating pair.

**Lemma 5.2.** *Every graph  $G$  with  $\text{ld}(G) \leq \lambda$  has a  $\lfloor \frac{\lambda}{2} \rfloor$ -dominating pair.*

*Proof.* Let  $f$  be an optimal line embedding for  $G = (V, E)$ . This embedding has a first vertex  $v_1$  and a last vertex  $v_n$ , i. e., for all  $u \in V$ ,  $f(v_1) \leq f(u) \leq f(v_n)$ . Let  $u$  be an arbitrary vertex of  $G$  and  $P$  an arbitrary path from  $v_1$  to  $v_n$  in  $G$ . If  $u$  is not on this path, there is an edge  $v_i v_j$  of  $P$  with  $f(v_i) < f(u) < f(v_j)$ . Without loss of generality, let  $f(u) - f(v_i) \leq f(v_j) - f(u)$ . Thus, by definition of line-distortion, we can say that

$$f(u) - f(v_i) \leq \left\lfloor \frac{f(v_j) - f(v_i)}{2} \right\rfloor \leq \left\lfloor \frac{\lambda}{2} \right\rfloor.$$

Therefore, each vertex  $u$  of  $G$  is in distance at most  $\lfloor \frac{\lambda}{2} \rfloor$  to each path from  $v_1$  to  $v_n$ , i. e.,  $(v_1, v_n)$  is a  $\lfloor \frac{\lambda}{2} \rfloor$ -dominating pair.  $\square$

Note that the difference between both factors can be arbitrary large. The complete graph  $K_n$  has line-distortion  $n - 1$ . However, each vertex pair is a 1-dominating pair.

## 5.2.2 Determining a $k$ -Dominating Pair

In this subsection, we present a polynomial time algorithm to determine if a given graph contains a  $k$ -dominating pair. Additionally, we show that, in a graph with path-length  $\lambda$ , one can find a  $2\lambda$ -dominating pair in linear time.

Consider a  $k$ -dominating pair  $(x, y)$  and an arbitrary vertex  $w$ . By definition, each path  $P$  from  $x$  to  $y$  is in distance at most  $k$  to  $w$ , i. e.,  $P \cap N^k[w] \neq \emptyset$ . Therefore, we get the following observation.

**Observation 5.1.** *A pair of vertices  $x$  and  $y$  is a  $k$ -dominating pair if and only if, for every vertex  $w \in V \setminus (N^k[x] \cup N^k[y])$ , the disk  $N^k[w]$  separates  $x$  and  $y$ .*

Based on Observation 5.1, Algorithm 5.1 below determines if a given graph contains a  $k$ -dominating pair for a given  $k$ . The idea is to (i) compute the connected components of  $G - N^k[v]$  for each  $v$  and (ii) iterate over all pairs  $(x, y)$  and check, for each vertex  $w$  with sufficiently large distance to  $x$  and  $y$ , if  $x$  and  $y$  are in different connected components of  $G - N^k[w]$ . If this is the case,  $x$  and  $y$  form a dominating pair.

---

**Algorithm 5.1:** Determines if a given graph contains a  $k$ -dominating pair.

---

**Input:** A graph  $G = (V, E)$  and a non-negative integer  $k$ .

**Output:** A  $k$ -dominating pair  $(x, y)$  if such a pair exists in  $G$ .

- 1 Determine the pairwise distances of all vertices.
  - 2 Create an empty  $n \times n$  matrix  $M$ .
  - 3 **For Each**  $v \in V$
  - 4     Determine the connected components of  $G - N^k[v]$ .
  - 5     Label each vertex  $x$  in  $G - N^k[v]$  with its connected component and store this label in  $M(v, x)$ . (Thus,  $M(v, x)$  is the label of the connected component of vertex  $x$  in  $G - N^k[v]$ .)
  - 6 **For Each** vertex pair  $(x, y)$
  - 7     **If for each**  $w \in V$  with  $\max\{d(x, w), d(y, w)\} > k$ ,  $M(w, x) \neq M(w, y)$  **Then**
  - 8         Output  $(x, y)$  and **Stop**.
- 

**Theorem 5.1.** *Given a graph  $G$  and an integer  $k$ , Algorithm 5.1 determines if  $G$  contains a  $k$ -dominating pair  $(x, y)$  in  $\mathcal{O}(n^3)$  time.*

*Proof.* By Observation 5.1,  $(x, y)$  is a  $k$ -dominating pair if, for each vertex  $w \in V \setminus (N^k[x] \cup N^k[y])$ ,  $x$  and  $y$  are in different connected components of  $G - N^k[w]$ . That is,  $M(w, x) \neq M(w, y)$ . Clearly,  $w \notin (N^k[x] \cup N^k[y])$  if and only if the larger of  $d(x, w)$

and  $d(y, w)$  is strictly larger than  $k$ . Therefore, line 7 is successful if and only if  $(x, y)$  is a dominating pair.

The pairwise distances of all vertices as well as the matrix  $M$  can easily be computed in  $\mathcal{O}(n(n+m))$  time by performing a BFS on each vertex. Because the pairwise distances are known, it can be checked in constant time for a vertex pair  $(x, y)$  and a vertex  $w$  if  $\max\{d(x, w), d(y, w)\} > k$  and  $M(w, x) \neq M(w, y)$ . Therefore, the algorithm runs in total  $\mathcal{O}(n^3)$  time.  $\square$

By performing a binary search over  $k$ , we get the following result:

**Corollary 5.1.** *There is a  $\mathcal{O}(n^3 \log n)$  time algorithm that computes a  $k$ -dominating pair with minimum  $k$  of a given graph.*

In [69], KRATSCHE and SPINRAD show that finding a dominating pair is essentially as hard as finding a triangle in a graph. Thus, it is unlikely that there is a linear time algorithm to find a dominating pair if it exists. Yet, by Lemma 5.1, one can search for  $k$ -dominating pairs in dependence of the path-length of a graph. We do not know how to find a  $k$ -dominating pair with  $k \leq \text{pl}(G)$  for an arbitrary graph  $G$  in linear time. However, we can prove the following weaker result which is useful in later sections. For this, recall that a *spread pair* is a vertex pair  $(x, y)$  such that, for some vertex  $s$ ,  $d(s, x) = \text{ecc}(s)$  and  $d(x, y) = \text{ecc}(x)$ .

**Theorem 5.2.** *In a graph  $G$ , any spread pair is a  $2 \text{pl}(G)$ -dominating pair.*

*Proof.* Consider a path-decomposition  $\mathcal{P} = \{X_1, X_2, \dots, X_q\}$  of  $G$  with length  $\text{pl}(G) = \lambda$ . Let  $(x, y)$  be a spread pair and let  $s$  be a vertex of  $G$  such that  $d(s, x) = \text{ecc}(s)$ . We claim that  $(x, y)$  is a  $2\lambda$ -dominating pair of  $G$ .

If there is a bag in  $\mathcal{P}$  containing both  $s$  and  $x$ , then  $d(s, x) \leq \lambda$  and, by the choice of  $x$ , each vertex of  $G$  is within distance at most  $\lambda$  from  $s$  and, hence, within distance at most  $2\lambda$  from  $x$ . Evidently, in this case,  $(x, y)$  is a  $2\lambda$ -dominating pair of  $G$ .

Assume now, without loss of generality, that  $x \in X_i$  and  $s \in X_l$  with  $i < l$ . Consider an arbitrary vertex  $v$  of  $G$  that belongs to only bags with indices smaller than  $i$ . We show that  $d(x, v) \leq 2\lambda$ . As  $X_i$  separates  $v$  from  $s$ , a shortest path  $P$  of  $G$  between  $s$  and  $v$  must have a vertex  $u$  in  $X_i$ . We have  $d(s, x) \geq d(s, v) = d(s, u) + d(u, v)$  and, by the triangle inequality,  $d(s, x) \leq d(s, u) + d(u, x)$ . Hence,  $d(u, v) \leq d(u, x)$  and, since both  $u$  and  $x$  belong to same bag  $X_i$ ,  $d(u, x) \leq \lambda$ . That is,  $d(x, v) \leq d(x, u) + d(u, v) \leq 2d(u, x) \leq 2\lambda$ .

If  $d(x, y) \leq 2\lambda$  then, by the choice of  $y$ , each vertex of  $G$  is within distance at most  $2\lambda$  from  $x$  and, hence,  $(x, y)$  is a  $2\lambda$ -dominating pair of  $G$ . So, assume that  $d(x, y) > 2\lambda$ , i. e., every bag of  $\mathcal{P}$  that contains  $y$  has an index greater than  $i$ . Consider a bag  $X_j$  containing  $y$ . We have  $i < j$ . Repeating the arguments of the previous paragraph, we can show that  $d(y, v) \leq 2\lambda$  for every vertex  $v$  that belongs to bags with indices greater than  $j$ .

Consider now an arbitrary path  $P$  of  $G$  connecting vertices  $x$  and  $y$ . By properties of path-decompositions,  $P$  has a vertex in every bag  $X_h$  of  $\mathcal{P}$  with  $i \leq h \leq j$ . Hence, for each vertex  $v$  of  $G$  that belongs to a bag  $X_h$  with  $i \leq h \leq j$ , there is a vertex  $u \in P \cap X_h$  such that  $d(v, u) \leq \lambda$ . As  $d(v, x) \leq 2\lambda$  for each vertex  $v$  from  $X_{i'}$  with  $i' < i$  and  $d(v, y) \leq 2\lambda$  for each vertex  $v$  from  $X_{j'}$  with  $j' > j$ , we conclude that  $P$  has eccentricity at most  $2\lambda$  in  $G$ .  $\square$

### 5.3 Bandwidth of Graphs with Bounded Path-Length

In this section, we show that there is an efficient algorithm which, for any graph  $G$  with  $\text{pl}(G) = \lambda$ , produces a layout  $f$  with bandwidth at most  $\mathcal{O}(\lambda) \text{bw}(G)$ . Moreover, this statement is true even for all graphs containing a shortest path with eccentricity  $\lambda$ . Recall that a shortest path  $P$  of a graph  $G$  has *eccentricity*  $k$  in  $G$  if every vertex  $v$  of  $G$  is at distance at most  $k$  from a vertex of  $P$ , i. e.,  $d(v, P) \leq k$ .

We need the following “local density” lemma.

**Lemma 5.3 (Räcke [82]).** *For each vertex  $v$  of an arbitrary graph  $G$  and each positive integer  $r$ ,*

$$\frac{|N^r[v]| - 1}{2r} \leq \text{bw}(G).$$

The main result of this section is the following.

**Theorem 5.3.** *For a given graph  $G$  and a given shortest path with eccentricity  $k$  in  $G$ , a layout  $f$  with bandwidth at most  $(4k + 2) \text{bw}(G)$  can be found in linear time.*

*Proof.* Let  $P = \{x_0, x_1, \dots, x_i, \dots, x_j, \dots, x_q\}$  be a given shortest path with eccentricity  $k$  in  $G = (V, E)$ . Based on  $P$ , determine a partition  $\mathcal{X} = \{X_1, X_2, \dots\}$  of  $V$  such that each subset  $X_i$  induces a connected subgraph and  $v \in X_i$  implies  $d(v, x_i) = d(v, P)$ . This can be done by running a single BFS starting at  $P$ . Then, a vertex  $v$  is in the subset  $X_i$  if  $x_i$  is an ancestor of  $v$  in the resulting BFS-tree. Now, create a layout  $f$  of  $G$  by placing all the vertices of  $X_i$  before all vertices of  $X_j$ , if  $i < j$ , and by placing the vertices within each  $X_i$  in an arbitrary order. See Figure 5.1 for an illustration.

Clearly, computing  $\mathcal{X}$  and  $f$  can be done in linear time if  $P$  is given.

We claim that this layout  $f$  has bandwidth at most  $(4k + 2) \text{bw}(G)$ . Consider any edge  $uv$  of  $G$  and assume  $u \in X_i$  and  $v \in X_j$  with  $i \leq j$ . For this edge  $uv$ , we have  $f(v) - f(u) \leq \left| \bigcup_{\ell=i}^j X_\ell \right| - 1$ . Since  $P$  is a shortest path with eccentricity  $k$ , we also know that  $d(x_i, x_j) = j - i \leq d(x_i, u) + 1 + d(x_j, v) \leq 2k + 1$ . Consider a vertex  $x_c$  of  $P$  with  $c = \lfloor (i + j)/2 \rfloor$ , i. e., a middle vertex of the subpath of  $P$  between  $x_i$  and  $x_j$ . Consider an arbitrary vertex  $w$  in  $X_\ell$  for  $i \leq \ell \leq j$ . We know that, by triangle inequality,  $d(x_c, w) \leq d(x_c, x_\ell) + d(x_\ell, w)$ , by definition of  $x_c$ ,  $d(x_c, x_\ell) \leq \lceil 2k + 1 \rceil / 2$ , and, because  $\text{ecc}(P) \leq k$ ,  $d(x_\ell, w) \leq k$ . Thus, we get  $d(x_c, w) \leq 2k + 1$ . In other words, for  $r := 2k + 1$ , the disk  $N^r[x_c]$  contains all vertices of  $\bigcup_{\ell=i}^j X_\ell$  and, hence,  $|N^r[x_c]| \geq \left| \bigcup_{\ell=i}^j X_\ell \right|$ . Applying



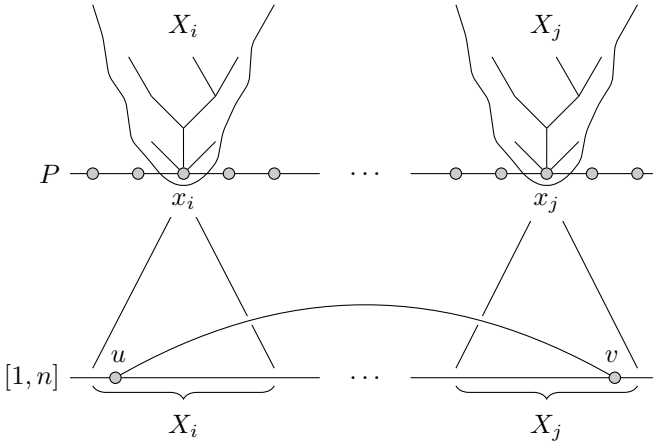


Figure 5.1. Illustration to the proof of Theorem 5.3.

Lemma 5.3, we conclude  $f(v) - f(u) \leq \left| \bigcup_{\ell=i}^j X_\ell \right| - 1 \leq |N^r[x_c]| - 1 \leq 2(2k + 1) \text{bw}(G) = (4k + 2) \text{bw}(G)$ .  $\square$

For a given graph  $G$ , one can find a shortest path with eccentricity  $k \leq \text{pl}(G)$  in  $\mathcal{O}(n^2m)$  time in the following way. Iterate over all vertex pairs of  $G$ . For each vertex pair  $(x, y)$ , pick a shortest  $(x, y)$ -path  $P$  and determine the eccentricity of  $P$ . Finally, output the path  $P$  for which  $\text{ecc}(P)$  is minimal. By Lemma 5.1, this minimum is at most  $\text{pl}(G)$ .

Alternatively, using Theorem 5.2, one can find a shortest path with eccentricity at most  $2 \text{pl}(G)$  in linear time. Therefore, Theorem 5.3 implies:

**Corollary 5.2.** *For every graph  $G$ , a layout with bandwidth at most  $(4 \text{pl}(G) + 2) \text{bw}(G)$  can be found in  $\mathcal{O}(n^2m)$  time and a layout with bandwidth at most  $(8 \text{pl}(G) + 2) \text{bw}(G)$  can be found in  $\mathcal{O}(n + m)$  time.*

The above results did not require a path-decomposition of length  $\text{pl}(G)$  of a graph  $G$  as input. We also avoided the construction of such a path-decomposition of  $G$  and just relied on the existence of a shortest path in  $G$  with eccentricity  $k$ . If, however, a path-decomposition with length  $\lambda$  of a graph  $G$  is given in advance together with  $G$ , then a better approximation ratio for the minimum bandwidth problem on  $G$  can be achieved.

**Theorem 5.4.** *If a graph  $G$  is given together with a path-decomposition of  $G$  of length  $\lambda$ , then a layout  $f$  with bandwidth at most  $\lambda \text{bw}(G)$  can be found in  $\mathcal{O}(n^2 + n \log^2 n)$  time.*

*Proof.* Let  $\mathcal{P} = \{X_1, X_2, \dots, X_q\}$  be a path-decomposition of length  $\lambda$  of  $G = (V, E)$ . We form a new graph  $G^+ = (V, E^+)$  from  $G$  by adding an edge between a pair of vertices  $u, v \in V$  if and only if  $u$  and  $v$  belong to a common bag in  $\mathcal{P}$ . From this construction, we conclude that  $G$  is a subgraph of  $G^+$  and  $G^+$  is a subgraph of  $G^\lambda$ . Note that  $G^+$  is an interval graph:  $\mathcal{P}$  gives a path-decomposition of  $G^+$  such that each

bag  $X_i$  is a clique of  $G^+$ . In [87], an  $\mathcal{O}(n \log^2 n)$  time algorithm to compute a minimum bandwidth layout of a graph is given. Let  $f$  be an optimal layout produced by that algorithm for our interval graph  $G^+$ . We claim that this layout  $f$ , when considered for  $G$ , has bandwidth at most  $\lambda \text{bw}(G)$ . Indeed, following [67], we have  $\max_{uv \in E} |f(u) - f(v)| \leq \max_{uv \in E^+} |f(u) - f(v)| = \text{bw}(G^+) \leq \text{bw}(G^\lambda) \leq \lambda \text{bw}(G)$ . Clearly, raising a graph to the  $\lambda$ -th power can only increase its bandwidth by a factor of  $\lambda$ .  $\square$

As shown in Lemma 3.2 (page 11), one can compute, for a given graph  $G$ , a path-decomposition with length at most  $2 \text{pl}(G)$  in  $\mathcal{O}(n^3)$  time. Combining this with Theorem 5.4, it follows:

**Corollary 5.3.** *For a given graph  $G$ , a layout  $f$  with bandwidth at most  $2 \text{pl}(G) \text{bw}(G)$  can be found in  $\mathcal{O}(n^3)$  time.*

Summarising the results of this section, we have the following interesting conclusion.

**Theorem 5.5.** *For every class of graphs with path-length bounded by a constant, there is an efficient constant-factor approximation algorithm for the minimum bandwidth problem.*

In Section 5.5, using some additional structural properties of AT-free graphs, we give a linear time 4-approximation algorithm for the minimum bandwidth problem for AT-free graphs. This result reproduces an approximation result by KLOKS et al. [67] with a better runtime.

## 5.4 Path-Length and Line-Distortion

In this section, we first show that the line-distortion of a graph gives an upper bound on its path-length and then demonstrate that, if the path-length of a graph  $G$  is bounded by a constant, there is an efficient constant-factor approximation algorithm for the minimum line-distortion problem on  $G$ .

### 5.4.1 Bound on Line-Distortion Implies Bound on Path-Length

In this subsection, we show that the path-length of an arbitrary graph never exceeds its line-distortion. The following inequalities are true.

**Theorem 5.6.** *For an arbitrary graph  $G$ ,  $\text{pl}(G) \leq \text{ld}(G)$ ,  $\text{pw}(G) \leq \text{ld}(G)$ , and  $\text{pb}(G) \leq \lceil \text{ld}(G)/2 \rceil$ .*

*Proof.* It is known (see, e. g., [62]) that every connected graph  $G$  has a minimum distortion embedding  $f$  into a line  $\ell$  (called a *canonic* embedding) such that  $|f(x) - f(y)| = d(x, y)$  for every two vertices  $x$  and  $y$  of  $G$  that are placed next to each other in  $\ell$  by  $f$ . Assume, in what follows, that  $f$  is such a canonic embedding and let  $k := \text{ld}(G)$ .

Consider the following path-decomposition of  $G$  created from  $f$ . For each vertex  $v$ , form a bag  $B_v$  consisting of all vertices of  $G$  which are placed by  $f$  in the interval  $[f(v), f(v) + k]$  of a line  $\ell$ . Order these bags with respect to the left ends of the corresponding intervals. Evidently, for every vertex  $v \in V$ ,  $v \in B_v$ , i. e., each vertex belongs to a bag. More generally, a vertex  $u$  belongs to a bag  $B_v$  if and only if  $f(v) \leq f(u) \leq f(v) + k$ . Since  $\text{ld}(G) = k$ , for every edge  $uv$  of  $G$ ,  $|f(u) - f(v)| \leq k$  holds. Hence, both ends of the edge  $uv$  belong either to the bag  $B_u$  (if  $f(u) < f(v)$ ) or to the bag  $B_v$  (if  $f(v) < f(u)$ ). Now, consider three bags  $B_a$ ,  $B_b$ , and  $B_c$  with  $f(a) < f(b) < f(c)$  and a vertex  $v$  of  $G$  that belongs to  $B_a$  and  $B_c$ . We have  $f(a) < f(b) < f(c) \leq f(v) \leq f(a) + k < f(b) + k$ . Hence,  $v$  belongs to  $B_b$  as well.

It remains to show that each bag  $B_v$ ,  $v \in V$ , has in  $G$  diameter at most  $k$ , radius at most  $\lceil k/2 \rceil$ , and cardinality at most  $k + 1$ . Indeed, for any two vertices  $x, y \in B_v$ , we have  $|f(x) - f(y)| \leq k$ , i. e.,  $d(x, y) \leq |f(x) - f(y)| \leq k$ . Furthermore, any interval  $[f(v), f(v) + k]$  of length  $k$  can have at most  $k + 1$  vertices of  $G$  as the distance between any two vertices placed by  $f$  to this interval is at least 1 ( $|f(x) - f(y)| \geq d(x, y) \geq 1$ ). Thus,  $|B_v| \leq k + 1$  for every  $v \in V$ .

