New Width Parameters of Graphs

Martin Vatshelle

Dissertation for the degree of Philosophiae Doctor (PhD)

Department of Informatics
University of Bergen

May 24, 2012

Abstract

The main focus of this thesis is on using the divide and conquer technique to efficiently solve graph problems that are in general intractable. We work in the field of parameterized algorithms, using width parameters of graphs that indicate the complexity inherent in the structure of the input graph. We use the notion of branch decompositions of a set function introduced by Robertson and Seymour to define three new graph parameters, boolean-width, maximum matching-width (MM-width) and maximum induced matching-width (MIM-width). We compare these new graph width parameters to existing graph parameters by defining partial orders of width parameters. We focus on tree-width, branch-width, clique-width, module-width and rank-width, and include a Hasse diagram of these orders containing 32 graph parameters.

We use the size of a maximum matching in a bipartite graph as a set function to define MM-width and show that MM-width never differs by more than a multiplicative factor 3 from tree-width. The main reason for introducing MM-width is that it simplifies the comparison between tree-width and parameters defined via branch decomposition of a set function.

We use the logarithm of the number of maximal independent sets in a bipartite graph as set function to define boolean-width. We show that boolean-width of a graph class is bounded if and only if rank-width is bounded, and show that the boolean-width of a graph can be as low as the logarithm of the rank-width of the graph. Given a decomposition of boolean-width k, we design FPT algorithms parameterized by k, for a large class of graph problems, whose runtime has a single exponential dependency in the boolean-width, i.e. $O^*(2^{O(k^2)})$. Moreover we solve Maximum Independent Set in time $O^*(2^{2k})$ and Minimum Dominating Set in time $O^*(2^{3k})$. These algorithms are in particular interesting in conjunction with the fact that many graph classes have boolean-width $O(log(n))$, e.g. interval graphs.

MIM-width is defined using the size of a maximum induced matching in a bipartite graph as set function. The main reason to introduce MIM-width is that its value is lower than any of the other parameters, in particular MIM-width is 1 on interval graphs, permutation graphs and convex graphs, and at

most $2k$ on circular k-trapezoid graphs, k-polygon graphs, Dliworth k graphs and complements of k-degenerate graphs. We show that the FPT algorithms designed for boolean-width are XP algorithms when parameterized by MIM-width, this shows that a large class of locally checkable vertex subset and vertex partitioning problems are polynomial time solvable on the mentioned graph classess with bounded MIM-width.

We give exact algorithms to compute optimal decompositions for all the three new width parameters and report on the implementation of a heuristic for finding decompositions of low boolean-width.

Acknowledgements

First and foremost, I would like to thank everyone in the Algorithms group at the University of Bergen for creating an excellent atmosphere for learning. It has been a great experience both scientifically and personally to get to know you all! I want to extend special gratitude to those that have also become my co-authors: Isolde Adler, Remy Belmonte, Binh-Minh Bui-Xuan, Fedor Fomin, Serge Gaspers, Petr Golovach, Eivind Magnus Hvidevold, Erik-Jan van Leeuwen, Daniel Meister, Sadia Sharmin and Yngve Villanger. And a special thanks to my office mates Mostofa Patwary and Sigve Sæther who made my office a cozy place to be.

I have had the privilege and pleasure of working with many people who are experts in their fields and from whom I have learnt a lot! I, therefore, would like to give special thanks to Therese Biedl, Lene Favrholt, Sang-il Oum, Yuri Rabinovich, and Johan M.M. van Rooij for sharing your valuable knowledge with me.

One of the things I enjoy the most from my academic experience is teaching, and I have had so much pleasure interacting and fostering scientific interest with a lot of students. Thanks to Pinar Heggernes, Daniel Lokshtanov, and Fredrik Manne, all of whom I have had the honour of teaching together with.

During my four years of PhD studies, I have had the fortune of travelling vastly and frequently. Out of the ten different countries I visited, Czech Republic has become my favourite. I would like to thank Jan Kratochvil and his entire group for organizing so many excellent workshops and conferences, Dan Kral for extending invitations, and all the other students for making Czech Republic feel like my second home. Also, I want to give special thanks to Naomi Nishimura of the University of Waterloo for making it possible for me to stay for five months in Waterloo - it was a unique experience at a great university. I hope to be able to spend much more time there in the future. Having said all these, thank you very much to UiB and NFR for generously funding all my travels!

Above all, (and of course), my greatest gratitude and appreciation goes

to my supervisor, Jan Arne Telle. Not only has he imparted his much valued knowledge, insights, and guidance to me, he has also managed to understand my complicated inner self and put up with all the things I have done over these past four years! None of the things I thanked for above would have been possible if it was not for his faith in me. Jan Arne, it has been such an honour, privilege, and enjoyment to learn from you and work with you. I am forever grateful for your open door.

I have, in periods, been very busy - a consequence that my family has had to endure. They have always been a great support throughout my studies and for that, I am very grateful. I wish I can promise to spend more time with them in the future, but at this point, there is much uncertainty about the future. Last but not least, I want to thank my girlfriend for patiently waiting for me to finish my PhD and enduring all those long months apart. You have given me so much inspiration and motivation.

I dedicate this thesis to a very special Sunflower.

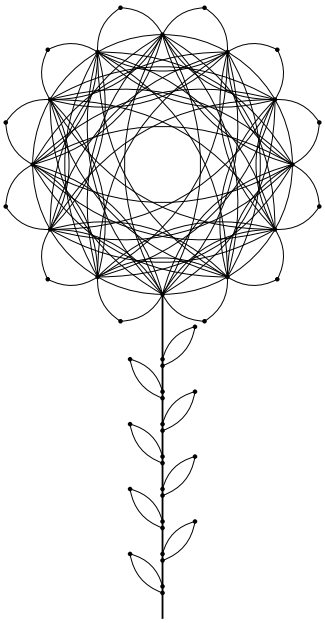


Figure 1: The Sunflower graph

Contents

Part I

Overview

Chapter 1

Introduction

Ever since they were first described by Leonard Euler in 1735 with his work on The Seven Bridges of Königsberg [27], graphs have been an important notion in discrete mathematics, and later in computer science. Graphs can be used to model any pairwise relationship between any kind of objects, like pieces of land connected by bridges or people connected by friendships. The field of graph algorithms has many applications, and divide and conquer is one of the fundamental algorithmic techniques. One of the earliest described uses of divide and conquer is an algorithm for discrete Fourier transformation of Gauss [37]. The main focus of this thesis is on using the divide and conquer technique to efficiently solve graph problems that are in general intractable.

1.1 Graph Decompositions and Width Parameters

When solving problems by divide and conquer on a graph G, it is common to store the information of how to divide the graph by a rooted decomposition tree of G. Constant size subgraphs of G are stored at the leaves of the decomposition tree, and at internal nodes of the tree the smaller subgraphs of G at its children are "glued" together to form bigger subgraphs of G. We then process the decomposition tree in a bottom up fashion solving the problem on subgraphs of G of increasing size using dynamic programming, with the solution found at the root, which stores the graph G. The runtime of such algorithms greatly depend on the choice of decomposition tree. Therefore it is common to measure the complexity of a decomposition tree by a carefully chosen parameter, called the width of a decomposition tree, with the width parameter of a graph being the width of an optimal decomposition tree. Tree-width, clique-width and rank-width are examples of

such graph width parameters for which there exists a vast litterature of both algorithmic and structural results, see [45] for an overview. In this thesis we focus on algorithmic applications and introduce three new graph width parameters called boolean-width, maximum matching-width and maximum induced matching-width.

When we analyse algorithms based on decomposition trees we use two parameters, n the number of vertices in the input graph and k the width of the decomposition tree. This makes it more complicated to compare the runtime of algorithms, e.g. $2^{k^3} \cdot n$ versus n^k. Such considerations are part of the field called Parameterized Complexity, see [24, 28, 56].

Graph algorithms based on decomposition trees have two stages. First compute a good decomposition tree. Second use dynamic programming on the decomposition tree to solve various NP-hard problems. Normally the first stage is allowed more runtime than the second since it can be viewed as a preprocessing and the same decomposition can be used to run many algorithms. There are three important aspects to consider when comparing the runtime of algorithms based on decomposition trees:

(1) The time spent to compute a decomposition tree (Chapter 6 gives an overview).

(2) The width of the decomposition found (Chapter 4 gives an overview).

(3) The time spent to solve the problem by dynamic programming (Chapter 5 gives an overview).

(1) is not a main focus of this thesis, however it is an important step. In some sense this is the hardest step and the best algorithms for computing optimal decompositions, in particular for tree-width and rank-width, involve deep results of graph theory originating from e.g. the Graph Minors project. These algorithms have limited viability in practice, but there is promising work in designing heuristics in particular for tree-width, and the ideas from this work might also carry over to other types of decompositions.

(2) is important because two different width parameters could have algorithms with the same runtime but the two parameter values on a given graph may differ greatly. We will compare many graph parameters, not only those defined via decomposition trees, and define partial orders on them to get an overview of how they relate to each other. The lower a parameter is the harder it will be to design algorithms that are efficient in terms of this parameter, and this constitutes the main challenge addressed in this thesis.

(3) has been studied intensely for all graph parameters discussed in this thesis, and several new results are presented. The main focus in this area

has been whether a problem can be solved in FPT time i.e. $f(k) \cdot poly(n)$ for a polynomial function $poly$ or in XP time i.e $n^{f(k)}$. For FPT algorithms there has been a focus on whether the $poly(n)$ in the runtime is linear in n or not. In this thesis we instead focus on $f(k)$ and on reducing the exponential dependency in k, in particular we want algorithms with runtime $2^{O(k)} \cdot poly(n)$.

1.2 New Width Parameters

The notion of branch decompositions introduced by Robertson and Seymour is a general framework that we will use to define several width parameters of a graph G. For our purposes these width parameters will be based on a set function assigning to each subset S of vertices of a graph G a number between 0 and $|V(G)|$. A subset $S \subseteq V(G)$ defines a cut of G consisting of the edges with one endpoint in S and one endpoint outside S. In this way a vertex subset S can be viewed as a cut which can be viewed as a bipartite graph. Any parameter of a bipartite graph can therefore be used as a set function to define a graph width parameter.

Rank-width fits this framework and is defined using the $GF(2)$ rank of the bipartite adjacency matrix of the cut as the set function. On the other hand, tree-width and clique-width do not easily fit in this framework. However, module-width which never differs by more than a multiplicative factor 2 from clique-width, can be defined using this framework with the number of twin-classes across the cut as the set function. Branch-width never differs from tree-width by more than a multiplicative factor 1.5 and fits in the frame-work of branch decompositions, but only by using a set function defined on subsets of edges. Can we define a parameter in this frame-work, using a set function defined on subsets of vertices, that never differs by more than a constant multiplicative factor from tree-width?

Yes, we can, and this is the new parameter we call the maximum matching width (MM-width). The Minimum Vertex Cover problem is a well studied problem, in particular in the field of parameterized algorithms. In bipartite graphs the Minimum Vertex Cover problem is equivalent to the Maximum Matching problem. We use the size of a maximum matching in the cut as a set function to define maximum matching width. The maximum matching width is interesting because a proof via a cops and robber game (equivalent to tree-width) shows that maximum matching-width never differs by more than a multiplicative factor 3 from tree-width. The proof relies on the monotonicity properties of this version of the cops and robber game, proven by [70].

When studying how to solve the Maximum Independent Set problem by dynamic programming on decomposition trees we discovered, with the help of Nathann Cohen, that the number of maximal independent sets in the bipartite graph of a cut was an important measure, and we use this to define boolean-width which help us solve several NP-complete problems in $O^*(2^{O(k)})$ time when given a decomposition tree of boolean-width k. These algorithms are in particular interesting in conjunction with the fact that many well-known graph classes, like interval graphs, have boolean-width $O(log(n))$.

We introduce a third new width parameter even lower than boolean-width, by using the size of a maximum induced matching in a bipartite graph as the set function to define maximum induced matching-width (MIM-width). MIM-width is constant on many interesting graph classes where none of the other parameters are constant, e.g. interval graphs.

1.3 Overview of Chapters

The thesis has two parts. Part I contains an overview of known results as well as new results that have not been published before. Part II contains five papers previously published or recently submitted.

1.3.1 Part I

In Chapter 2 we give a short overview of standard definitions.

In Chapter 3 we describe a general framework called binary decomposition trees and use this to define many graph width parameters. In particular we define the three new graph width parameters MM-width, boolean-width and MIM-width. Boolean-width was first introduced in the paper making up Chapter 8, while MM-width and MIM-width are introduced for the first time in this chapter.

In Chapter 4 we study the relationship between some well-known graph width parameters and also how they relate to the new graph width parameters. We introduce partial orders of width parameters in order to compare their values, presenting the orders in Hasse diagrams.

In Chapter 5 we compare the runtime of algorithms for various problems expressible in monadic second order logic using various graph parameters, sometimes assuming an appropriate decomposition given as part of the input. We give an overview of existing algorithms. We show that the algorithms given in Chapter 10 yield improved runtime parameterized by clique-width and rank-width and lead to XP algorithms parameterized by MIM-width. We also give an overview of runtimes for Maximum Independent Set,

Minimum Dominating Set and Feedback Vertex Set parameterized by the various graph width parameters.

In Chapter 6 we give exact algorithms to compute binary decomposition trees of optimal boolean-width and MIM-width. We also give an overview of the current best exact algorithms for other graph width parameters and also of the best decompositions achievable by an FPT algorithm. Finally we discuss heuristics to compute optimal decompositions, this is a big area which we have barely started to investigate, but it is part of ongoing research.

In Chapter 7 we summarize and list some open problems.

1.3.2 Part II

Part II consists of 5 papers appearing in separate chapters numbered 8 to 12.

Chapter 8 Binh-Minh Bui-Xuan, Jan Arne Telle and Martin Vatshelle, Boolean-width of Graphs, in *Theoretical Computer Science* 412(39), pages 5187–5204, 2011.

This paper introduces boolean-width, compares boolean-width to rank-width and gives dynamic programming algorithms for a handful of problems.

Chapter 9 Rémy Belmonte and Martin Vatshelle, Graph Classes with Structured Neighborhoods and Algorithmic Applications, submitted to journal. Extended abstract in *Proceedings of WG'11*, LNCS 6986, pages 47–58, 2011.

This paper shows that many well-known graph classes have logarithmic boolean width and that the algorithms in Chapter 10 run in polynomial time on these graph classes, i.e. graphs of bounded MIM-width.

Chapter 10 Binh-Minh Bui-Xuan, Jan Arne Telle and Martin Vatshelle, Fast Dynamic Programming for Locally Checkable Vertex Subset and Vertex Partitioning Problems, submitted to journal. Extended abstract (3 pages) in Isolde Adler, Binh-Minh Bui-Xuan, Yuri Rabinovich, Gabriel Renault, Jan Arne Telle and Martin Vatshelle, On the Boolean-width of a Graph: Structure and Applications, in *Proceedings of WG'10*, LNCS 6410, pages 159–170, 2010.

This paper gives algorithms for solving a large class of graph problems analyzing the runtime of these algorithms in terms of the width of a given decomposition tree.

Chapter 11 Binh-Minh Bui-Xuan, Ondra Suchý, Jan Arne Telle and Martin Vatshelle, Feedback Vertex Set on Graphs of low Clique-width, to appear in *European Journal of Combinatorics*.

This paper gives an algorithm for solving MINIMUM FEEDBACK VERTEX SET in time $O^*(k^{5k})$ when given a decomposition tree having module-width k.

Chapter 12 Eivind Magnus Hvidevold, Sadia Sharmin, Jan Arne Telle and Martin Vatshelle, Finding Good Decompositions for Dynamic Programming on Dense Graphs, *Proceedings of IPEC'11*, LNCS 7112, pages 219–231, 2011.

In this paper we give a first heuristic for computing boolean-decompositions and compare its performance to existing heuristics for tree-width.

Chapter 2

Preliminaries

In this Chapter we review some basic definitions. Most of the terminology in this thesis is standard and can be found in any textbook on the appropriate subject. First we consider set theory with special attention to the union, intersection and symmetric difference operators. Then basic graph theory and some well-known graph problems relevant to this thesis. Then a short section on how to measure the runtime of algorithms.

2.1 Set Theory

The cardinality of a set S denoted $|S|$ is the number of elements in the set. Given two sets A and B we use the following operations:

Subset $A \subseteq B$ if $\forall v \in A$ we have $v \in B$.

Equality $A = B$ if $A \subseteq B$ and $B \subseteq A$ otherwise $A \neq B$.

Strict subset $A \subset B$ if $A \subseteq B$ and $A \neq B$.

Intersection $A \cap B = \{v : v \in A \text{ and } v \in B\}$.

Union $A \cup B = \{v : v \in A \text{ or } v \in B\}$.

Difference $A \setminus B = \{v : v \in A \text{ and } v \notin B\}$.

Symmetric difference $A \triangle B = (A \cup B) \setminus (A \cap B) = (A \setminus B) \cup (B \setminus A)$.

A *set family* is a set of sets, as a convention to distinguish set families from sets we will when possible use calligraphic upper-case letters for set families, upper-case letters for sets and lower-case letters for elements. For three of the operations above the order or the sets is irrelevant and the operations can be extended to set families. Let $\mathcal{F} = X_1, X_2, \ldots, X_k$ be a set family then:

Intersection $\bigcap_{X\in\mathcal{F}} X = X_1 \cap X_2 \cap \cdots \cap X_k$.

Union $\bigcup_{X\in\mathcal{F}} X = X_1 \cup X_2 \cup \cdots \cup X_k$.

Symmetric difference $\triangle_{X\in\mathcal{F}} X = X_1 \triangle X_2 \triangle \cdots \triangle X_k$.

The theory of sets and set operations is wide and extensive, but not a topic of this thesis, for an extensive overview see [72].

A set family $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ is called a *partition* of U (called the universe) if $U = P_1 \cup P_2 \cup \cdots \cup P_k$ and for all i, j such that $1 \leq i < j \leq k$ we have $P_i \cap P_j = \emptyset$ and $P_i \neq \emptyset$.

We say a set S is *closed* under an operation $\oplus$ if for all $x, y \in S$ we have $x \oplus y \in S$. The $\oplus$-*closure* of a set S is the unique minimal set closed under $\oplus$ containing S. The two types of closure we will discuss in this thesis is union closure and $\triangle$-closure.

2.2 Graph Theory

In this section we give a short overview of standard graph terminology. For a more complete introduction to graph theory there are many good books for example an introductory book by R. J. Wilson [77] or a more advanced book by R. Diestel [22].

Graph A graph G is a pair $V(G)$ called the *vertices*, and $E(G)$ called the *edges* where edges are unordered pairs of the vertices. We will only consider loopless simple undirected graphs in this thesis.

Neighborhood For a graph G and a vertex $v \in V(G)$, the neighborhood of v, denoted $N(v)$, is the set of all vertices in G adjacent to v. The closed neighborhood is denoted $N[v] = N(v) \cup \{v\}$. The *neighborhood of a set* $S \subseteq V(G)$ is denoted $N(S) = \bigcup_{v\in S} N(v)$ and the closed neighborhood of S is denoted $N[S] = N(S) \cup S$.

Twins For a graph G, two vertices $x, y \in V(G)$ are *twins* if $N(x) \setminus y = N(y) \setminus x$.

Vertex complement For a graph G and $A \subseteq V(G)$, the complement of A is denoted $\overline{A} = V(G) \setminus A$.

The complement of a graph The complement of a graph G, denoted $\overline{G}$, is the graph where $V(\overline{G}) = V(G)$ and for any pair $u, v \in V(G)$ with $u \neq v$ we have $(u, v) \in E(\overline{G})$ if and only if $(u, v) \notin E(G)$.

Subgraph A graph H is a subgraph of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ denoted $H \subseteq G$. For $S \subseteq V(G)$ the subgraph of G induced by S denoted $G[S]$ is the maximal subgraph $H \subseteq G$ having $V(H) = S$.

Cut For a graph G and $A \subseteq V(G)$, the cut in G defined by A is the partition $(A, \overline{A})$ of $V(G)$. The neighborhood of $X \subseteq A$ across $(A, \overline{A})$ is $N(X) \cap \overline{A}$. Two vertices $x, y \in A$ are twins across $(A, \overline{A})$ if $N(x) \cap \overline{A} = N(y) \cap \overline{A}$.

Bipartite graph We say a graph $G = (V, E)$ is bipartite if there exist a subset $A \subseteq V(G)$ such that every edge in $E(G)$ has one endpoint in A and the other in $\overline{A}$.

Induced bipartite subgraph For a graph G and $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$. The bipartite graph induced by the two subsets is denoted $G[A, B] = (A \cup B, E')$ where $E' \subseteq E(G)$ are the edges with one endpoint in A and one endpoint in B. Note that $A \subseteq V(G)$ defines the induced bipartite subgraph $G[A, \overline{A}]$.

Bipartite adjacency matrix For a graph G and $A \subseteq V(G)$ let $A = \{v_1, v_2, \dots\}$ and $\overline{A} = \{u_1, u_2, \dots\}$. The bipartite adjacency matrix of $G[A, \overline{A}]$ is the matrix with $|A|$ rows and $|\overline{A}|$ columns where the entry in row i and column j is 1 if $(v_i, u_j) \in E(G[A, \overline{A}])$ and 0 otherwise.

Connected graph A graph G is connected if for every pair of vertices $u, v \in V(G)$ there exist a path from u to v. A graph that is not connected is called a disconnected graph.

Separator For a connected graph G, a set $S \subseteq V(G)$ is a separator of G if $G[V(G) \setminus S]$ is a disconnected graph. A separator of size 1 is called a cut vertex.

Tree A tree is a connected graph with no cycles. In a tree T, to avoid confusion with a graph, we call the elements in $V(T)$ nodes. A node with degree at most 1 is called a leaf and a node of degree at least 2 is called an internal node. A tree is called a rooted tree if one vertex has been designated the root, in which case the edges have a natural orientation, towards or away from the root. For a rooted tree T and $u \in V(T)$ the neighbor of u on the path to the root is called the parent of u and a vertex v is a child of u if u is the parent v.

Subcubic tree A *subcubic tree* is a tree where every node has degree at most 3.

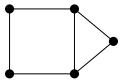


Figure 2.1: A graph with 5 vertices and 6 edges.

Binary tree A *binary tree* is a rooted tree where every node is either a leaf or has two children.

Contraction For a graph G and $(u, v) \in E(G)$ the graph G' is obtained by contracting (u, v) in G by adding a new vertex w and making w adjacent to $N(u) \cup N(v)$ and then deleting u and v.

Subdividing For a graph G and $(u, v) \in E(G)$ the graph G' is obtained by subdividing (u, v) in G by adding a new vertex w and making w adjacent to u and v and then deleting the edge (u, v).

Cycle By C_k, for $k \geq 2$, we denote the cycle of length k, i.e. a connected graph with $|V(C_k)| = k$ where every vertex has degree 2.

Clique By K_k, for $k \geq 1$, we denote the clique of size k, i.e. a graph with $|V(K_k)| = k$ where every vertex has degree $k-1$. K_3 is called a triangle.

2.2.1 Graph Properties and Graph Problems

For a graph G we have the following graph properties:

Vertex cover is a set $S \subseteq V(G)$ such that every edge in $E(G)$ has at least one endpoint in S.

Independent set is a set $S \subseteq V(G)$ such that every pair $u, v \in S$ where $u \neq v$ has u and v not adjacent in G.

q-coloring is a partition of the vertices into at most q independent sets.

Dominating set is a set $S \subseteq V(G)$ such that every vertex $v \in V(G)$ have $N[v] \cap S \neq \emptyset$.

Matching is a set of edges $M \subseteq E(G)$ such that for every vertex $v \in V(G)$ there is at most one edge in M having v as an endpoint.

Induced Matching is a set of edges $M \subseteq E(G)$ such that for every vertex $v \in V(G)$ either there is no edge in M having v as an endpoint or there is exactly one edge $(u, v) \in M$ having v as an endpoint and no other edge with an endpoint in $N(v)$.

Feedback vertex set is a set of vertices $S \subseteq V(G)$ such that $G[V(G) \setminus S]$ is a tree.

For any of the above graph properties we can define graph problems, we use Capital Letters for graph problems. E.g. Maximum Matching asks, given a graph G, for the maximum size of a matching in G and Minimum Dominating Set asks for the size of a minimum dominating set in G. We can also define weighted and counting versions of any of these problems.

2.3 Runtime Analysis

Let G be a graph with $n = |V(G)|$ vertices. We use big O and O^* notation to measure the runtime of algorithms. For functions f and g we say:

- $f(n) \in O(g(n))$ if there exist c and n_0 such that for all $n > n_0$ we have $f(n) \leq c \cdot g(n)$.
- $f(n) \in \Theta(g(n))$ if $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$.
- $f(n) \in O^*(g(n))$ if there exist a polynomial $poly$, such that $f(n) \in O(g(n) \cdot poly(n))$.

A graph parameter P is a function assigning a number to each graph. Two graph parameters P and Q are linearly bounded if for every graph G we have $P(G) \in O(Q(G))$ and $Q(G) \in O(P(G))$.

Let $P(G) = k$ be a parameter of a graph G, when we measure the runtime of an algorithm as a function of both n and k we call it a parameterized algorithm. We distinguish the runtime of parameterized algorithms as follows:

- A parameterized algorithm is FPT parameterized by k if there exists a function f and a polynomial function $poly$ such that the algorithm finishes in time $f(k)poly(n)$.
- A parameterized algorithm is XP parameterized by k if there exists a function f such that the algorithm finishes in time $n^{f(k)}$.
- A parameterized algorithm is *single exponential* parameterized by k if there exists a polynomial function $poly$ such that the algorithm finishes in time $2^{poly(k)} \cdot poly(n)$.

- A parameterized algorithm is *Linear single exponential* parameterized by k if there exists a polynomial function *poly* such that the algorithm finishes in $2^{O(k)} \cdot poly(n)$ time.

Chapter 3

Width Parameters

In this chapter we first describe a general framework called binary decomposition trees and use this to define many graph width parameters. In particular we define the three new graph width parameters MM-width, boolean-width and MIM-width.

3.1 Decomposition Trees

A *branch decomposition* based on a set function is by now a standard notion in graph and matroid theory, see [29, 38, 60, 66].

Definition 3.1.1 (Branch decomposition)**.** Let A be any finite set. Let $f\colon 2^A \to \mathbb{R}$ be a symmetric set function, i.e. f satisfies $f(X) = f(\overline{X})$ for all $X \subseteq A$. For a tree T we denote the set of leaves by $L(T)$. A *branch decomposition* of f on A is a pair (T, δ), for a subcubic tree T and a bijection $\delta : A \to L(T)$. For every edge $e \in E(T)$ let T_1, T_2 be the connected components of $T \setminus e$, then e yields a partition of A by the leaf labels of the two connected components:

$$\mathcal{P}_e = \left\{ \{\delta(v)^{-1} : v \in L(T) \cap V(T_1)\}, \{\delta(v)^{-1} : v \in L(T) \cap V(T_2)\} \right\}.$$

We extend the domain of f to edges e of T by letting $f(e) = f(X)$ for $\mathcal{P}_e = (X, \overline{X})$. This is well-defined because f is symmetric. The *width* of a branch decomposition (T, δ) is the maximum over all $e \in E(T)$ of $f(e)$. The *branch width* of f is the minimum width over all branch decompositions (T, δ). If $|A| \leq 1$, then f has a unique decomposition tree with no edges and we let the branch-width of f be $f(A)$.

Note that for any branch decomposition (T, δ) we can assume that T has no nodes of degree 2 since the two edges adjacent to a node of degree 2 yield

the same partition, hence contracting one of those edges would not change the branch width of (T, δ). We now define branch-width of a graph G.

Definition 3.1.2 (Branch-width of a graph)**.** For G a graph and $X \subseteq E(G)$ let the *middle set* of X be defined as $\{v \in V(G) : \exists (a, v) \in X \text{ and } (b, v) \in \overline{X}\}$. Let $mid\colon 2^{E(G)} \to \mathbb{N}$ be a function where $mid(X)$ is the size of the middle set of X. Let (T, δ) be branch decomposition of mid on $E(G)$. The branch width of (T, δ) is denoted $brw(T, \delta)$. The branch width of mid on $E(G)$, is called the branch-width of G and is denoted $brw(G)$.

Apart from branch-width most of the width parameters defined in this thesis involve a bijection to the vertices of a graph. Also, rooted decompositions are preferred when designing algorithms, so we will from now on be using the following general type of decomposition.