Now, consider the point  $p_v := f(v) + \lceil k/2 \rceil$  in the interval  $[f(v), f(v) + k]$  of  $\ell$ . Assume, without loss of generality, that  $p_v$  is between  $f(x)$  and  $f(y)$  which are the images of two vertices  $x$  and  $y$  of  $G$  placed next to each other in  $\ell$  by  $f$ . Let  $f(x) \leq p_v < f(y)$ . See Figure 5.2 for an illustration. Since  $f$  is a canonic embedding of  $G$ , there must exist a vertex  $c$  on a shortest path between  $x$  and  $y$  such that  $d(x, c) = p_v - f(x)$  and  $d(c, y) = f(y) - p_v = d(x, y) - d(x, c)$ . We claim that, for every vertex  $w \in B_v$ ,  $d(c, w) \leq \lceil k/2 \rceil$  holds. Assume  $f(w) \geq f(y)$  (the case when  $f(w) \leq f(x)$  is similar). Then, we have  $d(c, w) \leq d(c, y) + d(y, w) \leq (f(y) - p_v) + (f(w) - f(y)) = f(w) - p_v = f(w) - f(v) - \lceil k/2 \rceil \leq k - \lceil k/2 \rceil \leq \lceil k/2 \rceil$ .  $\square$

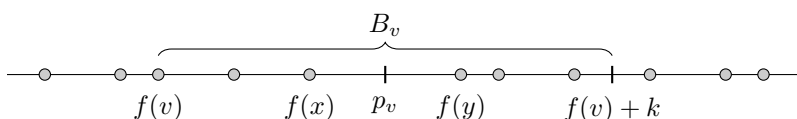


Figure 5.2. Illustration to the proof of Theorem 5.6.

It should be noted that the difference between the path-length and the line-distortion of a graph can be very large. The graph  $K_n$  has path-length 1, whereas the line-distortion of  $K_n$  is  $n - 1$ . Note also that the bandwidth and the path-length of a graph do not bound each other. The bandwidth of  $K_n$  is  $n - 1$  while its path-length is 1. On the other hand, the path-length of cycle  $C_{2n}$  is  $n$  while its bandwidth is 2.

## 5.4.2 Line-Distortion of Graphs with Bounded Path-Length

In this subsection, we show that there is an efficient algorithm that, for any graph  $G$  with  $\text{pl}(G) \leq \lambda$ , produces an embedding  $f$  of  $G$  into a line with distortion at most  $(8\lambda + 2)\text{ld}(G)$ . This statement is true even for all graphs with a shortest path with eccentricity  $\lambda$ .

We need the following simple “local density” lemma.

**Lemma 5.4.** *For every vertex set  $S \subseteq V$  of an arbitrary graph  $G = (V, E)$ ,*

$$|S| - 1 \leq \text{diam}(S) \text{ld}(G).$$

*Proof.* Consider an embedding  $f^*$  of  $G$  into a line  $\ell$  with distortion  $\text{ld}(G)$ . Let  $a$  and  $b$  be the leftmost and the rightmost, respectively, vertices of  $S$  in  $\ell$ , i. e.,  $f^*(a) \leq f^*(v) \leq f^*(b)$  for all  $v \in S$ . Consider a shortest path  $P$  in  $G$  from  $a$  to  $b$ . Since, for each edge  $xy$  of  $G$ ,  $|f^*(x) - f^*(y)| \leq \text{ld}(G)$  holds, we get  $f^*(b) - f^*(a) \leq d(a, b) \text{ld}(G) \leq \text{diam}(S) \text{ld}(G)$ . On the other hand, since all vertices of  $S$  are mapped to points of  $\ell$  between  $f^*(a)$  and  $f^*(b)$ , we have  $f^*(b) - f^*(a) \geq |S| - 1$ .  $\square$

The main result of this section is the following.

**Theorem 5.7.** *Every graph  $G$  with a shortest path of eccentricity  $k$  admits an embedding  $f$  of  $G$  into a line with distortion at most  $(8k + 2) \text{ld}(G)$ . If a shortest path of  $G$  of eccentricity  $k$  is given in advance, then such an embedding  $f$  can be found in linear time.*

*Proof.* Let  $P = \{x_0, x_1, \dots, x_i, \dots, x_j, \dots, x_q\}$  be a shortest path of  $G$  of eccentricity  $k$ . Build a BFS( $P, G$ )-tree  $T$  of  $G$  (i. e., a Breadth-First-Search tree of  $G$  started at path  $P$ ). Denote by  $\{X_0, X_1, \dots, X_q\}$  the decomposition of the vertex set  $V$  of  $G$  obtained from  $T$  by removing the edges of  $P$ . That is,  $X_i$  is the vertex set of a subtree (branch) of  $T$  growing from vertex  $x_i$  of  $P$ . See Figure 5.3a for an illustration. Since the eccentricity of  $P$  is  $k$ , we have  $d_G(v, x_i) \leq k$  for every  $i \in \{1, \dots, q\}$  and every  $v \in X_i$ .

We define an embedding  $f$  of  $G$  into a line  $\ell$  by performing a preorder traversal of the vertices of  $T$  starting at vertex  $x_0$  and visiting first the vertices of  $X_i$  and then the vertices of  $X_{i+1}$  for each  $i \in \{0, \dots, q - 1\}$ . We place the vertices of  $G$  on the line  $\ell$  in that order, and also, for each  $i \in \{0, \dots, q - 1\}$ , we leave a space of length  $d_T(v_i, v_{i+1})$  between any two vertices  $v_i$  and  $v_{i+1}$  placed next to each other (this can be done during the preorder traversal). Alternatively,  $f$  can be defined by creating a twice around tour of the tree  $T$ , which visits vertices of  $X_i$  prior to vertices of  $X_{i+1}$ ,  $i = 0, \dots, q - 1$ , and then returns to  $x_0$  from  $x_q$  along edges of  $P$ . Following vertices of  $T$  from  $x_0$  to  $x_q$  as shown in Figure 5.3b (i. e., using upper part of the twice around tour),  $f(v)$  can be defined as the first appearance of vertex  $v$  in that subtour.

We claim that  $f$  is a (non-contractive) embedding with distortion at most  $(8k+2) \text{ld}(G)$ . It is sufficient to show that  $d_G(x, y) \leq |f(x) - f(y)|$  for every two vertices of  $G$  that are placed by  $f$  next to each other in  $\ell$  and that  $|f(v) - f(u)| \leq (8k + 2) \text{ld}(G)$  for every edge  $uv$  of  $G$  (see, e. g., [18, 62]).

Let  $x$  and  $y$  be two arbitrary vertices of  $G$  that are placed by  $f$  next to each other in  $\ell$ . By construction, we know that  $|f(x) - f(y)| = d_T(x, y)$ . Since  $d_G(x, y) \leq d_T(x, y)$ , we get also  $d_G(x, y) \leq |f(x) - f(y)|$ , i. e.,  $f$  is non-contractive.

Consider now an arbitrary edge  $uv$  of  $G$  and assume  $u \in X_i$  and  $v \in X_j$  ( $i \leq j$ ). Note that  $d_P(x_i, x_j) = j - i \leq 2k + 1$ , since  $P$  is a shortest path of  $G$  and  $d_P(x_i, x_j) =$

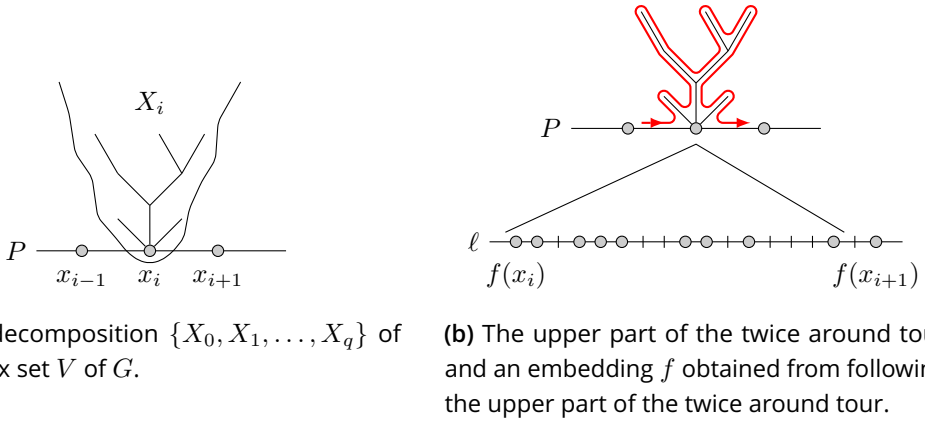


Figure 5.3. Illustration to the proof of Theorem 5.7.

$d_G(x_i, x_j) \leq d_G(x_i, u) + 1 + d_G(x_j, v) \leq 2k + 1$ . Set  $S = \bigcup_{h=i}^j X_h$ . For any two vertices  $x, y \in S$ ,  $d_G(x, y) \leq d_G(x, P) + 2k + 1 + d_G(y, P) \leq k + 2k + 1 + k = 4k + 1$  holds. Hence,  $\text{diam}_G(S) \leq 4k + 1$ . Consider subtree  $T_S$  of  $T$  induced by  $S$ . Clearly,  $T_S$  is connected and has  $|S| - 1$  edges. Therefore,  $f(v) - f(u) \leq 2(|S| - 1)$  since each edge of  $T_S$  contributes to  $f(v) - f(u)$  at most 2 units. Now, by Lemma 5.4,  $f(v) - f(u) \leq 2(|S| - 1) \leq 2 \text{diam}_G(S) \text{ld}(G) \leq (8k + 2) \text{ld}(G)$ .  $\square$

Recall that, by Lemma 5.1, each graph  $G$  with  $\text{pl}(G) \leq \lambda$  has a  $\lambda$ -dominating pair and, hence, a shortest path with eccentricity  $\lambda$ . Such a path can easily be found in  $\mathcal{O}(n^2m)$  time by iterating over all vertex pairs. Additionally, by Theorem 5.2, one can find a shortest path with eccentricity at most  $2\lambda$  in linear time. Thus, Theorem 5.7 implies:

**Corollary 5.4.** *For a given graph  $G$ , one can compute an embedding of  $G$  into a line with distortion at most  $(8 \text{pl}(G) + 2) \text{ld}(G)$  in  $\mathcal{O}(n^2m)$  time and with distortion at most  $(16 \text{pl}(G) + 2) \text{ld}(G)$  in  $\mathcal{O}(n + m)$  time.*

Thus, we have the following interesting conclusion.

**Theorem 5.8.** *For every class of graphs with path-length bounded by a constant, there is an efficient constant-factor approximation algorithm for the minimum line-distortion problem.*

Using the inequality  $\text{pl}(G) \leq \text{ld}(G)$  in Corollary 5.4 once more, we reproduce a result of [18].

**Corollary 5.5 (Bădoiu et al. [18]).** *For every graph  $G$  with  $\text{ld}(G) = c$ , an embedding into a line with distortion at most  $\mathcal{O}(c^2)$  can be found in polynomial time.*

It should be noted that, since the difference between the minimum eccentricity of a shortest path and the line-distortion of a graph can be very large (close to  $n$ ), the result in Theorem 5.7 seems to be stronger. In Chapter 6 (page 68), we investigate the problem of finding a shortest path with minimum eccentricity in a given graph.

## 5.5 Approximation for AT-Free Graphs

From Theorem 5.8 and results from Section 3.3 (page 25), it follows already that there is an efficient constant-factor approximation algorithm for the minimum line-distortion problem on such particular graph classes as permutation graphs, trapezoid graphs, cocomparability graphs as well as AT-free graphs. Recall that, for arbitrary graphs, the minimum line-distortion problem is hard to approximate within a constant factor [18]. Furthermore, the problem remains NP-complete even when the input graph is restricted to a chordal, cocomparability, or AT-free graph [61]. Polynomial-time constant-factor approximation algorithms were known only for split and cocomparability graphs; HEGGERNES and MEISTER [61] gave efficient 6-approximation algorithms for both graph classes. As far as we know, for AT-free graphs (the class which contains all cocomparability graphs), no prior efficient approximation algorithm was known.

In this section, we give a better approximation algorithm for all AT-free graphs using additional structural properties of AT-free graphs; more precisely, we give a 6-approximation algorithm that runs in linear time.

**Theorem 5.9.** *There is a linear time algorithm to compute an embedding with distortion at most  $6 \text{ld}(G)$  for a given AT-free graph  $G = (V, E)$ .*

*Proof.* Let  $s$  be an arbitrary vertex of  $G$ , let  $v$  be the vertex last visited (numbered 1) by a LexBFS starting at  $s$ , and let  $w$  be the vertex last visited by a LexBFS starting at  $v$ . One can compute, in linear time, a shortest path  $P = \{v = v_0, v_1, \dots, v_k = w\}$  from  $v$  to  $w$  such that, for all  $u \in L_i^{(v)}$  with  $i \geq 1$ ,  $uv_i \in E$  or  $uv_{i-1} \in E$  [67]. Based on  $P$ , we partition every layer  $L_i^{(v)}$  in three sets:  $\{v_i\}$ ,  $X_i = \{x \mid x \in L_i^{(v)}, v_i x \in E\}$ , and  $\bar{X}_i = L_i^{(v)} \setminus (\{v_i\} \cup X_i)$ . See Figure 5.4 for an illustration.

Note that, due to Lemma 2.12 (page 9) the diameter of each layer  $L_i^{(v)}$  is at most 2, i. e., for each  $x, y \in L_i^{(v)}$ ,  $d(x, y) \leq 2$ . The embedding  $f$  places vertices of  $G$  into a line  $\ell$  in the following order:  $\langle v_0, \dots, v_{i-1}, \bar{X}_i, X_i, v_i, \bar{X}_{i+1}, X_{i+1}, v_{i+1}, \dots, v_k \rangle$ . Between every two vertices  $x$  and  $y$  placed next to each other on the line  $\ell$ , to guarantee non-contractiveness,  $f$  leaves a space of length  $d(x, y)$  (which is either 1, or 2).

Now, we show that  $f$  approximates the minimum line-distortion of  $G$ . Since a layer  $L_i^{(v)}$  only contains vertices with distance  $i$  to  $v$ , there is no edge  $xy$  with  $x \in L_{i-1}^{(v)}$  and  $y \in L_{i+1}^{(v)}$ . Therefore, for all  $xy \in E$  with  $x, y \in L_i^{(v)} \cup L_{i+1}^{(v)}$ ,  $|f(x) - f(y)| < |f(v_{i-1}) - f(v_{i+1})|$ . Let  $S = \{v_{i-1}, v_i, v_{i+1}\} \cup \bar{X}_i \cup X_i \cup \bar{X}_{i+1} \cup X_{i+1}$ . Then, counting how many vertices are placed by  $f$  between  $f(v_{i-1})$  and  $f(v_{i+1})$  and the distance in  $G$  between vertices placed next to each other, we get  $|f(x) - f(y)| \leq 2|S| - 1$ .

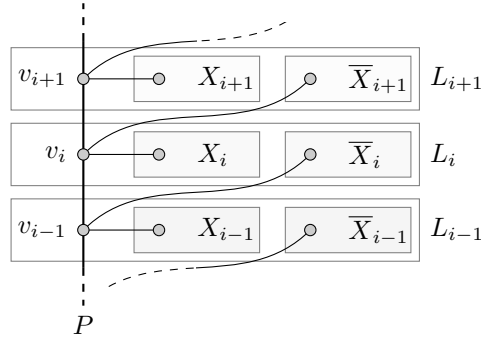


Figure 5.4. Illustration to the proof of Theorem 5.9. Layering of an AT-free graph.

*Claim 1.*  $\text{diam}(S) \leq 3$

*Proof (Claim).* Note that, by definition of  $S$ ,  $S = L_{i+1}^{(v)} \cup L_i^{(v)} \cup \{v_{i-1}\}$ . Due to Lemma 2.12 (page 9), the diameter of  $L_{i+1}^{(v)}$  and  $L_i^{(v)}$  is at most 2. Also, each vertex  $x \in L_{i+1}^{(v)}$  is adjacent to a vertex  $y \in L_i^{(v)}$ . Therefore, the diameter of  $L_{i+1}^{(v)} \cup L_i^{(v)}$  is at most 3. It remains to show that, for each  $x \in \bar{X}_{i+1} \cup X_{i+1}$ ,  $d(v_{i-1}, x) \leq 3$ . If  $x \in X_{i+1}$ , then there is a path  $\{x, v_{i+1}, v_i, v_{i-1}\}$ . If  $x \in \bar{X}_{i+1}$ , then there is a path  $\{x, v_i, v_{i-1}\}$ .  $\diamond$

From Claim 1 and Lemma 5.4, it follows that  $|f(x) - f(y)| \leq 2|S| - 1 \leq 6 \text{ld}(G)$  for all  $xy \in E$ .  $\square$

Algorithm 5.2 formalises the method described above.

---

**Algorithm 5.2:** A 6-approximation algorithm for the minimum line-distortion of an AT-free graph.

---

**Input:** An AT-free graph  $G = (V, E)$ .

**Output:** An embedding  $f$  of  $G$  into a line.

- 1 Compute the distance layers  $L_i^v$  a path  $P = \{v_0, \dots, v_k\}$  such that, for all  $u \in L_i^v$  with  $i \geq 1$ ,  $uv_i \in E$  or  $uv_{i-1} \in E$  (see [67]).
  - 2 Partition each layer  $L_i$  into three sets:  $\{v_i\}$ ,  $X_i = \{x \mid x \in L_i, v_i x \in E\}$ , and  $\bar{X}_i = L_i \setminus (\{v_i\} \cup X_i)$ .
  - 3 Create an embedding  $f$  by placing the vertices of  $G$  into a line  $\ell$  in the order  $\langle v_0, \dots, v_{i-1}, \bar{X}_i, X_i, v_i, \bar{X}_{i+1}, X_{i+1}, v_{i+1}, \dots, v_k \rangle$ .
  - 4 Between every two consecutive vertices  $x$  and  $y$  on the line  $\ell$ , leave a space of length  $d(x, y)$ .
  - 5 Output  $f$ .
- 

Note that  $S \subseteq N^2[v_i]$ . Therefore, it follows from Lemma 5.3 that the order in which the vertices of  $G$  are placed by  $f$  into the line  $\ell$  gives also a layout of  $G$  with bandwidth at most  $4 \text{bw}(G)$ . This reproduces an approximation result by KLOKS et al. [67]. Their

algorithm has complexity  $\mathcal{O}(m + n \log n)$  for a graph, since it involves an  $\mathcal{O}(n \log n)$  time algorithm by ASSMANN et al. [4] to find an optimal layout for a caterpillar with hair-length at most 1.

**Corollary 5.6 (Kloks et al. [67]).** *There is a linear time algorithm to compute a 4-approximation of the minimum bandwidth of an AT-free graph.*

Combining Theorem 3.13 (page 26) and Theorem 5.4, we also obtain the following result by KLOKS et al. [67] as a corollary.

**Corollary 5.7 (Kloks et al. [67]).** *There is an  $\mathcal{O}(m + n \log^2 n)$  time algorithm to compute a 2-approximation of the minimum bandwidth of an AT-free graph.*



# The Minimum Eccentricity Shortest Path Problem\*

In Section 5.3 (page 59) and Section 5.4 (page 61), we use the path-length of a graph to obtain a shortest path  $P$  with eccentricity  $\lambda$  and then use  $P$  to calculate approximations for the Bandwidth and Line-Distortion problems. However, after determining  $P$ , we do not use the bounded path-length again. This leads to the question if we can directly determine such a path. We call this the *Minimum Eccentricity Shortest Path* problem, *MESP* for short.

**Definition 6.1.** *For a given a graph  $G$ , the Minimum Eccentricity Shortest Path problem asks to find a shortest path  $P$  such that, for each shortest path  $Q$ ,  $\text{ecc}(P) \leq \text{ecc}(Q)$ .*

Note that our Minimum Eccentricity Shortest Path problem is close but different from the *Central Path* problem in graphs introduced in [85]. It asks, for a given graph  $G$ , to find a not necessarily shortest path  $P$  such that any other path of  $G$  has eccentricity at least  $\text{ecc}(P)$ . The Central Path problem generalizes the Hamiltonian Path problem and, therefore, is NP-complete even for chordal graphs [76]. Our problem, however, is polynomial time solvable for chordal graphs (see Corollary 6.7).

In this chapter, we investigate the Minimum Eccentricity Shortest Path problem. We analyse the hardness of the problem, show algorithms to compute an optimal solution, and present approximation algorithms differing in quality and runtime. This is done for general graphs as well as for special graph classes. Additionally, we show that, if a shortest path with eccentricity  $k$  is given, a  $k$ -dominating set can be found in pseudo-polynomial time.

## 6.1 Hardness

In this section, we show that finding a minimum eccentricity shortest path is NP-hard, even if restricted to planar bipartite graphs with maximum vertex-degree 3. Additionally, we show that the problem is W[2]-hard for sparse graphs. To do that, we define the decision version of the Minimum Eccentricity Shortest Path problem, named  $k$ -ESP, as follows: Given a graph  $G$  and an integer  $k$ , does  $G$  contain a shortest path  $P$  with eccentricity at most  $k$ ?

---

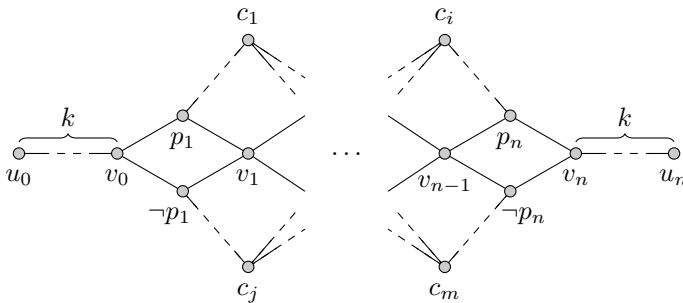
\* Results from this chapter have been published partially at WADS 2015, Victoria, Canada [41], at WG 2015, Munich, Germany [40], and in the *Journal of Graph Algorithms and Applications* [42].

**Theorem 6.1.** *The decision version of the Minimum Eccentricity Shortest Path problem is NP-complete.*

*Proof.* To prove Theorem 6.1, we use a version of 3-SAT which is called *Planar Monotone 3-SAT*. It was introduced by DE BERG and KHOSRAVI in [8]. Consider an instance of 3-SAT given in CNF with the variables  $\mathcal{P} = \{p_1, \dots, p_n\}$  and the clauses  $\mathcal{C} = \{c_1, \dots, c_m\}$ . A clause is called *positive* if it consists only of positive variables (i. e.,  $p_a \vee p_b \vee p_c$ ) and is called *negative* if it consists only of negative variables (i. e.,  $\neg p_a \vee \neg p_b \vee \neg p_c$ ). Consider the bipartite graph  $\mathcal{G} = (\mathcal{P}, \mathcal{C}, \mathcal{E})$  where  $p_i c_j \in \mathcal{E}$  if and only if  $c_j$  contains  $p_i$  or  $\neg p_i$ . An instance of 3-SAT is *planar monotone* if each clause is either positive or negative and there is a planar embedding for  $\mathcal{G}$  such that all variables are on a (horizontal) line  $L$ , all positive clauses are above  $L$ , all negative clauses are below  $L$ , and no edge is crossing  $L$ . Planar Monotone 3-SAT is NP-complete [8].

Now, assume that we are given an instance  $\mathcal{I}$  of Planar Monotone 3-SAT with the variables  $\mathcal{P} = \{p_1, \dots, p_n\}$  and the clauses  $\mathcal{C} = \{c_1, \dots, c_m\}$ . Also, let  $k = \max\{n, m\}$ . We create a graph  $G$  as shown in Figure 6.1. For each variable  $p_i$  create two vertices, one representing  $p_i$  and one representing  $\neg p_i$ . Create one vertex  $c_i$  for every clause  $c_i$ . Additionally, create two vertices  $u_0, u_n$  and, for each  $i$  with  $0 \leq i \leq n$ , a vertex  $v_i$ . Connect each variable vertex  $p_i$  and  $\neg p_i$  with  $v_{i-1}$  and  $v_i$  directly with an edge. Connect each clause with the variables contained in it with a path of length  $k$ . Also connect  $v_0$  with  $u_0$  and  $v_n$  with  $u_n$  with a path of length  $k$ .

Recall that, by definition of  $\mathcal{I}$ , the corresponding bipartite graph  $\mathcal{G}$  has a planar embedding where all variables are on a line. Therefore, we can clearly achieve a planar embedding for  $G$  when placing its vertices as shown in Figure 6.1.



**Figure 6.1.** Reduction from Planar Monotone 3-SAT to  $k$ -ESP. Illustration to the proof of Theorem 6.1.

Note that every shortest path in  $G$  not containing  $v_0$  and  $v_n$  has an eccentricity larger than  $k$ . Also, a shortest path from  $v_0$  to  $v_n$  has length  $2n$  ( $d(v_{i-1}, v_i) = 2$ , passing through  $p_i$  or  $\neg p_i$ ). Since  $k \geq n$ , no shortest path from  $v_0$  to  $v_n$  is passing through a vertex  $c_i$ ; in this case the minimal length would be  $2k + 2$ . Additionally, note that, for all vertices in  $G$  except the vertices which represent clauses, the distance to a vertex  $v_i$  with  $0 \leq i \leq n$  is at most  $k$ .

We now show that  $\mathcal{I}$  is satisfiable if and only if  $G$  has a shortest path with eccentricity  $k$ .

First, assume  $\mathcal{I}$  is satisfiable. Let  $f: \mathcal{P} \rightarrow \{T, F\}$  be a satisfying assignment for the variables. As shortest path  $P$ , we choose a shortest path from  $v_0$  to  $v_n$ . Thus, we have to choose between  $p_i$  and  $\neg p_i$ . We chose  $p_i$  if and only if  $f(p_i) = T$ . Because  $\mathcal{I}$  is satisfiable, there is a  $p_i$  for each  $c_j$  such that either  $f(p_i) = T$  and  $d(c_j, p_i) = k$ , or  $f(p_i) = F$  and  $d(c_j, \neg p_i) = k$ . Thus,  $P$  has eccentricity  $k$ .

Next, consider a shortest path  $P$  in  $G$  of eccentricity  $k$ . As mentioned above,  $P$  contains either  $p_i$  or  $\neg p_i$ . Now, we define  $f: \mathcal{P} \rightarrow \{T, F\}$  as follows:

$$f(p_i) = \begin{cases} T & \text{if } p_i \in P, \\ F & \text{else, i. e. } \neg p_i \in P. \end{cases}$$

Because  $P$  has eccentricity  $k$  and only vertices representing a variable in the clause  $c_j$  are at distance  $k$  to vertex  $c_j$ ,  $f$  is a satisfying assignment for  $\mathcal{I}$ .  $\square$

While the reduction works in principle for any version of SAT (given as CNF), choosing Planar Monotone 3-SAT allows to construct a planar graph  $G$ .

Note that the created graph is bipartite. Set the colour of each vertex  $v_i$  to black and of each  $p_j$  and  $\neg p_j$  to white. For some vertex  $x$  on the shortest path from  $c_i$  to  $p_j$  (or  $\neg p_j$ ), set the colour of  $x$  based on its distance to  $p_j$  (or  $\neg p_j$ ), i. e.,  $x$  is white if  $d(x, p_i)$  is even and black otherwise.

Additionally, V. B. LE<sup>1</sup> pointed out that, by slightly modifying the created graph as follows, it can be shown that the problem remains NP-complete even if the graph has the maximum vertex-degree 3. First, increase  $k$  to  $k = \max\{2n - 1, m\}$  and update all distances in the graph accordingly. Therefore,  $2k + 2 > 4n - 2$ . Then, replace each vertex  $v_i$  where  $1 \leq i \leq n - 1$  with three vertices  $v_i^-$ ,  $v_i'$ , and  $v_i^+$  such that  $N(v_i^-) = \{p_i, \neg p_i, v_i'\}$ ,  $N(v_i') = \{v_i^-, v_i^+\}$ , and  $N(v_i^+) = \{p_{i+1}, \neg p_{i+1}, v_i'\}$ . Note that a path from  $v_0$  to  $v_n$  which, for all  $i$ , passes through  $p_i$  or  $\neg p_i$  has length  $4n - 2$ . This is still a shortest path because each path from  $v_0$  to  $v_n$  passing through some  $c_i$  has length  $2k + 2 > 4n - 2$ . Also, since  $d(p_i, p_{i+1}) = 4$ , the graph remains bipartite. Next, to limit the degree of a vertex  $p_i$  (or  $\neg p_i$ ), instead of connecting it directly to all clauses containing it, make  $p_i$  adjacent to the root of a binary tree  $T_i$  with height  $\lceil \log_2 k \rceil$ . Then, connect each clause containing  $p_i$  to a leaf of  $T_i$  using a path with length  $k - \lceil \log_2 k \rceil - 1$  and, last, remove unused branches of  $T_i$ . Because this does not effect planarity or colouring, we get:

**Corollary 6.1.** *The decision version of the MESP problem remains NP-complete if restricted to planar bipartite graphs with maximum vertex-degree 3.*

We can slightly modify the MESP problem such that a start vertex  $s$  and an end vertex  $t$  of the path are given. That is, for a given a graph  $G$  and two vertices  $s$  and  $t$ , find a shortest  $(s, t)$ -path  $P$  such that, for each shortest  $(s, t)$ -path  $Q$ ,  $\text{ecc}(P) \leq \text{ecc}(Q)$ . We call this the  $(s, t)$ -MESP problem. From the reduction above, it follows that the decision version of this problem is NP-complete, too.

<sup>1</sup>University of Rostock, Germany

**Corollary 6.2.** *The decision version of the  $(s, t)$ -MESP problem is NP-complete, even if restricted to planar bipartite graphs with maximum vertex-degree 3.*

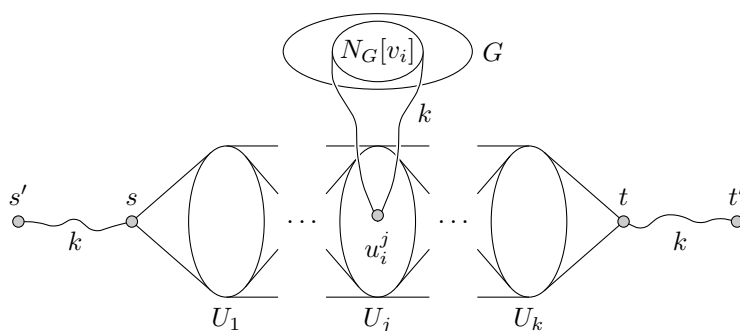
Note that the factor  $k$  in the reduction above depends on the input size. In [68], it was already mentioned that, for  $k = 1$ , the problem can be solved in  $\mathcal{O}(n^3m)$  time by modifying an algorithm given in [28]. There, the problem was called *Dominating Shortest Path* problem. Therefore, it is an interesting question how hard MESP is if  $k$  is bounded by a constant.

To answer this question, we show next that the problem is W[2]-hard in general and for sparse graphs. Therefore, we do not expect that MESP is Fixed Parameter Tractable, i. e., there is probably no algorithm that finds an optimal solution in  $f(k)n^{\mathcal{O}(1)}$  time. In Section 6.2, we generalise the result from [68] to show that MESP can be solved in pseudo-polynomial time.

**Theorem 6.2.** *The Minimum Eccentricity Shortest Path problem is W[2]-hard.*

*Proof.* To show W[2]-hardness, we make a parametrised reduction from the Dominating Set problem which is known to be W[2]-complete [34].

Consider a given graph  $G = (V, E)$  with  $V = \{v_1, v_2, \dots, v_n\}$  and a given  $k$ . Based on  $G$  and  $k$ , we construct a graph  $H$  (containing  $G$  as subgraph) as follows. Start with  $G$  and add  $k$  sets of vertices  $U_1, U_2, \dots, U_k$  with  $U_i = \{u_1^i, u_2^i, \dots, u_n^i\}$ . For each  $j$  with  $1 \leq j < k$ , make a join between  $U_j$  and  $U_{j+1}$ . Add the vertices  $s, s', t$ , and  $t'$  and connect  $s$  with  $s'$  and  $t$  with  $t'$ , respectively, with a path of length  $k$ . Additionally, make  $s$  adjacent to all vertices in  $U_1$  and make  $t$  adjacent to all vertices in  $U_k$ . Connect each vertex  $u_i^j \in U_j$  with each vertex in  $N_G[v_i]$  with a path of length  $k$  for all  $j$  with  $1 \leq j \leq k$ . Figure 6.2 gives an illustration.



**Figure 6.2.** Reduction from Dominating Set to  $k$ -ESP. Illustration to the graph  $H$  as constructed in the proof of Theorem 6.2.

Because  $d_H(s, s') = d_H(t, t') = k$ , each shortest path in  $H$  not containing  $s$  and  $t$  has an eccentricity larger than  $k$ . Also, a shortest path from  $s$  to  $t$  has length  $k + 1$ , intersects all sets  $U_j$ , and does not intersect  $V$ .

First, assume that  $H$  has a shortest path  $P$  with eccentricity  $k$ . By definition of  $P$  and construction of  $H$ , for every  $v \in V$ , there is a vertex  $u_i^j \in P$  such that  $d_H(v, u_i^j) = d_H(v, P) = k$  and, hence,  $v \in N_G[v_i]$ . Therefore, the set  $D = \left\{ v_i \in V \mid \text{there is a } j \text{ with } u_i^j \in P \right\}$  is a dominating set for  $G$  with cardinality at most  $k$ .

Next, assume there is a dominating set  $D$  for  $G$  with cardinality at most  $k$ . Without loss of generality, let  $D = \{v_1, v_2, \dots, v_k\}$ . Then, we define  $P = \{s, u_1^1, u_2^2, \dots, u_k^k, t\}$ . By construction of  $H$ , each vertex  $v \in N_G[v_i]$  is at distance  $k$  to  $u_i^i$ . Thus, because  $D$  is a dominating set, there is a vertex  $u_i^i \in P$  for each vertex  $v \in V$  with  $d_H(v, u_i^i) = k$ . Therefore,  $P$  has eccentricity  $k$  in  $H$ .  $\square$

Note that the graph  $H$  constructed in the reduction above has at most  $\mathcal{O}(kn^2)$  edges. Thus, we can transform  $H$  into a sparse graph  $H'$  by simply adding  $\Theta(kn^2)$  pendent vertices which are adjacent to  $s$ . Clearly,  $H'$  has a shortest path with eccentricity  $k$  if and only if  $H$  has a shortest path with eccentricity  $k$ . Thus, we get the following result.

**Corollary 6.3.** *The Minimum Eccentricity Shortest Path problem remains  $W[2]$ -hard when restricted to sparse graphs.*

Later, Corollary 6.5 shows that MESP is Fixed Parameter Tractable for graphs with bounded degree.

## 6.2 Computing an Optimal Solution

In this section, we investigate how to find an optimal solution for MESP. First, we present a pseudo-polynomial time algorithm to solve MESP on general graphs. Then, we analyse the problem for sparse graphs. Additionally, we present an approach to solve MESP for tree-structured graphs and show that, for some graph classes, the problem is solvable in polynomial time.

### 6.2.1 General Graphs

The next algorithm shows that the  $k$ -ESP problem remains polynomial for a fixed  $k$ . Our algorithm is a generalisation of the algorithm mentioned in [68]. It is based on Lemma 6.1 below. Informally, Lemma 6.1 states that, if a graph has a shortest path  $P$  with eccentricity  $k$  starting at  $s$ , each layer  $L_i^{(s)}$  is dominated by a subpath of  $P$  of length at most  $2k$ .

**Lemma 6.1.** *Let  $P = \{s = v_0, v_1, \dots, v_l\}$  be a shortest path with eccentricity  $k$ ,  $v_i \in L_i^{(s)}$ , and  $P_{i,k} = \{v_{\max\{0, i-k\}}, \dots, v_{\min\{i+k, l\}}\}$ . Then,  $L_i^{(s)} \subseteq N^k[P_{i,k}]$ .*

*Proof.* Assume there is a vertex  $u \in L_i^{(s)} \setminus N^k[P_{i,k}]$ . Consider any vertex  $v_j \in P \setminus P_{i,k}$ . By the definition of  $P_{i,k}$  it follows that  $|i - j| > k$ . Thus, because  $u \in L_i^{(s)}$  and  $v_j \in L_j^{(s)}$ ,  $d(v_j, u) \geq |i - j| > k$ . This contradicts with  $P$  having eccentricity  $k$ .  $\square$

For Algorithm 6.1 below, we say a shortest path  $\tau = \{v_{i-k}, \dots, v_j\}$  with  $i \leq j \leq i+k$  is a *layer-dominating path for a layer*  $L_i^{(s)}$  if

- $v_l \in L_i^{(s)}$  for  $i-k \leq l \leq j$ ,
- $j < i+k$  implies that there is no edge  $v_j w \in E$  with  $w \in L_{j+1}^{(s)}$ , and
- $N^k[\tau] \supseteq L_i^{(s)}$  with  $N^k[\tau] := \bigcup_{l=i-k}^j N^k[v_l]$ .

We say that a layer-dominating path  $\sigma = \{v_{i-k}, \dots, v_j\}$  for layer  $L_i^{(s)}$  is *compatible* with a layer-dominating path  $\tau = \{u_{i+1-k}, \dots, u_{j'}\}$  for layer  $L_{i+1}^{(s)}$  if  $j' - j \in \{0, 1\}$  and  $v_l = u_l$  for  $i+1-k \leq l \leq j$ . That is,  $\sigma$  and  $\tau$  share a path of length  $j - i - 1 + k$ .

---

**Algorithm 6.1:** Determines if there is a shortest path of eccentricity at most  $k$  starting at a given vertex  $s$ .

---

**Input:** A graph  $G = (V, E)$  and a positive integer  $k$ .

**Output:** A shortest path with eccentricity at most  $k$  if existent in  $G$ .

- 1 Calculate the layers  $L_i^{(s)} = \{v \in V \mid d_G(s, v) = i\}$  with  $0 \leq i \leq \text{ecc}_G(s)$ .
  - 2 **If**  $\text{ecc}_G(s) \leq 2k$  **Then**
  - 3     For each shortest path  $P$  from  $s$ , determine if  $\text{ecc}_G(P) \leq k$ . In this case,
    - Return  $P$ . If there is no such  $P$ , then  $G$  does not contain a shortest path of eccentricity at most  $k$  starting at  $s$ .
  - 4 **For**  $i = k$  **To**  $\text{ecc}_G(s) - k$
  - 5     Create an empty vertex set  $V'_i$ .
  - 6     **For Each** layer-dominating path  $\tau$  for layer  $L_i^{(s)}$
  - 7         Add a vertex  $v_\tau$ , representing the path  $\tau$ , to  $V'_i$ .
  - 8 **For Each**  $v_\tau \in V'_{\text{ecc}_G(s)-k}$
  - 9     If  $N_G^k[\tau] \not\supseteq \bigcup_{j=\text{ecc}_G(s)-k}^{\text{ecc}_G(s)} L_j^{(s)}$ , remove  $v_\tau$  from  $V'_{\text{ecc}_G(s)-k}$ .
  - 10 Create a graph  $G' = (V', E')$  with  $V' = V'_k \cup \dots \cup V'_{\text{ecc}_G(s)-k}$  and
  $E' = \{v_\sigma v_\tau \mid \sigma \text{ is compatible with } \tau\}$ .
  - 11  $G$  contains a shortest path of eccentricity at most  $k$  starting at  $s$  if and only if  $G'$  contains a path from a vertex  $v_\sigma \in V'_k$  to a vertex  $v_\tau \in V'_{\text{ecc}_G(s)-k}$ .
- 

**Theorem 6.3.** Algorithm 6.1 determines if there is a shortest path of eccentricity at most  $k$  starting from a given vertex  $s$  in  $\mathcal{O}(n^{2k+1}m)$  time.

*Proof (Correctness).* To show the correctness of the algorithm, we need to show that line 11 is correct. Without loss of generality, we can assume that  $\text{ecc}_G(s) > 2k$ . Otherwise, the algorithm would have stopped in line 3.

First, assume that there is a shortest path  $P = \{s = u_0, \dots, u_l\}$  of length  $l$  in  $G$  with  $\text{ecc}_G(P) \leq k$ . Note that  $\text{ecc}_G(s) - k \leq l \leq \text{ecc}_G(s)$ . Then, by Lemma 6.1, each subpath  $\tau = \{u_{i-k}, \dots, u_j\}$  ( $k \leq i \leq \text{ecc}_G(s) - k$ ,  $j = \min\{l, i+k\}$ ) is a layer-dominating path for layer  $L_i^{(s)}$ . Additionally, if  $j = l$ , then  $N_G^k[\tau] \supseteq \bigcup_{j=\text{ecc}_G(s)-k}^{\text{ecc}_G(s)} L_j^{(s)}$ . Thus, the

algorithm creates a vertex  $v_\tau \in V'_i$  in line 7 which represents a subpath of  $P$  for each  $i$  with  $k \leq i \leq \text{ecc}_G(s) - k$ . If  $v_\tau \in V'_i$  and  $v_\sigma \in V'_{i+1}$  represent subpaths of  $P$ ,  $v_\tau$  and  $v_\sigma$  are adjacent in  $G'$  because  $\tau$  and  $\sigma$  are compatible. Therefore, there is a path in  $G'$  from a vertex in  $V'_k$  to a vertex in  $V'_{\text{ecc}_G(s)-k}$ .

Next, assume that  $G'$  contains a path  $P'$  from a vertex  $u \in V'_k$  to a vertex  $v \in V'_{\text{ecc}_G(s)-k}$ . Each vertex  $v_\sigma \in V'_i \cap P'$  represents a layer-dominating path for layer  $L_i^{(s)}$  in  $G$ . By definition of layer-dominating paths, if  $v_\sigma \in V'_i$  is adjacent to  $v_\tau \in V'_{i+1}$ , the paths  $\sigma$  and  $\tau$  in  $G$  (of length  $2k$ ) can be combined to a longer path (of length  $2k + 1$ ). If  $\tau$  has length less than  $2k$ , it is a subpath of  $\sigma$ . Thus,  $P'$  represents a path  $P$  in  $G$  from  $s$  to a vertex  $w \in L_q^{(s)}$  with  $\text{ecc}_G(s) - k \leq q \leq \text{ecc}_G(s)$ .

Each vertex  $v_\tau \in V'_i \cap P'$  represents a layer-dominating path  $\tau$  for layer  $L_i^{(s)}$ . Because of line 9,  $v_\tau \in V'_{\text{ecc}_G(s)-k}$  implies  $N_G^k[\tau] \supseteq \bigcup_{j=\text{ecc}_G(s)-k}^k L_j^{(s)}$ . Thus,  $P$  is a shortest path starting from  $s$  with  $\text{ecc}_G(P) \leq k$ .  $\square$

*Proof (Complexity).* If  $\text{ecc}_G(s) \leq 2k$ , the algorithm stops after line 3. In this case there are at most  $\mathcal{O}(n^{2k})$  shortest paths starting from  $s$ . Thus, finding a shortest path with eccentricity  $k$  can be done in  $\mathcal{O}(n^{2k}m)$  time by deciding in  $\mathcal{O}(m)$  time if a path has eccentricity  $k$ .

Next, assume  $\text{ecc}_G(s) > 2k$ . The graph can only contain  $\mathcal{O}(n^{2k+1})$  layer-dominating paths because each such path has at most  $2k + 1$  vertices in it. Therefore, creating the vertices of  $G'$  (line 4 to line 9) can be done in  $\mathcal{O}(n^{2k+1}m)$  time.

Store the found layer-dominating paths in a forest structure  $\mathcal{T}$  as follows. For each vertex  $v$  of  $G$ ,  $\mathcal{T}$  contains a tree  $T_v$  rooted at  $v$  of depth at most  $2k$ . This tree  $T_v$  stores all layer-dominating paths of  $G$  starting at  $v$ . Any node  $u$  in  $T_v$  (including the root  $v$ ) at depth less than  $2k$  has as the children all neighbours  $w$  of  $u$  in  $G$  such that  $d_G(s, w) = d_G(s, u) + 1$ . Every node of  $T_v$  represents a unique path of  $G$  corresponding to the path of  $T_v$  from the root  $v$  to this node. A leaf  $t$  of  $T_v$  has a pointer to a layer-dominating path  $\tau$  (and, hence, to the corresponding vertex  $v_\tau$  in  $G'$ ) if the path  $\tau = \{v, \dots, t\}$  from the root  $v$  to the leaf  $t$  forms a layer-dominating path  $\tau$  in  $G$ .