Definition 3.1.3 (Binary decomposition tree)**.** For G a graph and $f\colon 2^{V(G)} \to \mathbb{R}$ a set function on $V(G)$. A *binary decomposition tree* of G is a pair (T, δ), for a binary tree T and a bijection $\delta : V(G) \to L(T)$, with $L(T)$ the leaves of T. For every node $a \in V(T)$ let L_a be the leaves of T having a as an ancestor and let $V_a = \{\delta^{-1}(x) : x \in L_a\}$ be the vertices of G mapped to L_a. The *f-width* of a binary decomposition tree (T, δ) is the maximum $f(V_a)$ over all $a \in V(T)$. The *f-width* of G is the minimum f-width over all binary decomposition trees (T, δ) of G.

We may also refer to a binary decomposition tree simply as a decomposition tree. When defining a graph width parameter using a symmetric set function f it usually does not matter which of the two definitions we use.

Observation 3.1.4. For any graph G and symmetric set function f on $V(G)$, if the branch width of f on $V(G)$ using Definition 3.1.1 is k and $f(V(G)) \leq k$ then the f-width of G using Definition 3.1.3 is also k.

Proof. Let (T, δ) be a branch decomposition of f on $V(G)$ of width k. We obtain a binary decomposition tree (T', δ) of f on $V(G)$ by subdividing an arbitrary edge of T by adding a node r and making r the root to obtain T'. For any node $v \in V(T')$ with $v \neq r$ there is a corresponding edge $e \in E(T)$, and vice versa for any edge $e \in E(T)$ there is a node $v \in V(T')$ such that $\mathcal{P}_e = \{V_v, \overline{V_v}\}$ and hence $f(V_v) = f(e)$, i.e. the unique edge $(x, y) \in E(T)$ with $\delta^{-1}(x) \in V_v$ and $\delta^{-1}(y) \in \overline{V_v}$. Since $f(V_r) \leq k$ the f-width of G is k. □

In many situations it is convenient to define a simpler type of binary decomposition tree.

Definition 3.1.5 (Caterpillar decomposition)**.** A *caterpillar decomposition* is a binary decomposition tree (T, δ) where every internal node of T has a child that is a leaf. See Figure 3.1. We can construct a caterpillar decomposition (T, δ) from any ordering σ of $V(G)$ by letting T be any binary tree where every internal node has a child that is a leaf such that T has $|V(G)|$ leaves and for all $1 \leq i \leq |V(G)|$ let δ map $\sigma(i)$ to the i'th leaf encountered by a breadth first search starting from the root of T.

A caterpillar decomposition is also referred to as a linear decomposition. For any width parameter defined via binary decomposition trees, the linear width of a graph is the minimum width over all caterpillar decompositions, see e.g. [33].

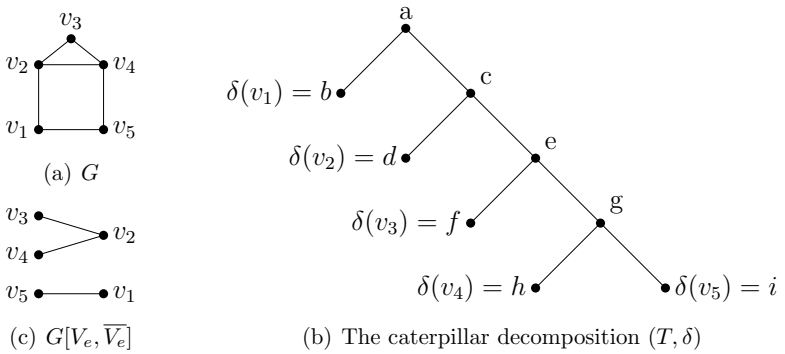


Figure 3.1: Subfigure (a) shows the graph G. Subfigure (c) shows the caterpillar decomposition (T, δ) of G, with a being the root of T. The ordering of $V(G)$ used to create (T, δ) is v_1, v_2, v_3, v_4, v_5. The node $e \in V(T)$ defines via δ the subset $V_e = \{v_3, v_4, v_5\}$. Subfigure (b) shows the bipartite graph $G[V_e, \overline{V_e}]$.

3.2 Module-width and Clique-width

Module-width is based on the definition of twins. Two vertices are twins if they have the same neighborhood, i.e. $x, y \in V(G)$ are *twins* if $N(x) \setminus y = N(y) \setminus x$ and they are *twins across a cut* $(A, \overline{A})$ if they have the same neighborhood across the cut, i.e. $x, y \in A$ are twins across the cut $(A, \overline{A})$ if $N(x) \cap \overline{A} = N(y) \cap \overline{A}$.

Definition 3.2.1 (Twin class partition)**.** Let G be a graph and $A \subseteq V(G)$ a subset of the vertices. The twin class partition of A, denoted $\mathcal{TC}_A$, is a partition of A such that $\forall x, y \in A$ we have x and y in the same partition class if and only if $N(x) \cap \overline{A} = N(y) \cap \overline{A}$. We define $ntc(A) = |\mathcal{TC}_A|$.

Definition 3.2.2 (Module-width)**.** For G a graph and $ntc\colon 2^{V(G)} \to \mathbb{N}$ the function defined above. Using Definition 3.1.3 with $f = ntc$ we define $modw(T, \delta)$ as the f-width of a binary decomposition tree (T, δ) and $modw(G)$ as the f-width of G, also called the *module-width* of G.

Note that $ntc(A)$ is not symmetric i.e. $ntc(A)$ is not necessarily equal to $ntc(\overline{A})$, hence it is important that the definition uses a rooted decomposition tree. Note that for the binary decomposition tree (T, δ) of the graph G in Figure 3.1 we have $modw(T, \delta) = modw(G) = 2$. The measure $modw(T, \delta)$ was first introduced in [63, Chapter 6.2] where it was called the bimodule-width of (T, δ).

We denote the clique-width of a graph G by $cw(G)$. We will not define clique-width, but since clique-width is a well-known graph width parameter we will mention results related to clique-width. All the results we will mention follows from the fact that module-width is linearly related to clique-width [64]. For an introduction to clique-width refer to [17]. Clique-width was introduced because many NP-hard graph problems are polynomial time solvable on "clique-like" graphs, however earlier width parameters were not able to handle these graphs, e.g. the tree-width of a clique on $n \geq 2$ vertices is $n - 1$ while the clique-width is 2.

3.3 Rank-width

Rank-width was introduced in [57, 60] and is based on the definition of the rank of a 0-1 matrix over $GF(2)$.

Definition 3.3.1 (Rank-width)**.** For G a graph, let $\textit{cut-rank}\colon 2^{V(G)} \to \mathbb{N}$ be a function where $\textit{cut-rank}(A)$ for $A \subseteq V(G)$ is the $GF(2)$ rank of the bipartite adjacency matrix of $G[A, \overline{A}]$. Using Definition 3.1.3 with $f = \textit{cut-rank}$ we define $rw(T, \delta)$ as the f-width of (T, δ) and $rw(G)$ as the f-width of G, also called the *rank-width* of G.

In Definition 3 of Chapter 8 we give an alternative definition of the *cut-rank* function, based on symmetric differences of neighborhoods across a cut, i.e. $\mathcal{DS}(A) = \{\triangle_{v \in X} N(v) \cap \overline{A} : X \subseteq A\}$. Note that for the binary decomposition tree (T, δ) of the graph G in Figure 3.1 we have $rw(T, \delta) =$

$rw(G) = 2$. An important application of rank-width is to efficiently approximate the clique-width of a graph [60], but these two parameters are not linearly related.

3.4 Tree-width

Since tree-width is the most well-known width-parameter we will relate our results to tree-width. Tree-width can be defined in various ways, also via a cops and robber game. The only place where we require the definition of treewitdh is in the proof of Lemma4.2.4 and that proof actually relies on the monotonicity property of the cops and robbers game, see [70]. We therefore define treewidth via the cops and robbers game, where the question is how many cops are needed to catch a visible and fast robber (minus one) on a given graph.

Definition 3.4.1 (Cops and robber game [70])**.** The robber occupies one vertex of the graph at any time and can at any time run at great speed to any other vertex along a path of the graph. The robber is not permitted to run through a vertex occupied by a cop. There are k cops, each of whom at any time either occupies a vertex or is being relocated (moved by a helicopter and can not capture the robber at this point). The objective of the player controlling the movement of the cops is to land a cop via helicopter on the vertex occupied by the robber, and the robber's objective is to elude capture. (The point of the helicopters is that cops are not constrained to move along paths of the graph – they move from vertex to vertex arbitrarily.) The robber can see the helicopter approaching its landing spot and may run to a new vertex before the helicopter actually lands but has to immediately find a new vertex to occupy. The cops have good intelligence services and know at all time which vertex the robber occupy, however the robber is faster than the police so they need to corner him by occupying all vertices adjacent to the vertex occupied by the robber and then send one cop to capture the robber.

A graph G has tree-width k if the minimal number of cops needed to capture a robber is $k + 1$. We denote by $tw(G)$ the tree-width of a graph. The linear version of tree-width is called path-width and can be defined via a similar cops and robbers game where the only difference is that the cops do not see where the robber is located [26].

3.5 Boolean-width

Boolean-width is a graph parameter first introduced in the paper making up Chapter 8 of this thesis. There are several ways to define boolean-width, we will now prove the equivalence of four set functions which can be used to define boolean-width. The first set function relates to neighborhoods across a cut.

Definition 3.5.1 (Union of Neighborhoods)**.** For a graph G and $A \subseteq V(G)$, we define the union of neighborhoods across the cut $(A, \overline{A})$ as

$$\mathcal{UN}(A) = \{N(X) \cap \overline{A} : X \subseteq A\}$$

The second set function comes from Boolean matrix theory (see [50]) and is the one that gave the name to boolean-width. A boolean matrix is a matrix with entries that are either 0 or 1, and in the boolean sum we have $1+1 = 1$.

Definition 3.5.2 (Boolean row space)**.** Let M be a boolean matrix. The boolean row space of M denoted $\mathcal{R}(M)$ is the smallest set of vectors containing the rows of M and the all-0 vector, and is closed under componentwise boolean sum. $|\mathcal{R}(M)|$ is called the boolean row span of M. Let G be any graph and $A \subseteq V(G)$. We define the boolean row span of A as the row span of the bipartite adjacency matrix of $G[A, \overline{A}]$.

It is known that the boolean row span is symmetric [50] i.e. the boolean row span of A equals the boolean row span of $\overline{A}$.

The third set function is via equivalence classes and is the most useful when designing algorithms.

Definition 3.5.3 (Neighborhood equivalence)**.** Let G be a graph and $A \subseteq V(G)$. Two vertex subsets $S_1, S_2 \subseteq A$ are neighborhood equivalent with respect to $(A, \overline{A})$, denoted by $S_1 \equiv_A S_2$, if $N(S_1) \cap \overline{A} = N(S_2) \cap \overline{A}$.

Definition 3.5.4 (Number of equivalence classes of $\equiv_A$)**.** For a graph G and $A \subseteq V(G)$ we denote by $nec(\equiv_A)$ the number of equivalence classes of $\equiv_A$. We also define for a binary decomposition tree (T, δ) the value $nec(T, \delta)$ as the max of $nec(\equiv_{V_a})$ over $a \in V(T)$.

The fourth set function is based on a well-known graph theoretic concept. Let $mis(G)$ denote the number of maximal independent sets in G, i.e. maximal under set inclusion. For $A \subseteq V(G)$ the set function we are interested in will be $mis(G[A, \overline{A}])$.

Now we are ready to show that all of these four set functions are equivalent.

Theorem 3.5.5. *Let G be a graph and $A \subseteq V(G)$. The following four values are equal:*

1. $|\mathcal{UN}(A)|$.
2. *The boolean row span of A.*
3. $nec(\equiv_A)$
4. $mis(G[A, \overline{A}])$

Proof. To see that $nec(\equiv_A) = |\mathcal{UN}(A)|$ see that for every $S \in \mathcal{UN}(A)$ there exist $X \subseteq A$ such that $N(X) \cap \overline{A} = S$ and all such X belong to the same equivalence class of $\equiv_A$. For every $X \subseteq A$ there is a unique element $S \in \mathcal{UN}(A)$ such that $N(X) \cap \overline{A} = S$, hence we have a bijection.

That $|\mathcal{UN}(A)|$ equals the boolean row span of A is easy to see since in a boolean sum $0+0 = 0$, $0+1 = 1+0 = 1+1 = 1$ and in an adjacency matrix 0 represents a non-neighbor and 1 represents a neighbor. Hence a boolean sum of a set of rows is equal to the union of the neighbors of the vertices corresponding to the rows. And the row space is then the same as the union closure of the neighborhoods across the cut, i.e. $\mathcal{UN}(A)$.

Finally we show that there is a bijection between $\mathcal{UN}(A)$ and the maximal independent sets of $G[A, \overline{A}]$ (this equivalence was first pointed out to us by Nathann Cohen [13]). For every $S \in \mathcal{UN}(A)$ we define $M(S) = \{v \in A : N(v) \cap \overline{A} \subseteq S\}$ the unique maximal subset of A having S as neighborhood. Clearly $I = M(S) \cup \overline{A} \setminus S$ is an independent set in $G[A, \overline{A}]$. Since $M(S)$ is maximal every vertex in $A \setminus I$ must have a neighbor in $\overline{A} \setminus S$ hence no vertices in $A \setminus I$ can be added to I. Since $\overline{A} \setminus I = S$ no vertices in $\overline{A}$ can be added to I. This shows that I is a maximal independent set of $G[A, \overline{A}]$ and hence $|\mathcal{UN}(A)| \leq mis(G[A, \overline{A}])$. For the other direction, for every maximal independent set I in $G[A, \overline{A}]$ we know that $N(I \cap A) = \overline{A} \setminus I$ otherwise I would not be a maximal independent set. Hence $\overline{A} \setminus I \in \mathcal{UN}(A)$ showing that there is a bijection between $\mathcal{UN}(A)$ and the maximal independent sets of $G[A, \overline{A}]$. This completes the proof. □

Definition 3.5.6 (Boolean-width)**.** For G a graph, let *bool-dim*: $2^{V(G)} \to \mathbb{R}$ be a function where $bool\text{-}dim(A) = \log_2(|\mathcal{UN}(A)|)$ for $A \subseteq V(G)$. Using Definition 3.1.3 with $f = bool\text{-}dim$ we define $boolw(T, \delta)$ as the f-width of a binary decomposition tree (T, δ) and $boolw(G)$ as the f-width of G, also called the *boolean-width* of G.

Note that for the binary decomposition tree (T, δ) of the graph G in Figure 3.1 we have $boolw(T, \delta) = 2$, but this is not a decomposition of optimal boolean width, instead using the ordering v_2, v_5, v_1, v_3, v_4 to construct a caterpillar decomposition shows that $boolw(G) = \log_2(3)$.

Theorem 3.5.5 gives the possibility of connecting boolean-width to results in various fields. There is a vast literature on number of maximal independent sets, the cardinality of the union closure of a set family and boolean matrix theory. See Chapter 8 for more about boolean-width. Let us mention a useful result that does not appear in that chapter.

Lemma 3.5.7. *For G a graph and $v \in V(G)$ it holds that:*

$$boolw(G \setminus v) \leq boolw(G) \leq boolw(G \setminus v) + 1$$

Proof. For any boolean decomposition (T, δ) of $G \setminus v$ we can add v by subdividing any edge and attaching a new leaf and let δ map v to the new leaf to obtain (T', δ'). Then clearly every graph $G[A, \overline{A}]$ defined by (T, δ) will have a corresponding graph defined by (T', δ') which can be obtained from $G[A, \overline{A}]$ by adding one vertex and the edges incident to that vertex, and $bool\text{-}dim(v) \leq 1$. We will show that adding a vertex to a bipartite graph H with color classes L, R can not decrease nor more than double $|\mathcal{UN}(L)|$. Since *bool-dim* is symmetric we may assume v is added to L. Every neighborhood in $\mathcal{UN}(L)$ is also in $\mathcal{UN}(L \cup v)$ therefore the first inequality holds. There can not be more than $|\mathcal{UN}(L)|$ elements $S \in \mathcal{UN}(L \cup v)$ such that $N(v) \cap \overline{A} \subseteq S$ since then for each such S there is a $X \subseteq L$ such that $N(X \cup v) \cap \overline{A} = S$ and all such X would yield a different element of $\mathcal{UN}(L)$. □

Corollary 3.5.8. *For G a graph and $e \in E(G)$ it holds that:*

$$boolw(G \setminus e) - 1 \leq boolw(G) \leq boolw(G \setminus e) + 1$$

Proof. If we remove one endpoint of e and then add that vertex again with all its adjacent edges except e we get $G \setminus e$. From Lemma 3.5.7 boolean-width can only go down at most 1 while removing a vertex and only go up at most 1 when adding a vertex, hence the corollary follows. □

3.6 Maximum Matching-width

To our knowledge, neither tree-width nor branch-width of a graph G has a characterization via a binary decomposition tree of G (based on a set function on vertex subsets). We will introduce a new parameter called Maximum Matching width (MM-width) defined via a binary decomposition tree of G and in Chapter 4 we will show that MM-width is linearly related to tree-width and branch-width. This parameter will help us understand the relationship between tree-width and the other parameters defined via binary decomposition trees.

Definition 3.6.1 (MM-width)**.** For G a graph, let $mm\colon 2^{V(G)} \to \mathbb{N}$ be a function where $mm(A)$ for $A \subseteq V(G)$ is the size of a maximum matching in $G[A, \overline{A}]$. Using Definition 3.1.3 with $f = mm$ we define $mmw(T, \delta)$ as the f-width of a binary decomposition tree (T, δ) and $mmw(G)$ as the f-width of G, also called the *MM-width* of G, or maximum matching width of G.

Note that for the binary decomposition tree (T, δ) of the graph G in Figure 3.1 we have $mmw(T, \delta) = mmw(G) = 2$.

3.7 Maximum Induced Matching-width

We will now introduce the third and last new width parameter of this thesis. It is called MIM-width and is based on the size of a maximum induced matching. There are several reasons to introduce MIM-width, one being that there are big classes of graphs where MIM-width is constant while none of the other parameters discussed in this thesis are constant. Another reason is that MIM-width is a natural extension of MM-width.

The third reason which lead us to the definition of MIM-width is a natural algorithmic application. The parameter boolean-width is defined via the number of neighborhoods across a cut. For a cut $(A, \overline{A})$, every neighborhood $S \in \mathcal{UN}(A)$ can be represented by a set $R \subseteq A$ with $N(R) \cap \overline{A} = S$. To achieve a more compact representation of the neighborhood we will choose an inclusion minimal set $R' \subseteq R$ such that $N(R') \cap \overline{A} = S$. It is then clear that every vertex in R' has a neighbor not shared with any other vertices in R'. Let M be a set of edges containing for each vertex v in R' an edge between v and one of its private neighbors. Then M forms an induced matching in $G[A, \overline{A}]$, also if there is an induced matching M such that $M \cap A = R'$ then no subset of R' has the same neighborhood as R' across the cut $(A, \overline{A})$. Hence the size of an maximum induced matching in $G[A, \overline{A}]$ equal the maximum size of such an inclusion minimal R'.

Definition 3.7.1 (MIM-width)**.** For G a graph and $A \subseteq V(G)$ let $mim\colon 2^{V(G)} \to \mathbb{N}$ be a function where $mim(A)$ is the size of a maximum induced matching in $G[A, \overline{A}]$. Using Definition 3.1.3 with $f = mim$ we define $mimw(T, \delta)$ as the f-width of a binary decomposition tree (T, δ) and $mimw(G)$ as the f-width of G, also called the *MIM-width* of G or the maximum induced matching width.

Note that for the binary decomposition tree (T, δ) of the graph G in Figure 3.1 we have $mimw(T, \delta) = 2$, but this is not a decomposition of optimal MIM-width, instead using the ordering v_2, v_5, v_1, v_3, v_4 to construct a caterpillar decomposition shows that $mimw(G) = 1$.

It is known that computing $mim(A)$ is NP-complete [71], moreover it is APX-hard [25]. See [10, 49] for more recent results on the induced matching problem. Therefore it is likely that computing $mimw(G)$ also is NP-hard.

The notion $mim(A)$ is equivalent to VC-dimension [76] of the set family $\mathcal{N}_A = \{N(v) \cap \overline{A} : v \in A\}$. A similar notion called the VC-dimension of a graph [53] is the VC-dimension of $\{N[v] : v \in V(G)\}$.

Adding a vertex to a graph does not alter MIM-width by more than 1.

Lemma 3.7.2. *For G a graph and $v \in V(G)$ it holds that:*

$$mimw(G \setminus v) \leq mimw(G) \leq mimw(G \setminus v) + 1$$

Proof. Let M be a maximum induced matching in $G \setminus v$, then M is also an induced matching in G, hence $mim(G \setminus v) \leq mim(G)$. Let M be a maximum induced matching in G, then there is at most one edge (u, v) in M having v as an endpoint. $M \setminus (u, v)$ is an induced matching in $G \setminus v$ hence $mim(G) \leq mim(G \setminus v) + 1$. □

Let us denote by MIM-1 the class of graphs having $mimw(G) = 1$. MIM-1 is an interesting class of graphs, little is known about this class. We show two lemmata initiating the research on this graph-class, in Chapter 9 we show that several well-known graph classes are contained in MIM-1.

Lemma 3.7.3. *For G a graph, if $mimw(G) = 1$ then $mimw(\overline{G}) = 1$.*

Proof. Let (T, δ) be a binary decomposition tree of G having $mimw(T, \delta) = 1$. Then we will show that (T, δ) is also decomposition tree of $\overline{G}$ having $mimw(T, \delta) = 1$. For $A \subseteq V(G)$ we have $mim(A) = 1$ if and only if for every pair of vertices $u, v \in A$ we have either $N(u) \cap \overline{A} \subseteq N(v) \cap \overline{A}$ or $N(v) \cap \overline{A} \subseteq N(u) \cap \overline{A}$ in G. Now, in $\overline{G}$ we have the opposite namely if $N(u) \cap \overline{A} \subseteq N(v) \cap \overline{A}$ in G then $N(v) \cap \overline{A} \subseteq N(u) \cap \overline{A}$ in $\overline{G}$ hence $mim(A) = 1$ also in $\overline{G}$. □

A graph G is called perfect if and only if neither G nor $\overline{G}$ contains an induced cycle of odd length at least 5 [12].

Corollary 3.7.4. *MIM-1 is perfect.*

Proof. Let C_k be a cycle on k vertices. It is easy to see that for any $k \geq 5$ we have $mimw(C_k) = 2$, hence by lemma 3.7.3 we have that $mimw(\overline{C_k}) = 2$. From Lemma 3.7.2 we see that any graph containing C_k as a subgraph has MIM-width at least 2, hence no graph in MIM-1 contains C_k for $k \geq 5$ as an induced subgraph. □

Open Problem 1. Can we recognize graphs of MIM-width 1 in polynomial time?

Chapter 4

Comparing the Values of Graph Width Parameters

In this chapter we start out in Section 4.1 by comparing the value of some well-known graph parameters, and defining partial orders of these parameters. Then in Section 4.2 we incorporate into these partial orders the three new parameters: Boolean-width, MM-width and MIM-width. In Section 4.3 we compare the parameters on some special classes of graphs, in particular random graphs, interval graphs, d-degenerate graphs and grid graphs.

4.1 Relations Between Some Well-known Graph Width Parameters

In this section we have chosen a set of seven well-known graph parameters that are typically used as parameters to design FPT algorithms for NP-complete graph problems. The parameters we will study are: path-width (pw), tree-width (tw), branch-width (brw), the size of a minimum feedback vertex set (fvs), clique-width (cw), module-width ($modw$) and rank-width (rw).

To get an overview of the values of these parameters on arbitrary graphs, we will map out their relations by defining partial orders of graph parameters and draw Hasse diagrams for these partial orders. The question of how various width parameters relate to each other has been well studied. For instance, tree-width relates linearly to branch-width and clique-width relates linearly to module-width, as seen by the following theorems:

Theorem 4.1.1. *[66] Let G be a graph then $brw(G) \leq tw(G)+1 \leq \frac{2}{3}brw(G)$.*

Theorem 4.1.2. *[64] Let G be a graph then $modw(G) \leq cw(G) \leq 2modw(G)$.*

This information is useful and tells us that if there is an algorithm solving a problem in $O^*(2^{O(brw(G))})$ time, then there is also an algorithm solving that problem in $O^*(2^{O(tw(G))})$ time. Likewise for clique-width and module-width. Let us consider a somewhat weaker relation between two graph parameters. Although clique-width is bounded on a class $\mathcal{C}$ if and only if rank-width is bounded on $\mathcal{C}$, they are not linearly related.

Theorem 4.1.3 ([60])**.** *Let G be a graph then $rw(G) \leq cw(G) \leq 2^{rw(G)+1}-1$.*

This kind of theorem tells us that a problem is FPT parameterized by rank-width if and only if it is FPT parameterized by clique-width. But a problem could have a runtime single exponential in clique-width while only double exponential in rank-width, i.e. for a graph G we could have runtime $O^*(2^{poly(cw(G))})$ as a function of clique-width but not better than $O^*(2^{2^{poly(rw(G))}})$ as a function of rank-width. Yet another possible relation is illustrated by clique-width and tree-width, since clique-width is bounded on any class where tree-width is bounded but not the other way around.

Theorem 4.1.4. *[14] Let G be a graph then $cw(G) \leq 1.5 \cdot 2^{tw(G)}$.*

Such a theorem tells us that if a problem is FPT parameterized by clique-width then it is also FPT parameterized by tree-width, but not necessarily the converse. Also, note that if we prove a problem W-hard parameterized by tree-width then it follows that the problem is also W-hard parameterized by clique-width. To capture these different kinds of relations we will use three partial orders.

Definition 4.1.5. Let P and Q be two graph parameters. We say $P \leq_f Q$ if there exists a function f such that for all graphs G we have $P(G) \leq f(Q(G))$.

Definition 4.1.6. Let P and Q be two graph parameters. We say $P \leq_{poly} Q$ if there exists a polynomial function *poly* such that for all graphs G we have $P(G) \leq poly(Q(G))$.

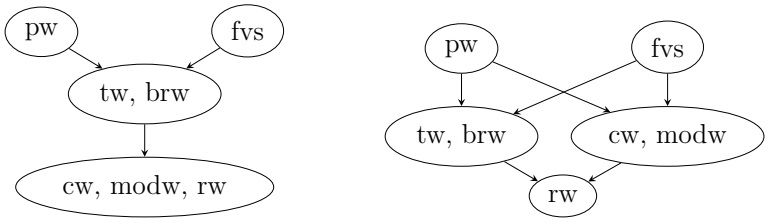
Definition 4.1.7. Let P and Q be two graph parameters. We say $P \leq_{lin} Q$ if for all graphs G we have $P(G) \in O(Q(G))$.

It is easy to see that these orders are reflexive and transitive, hence we have a pre-order. In order to obtain partial orders we define:

$$P =_f Q \text{ if } P \leq_f Q \text{ and } Q \leq_f P$$
$$P =_{poly} Q \text{ if } P \leq_{poly} Q \text{ and } Q \leq_{poly} P$$
$$P =_{lin} Q \text{ if } P \leq_{lin} Q \text{ and } Q \leq_{lin} P$$

Note that $\leq_{lin}$ is an extension of $\leq_{poly}$ and $\leq_{poly}$ is an extension of $\leq_f$, i.e. $P \leq_{lin} Q \to P \leq_{poly} Q \to P \leq_f Q$.

We will draw Hasse diagrams of these partial orders to get a better overview. Parameters which are equal in the partial order are drawn inside the same node and for each pair P, Q such that P is less than Q there is drawn an arc from Q to P, see Figure 4.1.



(a) Using $\leq_f$ (b) Using $\leq_{poly}$ or $\leq_{lin}$

Figure 4.1: Hasse diageams for the three partial orders $\leq_f$, $\leq_{poly}$ and $\leq_{lin}$ on seven well-known graph parameters. The diagrams for $\leq_{poly}$ and $\leq_{lin}$ are identical.

Lemma 4.1.8. *Figure 4.1 is the Hasse diagrams of the partial orders $\leq_f$ and $\leq_{lin}$ on the parameters $\{fvs, pw, tw, brw, cw, modw, rw\}$.*

Sketch of proof. To show that an arrow from P to Q in the diagrams should not be present is easy, one only needs a graph class where the relation does not hold, e.g. a graph class where parameter Q is unbounded while parameter P is bounded. Since Path-width is unbounded on trees while all other parameters in the diagram are bounded on trees no arrow points to path-width in any of the diagrams. Since Feedback Vertex Set is unbounded on a collection of triangles while all other parameters are bounded on this class no arrows points to fvs in any of the diagrams. From the definition it follows easily that $tw \leq_{lin} pw$ and since adding a vertex can increase tree-width by at most 1 and tree-width of trees is 1 it follows that $tw \leq_{lin} fvs$. Theorem 4.1.1 shows that $tw =_{lin} brw$.