Now, given a layer-dominating path  $\sigma = \{v_{i-k}, v_{i-k+1}, \dots, v_j\}$ , we can determine all layer-dominating paths  $\tau$  which  $\sigma$  is compatible with in  $\mathcal{O}(m)$  time as follows. Take the tree  $T_{v_{i-k+1}}$  in  $\mathcal{T}$  and, following path  $\sigma$ , decent to node  $v_j$  of  $T_{v_{i-k+1}}$  representing path  $\{v_{i-k+1}, \dots, v_j\}$ . Then, leaves of  $T_{v_{i-k+1}}$  attached to  $v_j$  have pointer to all paths  $\tau$  which  $\sigma$  is compatible with.

Since  $G'$  has at most  $\mathcal{O}(n^{2k+1})$  vertices, creating  $G'$  (in line 10) takes at most  $\mathcal{O}(n^{2k+1}m)$  time. Thus, the overall running time for Algorithm 6.1 is  $\mathcal{O}(n^{2k+1}m)$ .  $\square$

Algorithm 6.1 determines if there is a shortest path of eccentricity at most  $k$  for a given vertex  $s$ . If a start vertex is not given, iterating Algorithm 6.1 over each vertex determines if there is a shortest path of eccentricity at most  $k$  in a given graph  $G$  in  $\mathcal{O}(n^{2k+2}m)$  time. If  $k$  is unknown, a path with minimum eccentricity can be found by trying different values for  $k$  starting with 1. Then, the runtime is  $\mathcal{O}(n^4m) + \mathcal{O}(n^6m) + \dots + \mathcal{O}(n^{2k+2}m) = \mathcal{O}(n^{2k+2}m)$ .

**Corollary 6.4.** *If a given graph  $G$  contains a shortest path with eccentricity  $k$ , the MESP problem can be solved for  $G$  in  $\mathcal{O}(n^{2k+2}m)$  time, even if  $k$  is unknown.*

Algorithm 6.1 requires  $\mathcal{O}(n^{2k+1}m)$  time because there can be up-to  $\mathcal{O}(2k+1)$  layer-dominating paths with length  $2k$  (see complexity proof of Theorem 6.3). Now, consider the case that a given graph  $G$  has maximum vertex-degree  $\Delta$ . Therefore, such a graph contains at most  $\mathcal{O}(\Delta^{2k}n)$  layer-dominating paths of length  $2k$  and, hence, Algorithm 6.1 requires at most  $\mathcal{O}(\Delta^{2k}nm)$  time. Thus:

**Corollary 6.5.** *The Minimum Eccentricity Shortest Path problem is Fixed Parameter Tractable for graphs with maximum degree  $\Delta$ . If such a graph  $G$  contains a shortest path with eccentricity  $k$ , the MESP problem can be solved for  $G$  in  $\mathcal{O}(\Delta^{2k}n^2m)$  time, even if  $k$  is unknown.*

## 6.2.2 Distance-Hereditary Graphs

In this subsection, we present an algorithm that solves the Minimum Eccentricity Shortest Path problem for distance-hereditary graphs in linear time. Our algorithm is based on the following result.

**Theorem 6.4 (Dragan and Leitert [42]).** *Let  $x, y$  be a diametral pair of vertices of a distance-hereditary graph  $G$ , and  $k$  be the minimum eccentricity of a shortest path in  $G$ . Then, there is a shortest path  $P$  between  $x$  and  $y$  with  $\text{ecc}(P) = k$ .*

Recall that a diametral path in a tree can be found as follows. Select an arbitrary vertex  $v$ . Find a most distant vertex  $x$  from  $v$  and then a most distant vertex  $y$  from  $x$ . The shortest path from  $x$  to  $y$  is a diametral path. Thus, it follows from Theorem 6.4:

**Corollary 6.6.** *For a tree, a shortest path with minimum eccentricity can be computed in linear time by simply performing two BFS calls.*

It is known [45] that a diametral pair of a distance-hereditary graph can be found in linear time. Hence, according to Theorem 6.4, to find a shortest path of minimum eccentricity in a distance-hereditary graph in linear time, one needs to efficiently extract a best eccentricity shortest path for a given pair of end-vertices. In what follows, we demonstrate that, for a distance-hereditary graph, such an extraction can be done in linear time as well.

We will need few auxiliary lemmas.

**Lemma 6.2.** *In a distance-hereditary graph  $G$ , for each pair of vertices  $s$  and  $t$ , if  $x$  is a vertex on a shortest path from  $v$  to  $\Pi_v = \text{Pr}(v, I(s, t))$  with  $d(x, \Pi_v) = 1$ , then  $\Pi_v \subseteq N(x)$ .*

*Proof.* Let  $p$  and  $q$  be two vertices in  $\Pi_v$  and  $d(v, \Pi_v) = r$ . By Lemma 2.8 (page 8),  $N(p) \cap L_{r-1}^{(v)} = N(q) \cap L_{r-1}^{(v)}$ . Thus, each vertex  $x$  on a shortest path from  $v$  to  $\Pi_v$  with  $d(x, \Pi_v) = 1$  (which is in  $N(p) \cap L_{r-1}^{(v)}$  by definition) is adjacent to all vertices in  $\Pi_v$ , i. e.,  $\Pi_v \subseteq N(x)$ .  $\square$



For an interval  $I(s, t)$  between two vertices  $s$  and  $t$ , a *slice*  $S_i(s, t)$  is defined as the set of vertices in  $I(s, t)$  with distance  $i$  to  $s$ , i. e.,  $S_i(s, t) = L_i^{(s)} \cap I(s, t)$ .

**Lemma 6.3.** *In a distance-hereditary graph  $G$ , let  $S_i(s, t)$  and  $S_{i+1}(s, t)$  be two consecutive slices of an interval  $I(s, t)$ . Each vertex in  $S_i(s, t)$  is adjacent to each vertex in  $S_{i+1}(s, t)$ .*

*Proof.* Consider Lemma 2.8 (page 8) from perspective of  $t$ . Thus,  $S_i(s, t) \subseteq N(v)$  for each vertex  $v \in S_{i+1}(s, t)$ . Additionally, from perspective of  $s$ ,  $S_{i+1}(s, t) \subseteq N(u)$  for each vertex  $u \in S_i(s, t)$ .  $\square$

**Lemma 6.4.** *In a distance-hereditary graph  $G$ , if a projection  $\Pi_v = \text{Pr}(v, I(s, t))$  intersects two slices of an interval  $I(s, t)$ , each shortest  $(s, t)$ -path intersects  $\Pi_v$ .*

*Proof.* Because of Lemma 6.2, there is a vertex  $x$  with  $N(x) \supseteq \Pi_v$  and  $d(v, x) = d(v, \Pi_v) - 1$ . Thus,  $\Pi_v$  intersects at most two slices of interval  $I(s, t)$  and those slices have to be consecutive, otherwise  $x$  would be a part of the interval. Let  $S_i(s, t)$  and  $S_{i+1}(s, t)$  be these slices. Note that  $d(s, x) = i + 1$ . Thus, by Lemma 2.8 (page 8),  $N(x) \cap S_i(s, t) = N(u) \cap S_i(s, t)$  for each  $u \in S_{i+1}(s, t)$ . Therefore,  $S_i(s, t) \subseteq \Pi_v$ , i. e., each shortest path from  $s$  to  $t$  intersects  $\Pi_v$ .  $\square$

From the lemmas above, we can conclude that, for determining a shortest  $(s, t)$ -path with minimal eccentricity, a vertex  $v$  is only *relevant* if  $d(v, I(s, t)) = \text{ecc}(I(s, t))$  and the projection of  $v$  on the interval  $I(s, t)$  only intersects one slice. Algorithm 6.2 below uses this observation to find such a path in linear time.

**Lemma 6.5.** *For a distance-hereditary graph  $G$  and an arbitrary vertex pair  $s, t$ , Algorithm 6.2 computes a shortest  $(s, t)$ -path with minimal eccentricity in linear time.*

*Proof.* The loop in line 3 determines for each vertex  $v$  outside of the interval  $I(s, t)$  a *gate vertex*  $g(v)$  such that  $N(g(v)) \supseteq \text{Pr}(v, I(s, t))$  and  $d(v, I(s, t)) = d(v, g(v)) + 1$  (see Lemma 6.2). From Lemma 6.4 and Lemma 6.3, it follows that for a vertex  $v$  which is not in  $V_{\text{ecc}(I(s, t))}$  or its projection to  $I(s, t)$  is intersecting two slices of  $I(s, t)$ ,  $d(v, P(s, t)) \leq \text{ecc}(I(s, t))$  for every shortest path  $P(s, t)$  between  $s$  and  $t$ . Therefore, line 6 only marks  $g(v)$  if  $v \in V_{\text{ecc}(I(s, t))}$  and its projection  $\text{Pr}(v, I(s, t))$  intersects only one slice. Because only one slice is intersected and each vertex in a slice is adjacent to all vertices in the consecutive slice (see Lemma 6.3), in each slice the vertex of an optimal (of minimum eccentricity) path  $P$  can be selected independently from the preceding vertex. If a vertex  $x$  of a slice  $S_i(s, t)$  has the maximum number of *relevant* vertices in  $N(x)$ , then  $x$  is good to put in  $P$ . Indeed, if  $x$  dominates all relevant vertices adjacent to vertices of  $S_i(s, t)$ , then  $x$  is a perfect choice to put in  $P$ . Else, any vertex  $y$  of a slice  $S_i(s, t)$  is a good vertex to put in  $P$ . Hence,  $P$  is optimal if the number of *relevant* vertices adjacent to  $P$  is maximal. Thus, the path selected in line 8 to line 10 is optimal.  $\square$

---

**Algorithm 6.2:** Computes a shortest  $(s, t)$ -path  $P$  with minimal eccentricity for a given distance-hereditary graph  $G$  and a vertex pair  $s, t$ .

---

**Input:** A distance-hereditary graph  $G = (V, E)$  and two distinct vertices  $s$  and  $t$ .

**Output:** A shortest path  $P$  from  $s$  to  $t$  with minimal eccentricity.

- 1 Compute the sets  $V_i = \{v \mid d(v, I(s, t)) = i\}$  for  $1 \leq i \leq \text{ecc}(I(s, t))$ .
  - 2 Each vertex  $v \notin I(s, t)$  gets a pointer  $g(v)$  initialised with  $g(v) := v$  if  $v \in V_1$ , and  $g(v) := \emptyset$  otherwise.
  - 3 **For**  $i := 2$  **To**  $\text{ecc}(I(s, t))$
  - 4    $\lfloor$  For each  $v \in V_i$ , select a vertex  $u \in V_{i-1} \cap N(v)$  and set  $g(v) := g(u)$ .
  - 5 **For Each**  $v \in V_{\text{ecc}(I(s, t))}$
  - 6    $\lfloor$  If  $N(g(v))$  intersects only one slice of  $I(s, t)$ , flag  $g(v)$  as *relevant*.
  - 7 Set  $P := \{s, t\}$ .
  - 8 **For**  $i := 1$  **To**  $d(s, t) - 1$
  - 9    $\lfloor$  Find a vertex  $v \in S_i(s, t)$  for which the number of *relevant* vertices in  $N(v)$  is maximal.
  - 10    $\lfloor$  Add  $v$  to  $P$ .
- 

Running Algorithm 6.2 for a diametral pair of vertices of a distance-hereditary graph  $G$ , by Theorem 6.4, we get a shortest path of  $G$  with minimum eccentricity. Thus, we have proven the following result.

**Theorem 6.5.** *A shortest path with minimum eccentricity of a distance-hereditary graph can be computed in linear time.*

### 6.2.3 A General Approach for Tree-Structured Graphs

In a graph  $G$ , consider a shortest path  $P$  which starts at a vertex  $s$ . Each vertex  $x$  has a projection  $\Pi_x = \text{Pr}(x, P)$ . In case of a tree this is a single vertex. However, in general,  $\Pi_x$  can contain multiple vertices and does not necessarily induce a connected subgraph. In this case, there are two vertices  $u$  and  $w$  in  $\Pi_x$  such that all vertices  $v$  in the subpath  $Q$  between  $u$  and  $w$  are not in  $\Pi_x$ . Formally,  $u, w \in \Pi_x$ ,  $Q = \{v \in P \mid d(s, u) < d(s, v) < d(s, w)\}$ , and  $Q \cap \Pi_x = \emptyset$ .

Now, assume the cardinality of  $Q$  is at most  $\gamma$ , i. e.,  $d(u, w) \leq \gamma + 1$  for each  $P$ ,  $x$ ,  $u$  and  $w$ . Then, we refer to  $\gamma$  as the *projection gap* of  $G$ .

**Definition 6.2 (Projection Gap).** *In a graph  $G$ , let  $P = \{v_0, \dots, v_l\}$  be a shortest path with  $d(v_0, v_i) = i$ . The projection gap of  $G$  is  $\gamma$ ,  $\text{pg}(G) = \gamma$  for short, if, for every vertex  $x$  of  $G$  and every two vertices  $v_i, v_k \in \text{Pr}(x, P)$ ,  $d(v_i, v_k) > \gamma + 1$  implies that there is a vertex  $v_j \in \text{Pr}(x, P)$  with  $i < j < k$ .*

Based on this definition, we can make the following observation.

**Lemma 6.6.** *In a graph  $G$  with  $\text{pg}(G) = \gamma$ , let  $P$  be a shortest path starting at  $s$ ,  $Q$  be a subpath of  $P$ ,  $|Q| > \gamma$ ,  $u$  and  $w$  be two vertices in  $P \setminus Q$  such that  $d(s, u) < d(s, Q) < d(s, w)$ , and  $x$  be an arbitrary vertex in  $G$ . If  $d(x, u) < d(x, Q)$ , then  $d(x, w) \geq d(x, Q)$ .*

*Proof.* Assume that  $d(x, u) < d(x, Q)$  and  $d(x, w) < d(x, Q)$ . Without loss of generality, let  $d(x, u) = d(x, w) < d(x, v)$  for all  $v \in P$  with  $d(s, u) < d(s, v) < d(s, w)$ . Let  $P'$  be the subpath of  $P$  from  $u$  to  $w$ . Note that  $\text{Pr}(x, P') = \{u, w\}$  and  $Q \subset P'$ . Thus,  $d(u, w) \geq |Q| + 1 > \gamma + 1$ . This contradicts with  $\text{pg}(G) = \gamma$ .  $\square$

Informally, Lemma 6.6 says that, when exploring a shortest path  $P$ , if the distance to a vertex  $x$  did not decrease during the last  $\gamma + 1$  vertices of  $P$ , it will not decrease when exploring the remaining subpath. Based on this, we show that a minimum eccentricity shortest path can be found in polynomial time if  $\text{pg}(G)$  is bounded by some constant. Additionally, we show that for some graph classes the projection gap has an upper bound leading to polynomial time algorithms for these classes.

**Algorithm.** For the remainder of this subsection, we assume that we are given a graph  $G$  with  $\text{pg}(G) = \gamma$  containing a vertex  $s$ . We need the following notions and notations:

- $Q_i$  and  $Q_j$  are subpaths of length  $\gamma$  of some shortest paths starting at  $s$ . They do not need to be subpath of the same shortest path. Let  $v_i \in Q_i$  and  $v_j \in Q_j$  be the two vertices such that  $d(s, Q_i) = d(s, v_i)$  and  $d(s, Q_j) = d(s, v_j)$ . Without loss of generality, let  $d(s, v_i) \leq d(s, v_j)$ . We say,  $Q_i$  is *compatible* with  $Q_j$  (with respect to  $s$ ) if  $|Q_i \cap Q_j| = \gamma - 1$ ,  $v_i$  is adjacent to  $v_j$ , and  $d(s, v_i) < d(s, v_j)$ . Let  $\mathcal{C}_s(Q_j)$  denote the set of subpaths compatible with  $Q_j$ .
- $R_s(Q_j) = \{w \mid Q_j \subseteq I(s, w)\} \cup Q_j$  is the set of vertices  $w$  such that there is a shortest path from  $s$  to  $w$  containing  $Q_j$  (or  $w \in Q_j$ ).
- $I_s(Q_j) = I(s, v_j) \cup Q_j$  are the vertices that are on a shortest path from  $s$  to  $Q_j$  (or in  $Q_j$ ).
- $V_s^\downarrow(Q_j) = \{x \mid d(x, Q_j) = d(x, R_s(Q_j))\}$  is the set of vertices  $x$  which are closer to  $Q_j$  than to all other vertices in  $R_s(Q_j)$ . Thus, given a shortest path  $P$  containing  $Q_j$  and starting at  $s$ , expanding  $P$  beyond  $Q_j$  will not decrease the distance from  $x$  to  $P$ .

Note that  $Q_j \subseteq V_s^\downarrow(Q_j)$  and  $Q_j = I_s(Q_j) \cap R_s(Q_j)$ .

**Lemma 6.7.** *For each vertex  $x$  in  $G$ ,  $d(x, Q_j) = d(x, I_s(Q_j))$  or  $d(x, Q_j) = d(x, R_s(Q_j))$ .*

*Proof.* Assume that  $d(x, Q_j) > d(x, I_s(Q_j))$  and  $d(x, Q_j) > d(x, R_s(Q_j))$ . Then, there is a vertex  $u_i \in I_s(Q_j)$  and a vertex  $u_r \in R_s(Q_j)$  with  $d(x, u_i) < d(x, Q_j)$  and  $d(x, Q_j) > d(x, u_r)$ . Because  $u_i$ ,  $Q_j$ , and  $u_r$  are on a shortest path starting at  $s$  and  $|Q_j| > \gamma$ , this contradicts Lemma 6.6.  $\square$

**Lemma 6.8.** *If  $Q_i$  is compatible with  $Q_j$ , then  $V_s^\downarrow(Q_i) \subseteq V_s^\downarrow(Q_j)$ .*

*Proof.* Assume that  $V_s^\downarrow(Q_i) \not\subseteq V_s^\downarrow(Q_j)$ , i. e., there is a vertex  $x \in V_s^\downarrow(Q_i) \setminus V_s^\downarrow(Q_j)$ . Then,  $d(x, Q_j) > d(x, R_s(Q_j))$ . Thus, by Lemma 6.7,  $d(x, Q_j) = d(x, I_s(Q_j))$ . Because  $Q_i \subseteq I_s(Q_j)$ ,  $d(x, Q_i) \geq d(x, I_s(Q_j)) = d(x, Q_j)$ . Since  $x \in V_s^\downarrow(Q_i)$ ,  $d(x, Q_i) = d(x, R_s(Q_i))$ . Also, because  $x \notin V_s^\downarrow(Q_i)$ ,  $d(x, Q_j) > d(x, R_s(Q_j))$ . Thus,  $d(x, R_s(Q_i)) > d(x, R_s(Q_j))$ . On the other hand, because  $R_s(Q_i) \supseteq R_s(Q_j)$ ,  $d(x, R_s(Q_i)) \leq d(x, R_s(Q_j))$  and a contradiction arises.  $\square$

For a subpath  $Q_j$ , let  $\mathcal{P}_s(Q_j)$  denote the set of shortest paths  $P$  which start at  $s$  such that  $Q_j \subseteq P \subseteq I_s(Q_j)$ . Then, we define  $\varepsilon_s(Q_j)$  as follows:

$$\varepsilon_s(Q_j) = \min_{P \in \mathcal{P}_s(Q_j)} \max_{x \in V_s^\downarrow(Q_j)} d(x, P)$$

Consider a subpath  $Q_j$  for which  $R_s(Q_j) = Q_j$ , i. e., a shortest path containing  $Q_j$  cannot be extended any more. Then,  $V_s^\downarrow(Q_j) = V$ . Therefore, for any path  $P \in \mathcal{P}_s(Q_j)$ ,  $\max_{x \in V_s^\downarrow(Q_j)} d(x, P) = \text{ecc}(P)$ .

**Lemma 6.9.** *If  $\mathcal{C}_s(Q_j)$  is non-empty, then*

$$\varepsilon_s(Q_j) = \min_{Q_i \in \mathcal{C}_s(Q_j)} \max \left[ \max_{x \in V_s^\downarrow(Q_j) \setminus V_s^\downarrow(Q_i)} \min(d(x, Q_i), d(x, Q_j)), \varepsilon_s(Q_i) \right].$$

*Proof.* By definition,

$$\varepsilon_s(Q_j) = \min_{P \in \mathcal{P}_s(Q_j)} \max_{x \in V_s^\downarrow(Q_j)} d(x, P).$$

Let  $Q_i$  be compatible with  $Q_j$ . Because, by Lemma 6.8,  $V_s^\downarrow(Q_i) \subseteq V_s^\downarrow(Q_j)$ , we can partition  $V_s^\downarrow(Q_j)$  into  $V_s^\downarrow(Q_j) \setminus V_s^\downarrow(Q_i)$  and  $V_s^\downarrow(Q_i)$ . For simplicity, we write  $V_s^\downarrow(Q_j) \setminus V_s^\downarrow(Q_i)$  as  $V_s^\downarrow[Q_j|Q_i]$ . Thus,  $\varepsilon_s(Q_j) =$

$$\min_{Q_i \in \mathcal{C}_s(Q_j)} \min_{P \in \mathcal{P}_s(Q_i)} \max \left[ \max_{x \in V_s^\downarrow[Q_j|Q_i]} d(x, P \cup Q_j), \max_{x \in V_s^\downarrow(Q_i)} d(x, P \cup Q_j) \right].$$

Note that we changed the definition of  $P$  from  $P \in \mathcal{P}_s(Q_j)$  to  $P \in \mathcal{P}_s(Q_i)$ , i. e.,  $P$  may not contain the last vertex of  $Q_j$  any more.

If  $x \in V_s^\downarrow[Q_j|Q_i]$ , then  $d(x, Q_i) > d(x, R_s(Q_i))$ . Thus, by Lemma 6.7,  $d(x, Q_i) = d(x, I_s(Q_i))$ . Note that, by definition of  $P$ ,  $d(x, Q_i) \geq d(x, P) \geq d(x, I_s(Q_i))$ . Therefore,  $d(x, P) = d(x, Q_i)$  and

$$\max_{x \in V_s^\downarrow[Q_j|Q_i]} d(x, P \cup Q_j) = \max_{x \in V_s^\downarrow[Q_j|Q_i]} \min(d(x, Q_i), d(x, Q_j)).$$

For simplicity, we define

$$\varepsilon_s(Q_i, Q_j) := \max_{x \in V_s^\downarrow[Q_j|Q_i]} \min(d(x, Q_i), d(x, Q_j)).$$

Note that  $\varepsilon_s(Q_i, Q_j)$  does not depend on  $P$ . Therefore, because  $\min_u \max[c, f(u)] = \max[c, \min_u f(u)]$ ,

$$\varepsilon_s(Q_j) = \min_{Q_i \in \mathcal{C}_s(Q_j)} \max \left[ \varepsilon_s(Q_i, Q_j), \min_{P \in \mathcal{P}_s(Q_i)} \max_{x \in V_s^\downarrow(Q_i)} d(x, P \cup Q_j) \right].$$

If  $x \in V_s^\downarrow(Q_i)$ , then  $d(x, Q_i) = d(x, R_s(Q_i)) \leq d(x, R_s(Q_j)) = d(x, Q_j)$ . Therefore,

$$\min_{P \in \mathcal{P}_s(Q_i)} \max_{x \in V_s^\downarrow(Q_i)} d(x, P \cup Q_j) = \min_{P \in \mathcal{P}_s(Q_i)} \max_{x \in V_s^\downarrow(Q_i)} d(x, P) = \varepsilon_s(Q_i).$$

Thus,

$$\varepsilon_s(Q_j) = \min_{Q_i \in \mathcal{C}_s(Q_j)} \max \left[ \max_{x \in V_s^\downarrow(Q_j) \setminus V_s^\downarrow(Q_i)} \min(d(x, Q_i), d(x, Q_j)), \varepsilon_s(Q_i) \right]. \quad \square$$

Based on Lemma 6.9, Algorithm 6.3 computes a shortest path starting at  $s$  with minimal eccentricity. The algorithm has two parts. First, it computes the pairwise distance of all vertices and  $d(x, R_s(v))$  for each vertex pair  $x$  and  $v$  where, similarly to  $R_s(Q_j)$ ,  $R_s(v) = \{z \in V \mid v \in I(s, z)\}$ . This allows to easily determine if a vertex  $x$  is in  $V_s^\downarrow(Q_j)$ . Second, it computes  $\varepsilon_s(Q_j)$  for each subpath  $Q_j$ . For this, the algorithm uses dynamic programming. After calculating  $\varepsilon_s(Q_i)$  for all subpaths with distance  $i$  to  $s$ , the algorithm uses Lemma 6.9 to calculate  $\varepsilon_s(Q_j)$  for all subpaths  $Q_j$  which  $Q_i$  is compatible with.

**Theorem 6.6.** *For a given graph  $G$  with  $\text{pg}(G) = \gamma$  and a vertex  $s$ , Algorithm 6.3 computes a shortest path starting at  $s$  with minimal eccentricity. It runs in  $\mathcal{O}(n^{\gamma+3})$  time if  $\gamma \geq 2$ , in  $\mathcal{O}(n^2m)$  time if  $\gamma = 1$ , and in  $\mathcal{O}(nm)$  time if  $\gamma = 0$ .*

*Proof (Correctness).* The algorithm has two parts. The first part (line 1 to line 8) is a preprocessing which computes  $d(x, R_s(v))$  for each vertex pair  $x$  and  $v$ . The second part computes  $\varepsilon_s(Q_j)$  which is then used to determine a path with minimal eccentricity.

For the first part, without loss of generality, let  $d(s, v) = i$ ,  $N_s^\uparrow(v) = N(v) \cap L_{i+1}^{(s)}$ , and let  $x$  be an arbitrary vertex. By definition of  $R_s$ ,  $N_s^\uparrow(v) = \emptyset$  implies  $R_s(v) = \{v\}$ , i. e.,  $d(x, R_s(v)) = d(x, v)$ . Therefore,  $d(x, R_s(v))$  is correct for all vertices  $v$  with  $N_s^\uparrow(v) = \emptyset$  after line 3. By induction, assume that  $d(x, R_s(w))$  is correct for all vertices  $w \in N_s^\uparrow(v)$ . Because  $R_s(v) = \bigcup_{w \in N_s^\uparrow(v)} R_s(w) \cup \{v\}$ ,  $d(x, R_s(v)) = \min(\min_{w \in N_s^\uparrow(v)} d(x, R_s(w)), d(x, v))$ . Therefore, line 8 correctly computes  $d(x, R_s(v))$ .

The second part of Algorithm 6.3 iterates over all subpaths  $Q_j$  in increasing distance to  $s$ . Line 12 checks if a given vertex  $x$  is in  $V_s^\downarrow(Q_j)$ . By definition,  $R_s(Q_j) = Q_j \cup R_s(z_j)$  where  $z_j$  is the vertex in  $Q_j$  with the largest distance to  $s$ . Thus,  $d(x, R_s(Q_j)) = \min(d(x, R_s(z_j)), d(x, Q_j))$ . By definition of  $V_s^\downarrow$ ,  $x \in V_s^\downarrow(Q_j)$  if and only if  $d(x, Q_j) = d(x, R_s(Q_j))$ . Therefore,  $x \in V_s^\downarrow(Q_j)$  if and only if  $d(x, Q_j) \leq d(x, R_s(z_j))$ , i. e., line 12 computes  $V_s^\downarrow(Q_j)$  correctly.

---

**Algorithm 6.3:** Determines, for a given graph  $G$  with  $\text{pg}(G) \leq \gamma$  and a vertex  $s$ , a minimal eccentricity shortest path starting at  $s$ .

---

**Input:** A graph  $G = (V, E)$ , an integer  $\gamma$ , and a vertex  $s \in V$ .

**Output:** A shortest path  $P$  starting at  $s$  with minimal eccentricity.

- 1 Determine the pairwise distances of all vertices.
  - 2 **For Each**  $v, x \in V$
  - 3     $\lfloor$  Set  $d(x, R_s(v)) := d(x, v)$ .
  - 4 **For**  $i = \text{ecc}(s) - 1$  **DownTo** 0
  - 5     $\lfloor$  **For Each**  $v \in L_i^{(s)}$
  - 6         $\lfloor$  **For Each**  $w \in N(v) \cap L_{i+1}^{(s)}$
  - 7             $\lfloor$  **For Each**  $x \in V$
  - 8                 $\lfloor$  Set  $d(x, R_s(v)) := \min [d(x, R_s(v)), d(x, R_s(w))]$ .
  - 9 **For**  $j = 0$  **To**  $\text{ecc}(s) - \gamma$
  - 10     $\lfloor$  **For Each**  $Q_j$  with  $d(s, Q_j) = j$
  - 11         $\lfloor$  **For Each**  $x \in V$
  - 12             $\lfloor$  Let  $z_j$  be the vertex in  $Q_j$  with the largest distance to  $s$ . If
  - $\lfloor$   $d(x, Q_j) \leq d(x, R_s(z_j))$ , add  $x$  to  $V_s^\downarrow(Q_j)$  and store  $d(x, Q_j)$ .
  - 13         $\lfloor$  **If**  $j = 0$  **Then**
  - 14             $\lfloor$   $\varepsilon_s(Q_j) := \max_{x \in V_s^\downarrow(Q_j)} d(x, Q_j)$
  - 15         $\lfloor$  **Else**
  - 16             $\lfloor$   $\varepsilon_s(Q_j) := \infty$
  - 17         $\lfloor$  **For Each**  $Q_i \in \mathcal{C}_s(Q_j)$
  - 18             $\lfloor$   $\varepsilon'_s(Q_j) := \max \left[ \max_{x \in V_s^\downarrow(Q_j) \setminus V_s^\downarrow(Q_i)} \min (d(x, Q_i), d(x, Q_j)), \varepsilon_s(Q_i) \right]$
  - 19             $\lfloor$  **If**  $\varepsilon'_s(Q_j) < \varepsilon_s(Q_j)$  **Then**
  - 20                 $\lfloor$  Set  $\varepsilon_s(Q_j) := \varepsilon'_s(Q_j)$  and  $p(Q_j) := Q_i$ .
  - 21 Find a subpath  $Q_j$  such that a shortest path containing  $Q_j$  cannot be extended any more and for which  $\varepsilon_s(Q_j)$  is minimal.
  - 22 Construct a path  $P$  from  $Q_j$  to  $s$  using the  $p()$ -pointers and output it.
- 

Recall the definition of  $\varepsilon_s(Q_j)$ :

$$\varepsilon_s(Q_j) = \min_{P \in \mathcal{P}_s(Q_j)} \max_{x \in V_s^\downarrow(Q_j)} d(x, P)$$

If  $d(s, Q_j) = 0$ ,  $d(x, P) = d(x, Q_j)$ . Therefore,  $\varepsilon_s(Q_j) = \max_{x \in V_s^\downarrow(Q_j)} d(x, Q_j)$  as computed in line 14. Note that there is no subpath  $Q_i$  which is compatible with  $Q_j$ , if  $d(s, Q_j) = 0$ . Therefore, the loop starting in line 17 is skipped for these  $Q_j$ . Thus, the

algorithm correctly computes  $\varepsilon_s(Q_j)$ , if  $d(s, Q_j) = 0$ .

By induction, assume that  $\varepsilon_s(Q_i)$  is correct for each  $Q_i \in \mathcal{C}(Q_j)$ . Thus, Lemma 6.9 can be used to compute  $\varepsilon_s(Q_j)$ . This is done in the loop starting in line 17. Therefore, at the beginning of line 21,  $\varepsilon_s(Q_j)$  is computed correctly for each subpath  $Q_j$ .

Recall that, if  $P \in \mathcal{P}(Q_j)$  and  $R_s(Q_j) = Q_j$ , then  $V_s^\downarrow(Q_j) = V$  and, therefore,  $\max_{x \in V_s^\downarrow(Q_j)} d(x, P) = \text{ecc}(P)$ . Thus,  $R_s(Q_j) = Q_j$  implies that  $\varepsilon_s(Q_j)$  is the minimal eccentricity of all shortest paths starting in  $s$  and containing  $Q_j$ . Therefore, if  $Q_j$  is picked by line 21, then  $\varepsilon_s(Q_j)$  is the minimal eccentricity of all shortest paths starting in  $s$ .  $\square$

*Proof (Complexity).* First, we analyse line 1 to line 8. Line 1 runs in  $\mathcal{O}(nm)$  time. This allows to access the distance between two vertices in constant time. Thus, the total running time for line 3 is  $\mathcal{O}(n)$ . Because line 8 is called at most once for each vertex  $x$  and edge  $vw$ , implementing line 4 to line 8 can be done in  $\mathcal{O}(nm)$  time.

For the second part of the algorithm (starting in line 9), if  $\gamma \geq 2$ , let all subpaths be stored in a trie as follows: There are  $\gamma + 1$  layers of internal nodes. Each internal node is an array of size  $n$  (one entry for each vertex) and each entry points to an internal node of the next layer representing  $n$  subtrees. This requires  $\mathcal{O}(n^{\gamma+1})$  memory. Leafs are objects representing a subpath.

If  $\gamma = 1$ , a subpath is a single edge, and, if  $\gamma = 0$ , a subpath is a single vertex. Thus, no extra data structure is needed for these cases. In all three cases, a subpath can be accessed in  $\mathcal{O}(\gamma)$  time.

Next, we analyse the runtime of line 11 to line 16 for a single subpath  $Q_j$ . Accessing  $Q_j$  can be done in  $\mathcal{O}(\gamma)$  time. Line 12 requires at most  $\mathcal{O}(\gamma)$  time for a single call and is called at most  $\mathcal{O}(n)$  times. Line 14 requires  $\mathcal{O}(n\gamma)$  time and line 16 runs in constant time. Therefore, for a given subpath, line 11 to line 16 require  $\mathcal{O}(\gamma n + n)$  time.

For line 18 to line 20, consider a given pair of compatible subpaths  $Q_i$  and  $Q_j$ . Accessing both subpaths can be done in  $\mathcal{O}(\gamma)$  time. Assuming the vertices in  $V_s^\downarrow(Q_i)$  and  $V_s^\downarrow(Q_j)$  are sorted and stored with their distance to  $Q_i$  and  $Q_j$ , line 18 requires at most  $\mathcal{O}(n)$  time. Note that  $Q_i$  and  $Q_j$  intersect in  $\gamma - 1$  vertices. Thus,  $\min(d(x, Q_i), d(x, Q_j)) = \min(d(x, v_i), d(x, Q_j))$  where  $v_i$  is the vertex in  $Q_i$  closest to  $s$ . Line 19 and line 20 run in constant time. Therefore, for a given pair of compatible subpaths, line 18 to line 20 require  $\mathcal{O}(n)$  time.

Let  $\phi$  be the number of subpaths and  $\psi$  be the number of pairs of compatible subpaths. Then, the overall runtime for line 9 to line 20 is  $\mathcal{O}(\phi(\gamma n + n) + \psi n)$  time,  $\mathcal{O}(\phi)$  time for line 21, and  $\mathcal{O}(n)$  time for line 22. Together with the first part of the algorithm, the total runtime of Algorithm 6.3 is  $\mathcal{O}(mn + \phi(\gamma n + n) + \psi n)$ .

Because a subpath contains  $\gamma + 1$  vertices, there are up-to  $\mathcal{O}(n^{\gamma+1})$  subpaths and up-to  $\mathcal{O}(n^{\gamma+2})$  compatible pairs if  $\gamma \geq 2$ , i. e.,  $\phi \leq n^{\gamma+1}$  and  $\psi \leq n^{\gamma+2}$ . Therefore, if  $\gamma \geq 2$ , Algorithm 6.3 runs in  $\mathcal{O}(n^{\gamma+3})$  time.

If  $\gamma = 1$ , a subpath is a single edge and there are at most  $mn$  compatible pairs of subpaths, i. e.,  $\phi \leq m$  and  $\psi \leq nm$ . For the case when  $\gamma = 0$ , a subpath is a single vertex

( $\phi \leq n$ ) and a pair of compatible subpaths is an edge ( $\psi \leq m$ ). Therefore, Algorithm 6.3 runs in  $\mathcal{O}(n^2m)$  time if  $\gamma = 1$ , and in  $\mathcal{O}(nm)$  time if  $\gamma = 0$ .  $\square$

Note that Algorithm 6.3 computes a shortest path starting in a given vertex  $s$ . Thus, a shortest path with minimum eccentricity among all shortest paths in  $G$  can be determined by running Algorithm 6.3 for all start vertices  $s$ , resulting in the following:

**Theorem 6.7.** *For a given graph  $G$  with  $\text{pg}(G) = \gamma$ , a minimum eccentricity shortest path can be found in  $\mathcal{O}(n^{\gamma+4})$  time if  $\gamma \geq 2$ , in  $\mathcal{O}(n^3m)$  time if  $\gamma = 1$ , and in  $\mathcal{O}(n^2m)$  time if  $\gamma = 0$ .*

**Projection Gap for Some Special Graph Classes.** Above, we show that a minimum eccentricity shortest path can be found in polynomial time if the projection gap is bounded by a constant. In what follows, we determine the projection gap for chordal graphs, dually chordal graphs, graphs with bounded tree-length or tree-breadth, and  $\delta$ -hyperbolic graphs.

*Chordal Graphs.*

**Lemma 6.10.** *If  $G$  is a chordal graph, then  $\text{pg}(G) = 0$ .*

*Proof.* Assume that  $\text{pg}(G) \geq 1$ . Then, there is a shortest path  $P = \{u, \dots, w\}$  and a vertex  $x$  with  $\text{Pr}(x, P) = \{u, w\}$  and  $d(u, w) > 1$ . By Lemma 2.7 (page 2.7),  $u$  and  $w$  are adjacent. This contradicts with  $d(u, w) > 1$ .  $\square$

**Corollary 6.7.** *For chordal graphs, a minimum eccentricity shortest path can be found in  $\mathcal{O}(n^2m)$  time.*

*Dually Chordal Graphs.*

**Lemma 6.11.** *If  $G$  is a dually chordal graph, then  $\text{pg}(G) \leq 1$ .*

*Proof.* Assume there is a shortest path  $P = \{u, v_1, \dots, v_i, w\}$  and a vertex  $x$  with  $\text{Pr}(x, P) \supseteq \{u, w\}$ . To show that  $\text{pg}(G) \leq 1$ , we show that  $d(u, w) = i + 1 > 2$  implies there is a vertex  $v_k \in \text{Pr}(x, P)$  with  $1 \leq k \leq i$ .

Consider a family of disks  $\mathcal{D} = \{N[u], N[v_1], \dots, N[v_i], N[w], N^r[x]\}$  where  $r = d(x, P) - 1$ . Let  $H$  be the intersection graph of  $\mathcal{D}$ ,  $a$  be the vertex in  $H$  representing  $N[u]$ ,  $b_k$  representing  $N[v_k]$  (for  $1 \leq k \leq i$ ),  $c$  representing  $N[w]$ , and  $z$  representing  $N^r[x]$ . Because the intersection graph of disks of a dually chordal graph is chordal [16],  $H$  is chordal, too.  $H$  contains the edges  $za$  and  $zc$ ,  $ab_1$ ,  $cb_i$ , and  $b_k b_{k+1}$  for all  $1 \leq k < i$ . Note that, if  $d(u, w) > 2$ ,  $a$  and  $c$  are not adjacent in  $H$ . However, the path  $\{a, b_1, \dots, b_k, c\}$  connects  $a$  and  $c$ . Therefore, because  $H$  is chordal and by Lemma 2.7 (page 7), there is a  $k$  with  $1 \leq k \leq i$  such that  $z$  is adjacent to  $b_k$  in  $H$ . Thus,  $d(x, v_k) \leq r + 1$ , i. e.,  $v_k \in \text{Pr}(x, P)$ .  $\square$

**Corollary 6.8.** *For dually chordal graphs, a minimum eccentricity shortest path can be found in  $\mathcal{O}(n^3m)$  time.*



*Graphs with bounded Tree-Length or Tree-Breadth.*

**Lemma 6.12.** *If  $G$  has tree-length  $\lambda$  or tree-breadth  $\rho$ , a factor  $\gamma \geq \text{pg}(G)$  can be computed in  $\mathcal{O}(n^3)$  time such that  $\gamma \leq 3\lambda - 1$  or  $\gamma \leq 6\rho - 1$ , respectively.*

*Proof.* To compute  $\gamma$ , first determine the pairwise distances of all vertices. Then, compute a layering partition for each vertex  $x$ . Let  $\gamma + 1$  be the maximum diameter of all clusters of all layering partitions.

As shown in Lemma 3.1 (page 11), the diameter of each cluster is at most  $3\lambda$  if  $G$  has tree-length  $\lambda$  and at most  $6\rho$  if  $G$  has tree-breadth  $\rho$ . Therefore, for each shortest path  $P$ ,  $\text{diam}(\text{Pr}(x, P)) \leq 3\lambda$  and  $\text{diam}(\text{Pr}(x, P)) \leq 6\rho$ , respectively. Thus,  $\text{pg}(G) \leq \gamma \leq 3\lambda - 1$  and  $\text{pg}(G) \leq \gamma \leq 6\rho - 1$ .

Computing the pairwise distances of all vertices can be done in  $\mathcal{O}(nm)$  time. A layering partition can be computed in linear time (Lemma 2.5, page 7). For a given layering partition, the diameter of each cluster can be computed in  $\mathcal{O}(n^2)$  time if the pairwise distances of all vertices are known. Thus,  $\gamma$  can be computed in  $\mathcal{O}(n^3)$  time.  $\square$

Note that it is not necessary to know the tree-length or tree-breadth of  $G$  to compute  $\gamma$ . Thus, by computing  $\gamma$  and then running Algorithm 6.3 for each vertex in  $G$ , we get:

**Corollary 6.9.** *For graphs with tree-length  $\lambda$  or tree-breadth  $\rho$ , a minimum eccentricity shortest path can be found in  $\mathcal{O}(n^{3\lambda+3})$  time or  $\mathcal{O}(n^{6\rho+3})$  time, respectively.*

*$\delta$ -Hyperbolic Graphs.*

**Lemma 6.13.** *If  $G$  is  $\delta$ -hyperbolic, then  $\text{pg}(G) \leq 4\delta$ .*

*Proof.* Consider two vertices  $u$  and  $w$  such that  $u, w \in \text{Pr}(x, P)$  for some vertex  $x$  and shortest path  $P$ . Let  $v \in P$  be a vertex such that  $d(u, v) - d(v, w) \leq 1$  and  $d(u, v) \geq d(v, w)$ , i. e.,  $v$  is a middle vertex on the subpath from  $u$  to  $w$ .

Assume that  $d(u, w) > 4\delta + 1$ . Thus,  $d(u, v) \geq d(v, w) \geq 2\delta + 1$  and  $d(u, w) > d(u, v) + 2\delta$ . Therefore, by Lemma 2.10 (page 9),  $d(v, x) < \max\{d(x, u), d(x, w)\}$ . This contradicts that  $u, w \in \text{Pr}(x, P)$ . Hence, the diameter of a projection is at most  $4\delta + 1$  and, therefore,  $\text{pg}(G) \leq 4\delta$ .  $\square$

**Corollary 6.10.** *For  $\delta$ -hyperbolic graphs, a minimum eccentricity shortest path can be found in  $\mathcal{O}(n^{4\delta+4})$  time.*

## 6.3 Approximation Algorithms

In this section, we present different approximation algorithms. The algorithms differ in their approximation factor and runtime. First, we show algorithms for general graphs. Then, we present approaches to compute an approximation for graphs with bounded tree-length and graphs with bounded hyperbolicity. Additionally, we present an approach to compute an approximation based on the layering partition of a graph.

### 6.3.1 General Graphs

In [41], we show that a spread path gives an 8-approximation for MESP. Recently, BIRMELEÉ et al. [10] were able to improve this result.

**Theorem 6.8 (Birmelé et al. [10]).** *Let  $G$  be a graph having a shortest path of eccentricity  $k$ . Any spread path in  $G$  has eccentricity at most  $5k$ . This bound is tight.*

From Theorem 6.8, it follows that a 5-approximation for MESP can be computed in linear time by simply performing two BFS calls. BIRMELEÉ et al. [10] additionally show that Algorithm 6.4 below computes a 3-approximation in linear time. However, the runtime of this approach has a much higher constant factor.