From Theorem 4.1.4 it follows that $cw \leq_f tw$ and the fact that for cliques tree-width is $n-1$ while clique-width is 2 it follows that $tw \not\leq_f cw$. Theorem 4.1.2 show that $cw =_{lin} modw$. Theorem 4.1.3 shows that $rw =_f modw$. This shows that Figure 4.1(a) is correct.

That $cw \not\leq_{poly} tw$ follows from [14]. Clique-width of trees is at most 3 and adding a vertex to a graph can increase the clique-width by at most 1

therefore we have $cw \leq_{lin} fvs$. It is well-known that $cw \leq_{lin} pw$ and that $rw \leq_{lin} cw$ follows from [60] while that $rw \leq_{lin} tw$ follows from [58] (an alternative proof given in [1], see Subsection 4.2.1). From [14] it also follows that $cw \not\leq_{poly} rw$ in Figure 4.1(b). This completes the proof. □

4.2 Relating the New Width Parameters

We now incorporate boolean-width, MM-width and MIM-width into the partial orders displayed in Figure 4.1. First we will prove the relations we need, then the updated Hasse diagram is presented in Figure 4.2.

4.2.1 MM-width

We start by relating MM-width to branch-width. To do this we need the following well-known theorem relating the Maximum Matching (MM) problem to the Minimum Vertex Cover (Min VC) problem.

Theorem 4.2.1 (Königs theorem [52])**.** *Let H be a bipartite graph, $M \subseteq E(H)$ a maximum matching and $C \subseteq V(H)$ a minimum vertex cover, then $|M| = |C|$.*

Finding a maximum matching in a bipartite graph is a well-known problem that can be solved in $O(m\sqrt{n})$ time using the Hopcroft-Karp algorithm [46]. Hence computing the MM-width of a binary decomposition tree can be done in $O(nm\sqrt{n}) \in O(n^{3.5})$ time. The next lemma shows a close connection between minimum vertex covers and separators. Since separators are closely related to branch-width this indicates a close relation between MM-width and branch-width.

Lemma 4.2.2. *Let G be a graph, $A \subseteq V(G)$ and C a vertex cover of $G[A, \overline{A}]$. Then C is a separator of G into $A \setminus C$ and $\overline{A} \setminus C$.*

Proof. Assume for contradiction that C is not a separator of G into $A \setminus C$ and $\overline{A} \setminus C$. Then there must exist $u \in A \setminus C$ and $v \in \overline{A} \setminus C$ such that $(u, v) \in E(G)$, but then we also have $(u, v) \in E(G[A, \overline{A}])$. Hence (u, v) would have no endpoint in C contradicting that C is a vertex cover. □

We now show how MM-width relates to tree-width and branch-width, first we prove a lemma for branch-width.

Lemma 4.2.3. *Let G be a graph, then $mmw(G) \leq \max(brw(G), 1)$.*

Proof. Since both $mmw(G)$ and $brw(G)$ for a disconnected graph G is the maximum over all its connected components, it suffices to prove the lemma for connected graphs.

Letting (T_B, δ_B) be a branch decomposition of G, we will construct a binary decomposition tree (T_M, δ_M). Make a binary tree T'_M from T_B by subdividing an arbitrary edge of T_B and making the new vertex r the root of T'_M, then connecting two new leaves to each leaf in T'_M so that T'_M will be a binary tree with $2|E(G)|$ leaves. Note that $V(T_B) \subseteq V(T'_M)$ hence every node in T_B is also a node in T'_M. For each leaf l in T_B let δ'_M map the two endpoints of the edge mapped to l to the two children of l in T'_M. Now δ'_M may map some vertex $v \in V(G)$ to more than one leaf of T'_M, while that is the case, arbitrarily remove one of the leaves which v is mapped to (if one of the leaves is a child of the root r, remove a leaf that is not a child of the root). This will leave the parent x of the removed leaf ($x \neq r$) with degree 2. Make the unique child of x a child of the parent of x and delete x. Continue in this way decreasing the number of leaves in T'_M until we obtain the binary tree T_M with $|V(G)|$ leaves and δ_M a bijection between $V(G)$ and $V(T_M)$.

Note that the internal nodes of T_M is a subset of $V(T_B)$, hence every internal node of T_M is also a node in T_B. For any vertex w in T_M, recall that V_w denotes the vertices mapped to leaves of the subtree rooted at w. If $w \in V(T_M)$ is a leaf or $w = r$ (for r, the root of T_M) then the value $mm(V_w)$ is at most 1. If $w \in V(T_M)$ is not a leaf and $w \neq r$ then w has a parent p. First we will find an edge $e_w \in E(T_B)$, if $p = r$ let e_w be the edge of T_B that was subdivided to create r. Else both w and p are in T_B (but not necessarily adjacent in T_B). Let e_w be the edge adjacent to w on the path in T_B from w to p. Denote the edges of G mapped to leaves of the sub-tree of $T_B \setminus e_w$ containing w by E_w.

Let M be a maximum matching in $G[V_w, \overline{V_w}]$, then for every edge (u, v) in M with $u \in V_w$ and $v \in \overline{V_w}$ there must exist an edge $(u, x) \in E_w$ and $(v, y) \in E(G) \setminus E_w$. Since either $(u, v) \in E_w$ or $(u, v) \in E(G) \setminus E_w$ either u or v must be in the middle set of E_w. Hence by definition of branch-width $mmw(T_M, \delta_M) \leq brw(T_B, \delta_B)$. □

The above bound is tight up to an additive constant factor e.g. on grid graphs. The next lemma will upper bound tree-width in terms of MM-width and hence show that tree-width and MM-width are linearly related. To show this we rely on the min-max theorem for tree-width by Seymour and Thomas [70] and design a strategy for the cops and robber game. The strategy we design is non-monotone, i.e. for $v \in V(G)$ a cop may be placed on a vertex v, then removed again and later again a cop is placed on v. It

is non-trivial to transform a non-monotone strategy for the cops and robber game into a tree decomposition.

Lemma 4.2.4. *Let G be a graph, then $tw(G) \leq 3 \cdot mmw(G) - 1$.*

Proof. We will use the equivalence between tree-width and cops and robber games, see [70]. Given a binary decomposition tree (T, δ) of G we will use (T, δ) to design a strategy for catching a robber using at most $3mmw(T, \delta)$ cops. For every node $u \in V(T)$ let VC_u be a minimum vertex cover of $G[V_u, \overline{V_u}]$, we know from Königs theorem that $|VC_u| \leq mmw(T, \delta)$.

We will design a strategy in stages such that at the beginning of every stage there is a node $w \in T$ such that the cops occupy only VC_w and the robber is hiding in $V_w \setminus VC_w$. This is initially true at the root r since $V_r = V(G)$ and $VC_r = \emptyset$, hence the strategy we design will start with $w = r$ and move down the tree every stage. Lemma 4.2.2 shows that when VC_w is occupied by cops there is no way for the robber to move from $V_w \setminus VC_w$ to $\overline{V_w} \setminus VC_w$.

Let a and b be the children of w. Place cops on all vertices in $VC_a \cup VC_b$. Now the robber is either in $V_a \setminus VC_a$ or in $V_b \setminus VC_b$. Assuming without loss of generality that the robber is in $V_a \setminus VC_a$, remove all cops except those placed on VC_a. Now we know that the robber is hiding in $V_a \setminus VC_a$ and we have only placed cops on VC_a hence this is the beginning of the next stage and we can repeat the process with $w = a$.

At every stage w is moved further away from the root, hence this process will end in a linear number of stages. This process catches the robber using at most $3mmw(G)$ cops, if given a binary decomposition tree of optimal MM-width. Then it follows from [70] that $tw(G) \leq 3mmw(G) - 1$. □

Note that due to the nature of decompositions defined by symmetrical cut functions it is trivial to bound MM-width by $n/3$, see Subsection 4.3.3. Hence the above bound is tight e.g. on cliques where if n is a multiple of 3 then $n - 1 = tw(K_n)$ and $mmw(K_n) = \lceil \frac{n}{3} \rceil$.

Open Problem 2. Given a binary decomposition tree (T, δ) can we compute a tree decomposition of tree-width at most $3mmw(T, \delta) - 1$ in $O(n^{3.5})$ time?

Lemmata 4.2.3 and 4.2.4 gives us:

Theorem 4.2.5. *Let G be a graph, then*

$$\frac{1}{3}(tw(G) + 1) \leq mmw(G) \leq max(brw(G), 1) \leq tw(G) + 1$$

We can now with this new insight that MM-width is at most the branch-width give an alternative simpler proof of the fact that for any graph G we have $rw(G) \leq brw(G)$ [58].

Theorem 4.2.6. *Let G be a graph, then $rw(G) \leq mmw(G)$.*

Proof. For any $A \subseteq V(G)$ we show that $\textit{cut-rank}(A) \leq mm(A)$, from this the theorem follows. Let R be a basis of the rows corresponding to the vertices in A, then for any $X \subseteq R$ we have that the $GF(2)$ sum is unique i.e. $|\{\triangle_{v \in X} N(v) \cap \overline{A} : X \subseteq R\}| = 2^{|R|}$. This implies that for every $X \subseteq R$ we have $|N(X) \cap \overline{A}| \geq |X|$ otherwise X can not generate $2^{|X|}$ different sums. Then by Halls theorem [43] we have a matching of size at least $|R|$ hence this theorem holds. □

4.2.2 Boolean-width

We start by showing that boolean-width never exceeds MM-width. The following Lemma follows from [1], we give a simplified proof here.

Lemma 4.2.7. *Let G be a graph, then $boolw(G) \leq mmw(G)$.*

Proof. Fix $A \subseteq V(G)$ and let VC be a minimum vertex cover of $G[A, \overline{A}]$, from Königs theorem we know that $|VC| = mm(A)$. We will show that $mis(G[A, \overline{A}]) \leq 2^{|VC|}$, then by Theorem 3.5.5 it follows that $\textit{bool-dim}(A) \leq mm(A)$ and hence the lemma will follow.

Let $S \subseteq VC$ be any subset of the vertices in the vertex cover. For any maximal independent set X of $G[A, \overline{A}]$ such that $X \cap VC = S$ we can partition $V(G)$ into four types:

- VC,
- the isolated vertices which all must be in X,
- $N(S) \setminus VC$ of which none can be in X and
- $N(VC) \setminus (VC \cup N(S))$ of which all must be in X.

For a fixed $S \subseteq V(G)$ every $v \in V(G)$ fits into exactly one of the four types and X is uniquely defined by the partition of $V(G)$ into the four types. Hence there is a unique maximal independent set X of $G[A, \overline{A}]$ such that $X \cap VC = S$, and this shows that there are at most $2^{|VC|}$ maximal independent sets of $G[A, \overline{A}]$. Thus by Theorem 3.5.5 and Königs theorem we have:

$$\textit{bool-dim}(A) = log_2(mis(G[A, \overline{A}]) \leq log_2(2^{|VC|}) = |VC| = mm(A)$$

□

Theorem 4.2.8 ([1]). *Let G be a graph, then $boolw(G) \leq tw(G) + 1$.*

Proof. The theorem follows from Lemma 4.2.7 and Theorem 4.2.5. □

Open Problem 3. Is $boolw(G) \leq tw(G)$ for all graphs G?

It is known that any graph G with $boolw(G) = tw(G) + 1$ would need to have at least 8 vertices [2] and tree-width at least 2.

We now show that boolean-width is less than clique-width.

Theorem 4.2.9 (Chapter 8 Theorem 3). *Let G be a graph, then*

$$\log_2(cw(G)) - 1 \leq boolw(G) \leq modw(G) \leq cw(G)$$

Proof. We will first prove that for any cut $(A, \overline{A})$ we have:

$$ntc(A) \leq |\mathcal{UN}(A)| \leq 2^{ntc(A)}$$

Note that by definition any $x, y \in A$ belong to the same twin-class of A if and only if $N(x) \cap \overline{A} = N(y) \cap \overline{A}$, so every twin-class yields a unique element in $|\mathcal{UN}(A)|$. Therefore the number of twin classes of A is at most $|\mathcal{UN}(A)|$.

For the second inequality, note that if $X \subseteq A$ contains two vertices from the same twin class we can remove one of them without changing the neighborhood across $(A, \overline{A})$. Hence we can generate at most $2^{ntc(A)}$ unions of neighborhoods from the $ntc(A)$ twin classes, i.e. $|\mathcal{UN}(A)| \leq 2^{ntc(A)}$. This implies $bool\text{-}dim(A) \leq ntc(A)$.

Since for every binary decomposition tree (T, δ) this holds for every cut $(V_a, \overline{V_a})$ defined by (T, δ), it allows to conclude that $\log_2(modw(G)) \leq boolw(G)$. It is known that for any graph G we have $modw(G) \leq cw(G) \leq 2 \cdot modw(G)$ [64]. Hence the theorem follows. □

4.2.3 MIM-width

MIM-width is the smallest of all parameters we discuss in this Chapter.

Theorem 4.2.10. *Let G be a graph, then*

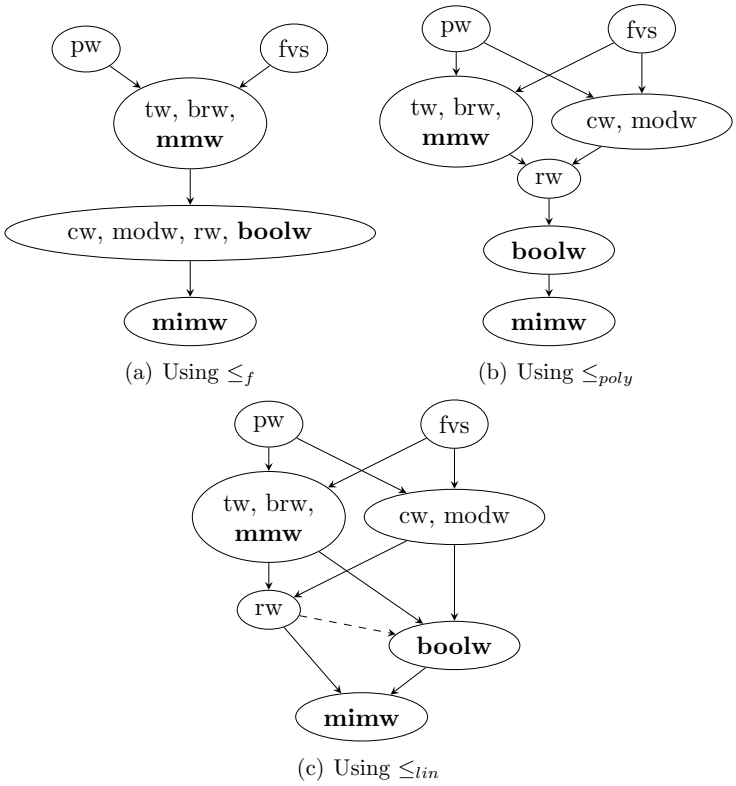
$$mimw(G) \leq boolw(G) \leq mimw(G) \log_2(n)$$

Proof. For $A \subseteq V(G)$ let M be a maximum induced matching in $G[A, \overline{A}]$. It is clear that no two subsets of $M \cap A$ have the same neighborhood in $M \cap \overline{A}$ hence $|\mathcal{UN}(A)| \geq 2^{mim(A)}$ which proves the first inequality. The second inequality holds by Chapter 9 Lemma 2. □

MIM-width is bounded on interval graphs, while boolean-width is not, hence $boolw \not\leq_f mim$.

4.2.4 Comparison Diagram of Graph Parameters

In Figure 4.2 the three new parameters boolean-width, MM-width and MIM-width have been added to the Hasse diagrams in Figure 4.1 and there are now a total of 10 parameters.



(a) Using $\leq_f$

(b) Using $\leq_{poly}$

(c) Using $\leq_{lin}$

Figure 4.2: The Hasse diagrams for the three partial orders on ten graph parameters. The three new graph parameters are in bold. The dotted line in Figure 4.2(c) is unknown. We conjecture that $boolw \not\leq_{lin} rw$ see Chapter 8 Section 4.

Theorem 4.2.11. *Figure 4.2 is the Hasse diagrams of* $\leq_f, \leq_{poly}$ *and* $\leq_{lin}$ *on* $\{fvs, pw, tw, brw, mmw, cw, modw, rw, boolw, mimw\}$.

Proof. That $mm =_{lin} tw$ follows from Theorem 4.2.5 and $boolw =_f cw$ follows from Theorem 4.2.9. That $boolw \leq_{poly} rw$ is shown in Chapter 8 Corollary 1

and that $rw \not\leq_{poly} boolw$ is shown in Chapter 8 Theorem 2. That $mim \leq_{lin} boolw$ follows from Theorem 4.2.10. That $mim \leq_{lin} rw$ follows from the fact that the adjacency matrix of an induced matching is the identity matrix. The remaining relations follow from Lemma 4.1.8. □

In Figure 4.3 we have included a total of 32 parameters in the Hasse diagram for $\leq_f$. It is part of an ongoing project to fill these Hasse diagrams with even more parameters. The explanation of the different abbreviations used in the figures can be found in Figure 4.4. Most of the relations follow from the results in this Chapter or from [69], some relatively easy ones are left without a proof. To make the Hasse diagrams easier to read we have divided the diagrams into four regions; **top left** is bounded on cliques and unbounded on trees, **top right** is unbounded on both cliques and trees, **lower left** is bounded on both cliques and trees, **lower right** is unbounded on cliques and bounded on trees.

4.3 The Parameter Value on Restricted Graph Classes

All time complexities considered in this thesis are worst case runtimes. For graph problems those worst case inputs could possibly be avoided, and it is common to consider also the runtime of graph problems when restricting the input to a certain graph class. In this section we will see some results on restricted graph classes that highlight the differences between several graph width parameters.

4.3.1 Random Graphs

Let us first consider random graphs generated by the Erdös-Rényi model. For a constant $0 < p < 1$ the Erdös-Rényi model generates a graph G_p on n vertices where independently for every pair of vertices an edge is added with probability p. This is not a restricted graph class since any graph in principle could be generated by the Erdös-Rényi model, however the probability for a specific graph to be generated tends to 0 as n increases. Therefore the probability that for a particular problem we get a worst case input may also tend to 0 as n increases. The following theorem shows that for random graphs the expected boolean-width is dramatically lower than tree-width, clique-width, rank-width and MM-width, and MIM-width is even lower than boolean-width.

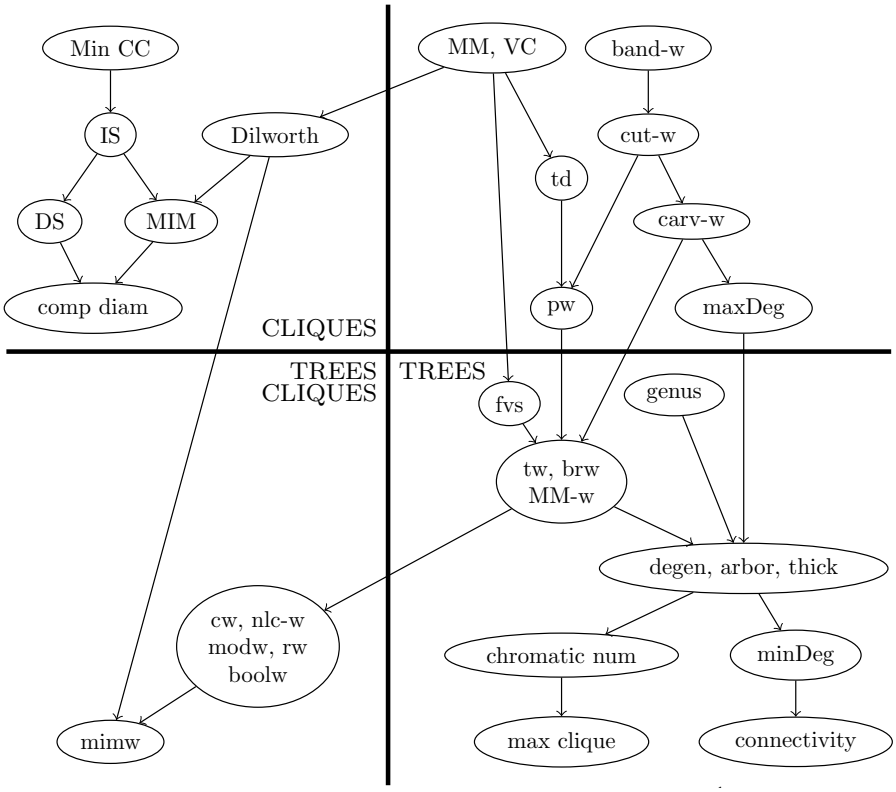


Figure 4.3: A Hasse diagram of the $\leq_f$ relation, with each quadrant containing parameters bounded on either cliques, trees, both, or none, as indicated. See Figure 4.4 for explanation of abbreviations.

Min CC Size of a minimum clique cover.
IS Size of a maximum independent set.
DS Size of a minimum dominating set.
MIM Size of a maximum induced matching.
comp diam Maximum diameter over all connected components.
Dilworth Dilworth number.
MM Size of a maximum matching.
VC Size of a minimum vertex cover.
td Tree Depth or equivalently Minimum Vertex Ranking.
band-w Band-width.
cut-w Cut-width.
pw Path-width.
carv-w Carving-width.
maxDeg Maximum Degree.
fvs Size of a minimum Feedback Vertex Set.
genus Genus.
tw Tree-width.
brw Branch-width.
MM-w Maximum matching width (MM-width).
degen Degeneracy.
abor Aboricity.
thick Thickness.
Chromatic num Chromatic number.
max clique The size of a maximum clique.
minDeg Minimum degree.
connectivity Connectivity.
cw Clique-width.
nlc-w NLC-width.
modw Module-width.
rw Rank-width.
boolw Boolean-width.
mimw Maximum Induced Matching width (MIM-width).

Figure 4.4: Abbreviations used in Figures 4.3

Theorem 4.3.1 ([1, 48, 51, 55]). *For a constant $0 < p < 1$, let G_p be generated by the Erdös-Rényi model, then almost surely:*

$tw(G_p) \in \Theta(n)$
$cw(G_p) \in \Theta(n)$
$rw(G_p) \in \Theta(n)$
$mmw(G_p) \in \Theta(n)$
$boolw(G_p) \in \Theta(\log(n)^2)$
$mimw(G_p) \in \Theta(\log n)$

Proof. The three first results follow from [51], [48] and [55] respectively. For MM-width the theorem follows since MM-width is linearly bounded by tree-width by Theorem 4.2.5. For boolean-width the theorem follows from [1, Theorem 4]. For MIM-width the Theorem follows from [1] Lemma 7. □

4.3.2 Graphs with an Intersection Model

Chapter 9 studies the boolean-width of several classes of graphs defined by having an intersection model. See Figure 4.5 for an overview of results. Let us consider in detail the class of interval graphs. To define interval graphs we need the notion of an interval. An interval $I = \langle i, j \rangle$ is represented by an ordered pair of real numbers with $i < j$ and represent the set of real numbers $\{x : i < x < j\}$. Let $I_1 = (a, b)$ and $I_2 = (c, d)$ be two intervals, then I_1 intersects I_2 if and only if $a < d$ and $c < b$.

Definition 4.3.2 (Interval graph). Let $\mathcal{I}$ be a family of intervals. For a graph G, we say G is an interval graph if we can associate to each vertex in $V(G)$ an interval in $\mathcal{I}$ such that two vertices are adjacent if and only if their corresponding intervals intersect.

Theorem 4.3.3 ([40], Chapter 9). *Let $\mathcal{C}$ be the family of interval graphs on n vertices. For any $n > 1$ we have:*

$\exists G \in \mathcal{C} : tw(G) = n - 1$
$\exists G \in \mathcal{C} : mmw(G) = \lceil \frac{n}{3} \rceil$
$\exists G \in \mathcal{C} : cw(G) \in \Theta(\sqrt(n))$
$\exists G \in \mathcal{C} : rw(G) \in \Theta(\sqrt(n))$
$\forall G \in \mathcal{C} : boolw(G) \leq \log_2(n)$
$\forall G \in \mathcal{C} : mimw(G) = 1$

Proof. To prove the theorem for tree-width and MM-width let G be a clique on n vertices. For clique-width the theorem follows from [40]. For rank-width the theorem follows from Chapter 9 Corollary 18. For boolean-width and MIM-width the theorem follows from Chapter 9 Lemma 3. □

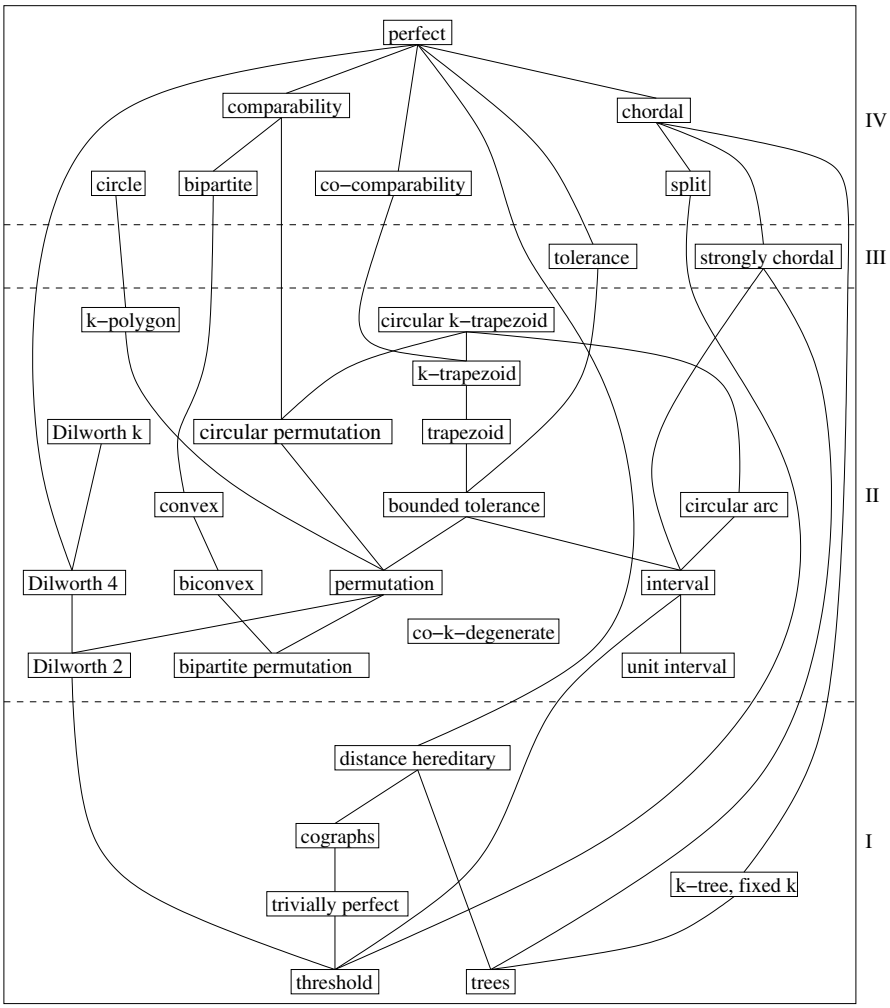


Figure 4.5: Inclusion diagram of some well-known graph classes.
(I) Graph classes where clique-width is bounded by a constant.
(II) Graph classes where MIM-width is bounded by a constant and boolean-width is $O(\log n)$.
(III) It is unknown whether these classes have constant MIM-width or boolean-width $O(\log n)$.
(IV) Either boolean-width is not $O(\log n)$ or it is NP-hard to compute such decompositions.

A similar theorem can be proven for any of the following graph classes: Bipartite permutation graphs, Unit interval graphs, Circular k-trapezoid graphs, k-polygon graphs, Convex graphs and complements of k-degenerate graphs, since all these graph classes have MIM-width $O(1)$ and contains graphs with rank-width $\Theta(\sqrt{(}n))$. See Figure 4.5 and Chapter 9. These theorems show that there are many graph classes where tree-width, clique-width and rank-width is exponentially larger than boolean-width, and MIM-width is bounded while none of the other parameters are bounded.