---

**Algorithm 6.4:** [10] Computes a 3-approximation for MESP.

---

**Input:** A graph  $G = (V, E)$ .

**Output:** A shortest path with eccentricity at most  $2k$ , where  $k$  is the minimum eccentricity of all paths in  $G$ .

- 1 Compute a spread path  $P$  in  $G$  from  $x$  to  $y$  and determine its eccentricity.
  - 2 Initialise an empty queue  $Q$  of triples  $(u, v, s)$  where  $u$  and  $v$  are vertices of  $G$  and  $s$  is an integer.
  - 3 Insert  $(x, y, 0)$  into  $Q$ .
  - 4 **While**  $Q$  is non-empty.
  - 5     Remove a triple  $(u, v, s)$  from  $Q$ .
  - 6     Compute a shortest path  $P'$  from  $u$  to  $v$  and determine a vertex  $w$  with maximal distance to  $P'$ .
  - 7     **If**  $d(P', w) < \text{ecc}(P)$  **Then**
  - 8         Set  $P := P'$ .
  - 9     **If**  $s < 8$  **Then**
  - 10         Insert  $(u, w, s + 1)$  and  $(v, w, s + 1)$  into  $Q$ .
  - 11 Output  $P$ .
- 

**Theorem 6.9 (Birmelé et al. [10]).** *Algorithm 6.4 calculates a 3-approximation for the MESP problem in linear time.*

Next, we present a 2-approximation algorithm. It is based on the following two lemmas.

**Lemma 6.14.** *In a graph  $G$ , let  $P$  be a shortest path from  $s$  to  $t$  of eccentricity at most  $k$ . For each layer  $L_i^{(s)}$ , there is a vertex  $p_i \in P$  such that the distance from  $p_i$  to each vertex  $v \in L_i^{(s)}$  is at most  $2k$ . Additionally,  $p_i \in L_i^{(s)}$  if  $i \leq d(s, t)$ , and  $p_i = t$  if  $i \geq d(s, t)$ .*

*Proof.* For each vertex  $v$ , let  $p(v) \in P$  be a vertex with  $d(p(v), v) \leq k$ .

For each  $i \leq d(s, t)$ , let  $p_i \in P \cap L_i^{(s)}$  be the vertex in  $P$  with distance  $i$  to  $s$ . For an arbitrary vertex  $v \in L_i^{(s)}$ , let  $j = d(s, p(v))$ . Because  $\text{ecc}(P) \leq k$  and  $P$  is a shortest path,  $|i - j| \leq k$ . Thus,  $d(p_i, v) \leq d(p_i, p(v)) + d(p(v), v) \leq 2k$ .

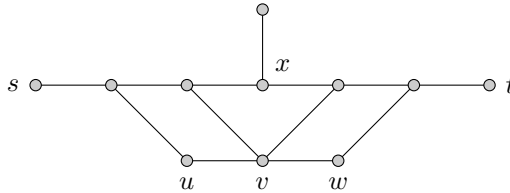
Let  $L' = \{v \mid d(s, v) \geq d(s, t)\}$ . Because  $P$  has eccentricity at most  $k$ ,  $d(p, t) \leq k$  for all  $p \in \{p(v) \mid v \in L'\}$ . Therefore,  $d(t, v) \leq 2k$  for all  $v \in L'$ .  $\square$

**Lemma 6.15.** *If  $G$  has a shortest path of eccentricity at most  $k$  from  $s$  to  $t$ , then every path  $Q$  with  $s \in Q$  and  $d(s, t) \leq \max_{v \in Q} d(s, v)$  has eccentricity at most  $3k$ .*

*Proof.* Let  $P$  be a shortest path from  $s$  to  $t$  with  $\text{ecc}(P) \leq k$  and  $Q$  an arbitrary path with  $s \in Q$  and  $d(s, t) \leq \max_{v \in Q} d(s, v)$ . Without loss of generality, we can assume that  $Q$  starts at  $s$ . Also, let  $u$  be an arbitrary vertex. Since  $\text{ecc}(P) \leq k$ , there is a vertex  $p \in P$  with  $d(u, p) \leq k$ . Because  $d(s, t) \leq \max_{v \in Q} d(s, v)$ , there is a vertex  $q \in Q$  with  $d(s, p) = d(s, q)$ . By Lemma 6.14, the distance between  $p$  and  $q$  is at most  $2k$ . Thus, the distance from  $q$  to  $u$  is at most  $3k$ .  $\square$

**Corollary 6.11.** *For a given graph  $G$  and two vertices  $s$  and  $t$ , each shortest  $(s, t)$ -path is a 3-approximation for the  $(s, t)$ -MESP problem.*

Note that the bounds given in Lemma 6.14 and Lemma 6.15 are tight. Figure 6.3 gives an example.



**Figure 6.3.** Example for Lemma 6.14 and Lemma 6.15. The shortest path from  $s$  to  $t$  which contains  $x$  has eccentricity 1 and the distance from  $x$  to  $v$  is 2. The shortest path from  $s$  to  $t$  which contains  $u$  and  $w$  has eccentricity 3.

For Algorithm 6.5 below, we say the *layer-wise eccentricity* of a shortest  $(s, t)$ -path  $Q$  is  $\phi$  if, for each layer  $L_i^{(s)}$ ,  $\max \{d(q_i, u) \mid u \in L_i^{(s)}\} \leq \phi$  where  $q_i \in Q \cap L_i^{(s)}$  if  $i \leq d(s, t)$  and  $q_i = t$  if  $i > d(s, t)$ . By Lemma 6.14, a shortest path with eccentricity  $k$  has a layer-wise eccentricity of  $2k$ . Therefore, determining a shortest path with minimum layer-wise eccentricity gives a 2-approximation for the MESP problem. To find such a path, Algorithm 6.5 computes, for each vertex  $s$ , the maximal distance of a vertex  $v$  to all other vertices  $u$  in the same layer  $L_i^{(s)}$  and uses a modified BFS to find a shortest path with minimal layer-wise eccentricity starting at  $s$ .

**Theorem 6.10.** *Algorithm 6.5 calculates a 2-approximation for the MESP problem in  $\mathcal{O}(n^3)$  time.*

---

**Algorithm 6.5:** A 2-approximation for the MESP problem.

---

**Input:** A graph  $G = (V, E)$ .

**Output:** A shortest path with eccentricity at most  $2k$ , where  $k$  is the minimum eccentricity of all paths in  $G$ .

- 1 Calculate the distances  $d(u, v)$  for all vertex pairs  $u$  and  $v$ , including  $L_i^{(u)} = \{v \in V \mid d(u, v) = i\}$  with  $0 \leq i \leq \text{ecc}(u)$  for each  $u$ .
  - 2 **For Each**  $s \in V$ 
    - 3     Set  $\phi(s) := 0$ .
    - 4     **For**  $i := 1$  **To**  $\text{ecc}(s)$ 
      - 5         **For Each**  $v \in L_i^{(s)}$ 
        - 6             Set  $\phi(v) := \max_{u \in L_i^{(s)}} d(u, v)$ .
        - 7             Let  $N^-(v) = L_{i-1}^{(s)} \cap N(v)$  denote the neighbours of  $v$  in the previous layer. Set  $\phi(v) := \max \left\{ \min_{u \in N^-(v)} \phi(u), \phi(v) \right\}$ .
        - 8             Set  $\phi^+(v) := \max \{ d(u, v) \mid d(s, u) \geq i \}$ .
    - 9     Calculate a BFS-tree  $T(s)$  starting from  $s$ . If multiple vertices  $u$  are possible as parent for a vertex  $v$ , select one with the smallest  $\phi(u)$ .
    - 10     Let  $t$  be the vertex for which  $\phi'(t) := \max \{ \phi(t), \phi^+(t) \}$  is minimal. Set  $k(s) := \phi'(t)$
  - 11 Among all computed pairs  $s$  and  $t$ , select a pair (and corresponding path in  $T(s)$ ) for which  $k(s)$  is minimal.
- 

*Proof (Correctness).* Assume a given graph  $G$  has a shortest path  $P$  from  $s$  to  $t$  with  $\text{ecc}(P) = k$  and  $s$  is the vertex selected by the loop starting in line 2. Let  $Q$  be a shortest path from  $s$  to  $v$ .

We show that lines 4 to 8 calculate, for each  $v$ , the minimal  $\phi(v)$  such that there is a shortest path  $Q$  from  $s$  to  $v$  with a layer-wise eccentricity  $\phi(v)$ .

By induction, assume this is true for all vertices  $u \in L_j^{(s)}$  with  $j \leq i - 1$ . Now let  $v$  be an arbitrary vertex in  $L_i^{(s)}$ . Line 6 calculates the maximal distance  $\phi(v)$  from  $v$  to all other vertices in  $L_i^{(s)}$ . Since  $v$  is the only vertex in  $Q \cap L_i^{(s)}$  for every shortest path  $Q$  from  $s$  to  $v$ , the layer-wise eccentricity of each  $Q$  is at least  $\phi(v)$ . Let  $u$  be a neighbour of  $v$  in the previous layer. By induction hypothesis,  $\phi(u)$  is optimal. Therefore,  $\phi(v) := \max \left\{ \min_{u \in N^-(v)} \phi(u), \phi(v) \right\}$  (line 7) is optimal for  $v$ .

Since line 9 selects the vertex  $u$  with the smallest  $\phi(u)$  as parent for  $v$ , each path  $Q$  from  $s$  to  $v$  in  $T(s)$  has an optimal layer-wise eccentricity of  $\phi(v)$ . Line 8 calculates the maximal distance from  $v$  to all vertices in  $\{u \mid d(s, u) \geq d(s, v)\}$ . Thus,  $\text{ecc}(Q) \leq \phi'(v)$  and line 10 and line 11 select a shortest path which has an eccentricity at most  $\phi'(v)$ .

By Lemma 6.14, we know that  $P$  has a layer-wise eccentricity of at most  $2k$ . Thus, the path  $Q$  from  $s$  to  $t$  in  $T(s)$  has a layer-wise eccentricity of at most  $2k$ . Additionally,

Lemma 6.14 says that  $t$  has distance at most  $2k$  to all vertices in  $\{v \mid d(s, v) \geq d(s, t)\}$ . Therefore,  $\text{ecc}(Q) \leq 2k$ . Thus, the path selected in line 11 is a shortest path with eccentricity at most  $2k$ .  $\square$

*Proof (Complexity).* Line 1 runs in  $\mathcal{O}(nm)$  time. If the distances are stored in an array, they can be later accessed in constant time. Therefore, line 6 and line 8 run in  $\mathcal{O}(n)$  time for a given  $s$  and  $v$  or in  $\mathcal{O}(n^3)$  time overall. For a given  $s$ , line 7 runs in  $\mathcal{O}(m)$  time and, therefore, has an overall runtime of  $\mathcal{O}(nm)$ . Line 9 has an overall runtime of  $\mathcal{O}(nm)$ , line 11 takes  $\mathcal{O}(n^2)$  time, and line 10 runs in  $\mathcal{O}(n)$  time. Adding all together, the total runtime is  $\mathcal{O}(n^3)$ .  $\square$

For the case that a start vertex  $s$  for a shortest path is given, Algorithm 6.5 can be simplified by having only one iteration of the loop starting in line 2. Then, the runtime is  $\mathcal{O}(nm)$ .

**Corollary 6.12.** *A 2-approximation for the  $(s, t)$ -MESPP problem can be computed in  $\mathcal{O}(nm)$  time.*

### 6.3.2 Graphs with Bounded Tree-Length and Bounded Hyperbolicity

In Section 6.2.3, we show how to find a shortest path with minimum eccentricity  $k$  for several graph classes. For graphs with tree-length  $\lambda$ , this can require up-to  $\mathcal{O}(n^{3\lambda+3})$  time. In this section, we show that, for graphs with tree-length  $\lambda$ , we can find a shortest path with eccentricity at most  $k + 2.5\lambda$  in at most  $\mathcal{O}(\lambda m)$  time and, for graphs with hyperbolicity  $\delta$ , we can find a shortest path with eccentricity at most  $k + \mathcal{O}(\delta \log n)$  in at most  $\mathcal{O}(\delta m)$  time.

**Lemma 6.16.** *Let  $G$  be a graph with hyperbolicity  $\delta$ . Two vertices  $x$  and  $y$  in  $G$  with  $\text{ecc}(x) = \text{ecc}(y) = d(x, y)$  can be found in  $\mathcal{O}(\delta m)$  time.*

*Proof.* Let  $u$  and  $v$  be two vertices in  $G$  such that  $d(u, v) = \text{diam}(G)$ . For an arbitrary vertex  $x_0$  and for  $i \geq 0$ , let  $y_i = x_{i+1}$  be vertices in  $G$  such that  $d(x_i, y_i) = \text{ecc}(x_i)$  and  $d(x_i, y_i) < d(x_{i+1}, y_{i+1})$ . To prove Lemma 6.16, we show that there is no vertex  $y_{2\delta+1}$ .

Because  $d(x_0, y_0) = \text{ecc}(x_0)$ ,  $d(x_0, y_0) \geq \max\{d(x_0, u), d(x_0, v)\}$ . Therefore, by Lemma 2.10 (page 9),  $d(u, v) \leq \max\{d(u, y_0), d(v, y_0)\} + 2\delta$  and, thus,  $\text{diam}(G) \leq \text{ecc}(x_1) + 2\delta$ . Since  $d(x_i, y_i) < d(x_{i+1}, y_{i+1})$ , there is no vertex  $y_j$  with  $j \geq 2\delta + 1$ , otherwise  $d(x_j, y_j) > \text{diam}(G)$ . Therefore, a vertex pair  $x, y$  with  $\text{ecc}(x) = \text{ecc}(y) = d(x, y)$  can be found in  $\mathcal{O}(\delta m)$  time as follows: Pick an arbitrary vertex  $x_0$  and find a vertex  $x_1$  with  $d(x_0, x_1) = \text{ecc}(x_0)$  using a BFS. Next, find a vertex  $x_2$  such that  $d(x_1, x_2) = \text{ecc}(x_1)$ . Repeat this (at most  $2\delta$  times) until  $d(x_i, x_{i+1}) = \text{ecc}(x_i) = \text{ecc}(x_{i+1})$ .  $\square$

Recall that, if a graph has tree-length  $\lambda$ , its hyperbolicity is at most  $\lambda$  (Theorem 2.4, page 9). Thus, it follows:

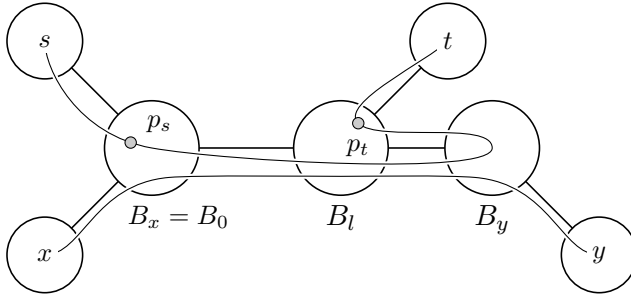
**Corollary 6.13.** *Let  $G$  be a graph with tree-length  $\lambda$ . Two vertices  $x$  and  $y$  in  $G$  with  $\text{ecc}(x) = \text{ecc}(y) = d(x, y)$  can be found in  $\mathcal{O}(\lambda m)$  time.*

The next lemma shows that, in a graph with bounded tree-length, a shortest path between two mutually furthest vertices gives an approximation for MESP.

**Lemma 6.17.** *Let  $G$  be a graph with tree-length  $\lambda$  having a shortest path with eccentricity  $k$ . Also, let  $x$  and  $y$  be two mutually furthest vertices, i. e.,  $\text{ecc}(x) = \text{ecc}(y) = d(x, y)$ . Then, each shortest path from  $x$  to  $y$  has eccentricity less than or equal to  $k + 2.5\lambda$ .*

*Proof.* Let  $P$  be a shortest path from  $s$  to  $t$  with eccentricity  $k$  and  $Q$  be a shortest path from  $x$  to  $y$ . Consider a tree-decomposition  $\mathcal{T}$  for  $G$  with length  $\lambda$ . We distinguish between two cases: (1) There is a bag in  $\mathcal{T}$  containing a vertex of  $P$  and a vertex of  $Q$  and (2) there is no such bag in  $\mathcal{T}$ .

*Case 1:* There is a bag in  $\mathcal{T}$  containing a vertex of  $P$  and a vertex of  $Q$ . We define bags  $B_x$  and  $B_y$  as follows: Both contain a vertex of  $P$  and a vertex of  $Q$ ,  $B_x$  is a bag closest to a bag containing  $x$ ,  $B_y$  is a bag closest to a bag containing  $y$ , and the distance between  $B_x$  and  $B_y$  in  $\mathcal{T}$  is maximal. Let  $\{B_0, B_1, \dots, B_l\}$  be a subpath of the shortest path from  $B_x$  to  $B_y$  in  $\mathcal{T}$  such that  $B_0$  is a bag closest to a bag containing  $s$ ,  $B_l$  is a bag closest to a bag containing  $t$ ,  $B_i$  is adjacent to  $B_{i+1}$  in  $\mathcal{T}$  ( $0 \leq i < l$ ), and the distance  $l$  between  $B_0$  and  $B_l$  is maximal. Without loss of generality, let  $d_{\mathcal{T}}(B_x, B_0) \leq d_{\mathcal{T}}(B_x, B_l)$ . Let  $p_s$  be the vertex in  $B_0 \cap P$  which is closest to  $s$  in  $G$  and let  $p_t$  be the vertex in  $B_l \cap P$  which is closest to  $t$  in  $G$ . Figure 6.4 gives an illustration.



**Figure 6.4.** Example for a possible tree-decomposition.

*Claim 1.* For each vertex  $p \in P$  with  $d_G(s, p_s) \leq d_G(s, p) \leq d_G(s, p_t)$ ,  $d_G(p, Q) \leq 1.5\lambda$ .

*Proof (Claim).* There is a vertex set  $\{p_s = p_0, p_1, \dots, p_l, p_{l+1} = p_t\} \subseteq P$ , where  $p_i \in B_{i-1} \cap B_i$  for all positive  $i \leq l$ . Because  $p_i, p_{i+1} \in B_i$  for  $0 \leq i \leq l$ ,  $d_G(p_i, p_{i+1}) \leq \lambda$ . Thus, because  $P$  is a shortest path, for all  $p' \in P$  with  $d_G(s, p_s) \leq d_G(s, p') \leq d_G(s, p_t)$  there is a vertex  $p_i$  with  $0 \leq i \leq l + 1$  such that  $d_G(p_i, p') \leq 0.5\lambda$ . By definition of  $\mathcal{T}$ , each bag  $B_i$  ( $0 \leq i \leq l$ ) contains a vertex  $q \in Q$ , i. e.,  $d_G(p_i, Q) \leq \lambda$  ( $0 \leq i \leq l + 1$ ). Therefore, for all  $p' \in P$  with  $d_G(s, p_s) \leq d_G(s, p') \leq d_G(s, p_t)$  there is a vertex  $p_i$  with  $0 \leq i \leq l + 1$  such that  $d_G(p', Q) \leq d_G(p_i, p') + d_G(p_i, Q) \leq 1.5\lambda$ .  $\diamond$

Consider an arbitrary vertex  $v$  in  $G$ . Let  $v'$  be a vertex in  $P$  closest to  $v$  and let  $P_v$  be a shortest path from  $v$  to  $v'$ . If  $v'$  is between  $p_s$  and  $p_t$ , i. e.,  $d_G(s, p_s) \leq d_G(s, v') \leq d_G(s, p_t)$ , then, by Claim 1,  $d_G(v, Q) \leq d_G(v, v') + d_G(v', Q) \leq k + 1.5\lambda$ . If  $P_v$  intersects a bag containing a vertex  $q \in Q$ ,  $d_G(v, Q) \leq k + \lambda$ .

Next, consider the case when  $P_v$  does not intersect a bag containing a vertex of  $Q$  and (without loss of generality)  $d_G(s, v') > d_G(s, p_t)$ . In this case, each path from  $x$  to  $v$  intersects  $B_l$ .

*Claim 2.* *There is a vertex  $u \in B_l$  such that  $d_G(u, y) \leq k + 0.5\lambda$ .*

*Proof (Claim).* Let  $y'$  be a vertex in  $P$  that is closest to  $y$  and let  $P_y$  be a shortest path from  $y$  to  $y'$ . If  $P_y$  intersects  $B_l$ , there is a vertex  $u \in P_y \cap B_l$  with  $d_G(y, u) \leq k$ .

If  $P_y$  does not intersect  $B_l$ , then there is a subpath of  $P$  starting at  $p_t$ , containing  $y'$ , and ending in a vertex  $p_l \in B_l$ . Because  $d_G(p_t, p_l) \leq \lambda$ ,  $d_G(y', \{p_t, p_l\}) \leq 0.5\lambda$ . Therefore,  $d_G(y, \{p_t, p_l\}) \leq d_G(y, y') + d_G(y', \{p_t, p_l\}) \leq k + 0.5\lambda$ .  $\diamond$

Let  $u, v'$ , and  $z$  be vertices in  $B_l$  such that  $d_G(u, y) \leq k + 0.5\lambda$ ,  $v'$  is on a shortest path from  $x$  to  $v$ , and  $z \in Q$ . Because  $d_G(x, y) = \text{ecc}(x)$ ,  $d_G(x, v') + d_G(v', v) \leq d_G(x, y)$ . Also, by the triangle inequality,  $d_G(x, y) \leq d_G(x, v') + d_G(v', y)$  and  $d_G(v', y) \leq d_G(v', u) + d_G(u, y)$ . Because  $\{u, v', z\} \subseteq B_l$  and  $d_G(u, y) \leq k + 0.5\lambda$ ,  $d_G(v', v) \leq k + 1.5\lambda$  and therefore  $d_G(z, v) \leq k + 2.5\lambda$ .

Thus, if there is a bag in  $\mathcal{T}$  containing a vertex of  $P$  and a vertex of  $Q$ , then  $d_G(v, Q) \leq k + 2.5\lambda$  for all vertices  $v$  in  $G$ .

*Case 2: There is no bag in  $\mathcal{T}$  containing vertices of  $P$  and  $Q$ .* Because there is no such bag,  $\mathcal{T}$  contains a bag  $B$  such that each path from  $x$  and  $y$  to  $P$  intersects  $B$  and there is a vertex  $z \in B \cap Q$ .