4.3.3 d-degenerate Graphs

A graph G is d-degenerate if there exists a total ordering of $V(G)$ such that no node has more than d neighbors occurring later in the order. We will now state a theorem bounding the different width parameters in terms of degeneracy of a graph. This theorem however, does not show much difference between the parameters. Let us start with the following lemma for MM-width. Since module-width, rank-width, boolean-width and MIM-width are all bounded by MM-width a corresponding lemma holds for any of these.

Lemma 4.3.4. *Let G be a graph and $S \subseteq V(G)$ a subset of the vertices, then if $|S| \leq \frac{n}{4}$ and $|N(S) \setminus S| \leq \frac{n-|S|}{3}$ them $mmw(G) \leq \left\lceil \frac{n-|S|}{3} \right\rceil$.*

Proof. We build a binary decomposition tree (T, δ). See Figure 4.6. When we build the decomposition we start by a root r with two children c_l and c_r. Let $V_{c_l} \subseteq V(G)$ be such that $|V_{c_l}| = \left\lceil \frac{n+2|S|}{3} \right\rceil$ and $N[S] \subseteq V_{c_l}$. Construct the binary subtree T_l rooted at c_l such that c_l has the two children a and b and T_l has $|V_{c_l}|$ leafs, half of the leaves have a as an ancestor and half of the leaves have b as an ancestor. Let δ arbitrarily map the vertices in V_{c_l} to the leaves of T_l. Let $V_{c_r} = \overline{V_{c_l}}$, construct the binary subtree T_r rooted at c_r such that c_r has the two children c and d and T_r has $|V_{c_r}|$ leafs, half of the leaves have c as an ancestor and half of the leaves have d as an ancestor. Let δ arbitrarily map the vertices in V_{c_r} to the leaves of T_r.

Both V_a and V_b have size at most $\lceil \frac{n+2|S|}{6} \rceil$. Since $\frac{2|S|}{3} \leq \frac{n}{6}$ we get $\frac{n+2|S|}{6} \leq \frac{n+2|S|}{6} - \frac{2|S|}{3} + \frac{n}{6} = \frac{n-|S|}{3}$. Both V_c and V_d have size at most $\lceil \frac{n-|S|}{3} \rceil$. Clearly since no vertex in S has a neighbor outside A we have:

$$mm(V_{c_r}) = mm(V_{c_l}) \leq |V_{c_l}| - |S| \leq \left\lceil \frac{n+2|S|}{3} \right\rceil - |S| = \left\lceil \frac{n-|S|}{3} \right\rceil$$

□

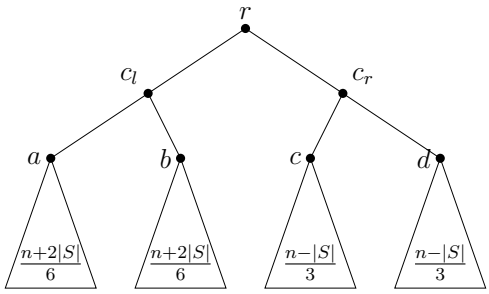


Figure 4.6: The decomposition tree of Lemma 4.3.4. The number of leaves in each subtree is marked.

Theorem 4.3.5. *For all graphs G such that G is d-degenerate:*

$tw(G) \leq n - \frac{n}{d+3} + 2$
$mmw(G) \leq \frac{n}{3} - \frac{n}{9d+3} + 1$
$cw(G) \leq \frac{2n}{3} - \frac{2n}{9d+3} + 1$
$rw(G) \leq \frac{n}{3} - \frac{n}{9d+3} + 1$
$boolw(G) \leq \frac{n}{3} - \frac{n}{9d+3} + 1$
$mimw(G) \leq \frac{n}{3} - \frac{n}{9d+3} + 1$

Proof. For $d = 0$ and $d = 1$ it is easy to see that the theorem holds. We only prove the theorem for mmw, then the rest will follow from the Lemmata in the previous section (to prove clique-width we know $cw(G) \leq 2modw(G) \leq 2mmw(G)$). Let S be the $\lfloor \frac{n}{3d+1} \rfloor$ first nodes in the degeneracy order. Then $|N_S(S)| \leq \frac{dn}{3d+1} = \frac{n}{3} - \frac{n}{9d+3}$. For all values of d greater than 1 it is easy to check that S satisfies the conditions in Lemma 4.3.4. We take the floor when finding $|S|$ which contributes at most $\frac{1}{3}$, and take ceiling in Lemma 4.3.4 which contributes at most $\frac{2}{3}$, hence the theorem follows. □

This theorem is tight on cliques for tree-width and MM-width, but for the other parameters the bounds are not tight.

Open Problem 4. Can we improve the bounds in Theorem 4.3.5?

4.3.4 Grid Graphs

For two integers p and q we define $G_{p,q}$ to be the $p \times q$ grid graph where for each $i, j : 1 \leq i \leq p$ and $1 \leq j \leq q$ we have $v_{i,j} \in V(G_{p,q})$ (called the vertex in the i'th row and j'th column) with two vertices $v_{i,j}$ and $v_{k,l}$ adjacent if and only if $|i - k| + |j - l| = 1$.

Theorem 4.3.6 ([32]). *Let P_n be a path on n vertices, then $mis(P_n) \in O(2^{0.406 \cdot n})$.*

Construct a total ordering σ of $V(G_{p,q})$ by sorting the vertices $v_{i,j} \in V(G_{p,q})$ according to the sum $i + j$, with ties broken by the value of i. Construct a caterpillar decomposition $(T, \delta)_\sigma$ of the grid graph $G_{p,q}$ as described in Definition 3.1.5 using the ordering σ. See Figure 4.7(a).

Lemma 4.3.7. *The boolean-width of the grid graph $G_{p,q}$ is at most $0.812 \cdot \min(p, q)$, for sufficiently large p and q, as shown by decomposition $(T, \delta)_\sigma$.*

Proof. For every cut $(A, \overline{A})$ defined by $(T, \delta)_\sigma$ the bipartite graph induced by the cut $G_{p,q}[A, \overline{A}]$ has at most $2 \cdot \min(p, q) + 2$ not isolated vertices, at most two components and max degree 2. In other words it is either one path or two paths of a combined length of at most $2 \min(p, q) + 2$. We get from Theorem 4.3.6 that $2^{boolw((T,\delta)_\sigma)} \in O(2^{0.812 \cdot \min(p,q)})$. □

For a long while we believed this to be the best possible way to decompose a grid graph and even backed up our conjecture with randomized computer aided searches. It turned out to not be the best way to decompose a grid graph in terms of boolean-width.

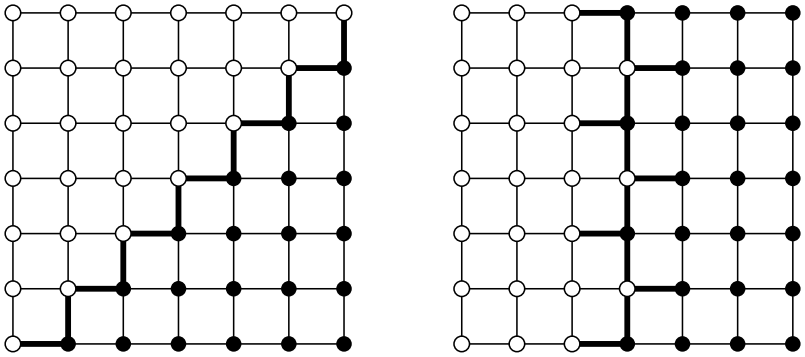
Assume without loss of generality that $p \leq q$, construct an ordering ρ of $V(G_{p,q})$ by sorting the vertices $v_{i,j} \in V(G_{p,q})$ according to $j + (i \ mod \ 2)$ with ties broken by the value of i. Construct a caterpillar decomposition $(T, \delta)_\rho$ of the grid graph $G_{p,q}$ as described in Definition 3.1.5 using the ordering ρ. See Figure 4.7(b).

Lemma 4.3.8. *The boolean-width of the grid graph $G_{p,q}$ is at most $0.695 \cdot \min(p, q)$, for sufficiently large p and q, as shown by decomposition $(T, \delta)_\rho$.*

Proof. For every cut $(A, \overline{A})$ defined by $(T, \delta)_\rho$ the graph $G_{p,q}[A, \overline{A}]$ after removing isolated vertices is at most 2 caterpillars with sum of the lengths at most $\min(p, q) + 2$. The number of maximal independent sets in a caterpillar is exactly the number of independent sets of its core path (i.e. the path induced by all vertices with degree at least 2). The number of independent sets in a path follows the Fibonacci sequence [62]. The Fibonacci sequence is defined by the recurrence $F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1$. Let $\phi = 1.618...$ denote the golden ratio, it is well-known that $F(n)$ can be described by the following formula:

$$F(n) = \frac{\phi^n - \left(\frac{-1}{\phi}\right)^n}{\sqrt{5}}$$

Since ϕ^n is the dominant term and $log_2(\phi) \leq 0.695$ the lemma follows. □



(a) Using σ (b) Using ρ

Figure 4.7: Two cuts $(A, \overline{A})$ described by the two discussed decompositions $(T, \delta)_\sigma$ and $(T, \delta)_\rho$ of a 7×7 grid. The edges across the cut are drawn bold, vertices of A are white while vertices of $\overline{A}$ are black.

The following lemma will help get a lower bound on the MIM-width of a grid graph.

Lemma 4.3.9. *Let G be graph and $A \subseteq V(G)$. If $G[A, \overline{A}]$ is d-degenerate and has a matching of size m, then $G[A, \overline{A}]$ has an induced matching of size $\frac{m}{d+1}$.*

Proof. Sort the m edges in the matching by their last endpoint in the degeneracy ordering. We construct an induced matching as follows. Greedily pick edges in that order if they are not adjacent to any already picked edge. Whenever an edge is put in the induced matching there is at most d later edges adjacent to that edge that hence can not be in the induced matching. Hence this greedy method will produce an induced matching of size $\frac{m}{d+1}$. □

Theorem 4.3.10 ([47])**.** *Let $G_{n,n}$ be a $n \times n$ grid graph (for sufficiently large n), then:*

$tw(G_{n,n}) = n$ *[51]*
$cw(G_{n,n}) = n + 1$ *[48]*
$rw(G_{n,n}) = n - 1$ *[55]*
$n - 1 \leq mmw(G_{n,n}) \leq n$
$0.33n \leq boolw(G_{n,n}) \leq 0.695n$
$0.33n \leq mim(G_{n,n}) \leq 0.5(n + 1)$

Proof. For MM-width it is easy to see that $mmw(G_{n,n}) \leq n$ using the caterpillar decomposition $(T, \delta)_\rho$ in Lemma 4.3.8, see Figure 4.7(b). Vít Jelínek showed in [47] that any decomposition tree of $G_{n,n}$ defines a cut with an acyclic matching of size at least $n-1$ and hence also a matching of size $n-1$. That $boolw(G_{n,n}) \leq 0.695n$ follows from Lemma 4.3.8. Since grid graphs are 2-degenerate Lemma 4.3.9 gives us that $0.33n \leq mim(G_{n,n}) \leq boolw(G)$. The decomposition in Figure 4.7(b) shows that $mim(G_{n,n}) \leq 0.5(n+1)$. □

It would be nice to close the gap in the bounds of $boolw(G_{n,n})$ and $mimw(G_{n,n})$ in Theorem 4.3.10, this is part of ongoing research.

Chapter 5

Parameterized Algorithms

In this chapter we give an overview of results that parameterize some well-known graph problems by various different graph parameters and consider the resulting parameterized complexity. We assume throughout the chapter that we are given a graph G with $n = |V(G)|$. When we parameterize a problem by a graph parameter P we use as parameter the value $k = P(G)$. For a particular problem parameterized by P we will consider the following three complexity behaviors:

para-NP-hard For some constant parameter value k the problem is NP-hard.

XP The problem is solvable in $O(n^{f(k)})$ time.

FPT The problem is solvable in $O(f(k) \cdot poly(n))$ time.

5.1 Monadic Second Order Logic

A large and well-known class of graph problems are the MSOL_2 problems, consisting of those graph problems that can be expressed in monadic second order logic. See [16] for a full definition. The following is arguably the most well-known result in the area of FPT algorithms for graph problems.

Theorem 5.1.1 ([15, 4])**.** *Any $MSOL_2$ problem is FPT parameterized by tree-width.*

The relation between graph parameters illustrated by the Hasse diagram in Figure 4.3 can be used to derive consequences of this Theorem.

Lemma 5.1.2. *Consider two graph parameters P and Q and assume that $P \leq_f Q$. If a problem Π is FPT parameterized by P then Π is FPT parameterized by Q. Moreover, if Π parameterized by Q is Para-NP-hard then so is Π parameterized by P.*

Proof. We know from definition of $\leq_f$ that there is a function g such that for any graph G we have $P(G) \leq g(Q(G))$ and from definition of FPT we know that there is a function f such that Π is solvable in time $O(f(P(G)) \cdot poly(n)$. Then it follows that Π is solvable in time $O(f(g(Q(G))) \cdot poly(n))$ which is FPT parameterized by Q.

If Π parameterized by Q is Para-NP-hard then there exists a constant c such that Π parameterized by Q is NP-hard on the class of graphs $\{G : Q(G) \leq c\}$. By assumption $P(G) \leq g(Q(G))$ and hence the problem is NP-hard on the class of graphs $\{G : P(G) \leq g(c)\}$ and thus Para-NP-hard parameterized by P. □

The derived consequences of Theorem 5.1.1 are given in the following Lemma and illustrated in Figure 5.1.

Lemma 5.1.3. *Consider Figure 5.1. For all red parameters there exists an $MSOL_2$ problem which is para-NP-hard and for all green parameters all $MSOL_2$ problems are FPT.*

Proof. The node containing tree-width is green since Theorem 5.1.1 states that any MSOL_2 problem is FPT parameterized by tree-width. All the nodes colored green are greater than tree-width in the partial order $\leq_f$ hence by Lemma 5.1.2 any MSOL_2 problem is FPT also on these. There exist MSOL_2 problems that are NP-hard on cliques [18], hence all the parameters bounded on cliques, i.e. on the left side, should be red. MAXIMUM INDEPENDENT SET is NP-hard on planar graphs of maximum degree 3 [35], and since both genus and maximum degree is bounded on this class it follows from Lemma 5.1.2 that all parameters below any of these two also should be red and this completes the proof. □

Another well-known result in the area of FPT algorithms for graph problems concerns parameterizing by clique-width. The MSOL_1 problems are those MSOL_2 problems that can be expressed in monadic second order logic without allowing logical quantifications over edge subsets. See [18] for a full definition. Certain problems, like HAMILTON CYCLE, belong to MSOL_2 but not MSOL_1.

Theorem 5.1.4 ([18],[60])**.** *Any $MSOL_1$ problem is FPT parameterized by clique-width.*

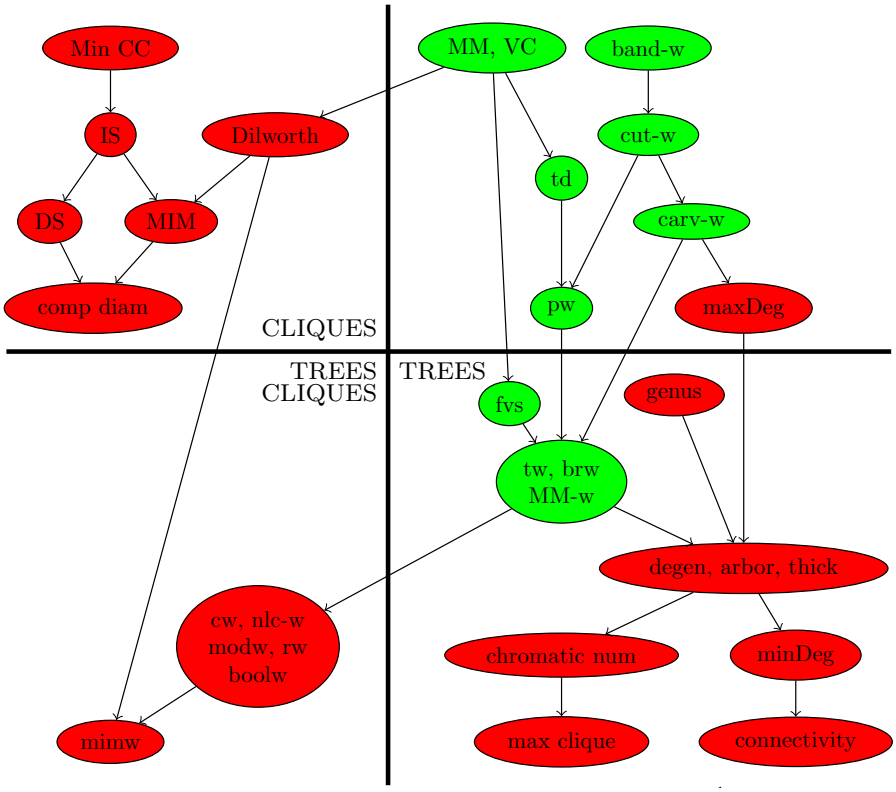


Figure 5.1: The parameterized complexity of MSOL_2 problems. Red means there exists an MSOL_2 problem which is para-NP-hard and green means all MSOL_2 problems are FPT.

Note that the result in Theorem 5.1.4 applies the FPT approximation algorithm to clique-width given by the FPT algorithm for rank-width of [60] and subsequently the result of [18]. Again, this Theorem has some derived consequences given by Lemma 5.1.5 and illustrated in Figure 5.2.

Lemma 5.1.5. *Consider Figure 5.2. For all red parameters there exists an MSOL_1 problem which is para-NP-hard, for all green parameters all MSOL_1 problems are FPT (Dilworth number it is open).*

Proof. Theorem 5.2 shows that all MSOL_1 problems are FPT parameterized by clique-width, and all the green nodes then follow from Lemma 5.1.2. Minimum Weight Dominating Set is in MSOL_1 and is NP-hard on complements of bipartite graphs [11]. Since complements of bipartite graphs have

clique cover of size 2, i.e. Min CC ≤ 2, all red nodes in the top left quadrant follows from Lemma 5.1.2.

Maximum Independent Set is NP-hard on planar graphs of max degree 3 [35], hence Maximum Clique is NP-hard on complements of planar graphs. Since both genus and max degree is bounded on planar graphs of max degree 3 all red nodes on the right side of the figure follows from Lemma 5.1.2. It can be proven in a way similar to Chapter 9 Lemma 13 that complements of planar graphs have MIM-width at most 6, hence Maximum Clique is NP-hard on graphs of MIM-width at most 6. □

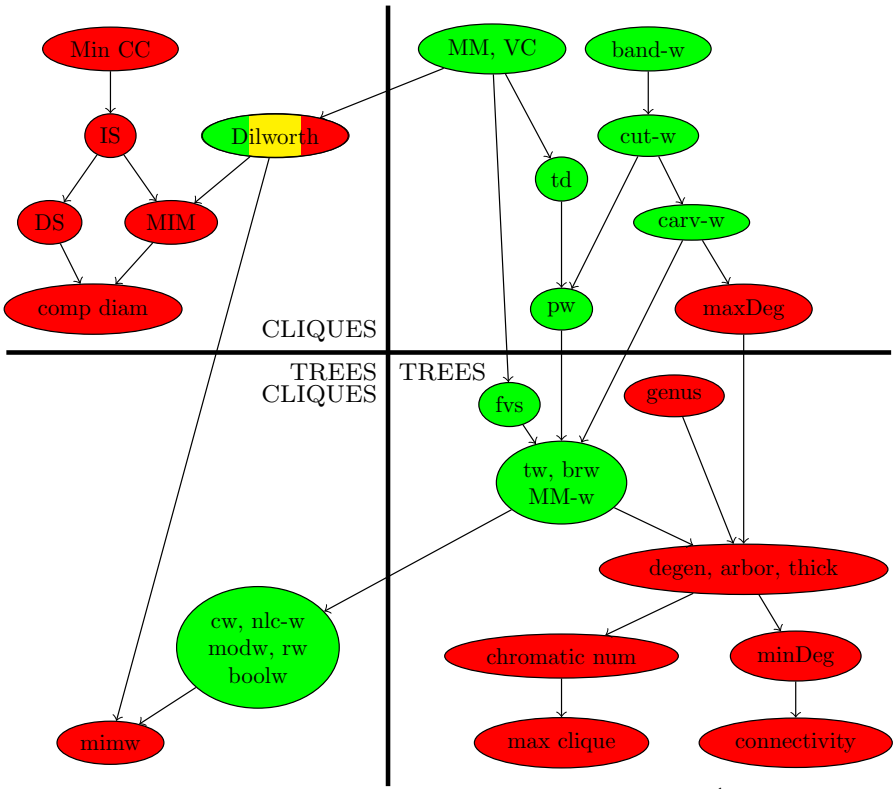


Figure 5.2: The parameterized complexity of MSOL_1 problems. Red means there exists an MSOL_1 problem which is para-NP-hard and green means all MSOL_1 problems are FPT. The complexity parameterized by Dilworth number is open.

The meta-algorithms for dynamic programming for MSOL problems parameterized by tree-width and clique-width have two stages, firstly finding a decomposition (which will be discussed in the next chapter) and secondly doing dynamic programming. The dynamic programming stage for general MSOL problems has a runtime which is not single exponential in the parameter value.

5.2 Some Locally Checkable Vertex Subset and Vertex Partitioning Problems

We will now consider a subclass of the MSOL_1 problems first introduced in [74], for which the dynamic programming stage has linear single exponential runtime as a function of the parameter, i.e. $O^*(2^{O(k)})$, for several parameters. These problems are defined formally in Chapter 10, where they are called Locally Checkable Vertex Subset and Vertex Partitioning problems, LC-VSVP for short. Well-known LC-VSVP problems can be found in Chapter 10 Tables 1 & 2. In Chapter 10 we give dynamic programming algorithms to solve any LC-VSVP problem. To express the runtime of these algorithms by the various width parameters we need some intermediate lemmata based on the following definition, which is also given in Chapter 10, motivated by the algorithmic application.

Definition 5.2.1 (d-neighbor equivalence)**.** Let d be a non-negative integer, G a graph and $A \subseteq V(G)$. Two vertex subsets $X \subseteq A$ and $Y \subseteq A$ are *d-neighbor equivalent* with respect to A, denoted by $X \equiv_A^d Y$, if:

$$\forall v \in \overline{A} : \min(d, |X \cap N(v)|) = min(d, |Y \cap N(v)|)$$

Let $nec(\equiv_A^d)$ be the number of equivalence classes of $\equiv_A^d$. For a binary decomposition tree (T, δ) we define $nec_d(T, \delta)$ as the maximum of $nec(\equiv_{V_w}^d)$ and $nec(\equiv_{\overline{V_w}}^d)$ over all nodes $w \in T$.

We now give two lemmata that will help us to bound $nec(\equiv_A^d)$ by $ntc(A)$ (see Definition 3.2.2) and $mim(A)$ (see Definition 3.7.1).

Lemma 5.2.2. *For G a graph, $A \subseteq V(G)$ and d a non-negative integer we have $nec(\equiv_A^d) \leq (d+1)^{min(ntc(A), ntc(\overline{A}))}$.*

Proof. Let $P \in \mathcal{TC}_{\overline{A}}$ and $S \subseteq A$. By definition of twin class partition, for any pair $u, v \in P$ we have $|N(u) \cap X| = |N(v) \cap S|$. Hence we can describe $N(S) \cap \overline{A}$ as a vector of length $ntc(\overline{A})$ with entries from $\{0, 1, \ldots, d-1, \geq d\}$

and two subsets of $\overline{A}$ are d-neighbor equivalent if they have the same vector. Hence $nec(\equiv_A^d) \leq (d+1)^{ntc(\overline{A})}$.

For $X, Y \subseteq A$ it is clear that if for all $P \in \mathcal{TC}_A$ we have $|P \cap X| = |P \cap Y|$ then $X \equiv_A^d Y$. Let $R \subseteq A$ be an inclusion minimal set of an equivalence class of $\equiv_A^d$, i.e. for all $u \in R$ we have $R \not\equiv_A^d (R \setminus u)$ and let $P \in \mathcal{TC}_A$. Then $|R \cap P| \leq d$. Assume for contradiction that $|R \cap P| \geq d+1$ and let $x \in R \cap P$, then for all $v \in \overline{A} \setminus N(u)$ we have $|R \cap N(v)| = |(R \setminus x) \cap N(v)|$ and for all $v \in \overline{A} \cap N(u)$ we have $|R \cap N(v)| \geq d+1$ and $|(R \setminus x) \cap N(v)| \geq d$. Hence $nec(\equiv_A^d) \leq (d+1)^{ntc(A)}$. □

Lemma 5.2.3. *For G a graph, $A \subseteq V(G)$ and d a non-negative integer we have $nec(\equiv_A^d) \leq ntc(A)^{mim(A) \cdot d}$.*

Proof. For any $S \subseteq A$ we know from Chapter 9 Lemma 2 that there exists $R \subseteq S$ such that $|R| \leq mim(A) \cdot d$ and $R \equiv_A^d S$. For $X, Y \subseteq A$ it is clear that if for all $P \in \mathcal{TC}_A$ we have $|P \cap X| = |P \cap Y|$ then $X \equiv_A^d Y$ hence the lemma holds. □

The following theorem summarizes what is known about the parameterized complexity of LC-VSVP problems when given a decomposition. Its proof relies on the $\leq_{lin}$ relation between the parameters, see Figure 4.2(c).

Theorem 5.2.4 ([74] Chapter 10)**.** *Given a graph G and a decomposition of (any of the following)-width k we can solve any LC-VSVP problem by:*

tree-width in $O^*(2^{O(k)})$
branch-width in $O^*(2^{O(k)})$
MM-width in $O^*(2^{O(k)})$
module-width in $O^*(2^{O(k)})$
clique-width in $O^*(2^{O(k)})$
rank-width in $O^*(2^{O(k^2)})$
boolean-width in $O^*(2^{O(k^2)})$
MIM-width in $n^{O(k)}$

Proof. For tree-width the theorem follows from [74]. For branch-width the theorem follows since branch-width and tree-width are linearly bounded (see Theorem 4.2.5) and we can transform a branch decomposition of branch-width k into a tree decomposition of tree-width at most $\frac{3}{2}k$ in polynomial time. For MM-width the theorem follows since MM-width and tree-width are linearly bounded (see Theorem 4.2.5) and we can compute a tree decomposition of width $O(k)$ in time $O^*(2^{O(k)})$ [65]. For module-width, whose set function is *ntc*, the theorem follows from Chapter 10 Theorems 1 & 2 and Lemma 5.2.2. For clique-width the theorem follows since module-width and

clique-width are linearly bounded (see Theorem 4.1.2) and we can transform a clique-expression of width k into a module decomposition of width at most $2k$ in polynomial time. For rank-width note that $ntc(A) \leq 2^{cut\text{-}rank(A)}$ [60] and $mim(A) \leq cut\text{-}rank(A)$ (the adjacency matrix on an induced matching is the identity matrix which has full rank). The theorem for rank-width follows from Lemma 5.2.3 and Chapter 10 Theorems 1 & 2. For boolean-width the theorem follows from Chapter 10 Corollaries 1 & 2. For MIM-width the theorem follows from Chapter 9 Lemma 2 and Chapter 10 Theorems 1 & 2. □

It is an interesting question whether all LC-VSVP problems are in XP parameterized by MIM-width. The above theorem answers the question positively for the dynamic programming stage, however the question of approximating the optimal MIM-width of a graph is open.

Open Problem 5. Is there a function f such that, given a graph G, we can in XP time parameterized by $k = mimw(G)$ compute a binary decomposition tree of MIM-width at most $f(k)$?

The runtime of the algorithms for LC-VSVP problems in Chapter 10 is expressed in terms of $nec(\equiv_A^d)$ where d is a problem specific constant. Note that by definition $nec(\equiv_A^1) = 2^{bool\text{-}dim(A)}$. The subclass of the LC-VSVP problems for which the problem specific constant d is 1 includes among others the problems Maximum Independent Set and Minimum Dominating Set, H-Coloring and H-Role Assignment, see Chapter 10 Table 1 & 2 for more problems. From Chapter 10 Theorem 1 & 2 we get the following FPT algorithm as a corollary.

Corollary 5.2.5. *Given a graph G and a binary decomposition tree of boolean-width k we can solve any LC-VSVP problem with the problem specific constant $d = 1$ in time $O^*(2^{O(boolw(G))})$.*

5.3 Independent Set and Dominating Set

For the well-studied LC-VSVP problems Maximum Independent Set and Minimum Dominating Set, we give a more precise statement of the runtime.

Theorem 5.3.1 ([23, 41, 73],Chapter 8)**.** *Given a graph G and a decomposition of (any of the following)-width k we can solve the* Maximum Independent Set *problem by:*

tree-width in $O^*(2^k)$
branch-width in $O^*(2.28^k)$
clique-width in $O^*(2^k)$
boolean-width in $O^*(4^k)$
module-width in $O^*(4^k)$
MM-width in $O^*(4^k)$
rank-width in $O^*(1.42^{k^2})$
MIM-width in $O^*(n^{2k})$

Proof. For tree-width the theorem follows from [73]. For branchwidth the theorem follows from [23]. For clique-width the theorem follows from [41]. For boolean-width the theorem follows from Chapter 8 Theorem 5. By Theorem 4.2.9 boolean-width is bounded by module-width (using the same decomposition tree), hence the theorem holds for module-width. By Theorem 4.2.5 boolean-width is bounded by MM-width (using the same decomposition tree), hence the theorem holds for MM-width. From Chapter 8 Corollary 1 we know that boolean-width is at most $\frac{1}{4}rw^2 + O(rw)$ where rw is the rank-width and since $4^{1/4} < 1.42$ the theorem holds for rank-width. By Theorem 4.2.10 we know that boolean-width is bounded by MIM-width times $log(n)$ and $4^{klog(n)} = n^{2k}$, hence the theorem holds for MIM-width. □

Theorem 5.3.2 ([75, 9] Chapter 8)**.** *Given a graph G and a decomposition of (any of the following)-width k we can solve the* Minimum Dominating Set *problem by:*

tree-width in $O^*(3^k)$
branch-width in $O^*(3.69^k)$
clique-width in $O^*(4^k)$
boolean-width in $O^*(8^k)$
module-width in $O^*(8^k)$
MM-width in $O^*(8^k)$
rank-width in $O^*(1.69^{k^2})$
MIM-width in $O^*(n^{3k})$

Proof. For tree-width the theorem follows from [75]. For branch-width and clique-width the theorem follows from [9]. For boolean-width the theorem follows from Chapter 8 Theorem 7. By theorem 4.2.9 boolean-width is bounded by module-width, hence the theorem holds for module-width. By Theorem 4.2.5 boolean-width is bounded by MM-width, hence the theorem holds

for MM-width. From Chapter 8 Corollary 1 we know that boolean-width is at most $\frac{1}{4}rw^2+O(rw)$ where rw is the rank-width and since $8^{1/4} < 1.69$ the theorem holds for rank-width. By Theorem 4.2.10 we know that boolean-width is bounded by MIM-width times $log(n)$ and $8^{klog(n)} = n^{3k}$. □

Open Problem 6. Given a graph G and a binary decomposition tree of rank-width k, can MAXIMUM INDEPENDENT SET be solved in time $O^*(2^{O(k)})$?

5.4 Feedback Vertex Set

We now ask the question if there are $MSOL_2$ problems that are not LC-VSVP problems but still are solvable in $O^*(2^{O(k)})$ time when given a decomposition of width k. HAMILTONIAN PATH, STEINER TREE, FEEDBACK VERTEX SET and CONNECTED DOMINATING SET are $MSOL_2$ problems, and none are LC-VSVP problems. Given a graph G of tree-width k any of these problems can be solved in $O^*(2^{O(k)})$ time by a randomized algorithm [19]. We focus on FEEDBACK VERTEX SET in this section.

Theorem 5.4.1 ([19, 34] Chapter 11)**.** *Given a graph G and a decomposition of (any of the following)-width k we can solve the* MINIMUM FEEDBACK VERTEX SET *problem parameterized by:*

tree-width in $O^*(3^k)$
branch-width in $O^*(5.2^k)$
MM-width in $O^*(2^{O(k)})$
clique-width in $O^*(32^{k\cdot\log_2(k)})$
module-width in $O^*(32^{k\cdot\log_2(k)})$
boolean-width in $O^*(32^{2^k\cdot k})$
rank-width in $O^*(32^{k^2})$

Proof. For tree-width the theorem follows from [19]. For branch-width the theorem follows since a branch decomposition of width k can be transformed into a tree decomposition of width at most $\frac{3}{2}k$ in polynomial time and $2^{3/2} < 5.2$. For module-width the theorem follows from Chapter 11. For clique-width the theorem follows since a clique-expression of width k can be transformed into a binary decomposition tree of module-width at most k in polynomial time. For boolean-width the theorem follows since a binary decomposition tree of boolean-width k has module-width at most 2^k. For rank-width the theorem follows from [34]. □

Chapter 6

Computing Decompositions

In this chapter we discuss algorithms for finding decompositions of a given graph G, for various width parameters. There are several ways to approach the problem of computing graph decompositions. We can design an algorithm to find an optimal decomposition for a specific parameter, and analyse the runtime of this algorithm either as a function of $|V(G)| = n$ only, which we call an exact algorithm, or we can analyse the runtime as a parameterized algorithm. We can also compute decomposition that only approximate the optimal width, or use heuristics. We give an overview of results regarding the different approaches.

6.1 Exact Algorithms

A general exponential time algorithm for computing an optimal branch decomposition of a set function is presented in [29] based on work by Oum [59].

Theorem 6.1.1 ([59])**.** *For any set A where $|A| = n$ and any set function $f\colon 2^A \to \mathbb{R}$, an optimal branch decomposition of f on A can be computed in $O^*(2^n \cdot t(f,n))$ time, where $t(f,n)$ is the time it takes to evaluate f.*

We showed in Observation 3.1.4 that branch decompositions are equivalent to binary decomposition trees for all set functions discussed in this thesis, so we can use this algorithm also to compute an optimal binary decomposition tree. For MM-width, module-width and rank-width the evaluation of the set functions can be done in polynomial time [42, 44, 46]. For boolean-width and MIM-width evaluating the set function is NP-hard [54, 35].

Lemma 6.1.2. *Given a graph G, we can compute a binary decomposition tree having optimal boolean-width in $O(2.52^n)$ time.*

Proof. The set function used to define boolean-width is *bool-dim*. For $A \subseteq V(G)$ we know from Theorem 3.5.5 that computing *bool-dim*(A) can be done by counting the number of maximal independent sets in the bipartite graph $G[A, \overline{A}]$. To compute an optimal decomposition we rely on Theorem 6.1.1. In general graphs, counting the number of maximal independent sets can be done in $O(1.3642^n)$ time [36], as far as we know this is also the best exact algorithm for bipartite graphs, hence $t(\textit{bool-dim}, n) \in O(1.3642^n)$. For the bipartite graphs appearing as cuts in binary decomposition trees having optimal boolean-width we can improve on this. This since, for decomposition trees (subcubic or binary) we can see, by balancing the decomposition tree properly, that any set function bounded by $min(|A|, \overline{A})$ yields a width of at most $\frac{n}{3}$.

We define a new cut function $\textit{minbool}\colon 2^{V(G)} \to \mathbb{R}$, with $\textit{minbool}(A) = min(\textit{bool-dim}(A), \frac{n}{3})$. For bipartite graphs, there are algorithms enumerating independent sets with polynomial delay [21], i.e. the time between each maximal independent set the algorithm outputs, is polynomial in n. We can stop the polynomial time delay algorithm as soon as it has output $2^{n/3}$ maximal independent sets, hence $t(\textit{minbool}, n) \in O^*(2^{n/3})$. Chapter 8 Algorithm 1 can also be used to evaluate $\textit{minbool}(A)$, however the algorithm of [21] is more memory efficient.

We can use Theorem 6.1.1 to compute a binary decomposition tree (T, δ) of optimal *minbool*-width in time $O^*(2^n \cdot 2^{n/3}) \in O(2.52^n)$. For every $u \in V(T)$ we know that $\textit{minbool}(V_u) \leq \frac{n}{3}$ hence $\textit{minbool}(V_u) = \textit{bool-dim}(V_u)$, thus (T, δ) has optimal boolean-width. □

The following Theorem gives an overview of the best exact algorithms for computing optimal decompositions for various graph width parameters.

Theorem 6.1.3 ([30, 31, 59])**.** *Given a graph G, we can compute a decomposition having optimal (any of the following)-width for:*

tree-width in $O^*(1.76^n)$
branch-width in $O^*(3.47^n)$
module-width in $O^*(2^n)$
rank-width in $O^*(2^n)$
MM-width in $O^*(2^n)$
boolean-width in $O^*(2.52^n)$
MIM-width in $O^*(2.79^n)$

Proof. For tree-width the theorem follows from [31]. For branch-width the theorem follows from [30]. For module-width, rank-width and MM-width the theorem follows from Theorem 6.1.1. For boolean-width the theorem follows

from Lemma 6.1.2. For MIM-width we know that computing $mim(A)$ for any $A \subseteq V(G)$ can be done in $O^*(1.392^n)$ time [3], combined with Theorem 6.1.1 we get that we can compute MIM-width in $O(1.392^n \cdot 2^n) \in O(2.79^n)$ time. □

6.2 Parameterized Algorithms

For graphs where the parameter value is low compared to the number of vertices we prefer parameterized algorithms. From a theoretical point of view an interesting question is whether an optimal decomposition can be computed in FPT time parameterized by k, the width of the optimal decomposition. If this is not the case, how good an approximation can we find in FPT time?

Theorem 6.2.1 ([3, 4, 8, 39, 61, 59])**.** *Given a graph G having (any of the following)-width k, we can in FPT time parameterized by k compute a decomposition having:*

tree-width k
branch-width k
MM-width at most $3k$
rank-width k
module-width at most 2^k
clique-width at most 2^{k+1}
boolean-width at most 4^k

Proof. For tree-width the theorem follows from [4]. For branch-width the theorem follows from [8]. For MM-width we use that we can compute an optimal branch decomposition in FPT time [8] and we know from Lemma 4.2.3 that a branch decomposition of branch-width w can be turned into a binary decomposition tree of MM-width at most w in polynomial time. From Theorem 4.2.5 we know that $w \leq 3k$, where k is the MM-width of G, hence the theorem follows. For rank-width the theorem follows from [61]. For module-width and clique-width the theorem follows from [61]. For boolean-width we know from [39] (see Chapter 8 Corollary 1) and [61] that if G has rank-width rw we can compute a binary decomposition tree having boolean-width at most $\frac{1}{4}rw^2 + rw$ and from Chapter 8 Lemma 1 we know that $rw \leq 2^k$ for k the boolean-width of G, and since $\frac{1}{4}(2^k)^2 + 2^k \leq 4^k$ for all $k \geq 1$ the theorem for boolean-width follows. □

The FPT algorithm for computing decompositions of optimal rank-width depends on the fact that the set function used for rank-width called *cut-rank*

is submodular. A set function $f : 2^A \to \mathbb{N}$ is submodular if for all $X, Y \subseteq A$ we have $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$. For the set functions *bool-dim* and *mim* we know that they are not submodular, see Figure 6.1, hence the approach used to compute rank-width is not directly applicable.

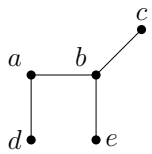


Figure 6.1: The sets $X = \{a, c\}$ and $Y = \{b, c\}$ certify that neither *bool-dim* nor *mim* are submodular set functions. $\textit{bool-dim}(X) = log_2(3)$, $\textit{bool-dim}(Y) = 1$, $\textit{bool-dim}(X \cup Y) = 2$, $\textit{bool-dim}(X \cap Y) = 1$, and $mim(X) = 1$, $mim(Y) = 1$, $mim(X \cup Y) = 2$, $mim(X \cap Y) = 1$.

Much of the literature on FPT algorithms to compute optimal graph decompositions focus on reducing the polynomial dependency in n, and often the dependency in k is not even stated, but rather hidden in the big-O notation. However, the dependency in k makes a big impact on the runtime in practice. The following algorithm controls both the dependency on k and n and the approximation ratio is linear.

Theorem 6.2.2 ([65])**.** *Given a graph G with tree-width k we can compute a tree decomposition of tree-width at most $4k$ in $O(27^k \cdot n \cdot \log(n))$ time.*

The parameters for which we can find decompositions of width at most a constant factor more than the optimal width k in time $O^*(2^{O(k)})$ are tree-width, branch-width and MM-width.

Open Problem 7. For a graph G with boolean-width k, rank-width k, module-width k or MIM-width k, can we in time $O^*(2^{O(k)})$ compute a decomposition having width $O(k)$?

6.3 Heuristics

In this section we discuss how to get around the fact that the algorithms we know for computing decompositions have too slow runtime (in seconds) to work in practice. In practice there are so many factors affecting the precise runtime (in seconds) of an algorithm that it is difficult to estimate, we should rather test it by making implementations. However, this is a lot of work. In the following we briefly discuss what would work in practice.

6.3.1 Practical Runtime

Consider the following thought experiment, we have a graph on $n = 1000$ vertices and we want to parameterize by a parameter with value k. We estimate a standard computer to do 10^9 operations per second.

Question: For which values of k are algorithms with various runtimes practical?

Table 6.3.1 shows what can be considered a practical value of k for various runtimes, all FPT, i.e. $f(k)\cdot poly(n)$ for varying functions f and $poly$ assuming no hidden constants. The practical values of k range between 3 (for runtime $2^{k^3} \cdot n$) and 35 (for runtime $2^k \cdot n$) These are rough estimates, taking into consideration the runtime but not the memory usage which often is a limiting factor. let us consider the practicality of some well-known algorithms.

runtime	max k	runtime	max k
$2^{2^k} \cdot n$	5	$2^{2k} \cdot n^2$	12
$2^{k^3} \cdot n$	3	$2^{2k} \cdot n$	17
$2^{k^2} \cdot n$	5	$3^k \cdot n$	21
$2^{\frac{3}{4}k^2} \cdot n$	6	$2^k \cdot n^3$	15
$2^{3k} \cdot n^2$	8	$2^k \cdot n^2$	25
$2^{3k} \cdot n$	11	$2^k \cdot n$	35

Table 6.1: Thought experiment on a graph with $n = 1000$ nodes. Runtime of different algorithms, and estimated maximum k for which we consider the algorithm practical (estimated to finish within 12 hours on a computer doing 10^9 operations per second).

For instance, Bodlaender's algorithm for computing an optimal tree decomposition [4] has a runtime of $O(c^{k^3} \cdot n)$ (for some large constant c, for our purpose it suffices to set $c = 2$), is this practical? If $c = 2$ and $k = 4$ we get a factor of 2^{64} and this many operations would take thousands of years to execute. Bodlaender's algorithm has been implemented [67], and it was shown that it can in practice only compute optimal decompositions of graphs of tree-width at most 3 [67]. For graphs of tree-width at most 4 there are other more practical algorithms, see [68] and the bibliography therein.

The algorithm for dominating set of Chapter 8 given a binary decomposition tree of boolean width k runs in time $O(k \cdot 2^{3k} \cdot n)$ (after preprocessing is done) and hence will be practical up to aproximately $k = 11$. Given a decomposition of tree-width k Minimum Dominating Set can be solved in time $O(3^k \cdot k^2 \cdot n)$ [75] which can be considered practical up to $k = 21$. In

general the dynamic programming step is practical for much larger values of k than the step of computing decompositions.

6.3.2 Reduction Rules

None of the FPT algorithms for computing optimal decompositions are linearly single exponential in the width of the graph. One way to get around this intractability is to do some preprocessing to remove the easier part of the graph. Much work has been done in this direction for computing tree decompositions of low tree-width [68, 5]. We now show one reduction rule for boolean-width, a similar rule also applies to rank-width [45], but not to tree-width.

Let G be a graph, a 1-join of G is a cut $(A, \overline{A})$ such that after removing all isolated vertices from the graph $G[A, \overline{A}]$, what is left is a complete graph.

Lemma 6.3.1. *Let G be a graph and $(A, \overline{A})$ a 1-join of G with $N(A) \cap \overline{A} = R$ and $N(\overline{A}) \cap A = L$. Let $u \in R$, $v \in L$ and H be the disjoint union of the two graphs $G[A \cup u]$ and $G[\overline{A} \cup v]$. Then $boolw(G) = boolw(H)$.*

Proof. Note that it suffices to prove this lemma for connected G. Since both $G[A \cup u]$ and $G[\overline{A} \cup v]$ are induced subgraphs of G we have from Lemma 3.5.7 that $boolw(G) \geq boolw(H)$, we only need to show $boolw(G) \leq boolw(H)$. Assume we have binary decomposition trees (T_u, δ_u) and (T_v, δ_v) of optimal boolean-width of each of the two components of H, we can make those two decompositions unrooted by deleting the root and connecting the two children of the root. Then we subdivide the two edges having either $\delta_u(u)$ or $\delta_v(v)$ as endpoints making two new vertices u' and v', now we add a new root r and connect the two decomposition trees by connecting r to u' and v' and merge δ_u and δ_v into a new bijection δ. For every node x except $\delta(u), u', \delta(v), v'$ and r we have that $bool\text{-}dim(V_x)$ in H is at most $boolw(G)$ since $G[V_x, V(H) \setminus V_x]$ after twin contraction equals $G[V_x, V(G) \setminus V_x]$ after twin contraction. Since $bool\text{-}dim(V_r) = 0$, $bool\text{-}dim(V_{\delta(u)}) = bool\text{-}dim(V_{\delta(v)}) = 1$, then the only cut to argue for is $(A, \overline{A})$, since this is a 1-join, i.e. $G[A, \overline{A}]$ is a complete graph plus some isolated vertices, we have $boolw(G) \geq 1$ and the lemma follows. □

Note that we can apply the above lemma as long as the graph has a cut vertex. The graphs of boolean-width 1 are the distance hereditary graphs and can be recognized in linear time [20].

Open Problem 8. Can we recognize graphs of boolean-width at most $log_2(3)$ in polynomial time?

6.3.3 Implementations

Implementations of algorithms for finding tree decompositions of relatively low tree-width have been successful [7, 6, 5], see Chapter 12 for more on this. We have started a similar project finding binary decomposition trees of relatively low boolean-width. The first results of this project are positive and are presented in Chapter 12. We refer to that chapter for further explanation. However, it is clear that a lot of work remains to be done before boolean-width decompositions can be used in the real world in any way resembling that of tree-width.

In order to develop good heuristics there has to be a deep theoretical understanding of the width parameter, with results such as Lemma 6.3.1. The three new parameters in this thesis lack in theoretical understanding compared to tree-width and rank-width. However, since MIM-width and boolean-width can be much smaller than the other parameters it might be easier to design heuristics for them. In particular, we believe that the heuristic for boolean-width given in Chapter 12 has much potential for improvements.

Chapter 7

Conclusions and Future Work

We have introduced three new width parameters, many questions relating to these are left open. In this Chapter we give an overview of the most interesting open questions.

Recall from Chapter 4 that the relation between MM-width and tree-width was proven via a non-monotone strategy for the cops and robber game characterizing tree-width. Computing the MM-width of a binary decomposition tree can be done in $O(n^{3.5})$ time using the Hopcroft-Karp algorithm [46].

Open Problem 2 (Chapter 4)**.** Given a binary decomposition tree (T, δ) can we compute a tree decomposition of tree-width at most $3mmw(T, \delta) - 1$ in $O(n^{3.5})$ time?

Maybe the most important theoretical questions is to decide the complexity of computing optimal decomposition trees or approximating the value of the new width parameters.

Open Problem 7 (Chapter 6)**.** For a graph G with boolean-width k, rank-width k, module-width k or MIM-width k, can we in time $O^*(2^{O(k)})$ compute a decomposition having width $O(k)$?

For MIM-width and boolean-width little is known about computing optimal decompositions, and the above question seems to be ambitious, it will be more realistic to start by asking:

Open Problem 5 (Chapter 5)**.** Is there a function f such that, given a graph G, we can in XP time parameterized by $k = mimw(G)$ compute a binary decomposition tree of MIM-width at most $f(k)$?

Even easier questions remain open.

Open Problem 1 (Chapter 3)**.** Can we recognize graphs of MIM-width 1 in polynomial time?

Open Problem 8 (Chapter 6)**.** Can we recognize graphs of boolean-width at most $log_2(3)$ in polynomial time?

We have not focused on the linear versions of the new width parameters in this thesis. However, all the above questions have corresponding questions for linear versions, which might be easier to answer.

There are also some special graph classes where the problem of deciding the MIM-width and boolean-width is left open. For strongly chordal graph we believe in a positive answer, while for tolerance graphs we believe in a negative answer.

Open Problem 9 (Chapter 9)**.** Is the MIM-width of strongly chordal graphs and tolerance graphs constant?

One thing that has been important in this thesis is to find linear exponential parameterized algorithms. There are many pairs of problems and parameters where we don't know the complexity, in particular the following.

Open Problem 6 (Chapter 5)**.** Given a graph G and a binary decomposition tree of rank-width k, can Maximum Independent Set be solved in time $O^*(2^{O(k)})$?

Linear single exponential FPT algorithms might be of practical interest, especially when parameterizing by boolean-width since boolean-width in general is much lower than other well-known graph width parameters. From a practical point of view heuristics seem like the best approach to computing decomposition trees of low boolean-width. Our heuristic for finding decompositions of low boolean-width is time and memory consuming and needs to be improved in order to be of practical interest, this is an interesting direction of research.

We know that the boolean-width of a graph G never exceed $\frac{|V(G)|}{3}$, it is part of an ongoing project with Yuri Rabinovich and Jan Arne Telle to prove that there exist a constant $c < \frac{1}{3}$ such that every graph G has boolean-width at most $c \cdot |V(G)|$.

Bibliography

[1] Isolde Adler, Binh-Minh Bui-Xuan, Yuri Rabinovich, Gabriel Renault, Jan Arne Telle, and Martin Vatshelle. On the boolean-width of a graph: Structure and applications. In *Proceedings of WG*, volume 6410 of *Lecture Notes in Computer Science*, pages 159–170. Springer-Verlag, 2010.

[2] Rémy Belmonte. Boolean-width of special graph classes, applications for solving np-hard problems on graph classes in polynomial time. Master's thesis, University Montpellier II, 2010. `http://remybelmonte.files.wordpress.com/2011/10/master_thesis.pdf`.

[3] Daniel Binkele-Raible, Ljiljana Brankovic, Marek Cygan, Henning Fernau, Joachim Kneis, Dieter Kratsch, Alexander Langer, Mathieu Liedloff, Marcin Pilipczuk, Peter Rossmanith, and Jakub Onufry Wojtaszczyk. Breaking the 2^n-barrier for irredundance: Two lines of attack. *Journal of Discrete Algorithms*, 9(3):214–230, 2011.

[4] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305 – 1317, 1996.

[5] Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *Proceedings of WG*, volume 4271 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 2006.

[6] Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth Computations I Upper Bounds. *Information and Computation*, 208(3):259–275, 2010.

[7] Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations II. lower bounds. *Information and Computation*, 209(7):1103–1119, 2011.

[8] Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In *Proceedings of ICALP*, volume 1256

of *Lecture Notes in Computer Science*, pages 627–637. Springer-Verlag, 1997.

[9] Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. Faster algorithms on clique and branch decompositions. In *Proceedings of MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 174–185. Springer-Verlag, 2010.

[10] Kathie Cameron and Tracy Walker. The graphs with maximum induced matching and maximum matching the same size. *Discrete Mathematics*, 299(13):49–55, 2005.

[11] Maw-Shang Chang. Weighted domination of cocomparability graphs. *Discrete Applied Mathematics*, 80(3):135–148, 1997.

[12] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164(1):51–229, 2006.

[13] Nathann Cohen. Personal communication.

[14] Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.

[15] Bruno Courcelle. The monadic second-order logic of graphs iii: tree-decompositions, minor and complexity issues. *Informatique Théorique et Applications*, 26:257–286, 1992.

[16] Bruno Courcelle. *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1, chapter The Expression Of Graph Properties And Graph Transformations In Monadic Second-Order Logic, pages 313–400. World Scientific, 1997.

[17] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993.

[18] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[19] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. pages 150–159. IEEE, 2011.

[20] Guillaume Damiand, Michel Habib, and Christophe Paul. A simple paradigm for graph recognition: application to cographs and distance hereditary graphs. *Theoretical Computer Science*, 263(1-2):99–111, 2001.

[21] Vânia M.F. Diasa, Celina M.H. de Figueiredo, and Jayme L. Szwarcfiter. On the generation of bicliques of a graph. *Discrete Applied Mathematics*, 155:1826–1832, 2007.

[22] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer-Verlag, fourth edition, 2010.

[23] Frederic Dorn. Dynamic programming and fast matrix multiplication. In *Proceedings of ESA*, volume 4168 of *Lecture Notes in Computer Science*, pages 280–291. Springer-Verlag, 2006.

[24] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[25] W. Duckworth, D. Manlove, and M. Zito. On the approximability of the maximum induced matching problem. *Journal of Discrete Algorithms*, 3:79–91, 2000.

[26] John A. Ellis, Ivan H. Sudborough, and Jonathan S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.

[27] Leonard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1736.

[28] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[29] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer-Verlag, 1st edition edition, 2010.

[30] Fedor V. Fomin, Frédéric Mazoit, and Ioan Todinca. Computing branchwidth via efficient triangulations and blocks. *Discrete Applied Mathematics*, 157(12):2726–2736, 2009.

[31] Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. In *Proceedings of ICALP*, volume 5125 of *Lecture Notes in Computer Science*, pages 210–221. Springer-Verlag, 2008.

[32] Zoltán Füredi. The number of maximal independent sets in connected graphs. *Journal of Graph Theory*, 11(4):463–470, 1987.

[33] Robert Ganian. Thread graphs, linear rank-width and their algorithmic applications. In *Proceedings of IWOCA*, volume 6460 of *Lecture Notes in Computer Science*, pages 38–42. Springer, 2011.

[34] Robert Ganian and Petr Hliněný. On Parse Trees and Myhill-Nerode-type Tools for handling Graphs of Bounded Rank-width. *Discrete Applied Mathematics*, 158(7):851–867, 2010.

[35] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman and Co., 1979.

[36] Serge Gaspers, Dieter Kratsch, and Mathieu Liedloff. On independent sets and bicliques in graphs. *Algorithmica*, 62(3-4):637–658, 2012.

[37] Carl Friedrich Gauss. *Nachlass: Theoria interpolationis methodo nova tractata*, volume 3. Knigliche Gesellschaft der Wissenschaften, Gttingen, 1866.

[38] James F. Geelen, Bert Gerards, and Geoff Whittle. Branch-width and well-quasi-ordering in matroids and graphs. *Journal of Combinatorial Theory, Series B*, 84(2):270–290, 2002.

[39] Jay Goldman and Gian-Carlo Rota. The number of subspaces of a vector space. *Recent Progress in Combinatorics*, pages 75–83, 1969.

[40] Martin C. Golumbic and Udi Rotics. On the clique-width of perfect graph classes. In *Proceedings of WG*, Lecture Notes in Computer Science, pages 135–147. Springer-Verlag, 1999.

[41] Frank Gurski. A comparison of two approaches for polynomial time algorithms computing basic graph parameters. *CoRR*, abs/0806.4073, 2008.

[42] Michel Habib, Christophe Paul, and Laurent Viennot. Partition refinement techniques: An interesting algorithmic tool kit. *International Journal of Foundations of Computer Science*, 10(2):147–170, 1999.

[43] Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10(1):26–30, 1935.

[44] Roger Hart. *The Chinese Roots of Linear Algebra*. The history of mathematics. Johns Hopkins University Press, 2010.

[45] Petr Hlinený, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal*, 51(3):326–362, 2008.

[46] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[47] Vít Jelínek. The rank-width of the square grid. *Discrete Applied Mathematics*, 158(7):841–850, 2010.

[48] Öjvind Johansson. Clique-decomposition, NLC-decomposition and modular decomposition – Relatiohships and results for random graphs. *Congressus Numerantium*, 132:39–60, 1998.

[49] Iyad Kanja, Michael J. Pelsmajerb, Marcus Schaefera, and Ge Xiac. On the induced matching problem. *Journal of Computer and System Sciences*, 77(6):1058–1070, 2011.

[50] Ki Hang Kim. *Boolean matrix theory and its applications*. Monographs and textbooks in pure and applied mathematics. Marcel Dekker, 1982.

[51] Ton Kloks and Hans L. Bodlaender. Only few graphs have bounded treewidth. Technical Report UU-CS-92-35, Department of Information and Computing Sciences, Utrecht University, 1992.

[52] Dénes König. Gráfok és mátrixok. *Matematikai és Fizikai Lapok*, 38:116–119, 1931.

[53] Evangelos Kranakis, Danny Krizanc, Berthold Ruf, Jorge Urrutia, and Gerhard Woeginger. The vc-dimension of set systems defined by graphs. *Discrete Applied Mathematics*, 77(3):237–257, 1997.