Consider an arbitrary vertex  $v$ . If there is a shortest path  $P_v$  from  $v$  to  $P$  which intersects  $B$ , then  $d_G(z, v) \leq k + \lambda$ . If there is no such path, each path from  $x$  to  $v$  intersects  $B$ . Let  $v' \in B$  be a vertex on a shortest path from  $x$  to  $v$  and let  $u \in B$  be a vertex on a shortest path from  $y$  to  $P$ . Note that  $d_G(u, y) \leq k$ .

Because  $d_G(x, y) = \text{ecc}(x)$ ,  $d_G(x, v') + d_G(v', v) \leq d_G(x, y)$ . Also, by the triangle inequality,  $d_G(x, y) \leq d_G(x, v') + d_G(v', y)$  and  $d_G(v', y) \leq d_G(v', u) + d_G(u, y)$ . Because  $\{u, v', z\} \subseteq B$  and  $d_G(u, y) < k$ ,  $d_G(v', v) < k + \lambda$  and therefore  $d_G(z, v) < k + 2\lambda$ .

Thus, if there is no bag in  $\mathcal{T}$  containing vertices of  $P$  and  $Q$ ,  $d_G(v, Q) < k + 2\lambda$  for all vertices  $v$  in  $G$ .  $\square$

Recall that a  $\delta$ -hyperbolic graph has tree-length at most  $\mathcal{O}(\delta \log n)$  (Theorem 2.4, page 9).

**Corollary 6.14.** *Let  $G$  be a graph with hyperbolicity  $\delta$  having a shortest path with eccentricity  $k$ . Also, let  $x$  and  $y$  be two mutually furthest vertices, i. e.,  $\text{ecc}(x) = \text{ecc}(y) = d(x, y)$ . Each shortest path from  $x$  to  $y$  has eccentricity less than or equal to  $k + \mathcal{O}(\delta \log n)$ .*

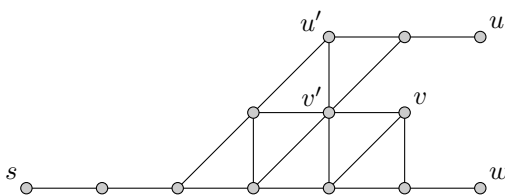
Lemma 6.16, Lemma 6.17, Corollary 6.13, and Corollary 6.14 imply our next result:

**Theorem 6.11.** *Let  $G$  be a graph having a shortest path with eccentricity  $k$ . If  $G$  has tree-length  $\lambda$ , a shortest path with eccentricity at most  $k+2.5\lambda$  can be found in  $\mathcal{O}(\lambda m)$  time. If  $G$  has hyperbolicity  $\delta$ , a shortest path with eccentricity at most  $k + \mathcal{O}(\delta \log n)$  can be found in  $\mathcal{O}(\delta m)$  time.*

Recall that a graph is chordal if and only if it has tree-length 1 (Theorem 2.1, page 8).

**Corollary 6.15.** *If  $G$  is a chordal graph and has a shortest path with eccentricity  $k$ , a shortest path in  $G$  with eccentricity at most  $k + 2$  can be found in linear time.*

Figure 6.5 gives an example that, for chordal graphs,  $k + 2$  is a tight upper bound for the eccentricity of the determined shortest path.



**Figure 6.5.** Example for Corollary 6.15. For the shown chordal graph  $G$ , a shortest path from  $s$  to  $v$  passing  $v'$  has eccentricity 2. This is the minimum for all shortest paths in  $G$ . The diametral path from  $s$  to  $u$  passing  $u'$  has eccentricity 4 because of its distance to  $w$ .

### 6.3.3 Using Layering Partition

In this subsection, we present an algorithm to compute an approximation for MESP using the layering partition of a graph. For the remainder of this subsection, let  $G$  be a graph, let  $T$  be a layering partition of  $G$ , and let  $\Delta$  be the maximum cluster diameter of  $T$ . Additionally, let  $k_G$  and  $k_T$  be the minimum eccentricities of any shortest path in  $G$  and  $T$ , respectively.

First, we show that

$$k_T - \frac{1}{2}\Delta \leq k_G \leq k_T + \Delta.$$

**Lemma 6.18.**

$$k_T - \frac{1}{2}\Delta \leq k_G$$

*Proof.* Let  $P$  be a path in  $G$  with the start vertex  $s$ , the end vertex  $t$ , and with eccentricity  $k_G$ . Let  $T_P$  be the subtree of  $T$  induced by the clusters of  $T$  which intersect  $P$ . Let  $Q$  be the shortest path in  $T$  from  $C_s$  to  $C_t$ , where  $C_s$  and  $C_t$  are the clusters containing  $s$  and  $t$ , respectively. Clearly, by the properties of a layering partition, each cluster of  $Q$  is also a cluster of  $T_P$ .



*Claim 1.* For all  $p \in P$ ,  $d_G(p, Q) \leq \frac{1}{2}\Delta$ .

*Proof (Claim).* Let  $C_P$  be a cluster of  $T_P$  that is not part of  $Q$ . If no such cluster exists, then  $T_P = Q$  and, thus,  $d_G(p, Q) = 0$  for all  $p \in P$ . Additionally, let  $C_Q$  be the cluster of  $Q$  which is closest to  $C_P$  in  $T$  and  $p$  be a vertex of  $P$  in  $C_P$ .

Due to the properties of a layering partition, there are two vertices  $p_s, p_t \in P \cap C_Q$  with  $d_G(s, p_s) < d_G(s, p) < d_G(s, p_t)$ . Because  $P$  is a shortest path and the diameter of a cluster is at most  $\Delta$ ,  $d_G(p, \{p_s, p_t\}) \leq \frac{1}{2}\Delta$  and, therefore,  $d_G(p, Q) \leq \frac{1}{2}\Delta$ .  $\diamond$

Let  $v$  be an arbitrary vertex of  $G$  and  $v'$  be a vertex in  $P$  with minimal distance to  $v$ . By definition of  $P$ ,  $d_G(v, v') \leq k_G$ . Then, by triangle inequality,  $d_G(v, Q) \leq d_G(v, v') + d_G(v', Q)$ , and, by Claim 1,  $d_G(v, Q) \leq k_G + \frac{1}{2}\Delta$ . Recall that, for any two vertices  $u$  and  $v$  of  $G$ ,  $d_T(u, v) \leq d_G(u, v)$  (Lemma 2.6, page 7). Thus, for any vertex  $v$  of  $G$ ,  $d_T(v, Q) \leq k_G + \frac{1}{2}\Delta$ .

Let  $v$  be a vertex of  $G$  with maximal distance to  $Q$  in  $T$ , i. e.,  $d_T(v, Q) = \text{ecc}_T(Q)$ . Therefore, because  $k_T \leq \text{ecc}_T(Q)$ ,  $k_T \leq d_T(v, Q) \leq k_G + \frac{1}{2}\Delta$ .  $\square$

**Lemma 6.19.**

$$k_G \leq k_T + \Delta$$

*Proof.* Let  $Q$  be a path in  $T$  with minimum eccentricity, i. e.,  $\text{ecc}_T(Q) = k_T$ , such that  $Q$  starts at the cluster  $C_s$  and ends at the cluster  $C_t$ . Additionally, let  $s$  and  $t$  be two vertices of  $G$  with  $s \in C_s$  and  $t \in C_t$  and let  $P$  be a shortest path from  $s$  to  $t$  in  $G$ .

Consider a vertex  $v$  of  $G$  with maximal distance to  $P$  in  $G$ . Hence,  $k_G \leq \text{ecc}_G(P) = d_G(v, P)$ . Because  $d_G(u, v) \leq d_T(u, v) + \Delta$  for each vertex  $u$  of  $G$  (Lemma 2.6, page 7), it follows that  $d_G(v, P) \leq d_T(v, P) + \Delta$ . Note that, by the properties of a layering partition,  $P$  intersects all clusters of  $Q$ . Thus,  $d_T(v, P) \leq d_T(v, Q)$ . Additionally, by definition of  $Q$ ,  $d_T(v, Q) \leq \text{ecc}_T(Q) = k_T$ . From combining these observation, it follows that

$$k_G \leq \text{ecc}_G(P) = d_G(v, P) \leq d_T(v, P) + \Delta \leq d_T(v, Q) + \Delta \leq \text{ecc}_T(Q) + \Delta = k_T + \Delta. \quad \square$$

Based on Lemma 6.18 and Lemma 6.19, we can compute an approximation for MESP in linear time.

**Theorem 6.12.** For a graph  $G$  with minimum eccentricity  $k$ , a shortest path  $P$  with eccentricity at most  $k + \frac{3}{2}\Delta$  can be computed in linear time.

*Proof.* First, construct a layering partition  $T$  for  $G$ , starting at an arbitrary vertex. Next, find a path  $Q$  in  $T$  with minimum eccentricity. Let  $C_s$  and  $C_t$  be the start and end clusters of  $Q$ . Pick two arbitrary vertices  $s \in C_s$  and  $t \in C_t$ . Then, compute a shortest path  $P$  from  $s$  to  $t$  in  $G$  and output it. Note that each of these steps can be done in linear time, including the construction of  $T$  (Lemma 2.5, page 7) as well as finding  $Q$  in  $T$  (Corollary 6.6).

Note that the construction of  $P$  is the same as in the proof of Lemma 6.19. Therefore, as shown in the proof,  $\text{ecc}_G(P) \leq \text{ecc}_T(Q) + \Delta$  and, by Lemma 6.18,  $\text{ecc}_G(P) \leq k + \frac{3}{2}\Delta$ .  $\square$

## 6.4 Relation to Other Parameters

Similar to path-length, one can see the minimum eccentricity  $k$  of all shortest paths of a graph  $G$  as a parameter describing the structure of  $G$ . In this section, we show how  $k$  relates to the path-length of  $G$  and other parameters.

**Theorem 6.13.** *Let  $G$  be a graph with path-length  $\lambda$  and path-breadth  $\rho$  and let the minimum eccentricity of all shortest paths be  $k$ . Then,*

$$k \leq \lambda \leq 4k + 1 \quad \text{and} \quad \frac{1}{2}k \leq \rho \leq 2k + 1.$$

*Proof.* As shown in Lemma 5.1 (page 56), it clearly follows from the definition of path-length that a graph with path-length  $\lambda$  has a shortest path with eccentricity at most  $\lambda$ . Therefore,  $k \leq \lambda$  and, since  $\lambda \leq 2\rho$ ,  $\frac{1}{2}k \leq \rho$ .

Next, let  $P$  be a shortest path in  $G$  with the start vertex  $s$  and  $\text{ecc}(P) = k$ . As shown in Lemma 6.14, the radius of  $L_i^{(s)}$  is at most  $2k$ . Thus, its diameter is not larger than  $4k$ . One can now create a path decomposition  $\mathcal{X} = \{X_1, \dots, X_n\}$  by creating a set  $X_i$  including layer  $L_i^{(s)}$  and all vertices from  $L_{i-1}^{(s)}$  which are adjacent to vertices in  $L_i^{(s)}$ , i. e.,  $X_i = L_i^{(s)} \cup \{u \mid uv \in E, u \in L_{i-1}^{(s)}, v \in L_i^{(s)}\}$ . It is easy to see that the radius of  $X_i$  is at most  $2k + 1$ , the diameter of  $X_i$  is at most  $4k + 1$ , and that  $\mathcal{X}$  is a valid path decomposition for  $G$ , i. e.,  $\rho \leq 2k + 1$  and  $\lambda \leq 4k + 1$ .  $\square$

Recently, VÖLKELE et al. [92] defined a closely related problem called  $k$ -Laminarity problem. It asks if a given graph contains a diametral path with eccentricity at most  $k$ . If a graph contains such a path, it is called  $k$ -laminar. Similar, if every diametral path of a graph has eccentricity at most  $k$ , it is called  $k$ -strongly laminar.

**Theorem 6.14 (Birmelé et al. [10]).** *Let  $G$  be a graph where the minimum eccentricity of all shortest paths is  $k$ . Additionally, let  $l$  and  $s$  be the minimal values such that  $G$  is  $l$ -laminar and  $s$ -strongly laminar. Then,*

$$k \leq l \leq 4k - 2 \quad \text{and} \quad k \leq s \leq 4k.$$

*These bounds are tight.*

## 6.5 Solving $k$ -Domination using a MESP

In [68], an  $\mathcal{O}(n^7)$  time algorithm is presented which finds a minimum dominating set for graphs containing a shortest path with eccentricity 1. Using a similar approach, we generalise this result to find a minimum  $k$ -dominating set for graphs containing a shortest path with eccentricity  $k$ .

Recall that, for a graph  $G = (V, E)$ , a vertex set  $D$  is a  $k$ -dominating set if  $N^k[D] = V$ . Additionally,  $D$  is a *minimum*  $k$ -dominating set if there is no  $k$ -dominating set  $D'$  for  $G$  with  $|D'| < |D|$ .

**Lemma 6.20.** *Let  $D$  be a minimum  $k$ -dominating set of a graph  $G$  and let  $G$  have a shortest path of eccentricity at most  $k$  starting at a vertex  $s$ . Then, for all non-negative integers  $i \leq \text{ecc}(s)$ ,*

$$\left| D \cap \bigcup_{l=i-k}^{i+k} L_l^{(s)} \right| \leq 6k + 1.$$

*Proof.* Let  $P = \{s = v_0, v_1, \dots, v_j\}$  be a shortest path with  $j \leq \text{ecc}(s)$  and  $\text{ecc}(P) \leq k$ . Also, let  $D_i = D \cap \bigcup_{l=i-k}^{i+k} L_l^{(s)}$  be a set of  $k$ -dominating vertices in the layers  $L_{i-k}^{(s)}$  to  $L_{i+k}^{(s)}$ . Because  $D$  is  $k$ -dominating,  $D_i$  can only  $k$ -dominate vertices in the layers  $L_{i-2k}^{(s)}$  to  $L_{i+2k}^{(s)}$ . By Lemma 6.1, these layers are also  $k$ -dominated by  $P_i = \{v_{i-3k}, \dots, v_{i+3k}\}$ . Thus,

$$N^k[D_i] \subseteq \bigcup_{l=i-2k}^{i+2k} L_l^{(s)} \subseteq N^k[P_i].$$

Assume that  $|D_i| > |P_i|$ . Note that  $|P_i| \leq 6k + 1$ . Then, there is a  $k$ -dominating set  $D' = (D \setminus D_i) \cup P_i$  such that  $|D| > |D'|$ . Thus,  $D$  is not a minimum  $k$ -dominating set.  $\square$

Based on Lemma 6.20, Algorithm 6.6 below computes a minimum  $k$ -dominating set for a given graph  $G = (V, E)$  with a shortest path of eccentricity  $k$  starting at a vertex  $s$  as follows. In the  $i$ -th iteration, the algorithm knows all vertex sets  $S$  for which there is a vertex set  $S'$  such that (i)  $S = S' \cap \left( L_{i-k}^{(s)} \cup \dots \cup L_{i-1+k}^{(s)} \right)$ , (ii) the set  $S^* = S \cup \left( S' \cap L_{i-1-k}^{(s)} \right)$   $k$ -dominates  $L_{i-1}^{(s)}$  and has cardinality at most  $6k + 1$ , and (iii)  $S'$   $k$ -dominates the layers  $L_0^{(s)}$  to  $L_{i-1}^{(s)}$ . Due to Lemma 6.20, a set  $S^*$  with a larger cardinality cannot be a subset of a minimum dominating set of  $G$  and, hence, neither can be  $S$  or  $S'$ . Each such set  $S$  is stored as a pair  $(S, S')$  in a set  $X_{i-1}$  where  $S'$  is a corresponding set with minimum cardinality, i. e.,  $X_{i-1}$  does not contain two pairs  $(S, S')$  and  $(T, T')$  with  $S = T$ . Note that, since  $S'$  has minimum cardinality, it does not contain any vertices from any layer  $L_j^{(s)}$  with  $j > i - 1 + k$ . We show later that, this way,  $X_{i-1}$  always contains a pair  $(S, S')$  such that  $S'$  is subset of some minimum  $k$ -dominating set.

Then, for each pair  $(S, S') \in X_{i-1}$ , the algorithm computes all sets  $S \cup U$  which  $k$ -dominate the layer  $L_i^{(s)}$  and have cardinality at most  $6k + 1$ . For such a set, the sets  $R = (S \cup U) \setminus L_{i-k}^{(s)}$  and  $R' = S' \cup U$  are created and, if the set  $X_i$  does not contain a pair  $(P, P')$  with  $P = R$ , stored as the pair  $(R, R')$  in  $X_i$ . In the case that  $X_i$  already contains such a pair  $(P, P')$ , either, if  $|R'| < |P'|$ ,  $(P, P')$  is replaced by  $(R, R')$  or, if  $|R'| \geq |P'|$ ,  $(R, R')$  is not added to  $X_i$ .

Note that  $L_{i+k}^{(s)} = \emptyset$  for  $i > \text{ecc}(s) - k$ . Therefore, the algorithm can stop after  $\text{ecc}(s) - k$  iterations.

**Theorem 6.15.** *For a given graph  $G$  and a vertex  $s$  which is start vertex of a shortest path with eccentricity  $k$ , Algorithm 6.6 determines a minimum  $k$ -dominating set in  $n^{O(k)}$  time.*

---

**Algorithm 6.6:** Determines a minimum  $k$ -dominating set in a given graph  $G$  containing a shortest path of eccentricity  $k$  starting at  $s$ .

---

**Input:** A graph  $G$ , an integer  $k$ , and a vertex  $s$  which is start vertex of a shortest path with eccentricity  $k$ .

**Output:** A minimum  $k$ -dominating set.

- 1 Compute the layers  $L_0^{(s)}, L_1^{(s)}, \dots, L_{\text{ecc}(s)}^{(s)}$ .
  - 2 Create the set  $X_0 := \left\{ (S, S') \mid S \subseteq N^k[s]; 0 < |S| \leq 6k + 1 \right\}$ .
  - 3 **For**  $i := 1$  **To**  $\text{ecc}(s) - k$ 
    - 4 Create  $X_i := \emptyset$ .
    - 5 **For Each**  $(S, S') \in X_{i-1}$ 
      - 6 **For Each**  $U \subseteq L_{i+k}^{(s)}$  with  $|S \cup U| \leq 6k + 1$ 
        - 7 **If**  $N^k[S \cup U] \supseteq L_i^{(s)}$  **Then**
          - 8  $R := (S \cup U) \setminus L_{i-k}^{(s)}$
          - 9  $R' := S' \cup U$
          - 10 **If** There is no pair  $(P, P') \in X_i$  with  $P = R$  **Then**
            - 11  $\lfloor$  Insert  $(R, R')$  into  $X_i$ .
          - 12 **If** There is a pair  $(P, P') \in X_i$  with  $P = R$  and  $|R'| < |P'|$  **Then**
            - 13  $\lfloor$  Replace  $(P, P')$  in  $X_i$  by  $(R, R')$ .
  - 14 Among all pairs  $(S, S') \in X_{\text{ecc}(s)-k}$  for which  $S'$   $k$ -dominates  $G$ , determine one with minimum  $|S'|$ , say  $(D, D')$ .
  - 15 Output  $D'$ .
- 

*Proof (Correctness).* To prove the correctness, we show by induction that, for each  $i$  with  $0 \leq i \leq \text{ecc}(s) - k$ , there is a minimum  $k$ -dominating set  $D$  and a pair  $(S, S') \in X_i$  such that  $S' = D \cap \bigcup_{l=0}^{i+k} L_l^{(s)}$ . If this is true for  $i = \text{ecc}(s) - k$ , then  $S' = D$ . Hence, if  $(S, S')$  is a pair in  $X_{\text{ecc}(s)-k}$  such that  $S'$   $k$ -dominates  $G$  and has minimum cardinality, then  $S'$  is a minimum  $k$ -dominating set of  $G$ .

By construction in line 2,  $X_0$  contains all pairs  $(S, S')$  such that  $S'$  is a vertex set with cardinality at most  $6k + 1$  which  $k$ -dominates  $L_0^{(s)}$ . Thus, the base case is true. Next, by induction hypothesis and by definition of the pairs  $(S, S')$ , there is a minimum dominating set  $D$  and a pair  $(S, S') \in X_{i-1}$  such that  $S = S' \cap \bigcup_{l=i-k}^{i+k-1} L_l^{(s)} = D \cap \bigcup_{l=i-k}^{i+k-1} L_l^{(s)}$ . Let  $M = D \cap L_{i+k}^{(s)}$ . By Lemma 6.20,  $\left| D \cap \bigcup_{l=i-k}^{i+k} L_l^{(s)} \right| = |S \cup M| \leq 6k + 1$ . Therefore, there is an iteration of the loop starting in line 6 with  $U = M$ . Because  $S \cup M = D \cap \bigcup_{l=i-k}^{i+k} L_l^{(s)}$ ,  $S \cup M$   $k$ -dominates  $L_i^{(s)}$ , i. e.,  $N^k[S \cup M] \supseteq L_i^{(s)}$ . Thus, the algorithm creates a pair  $(R, R')$  with  $R' = D \cap \bigcup_{l=0}^{i+k} L_l^{(s)}$  (see line 7 to line 9).