[54] Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. Generating all maximal independent sets: Np-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

[55] Lee J. Lee, C. and Sang-il Oum. Rank-width of random graphs. *Journal of Graph Theory*, 2011. doi: 10.1002/jgt.20620.

[56] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2006.

[57] Sang-il Oum. *Graphs of Bounded Rank-width*. PhD thesis, Princeton University, 2005.

[58] Sang-il Oum. Rank-width is less than or equal to branch-width. *Journal of Graph Theory*, 57(3):239–244, 2008.

[59] Sang-il Oum. Computing rank-width exactly. *Information Processing Letters*, 109:745–748, 2009.

[60] Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

[61] Sang-il Oum and Paul D. Seymour. Testing branch-width. *Journal of Combinatorial Theory, Series B*, 97(3):385–393, 2007.

[62] Helmut Prodinger and Robert F. Tichy. Fibonacci numbers of graphs. *The Fibonacci Quarterly*, 20(1):16–21, 1982.

[63] Michaël Rao. *Décomposition de graphes et algorithmes efficaces*. PhD thesis, Université Paul Verlaine, Metz, 2006.

[64] Michaël Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008.

[65] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of STOC*, pages 221–228. ACM.

[66] Neil Robertson and Paul D. Seymour. Graph minors X. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

[67] Hein Röhrig. Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik in Saarbrücken, 1998.

[68] Daniel P. Sanders. On linear recognition of tree-width at most four. *SIAM Journal on Discrete Mathematics*, 9(1):101–117, 1996.

[69] Robert Sasak. Comparing 17 graph parameters. Master's thesis, University of Bergen, 2010. http://hdl.handle.net/1956/4329.

[70] Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

[71] Larry J. Stockmeyer and Vijay V. Vazirani. Np-completeness of some generalizations of the maximum matching problem. *Information Processing Letters*, 15(1):14–19, 1982.

[72] Robert R. Stoll. *Set theory and logic*. Dover books on advanced mathematics. Dover Publications, 1979.

[73] Jan Arne Telle. *Vertex Partitioning Problems: Characterization, Complexity and Algorithms on Partial k-Trees*. PhD thesis, University of Oregon, 1994.

[74] Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997.

[75] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Proceedings of ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer-Verlag, 2009.

[76] Vladimir N. Vapnik and Alexey J. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[77] Robin J. Wilson. *Introduction to Graph Theory*. Prentice Hall, fourth edition, 1996.

Part II

Papers

Boolean-width of graphs[☆]

Binh-Minh Bui-Xuan[a], Jan Arne Telle[a,*], Martin Vatshelle[a]

[a] *Department of Informatics, University of Bergen, Norway.*
buixuan@lip6.fr [telle,vatshelle]@ii.uib.no

Abstract

We introduce the graph parameter boolean-width, related to the number of different unions of neighborhoods – Boolean sums of neighborhoods – across a cut of a graph. For many graph problems this number is the runtime bottleneck when using a divide-and-conquer approach. For an n-vertex graph given with a decomposition tree of boolean-width k, we solve Maximum Weight Independent Set in time $O(n^2k2^{2k})$ and Minimum Weight Dominating Set in time $O(n^2 + nk2^{3k})$. With an additional n^2 factor in the runtime we can also count all independent sets and dominating sets of each cardinality.

Boolean-width is bounded on the same classes of graphs as clique-width. Boolean-width is similar to rank-width, which is related to the number of $GF(2)$-sums of neighborhoods instead of the Boolean sums used for boolean-width. We show for any graph that its boolean-width is at most its clique-width and at most quadratic in its rank-width. We exhibit a class of graphs, the Hsu-grids, having the property that a Hsu-grid on $\Theta(n^2)$ vertices has boolean-width $\Theta(\log n)$ and rank-width, clique-width, tree-width, and branch-width $\Theta(n)$.

Keywords: graph decomposition, FPT algorithm, width parameter, Boolean algebra

1. Introduction

Width parameters of graphs, like tree-width, branch-width, clique-width and rank-width, are important in the field of graph algorithms. Many NP-hard graph optimization problems have fixed-parameter tractable (FPT)

[☆]Supported by the Norwegian Research Council, project PARALGO.
[*]Corresponding author. Tel: (+47) 55 58 40 36. Fax: (+47) 55 58 41 99.

algorithms when parameterized by these parameters (see [23] for a recent overview, and [15, 16] for extensive ones).

The most widely known parameter is the tree-width $tw(G)$ of a graph G, which was introduced along with branch-width $bw(G)$ in [41]. In time[1] $O^*(2^{3.7tw(G)})$ a decomposition of tree-width at most $3.7tw(G)$ can be computed [3], and once a decomposition of tree-width k is given, there are case-specific algorithms solving many NP-hard problems in time $O^*(2^{c \cdot k})$ for c a small constant, *e.g.* $c = 1.58$ for Minimum Dominating Set [42]. Similar results hold for branch-width since $bw(G) \leq tw(G)+1 \leq 1.5bw(G)$. A drawback of tree-width and branch-width arises with dense graphs, where their value is high, *e.g.* the complete graph K_n has tree-width $n-1$ and $2^{tw(K_n)}$ is thus exponential in n.

The introduction of clique-width $cw(G)$ in [12] was in this context a big improvement. A class of graphs has clique-width bounded by a constant whenever tree-width/branch-width is bounded by a constant, but $cw(K_n) = 2$. Moreover, given a decomposition of clique-width k many NP-hard problems can still be solved reasonably fast, *e.g.* Minimum Dominating Set can be solved in $O^*(2^{4 \cdot k})$ time [30], very recently improved to $O^*(2^{2 \cdot k})$ [6]. A drawback of clique-width was that for a long time no algorithm was known for computing a decomposition of low clique-width. This situation improved in 2005 with an algorithm that in time $O^*(2^{3cw(G)})$ computed a decomposition of clique-width at most $2^{3cw(G)}$ [36]. Although this can be far from the optimal clique-width, it means there are FPT algorithms for all $MSOL_1$ problems when parameterized by clique-width [13].

This approximation for clique-width was achieved by introducing a new parameter, the so-called rank-width $rw(G)$, that is interesting in itself. Firstly, given an n-vertex graph a decomposition of optimal rank-width can be computed in time $O(f(rw(G))poly(n))$, for some polynomial function $poly$ and a function f at least exponential [22]. Secondly, rank-width is potentially much smaller than clique-width, tree-width and branch-width: for any graph G we have $\log cw(G) \leq rw(G) \leq cw(G)$, and $rw(G) \leq tw(G)+1$ and $rw(G) \leq bw(G)$ [35], in contrast to clique-width where there exist graphs with $cw(G)$ in $2^{\Theta(tw(G))}$ [11]. A possible drawback of rank-width is that so far no well-known NP-hard problems are solvable in time $O^*(2^{c \cdot k})$ on a decomposition of rank-width k, *e.g.* for Minimum Dominating Set the fastest

[1] We use O^* notation that hides polynomial factors.

runtime is $O^*(2^{0.75k^2+O(k)})$ [9, 17]. However, note that for graphs having rank-width much smaller than clique-width and tree-width it will still be preferable to use rank-width for *e.g.* Minimum Dominating Set.

In this paper we introduce a graph parameter called boolean-width. Its value is not only smaller than clique-width but also potentially much smaller than rank-width: $\log cw(G) \leq boolw(G) \leq cw(G)$ and $\log rw(G) \leq boolw(G) \leq 0.25rw^2(G) + O(rw(G))$, with both lower bounds tight to a multiplicative factor as shown here for Hsu-grid graphs. Very recently, also well-known classes of graphs, like random graphs and interval graphs, have been shown to have clique-width and rank-width exponential in their boolean-width [1, 4]. We show that there are NP-hard problems solvable in time $O^*(2^{c \cdot k})$ on a decomposition of boolean-width k, *e.g.* $c = 3$ for Minimum Dominating Set. A drawback of boolean-width is the same as with clique-width: so far the best algorithm for computing a decomposition of low boolean-width is based on the algorithm for rank-width. It will in the worst case, and in particular for Hsu-grids, return a decomposition having boolean-width exponential in the optimal boolean-width.

Our paper is organized as follows. In Section 2 we define boolean-width based on the number of unions of neighborhoods across the cuts given by a decomposition tree, and argue that it is a natural parameter if the goal is fast divide-and-conquer algorithms, at least for independence and domination problems. In Section 3 we compare boolean-width to other width parameters, and in particular to rank-width. We show that $\log rw(G) \leq boolw(G) \leq 0.25rw^2(G) + O(rw(G))$. This means that boolean-width is (constantly) bounded on the same classes of graphs as clique-width and rank-width, but for higher bounds the situation is different. For a class of graphs C say parameter P is logarithmic, resp. polylog, if the value of P for any n-vertex graph G in C is logarithmic in n, resp. polylog in n. For example, boolean-width is logarithmic on interval graphs and polylog on random graphs. Whenever P is logarithmic on C, resp. polylog on C, any algorithm with runtime $O^*(2^{c \cdot P(G)})$, resp. $O^*(2^{poly(P(G))})$, will on input a graph G in C have polynomial runtime, resp. quasi-polynomial runtime. From the results depicted in Figure 1 it follows that if any of tree-width, branch-width, clique-width or rank-width is polylog on a class of graphs then so is boolean-width, while we show in Section 3 that boolean-width is logarithmic on Hsu-grids but the other parameters are not even polylog on Hsu-grids. Recent results showing that $boolw(G) \leq bw(G)$ [1] imply that if any of tree-width, branch-width, or clique-width is logarithmic on some graph class then so is

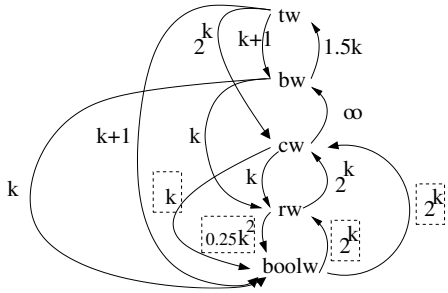


Figure 1: Upper bounds tying parameters tw=tree-width, bw=branch-width, cw=clique-width, rw=rank-width, and $boolw$=boolean-width. An arrow from P to Q labelled $f(k)$ means that any class of graphs having parameter P bounded by k will have parameter Q bounded by $f(k)$ or $O(f(k))$, and ∞ means that no such upper bound can be shown. The results surrounded by a box are shown in this paper. Most bounds are known to be tight, meaning there is a class of graphs for which the bound is $f(k)$ or $\Omega(f(k))$, except for the arrows $tw \to cw$ where an $\Omega(2^{k/2})$ bound is known [11], and $rw \to boolw$ where an $\Omega(k)$ bound is known (Theorem 2 of this paper).

boolean-width, but as the Hsu-grids show the converse is not always true.

The question whether logarithmic rank-width implies logarithmic boolean-width is left open, although in Section 4 we answer negatively a similar question at the level of graph cuts. More precisely, we show a sequence of bipartite graphs whose adjacency matrices have a Boolean row space of size equal to the number of their $GF(2)$ subspaces. This result in Boolean matrix theory implies that the measure for boolean-width can be quadratic in the measure for rank-width, when restricting to graph cuts. The use of Boolean-sums in the definition of boolean-width means a new application for the theory of Boolean matrices to the field of algorithms. Boolean matrices already have applications, *e.g.* in switching circuits, voting methods, applied logic, communication complexity, network measurements and social networks [14, 28, 32, 37].

Sections 5 and 6 are devoted to algorithms solving NP-hard problems on an n-vertex graph in time $O(2^{c \cdot k} poly(n))$ when given a decomposition tree of boolean-width k. Since the goal is to allow practical implementations of these algorithms we strive for simple descriptions, small constants c and low polynomial functions $poly$. In Section 5 we give a pre-processing routine setting up a data structure that will allow runtime at the combine step to be a function of the boolean-width of the decomposition tree, rather than

the number of vertices. In Section 6 we show how to apply dynamic programming on a decomposition tree while analysing runtime as a function of its boolean-width. We focus on the Maximum Independent Set problem where we get runtime $O(n^2k2^{2k})$ and Minimum Dominating Set with runtime $O(n^2+nk2^{3k})$. The algorithms can be deduced from similar algorithms in [9], that appeared before the introduction of boolean-width. We give the algorithms here using the new and simpler terminology and show that they have better runtime due to faster pre-processing and better data structures. We also give algorithms to handle the vertex weighted cases, also for Max and Min Weight Independent Dominating Set in the same runtime, and finally the case of counting all independent sets and dominating sets of given size.

The question of efficiently computing a decomposition of low boolean-width is left open. However, our algorithms take as input an easy-to-build decomposition tree, namely a layout of the input graph G by a tree having internal nodes of degree 3 and n leaves representing the n vertices of G, and runtimes are expressed as a function of the boolean-width of the decomposition tree. This paves the way for applying heuristics to build decomposition trees for boolean-width, as done in the TreewidthLIB project for tree-width [5], and research on boolean-width heuristics is underway [25]. We end the paper in Section 7 describing recent results and discuss some open problems.

2. Boolean-width

We deal with simple, loopless, undirected graphs and denote by $\{u, v\}$ an edge between vertices u and v. The complement of a vertex subset A of a graph $G = (V(G), E(G))$ is denoted by $\overline{A} = V(G) \setminus A$. The neighborhood of a vertex x is denoted by $N(x)$ and for a subset of vertices X we denote the union of their neighborhoods by $N(X) = \bigcup_{x \in X} N(x)$. A subset of vertices $X \subseteq V(G)$ is an independent set if there is no edge in G between any pair from X. A set $X \subseteq V(G)$ of vertices is a dominating set of G if $X \cup N(X) = V(G)$. A cut of G is a 2-partition $\{A, \overline{A}\}$ of $V(G)$. Two vertices $x, x' \in A$ are twins across $\{A, \overline{A}\}$ if $N(x) \cap \overline{A} = N(x') \cap \overline{A}$. A vertex subset $X \subseteq A$ is a twin class of A if X is a maximal set of vertices all of whom are twins across $\{A, \overline{A}\}$. The twin classes of A form a partition of A. For disjoint vertex subsets A, B of G we denote by $G(A, B)$ the bipartite graph on vertex set $A \cup B$ and edge set $\{\{u, v\} : u \in A \wedge v \in B \wedge \{u, v\} \in E(G)\}$. We denote by M_G the adjacency matrix of G, and by $M_G(A, B)$ the submatrix of M_G with the rows indexed by A and the columns by B. To ensure uniqueness

of certain algorithms, *e.g.* for computing representatives for vertex subsets, we assume a total ordering σ on the vertex set of G which stays the same throughout the entire paper. If vertex u comes before vertex v in the ordering then we say u is σ-smaller than v. For easy disambiguation, we usually refer to vertices of a graph and nodes of a tree.

We want to solve graph problems using a divide-and-conquer approach. To this aim, we need to store the information on how to recursively divide the input graph. A standard way to do this (see branch decompositions of graphs and matroids [18, 36, 41]) is to use a decomposition tree that is evaluated by a cut function.

Definition 1. A decomposition tree of a graph G is a pair (T, δ) where T is a tree having internal nodes of degree three and δ a bijection between the leaf set of T and the vertex set of G. Removing an edge from T results in two subtrees, and in a cut $\{A, \overline{A}\}$ of G given by the two subsets of $V(G)$ in bijection δ with the leaves of the two subtrees. Let $f : 2^V \to \mathbb{R}$ be a symmetric function that is also called a cut function: $f(A) = f(\overline{A})$ for all $A \subseteq V(G)$. The f-width of (T, δ) is the maximum value of $f(A)$ over all cuts $\{A, \overline{A}\}$ of G given by the removal of an edge of T. We work also on rooted trees. Subdivide an edge of T to get a new root node r, and denote by T_r the resulting binary rooted tree. For a node u let the subset of $V(G)$ in bijection δ with the leaves of the subtree of T_r rooted at u be denoted by A_u^r, or simply by A_u if the choice of subdivided edge and root r is clear or does not matter. For an edge $\{u, v\}$ of T, with u being the child of v in T_r, the cut given by removing edge $\{u, v\}$ from T can wlog be denoted $\{A_u, \overline{A_u}\}$.

We define the rooted tree T_r because divide-and-conquer on decomposition tree (T, δ) will solve the problem recursively, following the edges of T_r in a bottom-up fashion. In the conquer step we must combine solutions from two cuts given by the edges from a parent node to its children. The question of what 'solutions' we store is related to what problem we are solving. For a cut $\{A, \overline{A}\}$ note that if two independent sets $X \subseteq A$ and $X' \subseteq A$ have the same union of neighborhoods across the cut, i.e. $N(X) \cap \overline{A} = N(X') \cap \overline{A}$, then for any $Y \subseteq \overline{A}$ we have $X \cup Y$ an independent set if and only if $X' \cup Y$ an independent set. This suggests that when solving independent set problems we do not need to treat such X and X' separately, and that we should look for a decomposition tree minimizing the number of different unions of neighborhoods across the cuts. This minimum value is given by the boolean-width of the graph.

Definition 2 (Boolean-width). Let G be a graph and $A \subseteq V(G)$. Define the set of unions of neighborhoods of A across the cut $\{A, \overline{A}\}$ as

$$U(A) = \{Y \subseteq \overline{A} : \ \exists X \subseteq A \ \wedge \ Y = N(X) \cap \overline{A}\}$$

The *bool-dim* : $2^{V(G)} \to \mathbb{R}$ function of a graph G is defined as *bool-dim*$(A) = \log_2 |U(A)|$. Using Definition 1 with $f =$ *bool-dim* we define the boolean-width of a decomposition tree, denoted by $boolw(T, \delta)$, and the boolean-width of a graph, denoted by $boolw(G)$.

See Figure 2 for an example of a cut. $U(A)$ is in a bijection with what is called the Boolean row space of $M_G(A, \overline{A})$, i.e. the set of vectors that are spanned via Boolean sum (1+1=1) by the rows of $M_G(A, \overline{A})$, see the monograph [28] on Boolean matrix theory. It is known that $|U(A)| = |U(\overline{A})|$, see [28, Theorem 1.2.3] and hence the *bool-dim* function is symmetric. The value *bool-dim*(A) will be referred to as the boolean dimension of the matrix $M_G(A, \overline{A})$ and of the bipartite graph $G(A, \overline{A})$. The Boolean row space of $M_G(A, \overline{A})$ may not have a basis of size *bool-dim*(A), but we do find representatives of that size; below Lemma 6 shows that for each $Y \in U(A)$ we find $R \subseteq A$ with $|R| \leq$ *bool-dim*(A) and $Y = N(R) \cap \overline{A}$. Let us consider some examples. If $|U(A)| = 2$ then $G(A, \overline{A})$ has boolean dimension 1 and $G(A, \overline{A})$ is the union of a complete bipartite graph and some isolated vertices. If $G(A, \overline{A})$ is a perfect matching of G then $|U(A)| = 2^{|V(G)|/2}$ and $G(A, \overline{A})$ has boolean dimension $|V(G)|/2$. If a graph has boolean-width 1 then it has a decomposition tree such that, for every cut defined by an edge of the tree, the edges crossing the cut, if any, induce a complete bipartite graph. Since we take the logarithm base 2 of $|U(A)|$ in the definition of boolean dimension we have for any graph G that $0 \leq boolw(G) \leq |V(G)|$, which eases the comparison of boolean-width to other parameters, and is in analogy with the definition of rank-width given in Definition 3 below.

The boolean-width of a graph is not always an integer; however, most of the analysis will address the value $2^{\textit{bool-dim}(A)}$, which is an integer.

In the next sections we compare boolean-width to other graph parameters, but the reader interested only in algorithms can skip this and go directly to Section 5.

3. Value of boolean-width compared to other width parameters

In this section we compare boolean-width $boolw$ to tree-width tw, branch-width bw, clique-width cw and rank-width rw. For any graph G, it holds

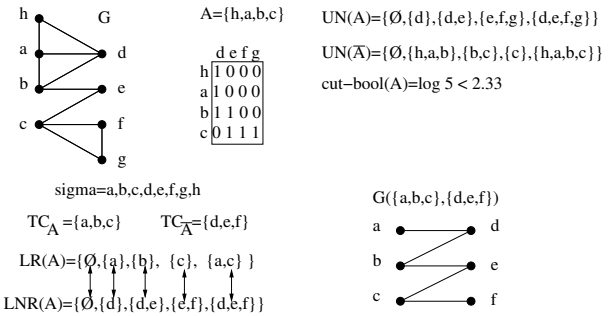


Figure 2: Example graph G and $A \subseteq V(G)$, with submatrix $M_G(A, \overline{A})$, unions of neighborhoods $U(A)$ and $U(\overline{A})$ with $\mathit{bool\text{-}dim}(A) = log_2|U(A)| < 2.33$, as defined in Section 2. Note that $\mathit{cut\text{-}rank}(A) = log_2|D(A)| = 3$, as defined in Section 3. Vertex ordering sigma yields twin class representatives TC_A and $TC_{\overline{A}}$, and the list of $\equiv_A$ representatives LR_A with pointers to list of their neighbors LNR_A, as defined in Section 5. Note that $\{h, b, c\}$ induces the largest independent set in G among the 7 subsets of A having $\equiv_A$ representative $\{a, c\}$. The graph $G(TC_A, TC_{\overline{A}})$ captures the essential information across the cut $\{A, \overline{A}\}$.

that the rank-width of the graph is essentially the smallest parameter among tw, bw, cw, rw [35, 36, 41]: we have $rw(G) \leq cw(G)$ and $rw(G) \leq bw(G) \leq tw(G)+1$ for $bw(G) \neq 0$. Accordingly, we focus on comparing boolean-width to rank-width.

Rank-width was introduced in [34, 36] based on the $\mathit{cut\text{-}rank} : 2^{V(G)} \to \mathbb{N}$ function of a graph G, with $\mathit{cut\text{-}rank}(A)$ being the rank over $GF(2)$ of $M_G(A, \overline{A})$. To see the connection with boolean-width note this alternative equivalent definition of rank-width.

Definition 3. Let G be a graph and $A \subseteq V(G)$. Let $\bigtriangleup$ be the symmetric difference operator, that applied to a family of sets gives the elements appearing in an odd number of sets. Define the set of symmetric differences of neighborhoods of A across the cut $\{A, \overline{A}\}$ as

$$D(A) = \{Y \subseteq \overline{A} : \ \exists X \subseteq A \ \wedge \ Y = \bigtriangleup_{x \in X} N(x) \cap \overline{A}\}$$

The $\mathit{cut\text{-}rank} : 2^{V(G)} \to \mathbb{R}$ function of a graph G is defined as $\mathit{cut\text{-}rank}(A) = \log_2 |D(A)|$. Using Definition 1 with $f = \mathit{cut\text{-}rank}$ we define the rank-width of a decomposition tree, denoted by $rw(T, \delta)$, and the rank-width of a graph, denoted by $rw(G)$.

We first investigate the relationship between the *bool-dim* and the *cut-rank* functions. Lemma 1 below can be derived from a reformulation of [9, Proposition 3.6]. We give a simplified proof here. Let

$$DS(A) = \{\mathcal{S} \subseteq D(A) : \ \mathcal{S} \text{ is closed under the symmetric difference}\}.$$

Lemma 1. *[9] For any graph G and $A \subseteq V(G)$ it holds that:*

$$\log_2 \textit{cut-rank}(A) \leq \textit{bool-dim}(A) \text{ and}$$

$$\textit{bool-dim}(A) \leq \log_2 |DS(A)| \leq \frac{1}{4}\textit{cut-rank}^2(A) + O(\textit{cut-rank}(A)).$$

Proof: Let $\{a_1, a_2, \ldots, a_{\textit{cut-rank}(A)}\}$ be a set of vertices of A whose corresponding rows in $M_G(A, \overline{A})$ define a $GF(2)$-basis of $M_G(A, \overline{A})$. Then clearly $N(a_1)$, $N(a_2)$, $\ldots$, $N(a_{\textit{cut-rank}(A)})$ are pairwise distinct. This allows to conclude about the first inequality.

To prove the second inequality we injectively map $U(A)$ to $DS(A)$. Let $U(A) = \{Y_1, Y_2, ..., Y_q\}$. For each i with $1 \leq i \leq q$ we fix some $X_i \subseteq A$ such that $N(X_i) \cap \overline{A} = Y_i$. Let $\Delta\textit{closure}(\mathcal{G})$ be the unique smallest family containing $\mathcal{G}$ that is closed under the symmetric difference of its members. For each $i \neq j$ with $1 \leq i, j \leq q$ we have $Y_i \neq Y_j$ and therefore $\Delta\textit{closure}(\{N(x) \cap \overline{A} : x \in X_i\}) \neq \Delta\textit{closure}(\{N(x) \cap \overline{A} : x \in X_j\})$. Note also that $\Delta\textit{closure}(\{N(x) \cap \overline{A} : x \in X_i\}) \in DS(A)$. But then $f : U(A) \to DS(A)$ defined by $f(Y_i) = \Delta\textit{closure}(\{N(x) \cap \overline{A} : x \in X_i\})$ is an injection, to conclude the proof of the second inequality in the lemma.

$DS(A)$ is in a bijection with the subspaces over $GF(2)$ of the space spanned over $GF(2)$ by the rows of $M_G(A, \overline{A})$. This space has dimension *cut-rank*(A) and for the number of subspaces the third inequality in the lemma is well-known in enumerative combinatorics, and can be derived from [19]. □

Lemma 1 holds for all edges of all decomposition trees, we therefore have the following corollary.

Corollary 1. *For any graph G and decomposition tree (T, δ) of G it holds that*

$$\log_2 rw(T, \delta) \leq \textit{boolw}(T, \delta) \leq \frac{1}{4}rw^2(T, \delta) + O(rw(T, \delta)),$$

$$\log_2 rw(G) \leq \textit{boolw}(G) \leq \frac{1}{4}rw^2(G) + O(rw(G)).$$

This corollary can be combined with a result of [22] to get an approximation algorithm for boolean-width, as follows. Let a graph G have decomposition trees (T, δ) and (T', δ') such that $rw(G) = rw(T, \delta)$ and $OPT = boolw(G) = boolw(T', \delta')$. We then have from Corollary 1 that $boolw(T, \delta) \leq rw^2(T, \delta) \leq rw^2(T', \delta') \leq (2^{OPT})^2$. Hence, any decomposition tree of G of optimal rank-width is also a decomposition tree of boolean-width within $2^{2 \cdot OPT}$ of the optimal boolean-width. There is an algorithm to compute a decomposition tree of G of optimal rank-width in $O(f(rw(G)) \times |V(G)|^3)$ time [22]. We thus have the following approximation for boolean-width, and we will see with below defined Hsu-grid graphs that this approximation bound is essentially tight for algorithms based on computing optimal rank-width.

Theorem 1. *Given an n-vertex graph G there is an algorithm to compute in $O(f(boolw(G))n^3)$ time a decomposition tree (T, δ) with $boolw(T, \delta) \leq 2^{2 \cdot boolw(G)}$, for some function f.*

We now address the interesting fact that there are graphs whose boolean-width is exponentially smaller than the value of the other main width parameters. In particular, we show that the lower bound $\log_2 rw(G) \leq boolw(G)$ in Corollary 1 is tight to a multiplicative factor, by employing the graphs used in the definition of Hsu's generalized join [24] to define the Hsu-grid. Firstly, for all $k \geq 1$, the Hsu graph H_k is defined as the bipartite graph having color classes $A_k = \{a_1, a_2, \ldots, a_{k+1}\}$ and $B_k = \{b_1, b_2, \ldots, b_{k+1}\}$ such that $N(a_1) = \emptyset$ and $N(a_i) = \{b_1, b_2, \ldots, b_{i-1}\}$ for all $i \geq 2$ (an illustration is given in Figure 3). We consider the cut $\{A_k, B_k\}$. A union of neighborhoods of vertices of A_k is always of the form $\{b_1, b_2, \ldots, b_l\}$, and as a consequence,

Fact: for any Hsu graph H_k it holds that:
$bool\text{-}dim(A_k) = \log_2 k$ *and* $cut\text{-}rank(A_k) = k$.

We lift this tightness result on graph cuts to the level of graph parameters in a standard way, by using the structure of a grid and the concept of a balanced cut (see *e.g.* [36, 40]): a cut $\{A, \overline{A}\}$ of a graph G is balanced if $\frac{1}{3}|V(G)| \leq |A| \leq \frac{2}{3}|V(G)|$. In any decomposition tree there exists an edge of the tree which induces a balanced cut in the graph and any balanced cut of a grid will contain either a large part of some row of the grid, or it contains a large matching using only horizontal edges. The formal definition of the Hsu-grid is given below while an illustration is given in Figure 3. Note that graphs with a similar definition have also been studied in relation with clique-width in a different context [7].

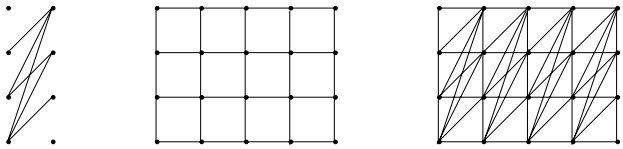


Figure 3: The Hsu graph H_3, the 4×5 grid, and the Hsu-grid $HG_{4,5}$.

Definition 4 (Hsu-grid $HG_{p,q}$). Let $p \geq 2$ and $q \geq 2$. The Hsu-grid $HG_{p,q}$ is defined by $V(HG_{p,q}) = \{v_{i,j} \mid 1 \leq i \leq p \ \wedge \ 1 \leq j \leq q\}$ with $E(HG_{p,q})$ being exactly the union of the edges $\{\{v_{i,j}, v_{i+1,j}\} \mid 1 \leq i < p \ \wedge \ 1 \leq j \leq q\}$ and the edges $\{\{v_{i,j}, v_{i',j+1}\} \mid 1 \leq i \leq i' \leq p \ \wedge \ 1 \leq j < q\}$. We say that vertex $v_{i,j}$ is at the i^{th} row and the j^{th} column.

We begin with a useful lemma.

Lemma 2. *Let $p \geq 2$ and $q \geq 2$. Let $\{A, \overline{A}\}$ be a balanced cut of the Hsu-grid $HG_{p,q}$. Then, either the cut-rank of A is at least $p/4$, or $HG_{p,q}(A, \overline{A})$ contains a $q/6$-matching as induced subgraph.*

Proof: We distinguish two self-exclusive cases.

- Case 1: for every row $1 \leq i \leq p$ there exists an edge $\{v_{i,j}, v_{i,j+1}\}$ crossing $\{A, \overline{A}\}$.
- Case 2: there is a row $1 \leq i \leq p$ containing only vertices of one side of the cut, w.l.o.g. $v_{i,j} \in A$ for all $1 \leq j \leq q$.

In case 1, we can suppose w.l.o.g. that there are at least $p/2$ row indices i's for which there exists j such that $v_{i,j} \in A$ and $v_{i,j+1} \in \overline{A}$. Therefore, there are at least $p/4$ row indices i's for which there exists j such that $v_{i,j} \in A$ and $v_{i,j+1} \in \overline{A}$ and that no two rows among those are consecutive (take every other row). Now we can check that the rank of the bipartite adjacency matrix of the subgraph of $HG_{p,q}(A, \overline{A})$ that is induced by the $p/4$ above mentioned pairs $v_{i,j}$ and $v_{i,j+1}$ is at least $p/4$. Hence, the cut-rank of A is at least $p/4$.

In case 2, from the balanced property of the cut $\{A, \overline{A}\}$ we have that there are at least $q/3$ columns each containing at least one vertex of $\overline{A}$. Then, for each such column j we can find an edge $\{v_{i,j}, v_{i+1,j}\}$ crossing $\{A, \overline{A}\}$. Choosing one such edge every two columns will lead to a $q/6$ matching that is an induced subgraph of $HG_{p,q}(A, \overline{A})$. $\square$

Lemma 3 below addresses the tightness of the lower bound on boolean-width as a function of rank-width. Note its additional stronger property that for a special class of Hsu-grids *any* decomposition tree of optimal rank-width has boolean-width exponential in the optimal boolean-width. Thus we cannot hope that an optimal rank-width algorithm will always return a decomposition tree whose boolean-width approximates the boolean-width of the graph by some polynomial function. This means that for approximation of boolean-width via rank-width Theorem 1 is essentially tight.

Lemma 3. *For large enough integers p and q it holds that: $boolw(HG_{p,q}) \leq \min\{2\log_2 p, q\}$ and $rw(HG_{p,q}) \geq \min\{\lfloor \frac{p}{4} \rfloor, \lfloor \frac{q}{6} \rfloor\}$. Moreover, if $q < \lfloor \frac{p}{8} \rfloor$ then any decomposition tree of $HG_{p,q}$ of optimal rank-width has boolean-width at least $\lfloor \frac{q}{6} \rfloor$.*

Proof: For simplicity, let m/n denote $\lfloor \frac{m}{n} \rfloor$. To prove Lemma 3, we will focus on two types of decomposition trees, that we call horizontal and vertical.

Let the k-comb be the tree we get from adding a new leaf node to each of the $k-2$ inner vertices in the path on k vertices. The k leaves of the k-comb are thus naturally ordered from left to right along the path. Let B_k be a binary tree with k leaves (its shape does not matter). Let $T_{p,q}$ be the tree having pq leaves that we get from identifying each leaf of a p-comb with the root of a B_q. The horizontal decomposition tree $(T_{p,q}, \delta_h)$ is defined by letting δ_h induce a bijection that assigns the leaves of the leftmost copy of B_q to the first row of $HG_{p,q}$, the leaves of the next copy to the second row, and so on, until the leaves of the rightmost copy of B_q that are assigned to the p'th row of $HG_{p,q}$. The vertical decomposition tree $(T_{q,p}, \delta_v)$ is defined by letting δ_v induce a bijection that assigns the leaves of the leftmost copy of B_p to the first column of $HG_{p,q}$, the leaves of the next copy to the second column, and so on, until the leaves of the rightmost copy of B_p that are assigned to the q'th column of $HG_{p,q}$.

It is straightforward to check that the boolean-width of any vertical decomposition tree of $HG_{p,q}$ is at most $2\log_2 p$ and the boolean-width of any horizontal decomposition tree of $HG_{p,q}$ is at most q. Therefore it follows that $boolw(HG_{p,q}) \leq \min\{2\log_2 p, q\}$. Besides, it follows directly from Lemma 2 that $rw(HG_{p,q}) \geq \min\{p/4, q/6\}$.

To prove the last statement of Lemma 3, we note that any horizontal decomposition tree of $HG_{p,q}$ has rank-width $2q$, and therefore the rank-width of $HG_{p,q}$ is at most $2q < p/4$. We consider a decomposition tree (T, δ) of

$HG_{p,q}$ of optimal rank-width, and an edge $\{u, v\}$ of T inducing a balanced cut $\{A, \overline{A}\}$ in $HG_{p,q}$. From Lemma 2 and the fact that the rank-width of $HG_{p,q}$ is at most $2q < p/4$, $HG_{p,q}(A, \overline{A})$ has a $q/6$-matching as induced subgraph. Therefore, the value of $bool\text{-}dim(A)$ is at least $q/6$, and the boolean-width of (T, δ) is at least $q/6$. □

The following theorem sumarizes the tightness bounds on boolean-width as a function of rank-width. Comparing with Corollary 1 note that the lower bound is tight to a multiplicative factor while for the upper bound there is a gap between a linear and a quadratic bound.

Theorem 2. *For large enough integer k, there are graphs L_k and U_k of rank-width at least k such that $boolw(L_k) \leq 2 \log rw(L_k) + 4$ and $boolw(U_k) \geq \lfloor \frac{1}{6} rw(U_k) \rfloor$.*

Proof: We define L_k as a Hsu-grid $HG_{p,q}$ such that $k \leq p/4 \leq q/6$ and $2 \log_2 p \leq q$. Then, from Lemma 3, we have that $rw(L_k) \geq p/4 \geq k$ and $boolw(L_k) \leq 2 \log_2 p$, which allows to conclude about L_k. We define U_k to be the $k \times k$ grid. It is known that the rank-width of U_k is $k - 1$ [26]. The same idea as in the proof of Lemma 2 can be used to prove that $boolw(U_k) \geq k/6$. □

One of the most important applications of rank-width is to approximate the clique-width $cw(G)$ of a graph by $\log_2(cw(G) + 1) - 1 \leq rw(G) \leq cw(G)$ [36]. Although we have seen that the difference between rank-width and boolean-width can be quite large, we now show that, w.r.t. clique-width, boolean-width behaves similarly as rank-width.

Theorem 3. *For any graph G it holds that $\log_2 cw(G) - 1 \leq boolw(G) \leq cw(G)$. For large enough integer k, there are graphs L_k and U_k of clique-width at least k such that $boolw(L_k) \leq 2 \log_2 cw(L_k) + 4$ and $boolw(U_k) \geq \lfloor \frac{1}{6} cw(U_k) \rfloor - 1$.*

Proof: For a proper introduction to clique-width refer to [12]. We will in fact not address directly clique-width, but a closely related parameter called module-width [39], whose definition is based on rooted binary trees and twin classes of a subset of vertices. Let (T, δ) be a decomposition tree of G. Let T_r be the rooted binary tree we get by subdividing any edge of T for a root r. The module-width of (T_r, δ), denoted $modw(T_r, \delta)$, is the maximum, over

all nodes a of T_r, of the number of twin classes of A^r_a. Note that $\overline{A^r_a}$ is not used in this definition and thus the choice of rooting is important. The module-width of G, denoted $modw(G)$, is the minimum module-width taken over every decomposition tree (T, δ) of G and over the subdivision of every edge e of T to obtain a rooted tree T_r [39].

We first prove that $\log_2 modw(T_r, \delta) \leq boolw(T, \delta)$. Let $\{w, a\}$ be an edge in T_r with w being the parent of a. Note from the definition of twins that $x \in A^r_a$ and $y \in A^r_a$ belong to the same twin class of A^r_a if and only if $N(\{x\}) \cap \overline{A^r_a} = N(\{y\}) \cap \overline{A^r_a}$. Therefore the number of twin classes of A^r_a is at most $|U(A^r_a)| = 2^{bool\text{-}dim(A^r_a)}$. Since this holds for every edge $\{w, a\}$ in the trees T and T_r, it allows to conclude that $\log_2 modw(T_r, \delta) \leq boolw(T, \delta)$.

To now prove $boolw(T, \delta) \leq modw(T_r, \delta)$, we consider an edge $\{w, a\}$ with w parent of a in T_r and denote by k the number of twin classes of A^r_a. Since twins of A^r_a have the same neighbors in $\overline{A^r_a}$, we can generate at most 2^k unions of neighborhoods from k twin classes, that is, $|U(A^r_a)| \leq 2^k$. In other words, $bool\text{-}dim(A^r_a) \leq k$, and since this holds for every edge $\{w, a\}$ in the trees T and T_r, it allows to conclude that $boolw(T, \delta) \leq modw(T_r, \delta)$.

It is known that for any graph G we have $modw(G) \leq cw(G) \leq 2 \cdot modw(G)$ [39]. Combining with the above bounds we obtain the inequalities in the statement of Theorem 3. Finally, we use the same graphs L_k and U_k as in the proof of Theorem 2 and the well-known fact that $rw(G) \leq cw(G)$ for any graph G [36] in order to conclude that $boolw(L_k) \leq 2 \log_2 cw(L_k) + 4$. Recall that U_k is the $k \times k$ square grid and so it is known that the clique-width of U_k is at most $k + 1$ [20]. Combining this with $boolw(U_k) \geq k/6$ allows to conclude. □

It would be nice to close the gap between the linear and quadratic upper bound on boolean-width as a function of rank-width, *i.e.* by either improving the bound $boolw(G) \leq \frac{1}{4} rw^2(G) + O(rw(G))$ in Corollary 1 or alternatively showing its tightness. We show in the next section tightness of quadratic upper bound on *bool-dim* as a function of *cut-rank*. However, we have not been able to lift this result on graph cuts to the level of graph parameters by using the structure of a grid, so we leave this as an open question.

Question: *Is the boolean-width of every graph subquadratic in its rank-width?*

4. The cardinality of the Boolean space can equal the number of $GF(2)$ subspaces

We prove in this section that the quadratic upper bound on *bool-dim* as a function of *cut-rank* from Lemma 1 is tight. More precisely, we exhibit a graph G and $A \subseteq V(G)$ where $|U(A)| = |DS(A)|$, leading to $\textit{bool-dim}(A) = \log_2 |DS(A)| = \Theta(\textit{cut-rank}^2(A))$. Note that $U(A)$ is in a bijection with the Boolean space spanned over the Boolean algebra by the rows of $M_G(A, \overline{A})$. The question of the possible cardinalities of the Boolean space of a given $\{0,1\}$-matrix has been studied by several researchers, see [44] and the bibliography therein. Recall that

$$DS(A) = \{\mathcal{S} \subseteq D(A) : \ \mathcal{S} \text{ is closed under the symmetric difference}\}$$

is in a bijection with the vector subspaces over $GF(2)$ of the vector space spanned over $GF(2)$ by the rows of $M_G(A, \overline{A})$. It follows from Definition 3 that this space has dimension, or rank, $k = \textit{cut-rank}(A)$. The fact that the number of its vector subspaces is therefore $\Theta(k^2)$ is well-known in enumerative combinatorics – sum of Gaussian binomials – and derivable from a recursion formula in [19]. The following are the important graphs and cuts.

Definition 5. *For any integer $k \geq 1$ the graph R_k is defined as a bipartite graph having color classes $A = \{a_S : \ S \subseteq \{1, 2, \ldots, k\}\}$ and $B = \{b_S : \ S \subseteq \{1, 2, \ldots, k\}\}$ such that a_S and b_T are adjacent if and only if $|S \cap T|$ is odd.*

For an example, note that R_2 is the disjoint union of 2 isolated vertices and a cycle of length 6. The "natural" cut of the bipartite graph R_k given by $\{A, B\}$ has *cut-rank* k and was used in [9] to give an alternative characterization of the graphs of rank-width at most k. The graph R_k helps in characterizing rank-width since any bipartite graph induced by a cut of *cut-rank* k is, after removing twins, an induced subgraph of R_k. Let us remark that the graph R_k has many interesting properties, and that graphs with a similar definition based on a parity check appear in the book of Alon and Spencer [2] and recently also in a paper by Charbit, Thomassé and Yeo [10]. Observation 1 and its Corollary 2 are two important properties of R_k, whose proofs are essentially a straightforward parity check.

Observation 1. *It holds for any pair of vertices a_S and a_T of R_k that $N(a_S) \Delta N(a_T) = N(a_{S \Delta T})$. The same holds for b_S and b_T.*

In terms of linear algebra, Observation 1 tells us that the $GF(2)$ sum of the two rows in $M_{R_k}(A,B)$ corresponding to a_S and a_T will result in the one corresponding to $a_{S\Delta T}$. This helps as a shortcut when dealing with adjacency issues in R_k. For any set family $\mathcal{G}$, we let $\Delta closure(\mathcal{G})$ be the unique smallest family containing $\mathcal{G}$ that is closed under the symmetric difference of its members. By convention we let $\emptyset \in \Delta closure(\mathcal{G})$ for all $\mathcal{G}$. In particular, $\Delta closure(\emptyset) = \{\emptyset\} = \Delta closure(\{\emptyset\})$.

Corollary 2. *It holds for any vertex subset $X \subseteq A$ of R_k that*

$$\Delta closure(\{N(a_S) : \ a_S \in X\}) \ \subseteq \ \{N(a_S) : \ a_S \in A\}.$$

Note that $a_\emptyset$ is a vertex of R_k and that $N(a_\emptyset) = \emptyset$. In terms of linear algebra, Corollary 2 tells us in particular that the row space over $GF(2)$ of $M_{R_k}(A,B)$ is exactly the set of rows of $M_{R_k}(A,B)$: roughly, when using Observation 1, we never "go outside" R_k by creating a fictive vertex because $a_{S\Delta T}$ is a vertex of R_k. Before proving the main claim of the section, we need the following tool.

Lemma 4. *Consider the graph R_k and any $X \subseteq A$ such that $\{N(a_Z) : \ a_Z \in X\} = \Delta closure(\{N(a_Z) : \ a_Z \in X\})$. Then, for any $a_S \in A$ with $N(a_S) \subseteq N(X)$ we have $a_S \in X$.*

Proof: Let $\mathcal{F} = \{N(a_Z) : \ a_Z \in X\}$. We note a technical remark. From $\mathcal{F} = \Delta closure(\mathcal{F})$ we have that $\emptyset \in \mathcal{F}$. The only vertex in A having an empty neighborhood is $a_\emptyset$. Therefore, we always have $a_\emptyset \in X$. We conduct a proof by induction on the notion of the dimension of $\mathcal{F}$. For a family $\mathcal{G}$ closed under the symmetric difference of its members, we let $B_\mathcal{G}$ be the smallest subfamily of $\mathcal{G}$ such that $\mathcal{G} = \Delta closure(B_\mathcal{G})$, and define the dimension $dim(\mathcal{G})$ of $\mathcal{G}$ as the cardinality of $B_\mathcal{G}$. Note by the minimality of $B_\mathcal{G}$ that $\emptyset \notin B_\mathcal{G}$. Let us prove Lemma 4 by induction on $p = dim(\mathcal{F})$.

If $p = 0$, then $B_\mathcal{F} = \emptyset$, which means $\mathcal{F} = \{\emptyset\}$. Therefore, $X = \{a_\emptyset\}$, and so $N(X) = \emptyset$. The only vertex in A having an empty neighborhood is $a_\emptyset$. In particular, $N(a_S) \subseteq \emptyset$ will directly mean $a_S = a_\emptyset \in X$.

If $p = 1$ then $\mathcal{F} \setminus \{\emptyset\}$ is a singleton. So X contains only one non-trivial vertex, say $X = \{a_\emptyset, a_T\}$ with $T \neq \emptyset$. If $a_S = a_\emptyset$ then trivially $a_S \in X$ so we suppose $a_S \neq a_\emptyset$. Since X has so few members $N(a_S) \subseteq N(X)$ simply means $N(a_S) \subseteq N(a_T)$. If $S \setminus T \neq \emptyset$, we define $W = \{i\}$ with $i \in S \setminus T$. If $S \subsetneq T$, we define $W = \{i, j\}$ with $i \in S$ and $j \in T \setminus S$. In both cases we

have $b_W \in N(a_S) \setminus N(a_T)$, contradicting to the fact that $N(a_S) \subseteq N(a_T)$. Hence, $S = T$, which in particular means $a_S \in X$.

We now assume that Lemma 4 holds whenever $dim(\mathcal{F}) \leq p-1$, with $p-1 \geq 1$, and want to prove that it also holds for the case where $dim(\mathcal{F}) = p$. In particular $p \geq 2$ and by this fact $X \setminus \{a_\emptyset\}$ contains at least two vertices. Like before, if $a_S = a_\emptyset$ then trivially $a_S \in X$ so we suppose $a_S \neq a_\emptyset$. Let a_T be a vertex in X such that $a_T \neq a_S$, and $a_T \neq a_\emptyset$. If $T \setminus S \neq \emptyset$, we define $W = \{i\}$ with $i \in T \setminus S$, otherwise $T \subsetneq S$ and we define $W = \{i, j\}$ with $i \in S \setminus T$ and $j \in T$, so that in any case we have $b_W \in N(a_T) \setminus N(a_S)$. Let $X' = \{a_Z \in X : \ b_W \notin N(a_Z)\}$.

We want to first prove that $N(a_S) \subseteq N(X')$. Assume for a contradiction that there exists $b_{W'} \in N(a_S) \setminus N(X')$. Then, we have b_W, $b_{W'}$, and $b_{W\Delta W'}$ are three distinct vertices (because $b_W \neq b_{W'}$). Observation 1 tells us that $N(b_{W\Delta W'}) = N(b_W)\Delta N(b_{W'})$, and therefore we deduce $b_{W\Delta W'} \in N(a_S) \setminus N(X')$. (It is easier here to see the property by looking at the corresponding $\{0,1\}$ values in the matrix M_{R_k} and use the "$GF(2)$ sum" $N(b_{W\Delta W'}) = N(b_W)\Delta N(b_{W'})$ on the coordinates a_S and a_Z, for every $a_Z \in X'$.) Since $\{b_{W'}, b_{W\Delta W'}\} \subseteq N(a_S) \subseteq N(X)$, there exist vertices $a_U \in X \setminus X'$ and $a_V \in X \setminus X'$ such that both $|U \cap W'|$ and $|V \cap (W\Delta W')|$ are odd. Note that not belonging to X' means both $|U \cap W|$ and $|V \cap W|$ are odd. Hence, U and V cannot be equal because on the one hand we can deduce that $|U \cap (W\Delta W')|$ is even (from the facts $|U \cap W|$ odd and $|U \cap W'|$ odd), while on the other hand we know that $|V \cap (W\Delta W')|$ is odd. But then $|(U\Delta V) \cap W|$ is even and $|(U\Delta V) \cap W'|$ is odd (a parity check or alternatively we can according to Observation 1 check the "$GF(2)$" sum $N(a_{U\Delta V}) = N(a_U)\Delta N(a_V)$ inside M_{R_k} at the coordinates b_W and $b_{W'}$). That $|(U\Delta V) \cap W|$ is even means $a_{U\Delta V}$ is a member of X' because $a_{U\Delta V}$ is not adjacent to b_W, and clearly $N(a_{U\Delta V}) = N(a_U)\Delta N(a_V)$ belongs to $\mathcal{F}$ by the symmetric difference closure of $\mathcal{F}$. That $|(U\Delta V) \cap W'|$ is odd means $a_{U\Delta V}$ is adjacent to $b_{W'}$. This contradicts the assumption $b_{W'} \in N(a_S) \setminus N(X')$. Hence, $N(a_S) \subseteq N(X')$.

We now want to prove that $\mathcal{F}' = \{N(a_Z) : \ a_Z \in X'\}$ is closed under the symmetric difference of its members. Pick $N(a_Z)$ and $N(a_{Z'})$ therein: both of them belong to $\mathcal{F}$, so from Observation 1 and the fact $\mathcal{F}$ is closed under symmetric difference, we deduce that $N(a_{Z\Delta Z'}) \in \mathcal{F}$, in other words $a_{Z\Delta Z'} \in X$. Besides, note that we can also write $X' = \{a_Z \in X, \ |W \cap Z| \text{ is even}\}$ and it is clear that if both $|W \cap Z|$ and $|W \cap Z'|$ are even, then $|W \cap (Z\Delta Z')|$ is even. Hence, $\mathcal{F}'$ is closed under the symmetric difference of its members. We also check that $dim(\mathcal{F}') \leq p-1$. Indeed, $\mathcal{F}' \subseteq \mathcal{F}$,

so we only need to show that $\Delta closure(\mathcal{F})$ properly contains $\Delta closure(\mathcal{F}')$. We consider a_T: clearly $N(a_T) \in \mathcal{F}$. Recall that $b_W \in N(a_T) \setminus N(a_S)$ and that $\mathcal{F}' = \{N(a_Z) : \ a_Z \in X \wedge b_W \notin N(a_Z)\}$. Therefore every member $N(a_{Z_0}) \in \Delta closure(\mathcal{F}')$ is such that $b_W \notin N(a_{Z_0})$. Since $b_W \in N(a_T)$, we deduce $N(a_T) \notin \Delta closure(\mathcal{F}')$. Hence, $dim(\mathcal{F}') \leq p-1$. Now, we can conclude by applying the inductive hypothesis on $\mathcal{F}'$. □

Theorem 4. *For the cut given by the bipartite graph R_k it holds that $|U(A)| = |DS(A)|$ and hence bool-dim$(A) = \log_2 |DS(A)| = \Theta($cut-rank$^2(A))$.*

Proof: As mentioned before, *cut-rank*(A) is the dimension of $D(A)$ and thus $\log_2 |DS(A)| = \Theta($*cut-rank*$^2(A))$ follows from [19]. Since by definition *bool-dim*$(A) = \log_2 |U(A)|$ it remains only to show $|U(A)| = |DS(A)|$. We do this by giving the following bijection between the two sets. Let $f : 2^{\overline{A}} \to 2^{2^{\overline{A}}}$ be defined as $f(Y) = \{N(a_S) : \ b_S \notin Y\}$. We claim that the restriction of f to $U(A)$ is a bijection between $U(A)$ and $DS(A)$.

By definition, it is clear that $f(Y) \subseteq D(A)$ for all $Y \subseteq 2^{\overline{A}}$. It can also be checked that the sets $N(a_S)$, taken over all $a_S \in A$, are pairwise distinct. Therefore, f is a well-defined injection from $2^{\overline{A}}$ into $2^{D(A)}$. Let us show that the image of $U(A)$ by f is included in $DS(A)$. Let $Y \in U(A)$ and let $X \subseteq A$ be such that $Y = N(X)$. Let $N(a_S) \in f(Y)$ and $N(a_T) \in f(Y)$. By definition, neither b_S nor b_T belong to $N(X)$. In particular, for every $a_W \in X$, we have that both $|S \cap W|$ and $|T \cap W|$ are even, which also means $|(S \Delta T) \cap W|$ is even. This implies $b_{S \Delta T} \notin N(X)$. Hence, $N(a_{S \Delta T}) \in f(Y)$. From Observation 1 we have $N(a_{S \Delta T}) = N(a_S) \Delta N(a_T)$. Hence, $f(Y)$ is closed under the symmetric difference of its members. In other words, $f(Y) \in DS(A)$. (Note that since $Y \in U(A)$, we have $b_\emptyset \notin Y$, hence $\emptyset = N(a_\emptyset) \in f(Y)$.)

Let $\mathcal{F} \in DS(A)$. In order to conclude Theorem 4, we only need to find a vertex subset $Y \in U(A)$ such that $f(Y) = \mathcal{F}$. It is a basic property in linear algebra that to any such $\mathcal{F}$ can be associated with a basis $B_\mathcal{F} \subseteq \{N(a_S) : \ a_S \in A\}$ so that $\mathcal{F} = \Delta closure(B_\mathcal{F})$. From Corollary 2, $\mathcal{F} \subseteq \{N(a_S) : \ a_S \in A\}$, so let $X \subseteq A$ be such that $\mathcal{F} = \{N(a_S) : \ a_S \in X\}$. We define $Y = \{b_S : \ a_S \notin X\}$, so that we clearly have from definition $f(Y) = \{N(a_S) : \ b_S \notin Y\} = \{N(a_S) : \ a_S \in X\} = \mathcal{F}$. Thus, the only thing left to show is that $Y \in U(A)$. More precisely, we will prove that $Y = N(X')$, where $X' = \{a_S : \ b_S \notin N(X)\}$.

- Let $b_S \in N(X')$. Then, there exists $a_T \in X'$ such that $|S \cap T|$ is odd. By definition of X', we know that $b_T \notin N(X)$. Since $|S \cap T|$ is odd (hence a_S and b_T are adjacent in R_k), we deduce that $a_S \notin X$. Then, by definition of Y, we deduce that $b_S \in Y$. Hence, $N(X') \subseteq Y$.

- Let $b_S \notin N(X')$. Then, for every $a_T \in X'$, $|S \cap T|$ is even. In other words, for every $b_T \notin N(X)$, $|S \cap T|$ is even. We can also say: for every $b_T \notin N(X)$, $b_T \notin N(a_S)$. Therefore, $N(a_S) \subseteq N(X)$. Lemma 4 then applies and tells us $a_S \in X$. This, by definition of Y, means $b_S \notin Y$. Hence, $Y \subseteq N(X')$.

□

5. A data structure for representatives bounded by boolean-width

In this section we give a pre-processing routine setting up a data structure useful for dynamic programming on any decomposition tree (T, δ) of a given graph G. It will allow runtime at the combine step to be a function of $boolw(T, \delta)$, rather than the number of vertices of G. The data structure is particularly useful when the goal is good runtime as a function of boolean-width. For a vertex subset A we will give in Definition 7 an equivalence relation on subsets of A, whose classes will be in a natural bijection with $U(A)$. It has in the worst case $2^{bool\text{-}dim(A)}$ equivalence classes, but we show how to represent each of them by a subset of A of size at most $\lfloor bool\text{-}dim(A) \rfloor$. We show how to compute these representatives and how to set up a data structure that given $X \subseteq A$ in $O(|X|)$ time will access its representative. This access is a main operation in the inner loop of many dynamic programming algorithms, and it must be fast if we want good overall runtime.

We begin with a pre-processing step that is useful also outside the context of boolean-width. Indeed, when solving an optimization problem on a graph G by divide-and-conquer along its decomposition tree (T, δ), the cuts of G given by edges of T are crucial. Since T is a tree having internal nodes of degree three and $n = |V(G)|$ leaves there will be $2n - 3$ such cuts. For the combine step, the important information across a cut $\{A, \overline{A}\}$ is captured by the bipartite subgraph $G(A, \overline{A})$ of G. In speeding up the handling of $G(A, \overline{A})$, a basic idea is that if two vertices have the same neighbors in $G(A, \overline{A})$ then we access the neighborhood information only for one of them.