Assume that there is a pair  $(P, P') \neq (R, R')$  with  $P = R$  and  $|P'| \leq |R'|$ , i. e.,  $(R, R')$  will not be stored in  $X_i$  or replaced by  $(P, P')$  (see line 10 to line 13). Because

$P = R$ ,  $P' \cap \bigcup_{l=i-k+1}^{i+k} L_l^{(s)} = D \cap \bigcup_{l=i-k}^{i+k} L_l^{(s)}$ . Let  $D' = P' \cup \left( D \cap \bigcup_{l=i-k+1}^{\text{ecc}(s)-k} L_l^{(s)} \right)$ . By definition,  $P'$   $k$ -dominates  $\bigcup_{l=0}^i L_l^{(s)}$ . Thus,  $D'$   $k$ -dominates  $\bigcup_{l=0}^i L_l^{(s)}$ . Note that  $D' \supseteq D \cap \left( \bigcup_{l=i-k+1}^{\text{ecc}(s)} L_l^{(s)} \right)$ . Thus,  $D'$  also  $k$ -dominates  $\bigcup_{l=i+1}^{\text{ecc}(s)} L_l^{(s)}$ . Therefore,  $D'$  is a minimum  $k$ -dominating set and there is a pair  $(P, P') \in X_i$  such that  $P' = D' \cap \bigcup_{l=0}^{i+k} L_l^{(s)}$ .  $\square$

*Proof (Complexity).* For a given  $i$ , there are no two pairs  $(S, S')$  and  $(T, T')$  in  $X_i$  with  $S = T$  (see line 10 to line 13). Thus, for each set  $U \subseteq L_{i+k}^{(s)}$ ,  $S \cup U \neq T \cup U$ . Additionally, since  $S$  and  $T$  intersect at most  $2k$  consecutive layers,  $S \neq T$  for all pairs  $(S, S') \in X_i$  and  $(T, T') \in X_j$  with  $|i - j| \geq 2k$ . Therefore, a set  $S \cup U$  is processed at most  $\mathcal{O}(k)$  times by the loop starting in line 6. Hence, because there are at most  $n^{6k+1}$  sets  $S \cup U$  with  $|S \cup U| \leq 6k + 1$ , the loop starting in line 6 has at most  $\mathcal{O}(n^{6k+1}k)$  iterations.

Next, we show that a single iteration of the loop starting in line 6 requires at most  $\mathcal{O}(m)$  time. It takes at most  $\mathcal{O}(m)$  time to check if  $N^k[S \cup U] \supseteq L_i^{(s)}$  (line 7) and at most  $\mathcal{O}(n)$  time to construct  $(R, R')$ . Determining if  $X_i$  contains a pair  $(P, P')$  with  $P = R$  and (if necessary) replacing it can be achieved in  $\mathcal{O}(n)$  time as follows. One way is to use a tree-structure similar to the one we used for Algorithm 6.1. An other (less memory efficient) option is to use an array  $A_i$  of size  $n^{6k+1}$  where each element can store a pair  $(S, S')$ . To determine the index of a pair, assume that each vertex of  $G$  has a unique identifier in the range from 0 to  $n - 1$ . Additionally, assume that the vertices in a set  $S$  are ordered by their identifier. Therefore, each set  $S$  can be represented by a unique  $(6k + 1)$ -digit number with base  $n$ . This number is the index of a pair  $(S, S')$  in  $A_i$ . Hence, it takes at most  $\mathcal{O}(n)$  time to add  $(R, R')$  to  $X_i$  and (if necessary) replace a pair  $(P, P')$ .

Therefore, the total runtime of the algorithm is  $\mathcal{O}(n^{6k+1}km)$ .  $\square$

If the start vertex  $s$  is unknown, one can use Algorithm 6.1 to, first, find a shortest path with eccentricity  $k$  and, then, use Algorithm 6.6 to find a minimum  $k$ -dominating set.

## 6.6 Open Questions

**Finding Start and End Vertex.** The Minimum Eccentricity Shortest Path problem can be naturally split into two subproblems. First, find the start and end vertices of an optimal path. Second, for a given vertex pair, find a shortest path between them with the minimum eccentricity. We know that the second subproblem remains NP-hard (Corollary 6.2). However, is it possible to determine the start and end vertices of an optimal path efficiently?

**APX-Complete?** In Section 6.3.1, we show multiple algorithms which compute a constant factor approximation in linear or polynomial time. Therefore, MESP is in APX. It remains an open question if the problem is APX-complete.

**Complexity for Planar Graphs.** We show that MESP is NP-complete for planar graphs (Corollary 6.1) and remains  $W[2]$ -hard for sparse graphs (Corollary 6.3). Additionally, we show that the problem is Fixed Parameter Tractable for graphs with bounded degree (Corollary 6.5). Since every induced subgraph of a planar graphs has an average degree less than 6, we conjecture that there is a Fixed Parameter Tractable algorithm to solve MESP on planar graphs.

**Computing Projection Gap.** In Section 6.2.3, we show that we can solve MESP in polynomial time, if the projection gap of a given graph bounded by a constant. Additionally, we determine an upper bound for the projection gap of some graph classes. It would be interesting to know how to compute the projection gap for a given graph.

---

# Bibliography

- [1] Abboud, A., Williams, V. V., Wang, J. R.: Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs. In: SODA. 377–391. SIAM, 2016.
- [2] Abu-Ata, M., Dragan, F. F.: Metric tree-like structures in real-life networks: an empirical study. *Networks* 67 (1), 49–68, 2016.
- [3] Arnborg, S., Corneil, D., Proskurowski, A.: Complexity of Finding Embeddings in a  $k$ -Tree. *SIAM Journal on Algebraic Discrete Methods* 8 (2), 277–284, 1987.
- [4] Assmann, S. F., Peck, G. W., Sysło, M. M., Zak, J.: The Bandwidth of Caterpillars with Hairs of Length 1 and 2. *SIAM Journal on Algebraic Discrete Methods* 2 (4), 387–393, 1981.
- [5] Bandelt, H.-J., Dress, A.: Reconstructing the shape of a tree from observed dissimilarity data. *Advances in Applied Mathematics* 7 (3), 309–343, 1986.
- [6] Bandelt, H.-J., Mulder, H. M.: Distance-hereditary graphs. *Journal of Combinatorial Theory, Series B* 41 (2), 182–208, 1986.
- [7] Belmonte, R., Fomin, F. V., Golovach, P. A., Ramanujan, M. S.: Metric Dimension of Bounded Tree-length Graphs. CoRR abs/1602.02610, 2016.
- [8] de Berg, M., Khosravi, A.: Optimal Binary Space Partitions in the Plane. In: COCOON. *Lecture Notes in Computer Science*, vol. 6196, 216–225. Springer, 2010.
- [9] Berge, C.: *Hypergraphs: Combinatorics of Finite Sets*. North-Holland Mathematical Library, Elsevier Science, 1984.
- [10] Birmelé, E., de Montgolfier, F., Planche, L.: Minimum Eccentricity Shortest Path Problem: An Approximation Algorithm and Relation with the  $k$ -Laminarity Problem. In: *Combinatorial Optimization and Applications*. *Lecture Notes in Computer Science*, vol. 10043, 216–229. Springer, 2016.
- [11] Blache, G., Karpinski, M., Wirtgen, J.: On Approximation Intractability of the Bandwidth Problem. *Electronic Colloquium on Computational Complexity (ECCC)* 5 (14), 1998.

- 
- [12] Bodlaender, H.: A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM Journal on Computing* 25 (6), 1305–1317, 1996.
- [13] Booth, K. S., Johnson, J. H.: Dominating Sets in Chordal Graphs. *SIAM Journal on Computing* 11 (1), 191–199, 1982.
- [14] Brandstädt, A., Chepoi, V., Dragan, F. F.: The Algorithmic Use of Hypertree Structure and Maximum Neighbourhood Orderings. *Discrete Applied Mathematics* 82 (1–3), 43–77, 1998.
- [15] Brandstädt, A., Chepoi, V., Dragan, F. F.: Distance Approximating Trees for Chordal and Dually Chordal Graphs. *Journal of Algorithms* 30 (1), 166–184, 1999.
- [16] Brandstädt, A., Dragan, F. F., Chepoi, V., Voloshin, V. I.: Dually Chordal Graphs. *SIAM Journal on Discrete Mathematics* 11 (3), 437–455, 1998.
- [17] Bădoiu, M., Chuzhoy, J., Indyk, P., Sidiropoulos, A.: Low-distortion embeddings of general metrics into the line. In: *STOC*. 225–233. ACM, 2005.
- [18] Bădoiu, M., Dhamdhere, K., Gupta, A., Rabinovich, Y., Räcke, H., Ravi, R., Sidiropoulos, A.: Approximation algorithms for low-distortion embeddings into low-dimensional spaces. In: *SODA*. 119–128. SIAM, 2005.
- [19] Chepoi, V., Dragan, F. F.: A Note on Distance Approximating Trees in Graphs. *European Journal of Combinatorics* 21 (6), 761–766, 2000.
- [20] Chepoi, V., Dragan, F. F., Estellon, B., Habib, M., Vaxès, Y.: Diameters, Centers, and Approximating Trees of  $\delta$ -Hyperbolic Geodesic Spaces and Graphs. In: *Symposium on Computational Geometry*. 59–68. ACM, 2008.
- [21] Chepoi, V., Estellon, B.: Packing and Covering  $\delta$ -Hyperbolic Spaces by Balls. In: *APPROX-RANDOM*. Lecture Notes in Computer Science, vol. 4627, 59–73. Springer, 2007.
- [22] Chlebík, M., Chlebíková, J.: Approximation hardness of dominating set problems in bounded degree graphs. *Information and Computation* 206 (11), 1264–1275, 2008.
- [23] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: *Introduction to Algorithms* (3. ed.). MIT Press, 2009.
- [24] Corneil, D. G., Olariu, S., Stewart, L.: Asteroidal Triple-Free Graphs. *SIAM Journal on Discrete Mathematics* 10 (3), 399–430, 1997.
- [25] Corneil, D. G., Olariu, S., Stewart, L.: Linear Time Algorithms for Dominating Pairs in Asteroidal Triple-free Graphs. *SIAM Journal on Computing* 28 (4), 1284–1297, 1999.



- 
- [26] Coudert, D., Ducoffe, G., Nisse, N.: To Approximate Treewidth, Use Treelength! *SIAM Journal on Discrete Mathematics* 30 (3), 1424–1436, 2016.
- [27] Damiand, G., Habib, M., Paul, C.: A simple paradigm for graph recognition: application to cographs and distance hereditary graphs. *Theoretical Computer Science* 263 (1–2), 99–111, 2001.
- [28] Deogun, J.S., Kratsch, D.: Diametral Path Graphs. In: WG. *Lecture Notes in Computer Science*, vol. 1017, 344–357. Springer, 1995.
- [29] Diestel, R.: *Graph Theory*. Springer-Verlag, 1997.
- [30] Dirac, G.: On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 25, 71–76, 1961.
- [31] Dourisboure, Y.: Compact Routing Schemes for Generalised Chordal Graphs. *Journal of Graph Algorithms and Applications* 9 (2), 277–297, 2005.
- [32] Dourisboure, Y., Dragan, F.F., Gavaille, C., Yan, C.: Spanners for bounded tree-length graphs. *Theoretical Computer Science* 383 (1), 34–44, 2007.
- [33] Dourisboure, Y., Gavaille, C.: Tree-decompositions with bags of small diameter. *Discrete Mathematics* 307 (16), 2008–2029, 2007.
- [34] Downey, R. G., Fellows, M. R.: *Fundamentals of Parameterized Complexity*. Springer, 2013.
- [35] Dragan, F.F.: HT-graphs: centers, connected  $r$ -domination and Steiner trees. *Computer Science Journal of Moldova* 1 (2), 64–83, 1993.
- [36] Dragan, F.F., Abu-Ata, M.: Collective additive tree spanners of bounded tree-breadth graphs with generalizations and consequences. *Theoretical Computer Science* 547, 1–17, 2014.
- [37] Dragan, F.F., Köhler, E.: An Approximation Algorithm for the Tree  $t$ -Spanner Problem on Unweighted Graphs via Generalized Chordal Graphs. *Algorithmica* 69 (4), 884–905, 2014.
- [38] Dragan, F.F., Köhler, E., Leitert, A.: Line-Distortion, Bandwidth and Path-Length of a Graph. In: SWAT. *Lecture Notes in Computer Science*, vol. 8503, 158–169. Springer, 2014.
- [39] Dragan, F.F., Köhler, E., Leitert, A.: Line-Distortion, Bandwidth and Path-Length of a Graph. *Algorithmica* 77 (3), 686–713, 2017.
- [40] Dragan, F.F., Leitert, A.: Minimum Eccentricity Shortest Paths in Some Structured Graph Classes. In: WG. *Lecture Notes in Computer Science*, vol. 9224, 189–202. Springer, 2015.

- 
- [41] Dragan, F. F., Leitert, A.: On the Minimum Eccentricity Shortest Path Problem. In: WADS. Lecture Notes in Computer Science, vol. 9214, 276–288. Springer, 2015.
- [42] Dragan, F. F., Leitert, A.: Minimum Eccentricity Shortest Paths in some Structured Graph Classes. *Journal of Graph Algorithms and Applications* 20 (2), 299–322, 2016.
- [43] Dragan, F. F., Lomonosov, I.: On compact and efficient routing in certain graph classes. *Discrete Applied Mathematics* 155 (11), 1458–1470, 2007.
- [44] Dragan, F. F., Matamala, M.: Navigating in a Graph by Aid of Its Spanning Tree Metric. *SIAM Journal on Discrete Mathematics* 25 (1), 306–332, 2011.
- [45] Dragan, F. F., Nicolai, F.: LexBFS-orderings of Distance-hereditary Graphs with Application to the Diametral Pair Problem. *Discrete Applied Mathematics* 98 (3), 191–207, 2000.
- [46] Dubey, C. K., Feige, U., Unger, W.: Hardness results for approximating the bandwidth. *Journal of Computer and System Sciences* 77 (1), 62–90, 2011.
- [47] Ducoffe, G., Legay, S., Nisse, N.: On computing tree and path decompositions with metric constraints on the bags. CoRR abs/1601.01958, 2016.
- [48] Ducoffe, G., Legay, S., Nisse, N.: On the Complexity of Computing Treewidth. In: IWOCA. Lecture Notes in Computer Science, vol. 9843, 3–15. Springer, 2016., full version: [47]
- [49] Farber, M., Jamison, R. E.: Convexity in Graphs and Hypergraphs. *SIAM Journal on Algebraic Discrete Methods* 7 (3), 433–444, 1986.
- [50] Feige, U.: Approximating the Bandwidth via Volume Respecting Embeddings. *Journal of Computer and System Science* 60 (3), 510–539, 2000.
- [51] Feige, U., Talwar, K.: Approximating the Bandwidth of Caterpillars. *Algorithmica* 55 (1), 190–204, 2009.
- [52] Fellows, M. R., Fomin, F. V., Lokshtanov, D., Losievskaja, E., Rosamond, F. A., Saurabh, S.: Distortion is Fixed Parameter Tractable. *TOCT* 5 (4), 16, 2013.
- [53] Fomin, F. V., Lokshtanov, D., Saurabh, S.: An exact algorithm for minimum distortion embedding. *Theoretical Computer Science* 412 (29), 3530–3536, 2011.
- [54] Frederickson, G. N.: Parametric Search and Locating Supply Centers in Trees. In: WADS. Lecture Notes in Computer Science, vol. 519, 299–319. Springer, 1991.
- [55] Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B* 16 (1), 47–56, 1974.
-

- 
- [56] Gilmore, P. C., Hoffman, A. J.: A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics* 16, 539–548, 1964.
- [57] Golovach, P. A., Heggenes, P., Kratsch, D., Lokshtanov, D., Meister, D., Saurabh, S.: Bandwidth on AT-free graphs. *Theoretical Computer Science* 412 (50), 7001–7008, 2011.
- [58] Gupta, A.: Improved Bandwidth Approximation for Trees and Chordal Graphs. *Journal of Algorithms* 40 (1), 24–36, 2001.
- [59] Halin, R.: S-functions for graphs. *Journal of Geometry* 8 (1–2), 171–186, 1976.
- [60] Heggenes, P., Kratsch, D., Meister, D.: Bandwidth of bipartite permutation graphs in polynomial time. *Journal of Discrete Algorithms* 7 (4), 533–544, 2009.
- [61] Heggenes, P., Meister, D.: Hardness and approximation of minimum distortion embeddings. *Information Processing Letters* 110 (8–9), 312–316, 2010.
- [62] Heggenes, P., Meister, D., Proskurowski, A.: Computing minimum distortion embeddings into a path for bipartite permutation graphs and threshold graphs. *Theoretical Computer Science* 412 (12–14), 1275–1297, 2011.
- [63] Indyk, P.: Algorithmic Applications of Low-Distortion Geometric Embeddings. In: FOCS. 10–33. IEEE Computer Society, 2001.
- [64] Indyk, P., Matousek, J.: Low-Distortion Embeddings of Finite Metric Spaces. In: Handbook of Discrete and Computational Geometry, 2nd Ed., 177–196. Chapman and Hall/CRC, 2004.
- [65] Karp, R.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, 85–103. Plenum Press, 1972.
- [66] Kleitman, D. J., Vohra, R.: Computing the Bandwidth of Interval Graphs. *SIAM Journal on Discrete Mathematics* 3 (3), 373–375, 1990.
- [67] Kloks, T., Kratsch, D., Müller, H.: Approximating the Bandwidth for Asteroidal Triple-Free Graphs. *Journal of Algorithms* 32 (1), 41–57, 1999.
- [68] Kratsch, D.: Domination and Total Domination on Asteroidal Triple-free Graphs. *Discrete Applied Mathematics* 99 (1–3), 111–123, 2000.
- [69] Kratsch, D., Spinrad, J.: Between  $O(nm)$  and  $O(n^\alpha)$ . *SIAM Journal on Computing* 36 (2), 310–325, 2006.
- [70] Kratsch, D., Stewart, L.: Approximating Bandwidth by Mixing Layouts of Interval Graphs. *SIAM Journal on Discrete Mathematics* 15 (4), 435–449, 2002.

- 
- [71] Leitert, A., Dragan, F.F.: On Strong Tree-Breadth. In: COCOA. Lecture Notes in Computer Science, vol. 10043, 62–76. Springer, 2016.
- [72] Lokshtanov, D.: On the complexity of computing treelength. *Discrete Applied Mathematics* 158 (7), 820–827, 2010.
- [73] Ma, T.-H., Spinrad, J.P.: On the 2-Chain Subgraph Cover and Related Problems. *Journal of Algorithms* 17 (2), 251–268, 1994.
- [74] McConnell, R. M., Spinrad, J. P.: Linear-Time Transitive Orientation. In: SODA. 19–25. ACM/SIAM, 1997.
- [75] Monien, B.: The Bandwidth Minimization Problem for Caterpillars with Hair Length 3 is NP-Complete. *SIAM Journal on Algebraic Discrete Methods* 7(4), 505–512, 1986.
- [76] Müller, H.: Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics* 156 (1-3), 291–298, 1996.
- [77] Olariu, S.: An Optimal Greedy Heuristic to Color Interval Graphs. *Information Processing Letters* 37 (1), 21–25, 1991.
- [78] Peleg, D., Reshef, E.: Low Complexity Variants of the Arrow Distributed Directory. *Journal of Computer and System Sciences* 63 (3), 474–485, 2001.
- [79] Robertson, N., Seymour, P.D.: Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory, Series B* 35 (1), 39–61, 1983.
- [80] Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B* 36 (1), 49–64, 1984.
- [81] Rose, D. J., Tarjan, R. E., Lueker, G. S.: Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing* 5 (2), 266–283, 1976.
- [82] Räcke, H.: Lecture notes at <http://ttic.uchicago.edu/~harry/teaching/pdf/lecture15.pdf>
- [83] Schaefer, T. J.: The Complexity of Satisfiability Problems. In: STOC. 216–226. ACM, 1978.
- [84] Shrestha, A. M. S., Tayu, S., Ueno, S.: Bandwidth of convex bipartite graphs and related graphs. *Information Processing Letters* 112 (11), 411–417, 2012.
- [85] Slater, P. J.: Locating central paths in a graph. *Transportation Science* 16, 1–18, 1982.
- [86] Spinrad, J. P.: *Efficient Graph Representations*. American Mathematical Society, 2003.

- [87] Sprague, A. P.: An  $O(n \log n)$  Algorithm for Bandwidth of Interval Graphs. *SIAM Journal on Discrete Mathematics* 7 (2), 213–220, 1994.
- [88] Tarjan, R. E., Yannakakis, M.: Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM Journal of Computing* 13 (3), 566–579, 1984.
- [89] Tenenbaum, J. B., de Silva, V., Langford, J. C.: A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* 290 (5500), 2319–2323, 2000.
- [90] Thorup, M., Zwick, U.: Compact Routing Schemes. In: *SPAA*. 1–10, 2001.
- [91] Venkatesan, G., Rotics, U., Madanlal, M. S., Makowsky, J. A., Pandu Rangan, C.: Restrictions of Minimum Spanner Problem. *Information and Computation* 136 (2), 143–164, 1997.
- [92] Völkel, F., Bapteste, E., Habib, M., Lopez, P., Vigliotti, C.: Read networks and  $k$ -laminar graphs. *CoRR* abs/1603.01179, 2016.
- [93] Yancey, M. P.: An Investigation into Graph Curvature’s Ability to Measure Congestion in Network Flow. *CoRR* abs/1512.01281, 2015.

---

# List of Figures

2.1	Example of a layering partition. . . . .	7
3.1	A graph $G$ where a layering partition creates a cluster $C$ with radius $3\text{tb}(G)$ and diameter $3\text{tl}(G)$ . . . . .	13
3.2	Two extended $C_5$ of degree 1 and degree 3. . . . .	17
3.3	Illustration to the proof of Theorem 3.7. . . . .	19
3.4	Illustration for the graph $G_\rho^+$ . . . . .	21
4.1	Example for the set $\mathcal{P}$ for a subtree of a layering partition. . . . .	34
4.2	Construction of the set $C_B$ for a branching bag $B$ . . . . .	47
5.1	Illustration to the proof of Theorem 5.3. . . . .	60
5.2	Illustration to the proof of Theorem 5.6. . . . .	62
5.3	Illustration to the proof of Theorem 5.7. . . . .	64
5.4	Illustration to the proof of Theorem 5.9. . . . .	66
6.1	Reduction from Planar Monotone 3-SAT to $k$ -ESP. . . . .	69
6.2	Reduction from Dominating Set to $k$ -ESP. . . . .	71
6.3	Example for Lemma 6.14 and Lemma 6.15. . . . .	86
6.4	Example for a possible tree-decomposition. . . . .	89
6.5	Example for Corollary 6.15. . . . .	91

---

# List of Tables

4.1	Implications of our results for the $p$ -Center problem. . . . .	51
4.2	Implications of our results for the Connected $p$ -Center problem. . . . .	51
5.1	Existing solutions for calculating bandwidth. . . . .	54
5.2	Existing hardness results for calculating bandwidth. . . . .	54
5.3	Existing solutions for calculating the minimum distortion. . . . .	55
5.4	Existing hardness results for calculating the minimum distortion. . . . .	56