Definition 6. Let G be a graph and $A \subseteq V(G)$. Denote by $TC_A \subseteq A$ the set containing for each twin class of A the σ-smallest vertex of the class. Define $ntc(T,\delta)$, the number of twin classes of a decomposition tree (T,δ), as the maximum value of $|TC_A|$ and $|TC_{\overline{A}}|$ taken over all the $2n-3$ cuts $\{A,\overline{A}\}$ obtained by removing an edge of T.

See Figure 2 for an example. The measure $ntc(T,\delta)$ was first introduced in [38, Chapter 6.2] where it was called the bimodule-width of (T,δ). The subgraph $G(TC_A, TC_{\overline{A}})$ together with twin class sizes, is a compact representation of the subgraph $G(A,\overline{A})$, and is stored as $M_G(TC_A, TC_{\overline{A}})$. Its use allows runtime at each cut to be bounded by a function of $ntc(T,\delta)$ rather than $|V(G)|$. To this purpose we also want a data structure that given any vertex $x \in A$ in constant time will find the vertex $y \in TC_A$ for x and y being in the same twin class of A. For a single cut there is a simple $O(m)$ time partition refinement algorithm for this task.

Lemma 5. *[9] Let G be an n-vertex m-edge graph and (T,δ) a decomposition tree of G. We can in time $O(nm)$ compute for every edge of T the two vertex sets TC_A and $TC_{\overline{A}}$ associated to the cut $\{A,\overline{A}\}$ given by the edge. We can also in the same time compute for every $x \in A$ a pointer to $y \in TC_A$ for x and y being in the same twin class of A, and similarly for every $x \in \overline{A}$.*

Note that we can avoid the above $O(nm)$ factor, and in Lemma 10 we will show an alternative with a faster runtime whenever $ntc(T,\delta) = o(\sqrt{m})$, which typically holds for a good decomposition tree.

After this first pre-processing step we are ready to consider the main data structure for representatives. Recall that one of the motivations behind the definition of boolean-width is that for many optimization problems two subsets of A having the same neighbors across the cut $\{A,\overline{A}\}$ do not need to be treated separately. This leads to the following equivalence relation on subsets of A, whose classes are in a natural bijection with $U(A)$.

Definition 7. Let G be a graph and $A \subseteq V(G)$. Two vertex subsets $X \subseteq A$ and $X' \subseteq A$ are neighborhood equivalent w.r.t. A, denoted by $X \equiv_A X'$, if $N(X) \setminus A = N(X') \setminus A$.

For each equivalence class of $\equiv_A$ we choose one element as a representative for that class. The representative should be a subset of TC_A and the lexicographically σ-smallest among the sets in the class having minimum cardinality. More formally we define for $A \subseteq V(G)$ the list LR_A of all representatives of $\equiv_A$.

Definition 8 (List of Representatives of $\equiv_A$ and their Neighbors). Given a graph G and $A \subseteq V(G)$ we define the list LR_A of representatives of $\equiv_A$ as the unique family $LR_A \subseteq 2^A$ satisfying:

1) $\forall X \subseteq A, \exists R \in LR_A$ such that $R \equiv_A X$
2) $\forall R \in LR_A$, if $R \equiv_A X$ then $|R| \leq |X|$
3) $\forall R \in LR_A$, if $R \equiv_A X$ and $|R| = |X|$ then R lexicographically σ-smaller than X.

Let LNR_A be the list containing the unions of neighbourhoods of members of LR_A in $G(TC_A, TC_{\overline{A}})$, i.e. $LNR_A = \{N(R) \cap TC_{\overline{A}} : \ R \in LR_A\}$.

See Figure 2 for an example. Note that LNR_A is the projection of $U(A)$ on $TC_{\overline{A}}$. It is straightforward to check that for any $R \in LR_A$ we have $R \subseteq TC_A$ (both LR_A and TC_A are defined using σ) and that there is a bijection between the members of LR_A and the equivalence classes of $\equiv_A$.

Lemma 6. *Let G be a graph, $A \subseteq V(G)$ and $R \in LR_A$. For any $Y, Z \subseteq R$ s.t. $Y \neq Z$, we have $Y \not\equiv_A Z$. Thus $|R| \leq \lfloor bool\text{-}dim(A) \rfloor$.*

Proof: Suppose, for a contradiction, that there are $Y \subseteq R$ and $Z \subseteq R$ such that $Y \neq Z$ and $Y \equiv_A Z$. W.l.o.g. $Y \setminus Z \neq \emptyset$ and so let $v \in Y \setminus Z$. Since $Y \equiv_A Z$ we have $N(v) \cap \overline{A} \subseteq N(Z) \cap \overline{A}$. Hence, $N(R \setminus \{v\}) \cap \overline{A} = N(R) \cap \overline{A}$, contradicting the minimum cardinality of R. Thus, there are $2^{|R|}$ mutually non-equivalent subsets of R, each yielding a distinct element of $U(A)$. Since $|U(A)| = 2^{bool\text{-}dim(A)}$ we have $|R| \leq \lfloor bool\text{-}dim(A) \rfloor$. □

We now describe an algorithm to compute at the same time LR_A, LNR_A, and pointers between the two lists in such a way that given an element N of LNR_A we can access the element R of LR_A such that $N = N(R) \cap \overline{A}$, and vice versa. Firstly, note that by brute force the graph $G(TC_A, TC_{\overline{A}})$ can be computed in time $O(|TC_A| \times |TC_{\overline{A}}|)$ after the preprocessing given in the previous section.

Lemma 7. *Let G be an n-vertex graph and (T, δ) a decomposition tree of G. Assume the pre-processing described in Lemma 5 has been done. Then, in time $O(n \cdot ntc^2(T, \delta) \cdot boolw(T, \delta) \cdot 2^{boolw(T,\delta)})$ we compute for every cut $\{A, \overline{A}\}$ associated to an edge of T the list of representatives LR_A, its neighbor list LNR_A, and pointers such that $R \in LR_A$ and $N \in LNR_A$ point to each other if and only if $N = N(R) \cap \overline{A}$.*

Algorithm 1 List of representatives and their neighborhood

```
Initialize LR_A, LNR_A, NextLevel to be empty
Initialize LastLevel = {∅}
while LastLevel != ∅ do
  for R in LastLevel do
    for every vertex v of TC_A do
      R' = R ∪ {v}
      compute N' = N(R') ∩ TC_Ā
      if R' ≢_A R and N' is not contained in LNR_A then
        add R' to both LR_A and NextLevel
        add N' to LNR_A at the proper position
        add pointers between R' and N'
      end if
    end for
  end for
  set LastLevel = NextLevel, and NextLevel = ∅
end while
```

Proof: We describe the operations needed for a cut $\{A, \overline{A}\}$ in Algorithm 1. Our global computation simply repeats this operation over the $2n-3$ cuts given by the edges of T.

Let us first argue for the correctness of the algorithm. The first iteration of the while-loop will set $\{v\}$ as representative, for every $v \in TC_A$, and there exist no other representatives of size 1 in LR_A. The algorithm computes all representatives of size i before it moves on to those of size $i+1$. LastLevel will contain all representatives of size i while NextLevel will contain all representatives of size $i+1$ found so far. Every representative will be expanded by every possible node and checked against all previously found representatives. The only thing left to prove is that any representative R can be written as $R' \cup \{v\}$ for some representative R'. Assume for contradiction that no R' exists such that $R = R' \cup \{v\}$. Then let v be the lexicographically largest element of R, then $R \setminus \{v\}$ can not be a representative so let R'' be the representative of $[R \setminus \{v\}]_{\equiv_A}$. We know that $R'' \cup \{v\} \equiv_A R$, we know that $|R'' \cup \{v\}| \leq |R|$ and that $R'' \cup \{v\}$ comes before R in a lexicographical ordering contradicting that R is a representative.

We now argue for the runtime. Let $k = \textit{bool-dim}(A)$. The three loops are executed once for each pair consisting of an element $R \in LR_A$ and a vertex

Algorithm 2 Initialize datastructure used for finding representative R of $[X]_{\equiv_A}$

```
Initialize M to a two dimensional table with |LR_A| × |TC_A| elements.
for every vertex v of TC_A do
  for R in LR_A do
    R' = R ∪ {v}
    find R_U in LR_A that is linked to the neighborhood N(R') ∩ TC_Ā in LNR_A
    add a pointer from M[R][v] to R_U
  end for
end for
```

$v \in TC_A$. The number of representatives is exactly 2^k, while the number of vertices is $|TC_A|$, hence at most $O(|TC_A|2^k)$ iterations in total. Inside the innermost for-loop we need to calculate the neighborhood of R'. Note when processing R' that we have already computed $N(R)$, so that we can find $N(R') \cap TC_{\overline{A}}$ in $O(|TC_{\overline{A}}|)$ time. Then to see if $R' \equiv_A R$ we compare the two neighborhoods in $O(|TC_{\overline{A}}|)$ time. Then we want to check if the neighborhood is contained in the list LNR_A. For fast runtime we can represent LNR_A using the so-called self-balancing binary search tree (or AVL tree): searching takes $\log_2(2^k) = k$ steps where for each step comparing two neighborhoods takes $O(|TC_{\overline{A}}|)$ time, yielding $O(|TC_{\overline{A}}|k)$ in total. Inserting into the sorted list LNR_A takes $O(|TC_{\overline{A}}|k)$ time, and in the other lists $O(1)$ time. This means all operations in the inner for-loop can be done in $O(|TC_{\overline{A}}|k)$ time, giving a runtime of $O(|TC_A||TC_{\overline{A}}|k2^k)$ for each cut $\{A, \overline{A}\}$. □

Given $X \subseteq A$ we will now address the question of computing the representative R of $[X]_{\equiv_A}$, in other words accessing the entry R of LR_A such that $X \equiv_A R$. The naive way to do this is to search LNR_A for the set $N(X) \cap \overline{A}$. However, we want to do this faster, namely in $O(|X|)$ time. To this aim we construct an auxiliary data-structure that maps a pair (R, v), consisting of one representative R from LR_A and one vertex from TC_A, to the representative R' of the class $[R \cup \{v\}]_{\equiv_A}$.

Lemma 8. *Let G be an n-vertex graph and (T, δ) a decomposition tree of G. Assume the pre-processing described in Lemmata 5 and 7 has been done. Then, in time $O(n \cdot ntc^2(T, \delta) \cdot boolw(T, \delta) \cdot 2^{boolw(T,\delta)})$ we compute for every cut*

Algorithm 3 Finding representative R of $[X]_{\equiv_A}$

Initialize R to be empty.
for every vertex u of X **do**
  find $v \in TC_A$ with u and v in same twin class of A, using pointer described in Lemma 5
  $R = M[R][v]$ for M computed in Algorithm 2
end for

$\{A, \overline{A}\}$ associated to an edge of T a datastructure allowing, for any $X \subseteq A$, to access in $O(|X|)$ time the entry R of LR_A such that $X \equiv_A R$.

Proof: As with Lemma 7, we only describe the algorithm for a cut $\{A, \overline{A}\}$. The computation of the datastructure is described in Algorithm 2, while Algorithm 3 describes how to use it. The idea is to build R from an "incremental" scanning of the elements of $X = \{x_1, x_2, \ldots, x_p\}$ (see Algorithm 3): an algorithmic invariant is that at step i the value of R is exactly the representative of $\{x_1, x_2, \ldots, x_i\}$. The correctness of this invariant (of Algorithm 3) depends on the correctness of the computation of table M in Algorithm 2. To prove the latter correctness, just notice that the algorithm essentially exploites the bijection between the elements of LR_A and LNR_A.

Let us analyse the complexity of Algorithm 2. Let $k = \textit{bool-dim}(A)$. There are two for loops in the algorithm iterating $O(|TC_A|2^k)$ times in total. For each iteration, finding the neighborhood of R' takes $O(|TC_{\overline{A}}|)$ time, searching LNR_A takes $O(|TC_{\overline{A}}|k)$, and comparing neighborhoods takes $O(|TC_{\overline{A}}|)$ time, and the remaining operations take constant time. Hence, the runtime is $O(|TC_A||TC_{\overline{A}}|k2^k)$ for each cut $\{A, \overline{A}\}$. The complexity analysis for Algorithm 3 is straightforward. □

6. Dynamic programming for fast runtime by boolean-width

We show in this section how in general to apply dynamic programming on a decomposition tree (T, δ) of a graph G while analysing runtime as a function of $\textit{boolw}(T, \delta)$. We focus on the Maximum Indpendent Set (Max IS) and Minimum Dominating Set (Min DS) problems. The algorithms given for Max IS and Min DS can be deduced from similar algorithms in [9], that appeared before the introduction of boolean-width. We give the algorithms here using the new and simpler terminology and show that they have better runtime due to faster pre-processing and better data structures. We also give

algorithms to handle the vertex weighted cases and the case of counting all independent sets and dominating sets of given size.

Note that we do not assume any further information from the input of (T, δ) other than T being a tree with internal nodes of degree three and δ a bijection between its leaves and $V(G)$. As is customary, and as in Definition 1, we first subdivide an arbitrary edge of T to get a new root node r, denote by T_r the resulting rooted tree, and let the algorithm follow a bottom-up traversal of T_r. Recall that for a node a of T we denote by A_a the subset of $V(G)$ in bijection δ with the leaves of the subtree of T_r rooted at a. For any dynamic programming on decomposition trees it is important to keep in mind the below observation, that follows directly from definitions.

Observation 2. *If in the tree T_r the node w has children a and b then $\{A_a, A_b, \overline{A_w}\}$ forms a 3-partition of $V(G)$.*

Another crucial observation is the coarsening of neighborhood equivalence classes when traversing from a child node a to its parent node w.

Observation 3. *Let G be a graph with $A_a \subseteq A_w \subseteq V(G)$ and let $X, Y \subseteq A_a$. If $X \equiv_{A_a} Y$ then $X \equiv_{A_w} Y$.*

Proof: Since $X \equiv_{A_a} Y$ we have $N(X) \cap \overline{A_a} = N(Y) \cap \overline{A_a}$. Since $A_a \subseteq A_w$ we have $\overline{A_w} \subseteq \overline{A_a}$ and thus $N(X) \cap \overline{A_w} = N(Y) \cap \overline{A_w}$ implying $X \equiv_{A_w} Y$. □

With each node w of T_r we associate a table data structure Tab_w. In general, the table will store optimal solutions to subproblems related to the cut $\{A_w, \overline{A_w}\}$. To simplify the initialization of Tab_l (for every leaf l of T_r) we assume throughout the section that G has no isolated vertices: there are straightforward preprocessings in order to remove isolated vertices for any of the problems we consider.

6.1. Maximum Independent Set

Let us first consider the Maximum Independent Set (Max IS) problem. For Max IS the table Tab_w is particularly easy to define since it will be indexed by the representatives of the classes of $\equiv_{A_w}$.

Definition 9. The table Tab_w used for Max IS at a node w of T_r has index set LR_{A_w}. For $R \in LR_{A_w}$ the table should store

$$Tab_w[R] = \max_{S \subseteq A_w} \{|S| : S \equiv_{A_w} R \text{ and } S \text{ an IS of } G\}.$$

Note that Tab_w has exactly $2^{bool\text{-}dim(A_w)}$ entries. For a leaf l of T_r, $A_l = \{\delta(l)\}$ and $\equiv_{A_l}$ has two equivalence classes: one containing $\emptyset$ and the other containing A_l, and these are also the representatives. We initialize tables at leaves of T_r brute-force by setting $Tab_l[\emptyset] = 0$ and $Tab_l[\{\delta(l)\}] = 1$. The combine step filling the table at an inner node after tables of its children have been filled is given in Algorithm 4.

Algorithm 4 Combine step for Max IS at node w with children a, b

```
for all R_w ∈ LR_{A_w} do
  initialize Tab_w[R_w] = 0
end for
for all pairs R_a ∈ LR_{A_a}, R_b ∈ LR_{A_b} do
  if R_a ∪ R_b is an IS in G(R_a, R_b) then
    find the representative R_w of the class [R_a ∪ R_b]_{≡_{A_w}}
    Tab_w[R_w] = max(Tab_w[R_w], Tab_a[R_a] + Tab_b[R_b])
  end if
end for
```

Lemma 9. *The Combine step for Max IS is correct.*

Proof: Let node w have children a, b and assume Tab_a, Tab_b have been filled correctly. We show that after executing the Combine step in Algorithm 4 the table Tab_w is filled according to Definition 9. Let $R_w \in LR_{A_w}$ and assume $I_w \subseteq A_w$ is an IS of G such that $R_w \equiv_{A_w} I_w$. We first show that $Tab_w[R_w] \geq |I_w|$. Let $I_a = I_w \cap A_a$ and $I_b = I_w \cap A_b$ and let $R_a \in LR_{A_a}, R_b \in LR_{A_b}$ be such that $R_a \equiv_{A_a} I_a$ and $R_b \equiv_{A_b} I_b$. Thus $I_a \cup I_b$ is an IS in G and $R_a \cup R_b$ is an IS in $G(R_a, R_b)$. Also, I_a and I_b are independent sets in G, and therefore $Tab_a[R_a] \geq |I_a|$ and $Tab_b[R_b] \geq |I_b|$. Thus, when considering the pair R_a, R_b the combine step will ensure that the entry for the representative of the class $[R_a \cup R_b]_{\equiv_{A_w}}$ is at least $|I_a| + |I_b| = |I_w|$. It remains to show that this representative is R_w. By Observation 3 we have $R_a \equiv_{A_w} I_a$ and $R_b \equiv_{A_w} I_b$ so that $R_a \cup R_b \equiv_{A_w} I_a \cup I_b$. Since $I_w = I_a \cup I_b$ and we assumed $R_w \equiv_{A_w} I_w$ we therefore have $R_a \cup R_b \equiv_{A_w} R_w$ as desired.

To finish the correctness proof, we need to show that if $Tab_w[R_w] = k$ then there exists $I_w \subseteq A_w$ with $|I_w| = k$ and $I_w \equiv_{A_w} R_w$ and I_w an IS in G. For this, note that the Combine step increases the value of $Tab_w[R_w]$ only if there exist indices $R_a \in LR_{A_a}$ and $R_b \in LR_{A_b}$ such that $R_a \cup R_b$ is an IS in

$G(R_a, R_b)$, and $R_a \cup R_b \equiv_{A_w} R_w$, and $Tab_a[R_a] = k_a$, and $Tab_b[R_b] = k_b$, and $k_a + k_b = k$. Since Tab_a, Tab_b are filled correctly we have two independent sets I_a, I_b in G with $R_a \equiv_{A_a} I_a$ and $R_b \equiv_{A_b} I_b$ and $|I_a| = k_a$ and $|I_b| = k_b$. We claim that $I_a \cup I_b$ is the desired I_w. Since $R_a \cup R_b$ is an IS of $G(R_a, R_b)$ it is clear that $I_a \cup I_b$ is an IS in G of size $k_a + k_b = k$. It remains to show that $I_a \cup I_b \equiv_{A_w} R_w$. By Observation 3 we have $R_a \equiv_{A_w} I_a$ and $R_b \equiv_{A_w} I_b$ so that $R_a \cup R_b \equiv_{A_w} I_a \cup I_b$. Since we assumed $R_a \cup R_b \equiv_{A_w} R_w$ we therefore have $I_a \cup I_b \equiv_{A_w} R_w$ as desired. □

Theorem 5. *Given an n-vertex graph G and a decomposition tree (T, δ) of G, we can solve the Maximum Independent Set problem on G in time $O(n(n + ntc^2(T, \delta) \cdot k2^k + k^2 2^{2k}))$ where $k = boolw(T, \delta)$. The runtime can also be written $O(n^2 k 2^{2k})$.*

Proof: We start by running, for all cuts $\{A, \overline{A}\}$ given by edges of T, the pre-processing routines described in Section 5. However, for a faster runtime we replace Lemma 5 by below Lemma 10. That is, we first compute for all such cuts the twin classes TC_A and $TC_{\overline{A}}$ as described in Lemma 10, for a global runtime in $O(n(n+ntc^2(T, \delta)))$. Second, we compute representatives of neighborhood equivalence classes LR_A, $LR_{\overline{A}}$, LNR_A, and $LNR_{\overline{A}}$ as described in Lemma 7. This takes time $O(n \cdot ntc^2(T, \delta) \cdot k2^k)$. Third, set up the datastructure for finding a representative of $[X]_{\equiv_A}$ and $[Y]_{\equiv_{\overline{A}}}$ as described in Lemma 8. This takes the same time as the latter operation, namely $O(n \cdot ntc^2(T, \delta) \cdot k2^k)$.

We then perform the dynamic programming described in this section, subdividing an arbitrary edge of T by a new root node r to get T_r, initializing the table for every leaf of T_r, and traversing T_r in a bottom-up fashion filling the table for every internal node based on already filled tables of its children. At the root r we have $A_r = V(G)$ so that by induction on the rooted tree applying Lemma 9 the size of the maximum IS in G is found at the unique entry of Tab_r.

The combine step is executed $O(n)$ times and loops over $O(2^{2k})$ pairs of representatives. In each execution of this loop we must check that there are no edges between R_{A_a} and R_{A_b}, and this can be done in time $O(k^2)$. Also we must find the representative of the class $[R_a \cup R_b]_{\equiv_{A_w}}$, which using the data structure of Lemma 8 takes time $O(|R_a \cup R_b|)$ which is $O(k)$ by Lemma 6. The runtime is therefore $O(n(n+ntc^2(T, \delta) \cdot k2^k + k^2 2^{2k}))$, and also $O(n^2 k 2^{2k})$ since $ntc(T, \delta) \leq \min\{n, 2^k\}$ and $k \leq n$. □

To avoid the nm factor in the runtimes we had to replace the simple computation of twin classes given by Lemma 5 by the following Lemma.

Lemma 10. *Let G be an n-vertex graph and (T, δ) a decomposition tree of G. In time $O(n(n + ntc^2(T, \delta)))$ we compute for every edge of T the two vertex sets TC_A and $TC_{\overline{A}}$ associated to the cut $\{A, \overline{A}\}$ given by the edge. In the same time we compute for every $x \in A$ a pointer to $y \in TC_A$ for x and y being in the same twin class of A, and similarly for every $x \in \overline{A}$.*

Proof: It is more convenient to deal with rooted trees here, so we address the rooted tree T_r as in Definition 1. The idea is to proceed in two steps. In the first step we compute in a top-down traversal of T_r the set TC_{A_a} for every node a of T_r. Then, in a second top-down traversal of T_r we compute all sets $TC_{\overline{A_a}}$.

A refinement operation of an ordered partition $\mathcal{P} = (P_1, P_2, \ldots, P_k)$ using X as pivot is the act of splitting every part P_i of $\mathcal{P}$ into $P_i \cap X$ and $P_i \setminus X$. With the appropriate use of data structure [21], such an operation can be implemented to run in $O(|X|)$ time. If the elements of each P_i are initially ordered, the refinement operations will preserve their order. We initialize $\mathcal{P}_r = \{V(G)\}$, where the elements of $V(G)$ follow in order σ. The following claim constitutes the first top-down traversal of T_r.

Claim 10.1. ([8, Lemma 2]) *We can compute TC_{A_a} for every a in T_r in $O(n^2)$ time.*

The full proof of Claim 10.1. is given in [8] but let us sketch the idea. If $a = r$ there is nothing to show, otherwise let w be the parent of a and let b be the sibling of a. Suppose by induction that the twin-class partition $\mathcal{P}_w = \{P_1, P_2, \ldots, P_k\}$ of A_w has been computed before a is visited (when w was visited). Then, refining $\mathcal{P}_w[A_a] = \{P_1 \cap A_a, P_2 \cap A_a, \ldots, P_k \cap A_a\}$ using $N(z) \cap A_a$ as pivot, for every $z \in A_b$, will result in exactly the twin-class partition of A_a. This idea can be implemented to run globally in $O(n^2)$ time (the main trick is to compute $N(z) \cap A_a$ for any $z \in A_b$ since $P_w[A_a]$ can be computed simply by refining P_w using A_a as pivot). The implementation details are described in [8, Section 3]. After this, we scan every class P of $\mathcal{P}_a$ and pick the first element of P in order to build the list TC_{A_a}.

We now compute $TC_{\overline{A_a}}$ for every node a of T_r by a second top-down traversal of T_r. Recall w is the parent of a and b. Clearly, $\overline{A_a} = A_b \cup \overline{A_w}$. The twin-class partitions $\mathcal{P}_a$ and $\mathcal{P}_b$ of A_a and A_b have already been computed as

described above. By induction we suppose that, before visiting a, the twin-class partition $\mathcal{P}_{\overline{w}}$ of $\overline{A_w}$ has also been computed (when w was visited). Pick one representative vertex per part in $\mathcal{P}_a$ and put them together in a list R_a (we can also use $R_a = TC_{A_a}$). Likewise, pick one representative vertex per part in $\mathcal{P}_b \cup \mathcal{P}_{\overline{w}}$ and put them together in a list $R_{\overline{a}}$, with additional pointers so that from every element x of $R_{\overline{a}}$ we can trace back the partition class of $\mathcal{P}_b \cup \mathcal{P}_{\overline{w}}$ containing x. We then compute $H = G(R_a, R_{\overline{a}})$. We now initialize $\mathcal{P}_{\overline{a}} = \{R_{\overline{a}}\}$ and, for every $z \in R_a$, refine $\mathcal{P}_{\overline{a}}$ using the neighborhood of z in H as pivot. Finally, for every class P of $\mathcal{P}_{\overline{a}}$, we replace every element x of P by all the elements belonging to the partition class in $\mathcal{P}_b \cup \mathcal{P}_{\overline{w}}$ for which x is representative. It is then straightforward to check that $\mathcal{P}_{\overline{a}}$ is now exactly the twin-class partition of $\overline{A_a}$. After this, we scan every class P of $\mathcal{P}_{\overline{a}}$ and pick the first element of P in order to build the list $TC_{\overline{A_a}}$.

We now analyse the time complexity of the global computation. First we have to run the algorithm mentioned in Claim 10.1, which takes $O(n^2)$ time. For the rest, note that $|R_a| \leq ntc(T, \delta)$ and $|R_{\overline{a}}| \leq 2 \times ntc(T, \delta)$, *i.e.* we can compute R_a, $R_{\overline{a}}$ and H in $O(ntc^2(T, \delta))$ time (brute-force adjacency check for H). The time for initializing the data structure for partition refinement, and for subsequently performing all refinement operations is globally linear in the size of H, namely in $O(ntc^2(T, \delta))$. The remaining operations consist basically in following the pointers, whose total sum is bounded by the size of $R_{\overline{a}}$. Whence, $TC_{\overline{A_a}}$ can be computed in $O(ntc^2(T, \delta))$ for every such a, leading to an $O(n \cdot ntc^2(T, \delta))$ runtime on T_r. □

6.2. Counting independent sets

Let α be the size of the max IS in G. Counting the number of independent sets in G of cardinality k for each $0 \leq k \leq \alpha$ can be accomplished by a similar algorithm having runtime with an additional factor α^2. The table Tab_w must be indexed by $LR_{A_w} \times \{0, 1, ..., |A_w|\}$ and store

$$Tab_w[R][k] = |\{S : S \subseteq A_w \text{ and } S \equiv_{A_w} R \text{ and } S \text{ an IS of } G \text{ and } |S| = k\}|.$$

The initialization at a leaf l of T_r should be:
$Tab_l[\delta(l)][0] = 0$
$Tab_l[\delta(l)][1] = 1$
$Tab_l[\emptyset][0] = 1$
$Tab_l[\emptyset][1] = 0$

The combine step is given in Algorithm 5. Note that two families F_a and F_b of vertex subsets, taken from two disjoint sets of vertices, can be

combined into $|F_a| * |F_b|$ larger vertex subsets. Note also that in the inner loop of the combine step $k_a, k_b \leq \alpha$. The proof of correctness and runtime remains otherwise much the same.

Algorithm 5 Combine step for Counting number of IS at node w with children a, b

for all $R_w \in LR_{A_w}$ and all $k : 0 \leq k \leq |A_w|$ **do**
  initialize $Tab_w[R_w][k] = 0$
end for
for all pairs $R_a \in LR_{A_a}, R_b \in LR_{A_b}$ **do**
  if $R_a \cup R_b$ is an IS in $G(R_a, R_b)$ **then**
    find max k_a and k_b such that $Tab_a[R_a][k_a] > 0$ and $Tab_b[R_b][k_b] > 0$
    find the representative R_w of the class $[R_a \cup R_b]_{\equiv_{A_w}}$
    for all pairs $i, j : 0 \leq i \leq k_a$ and $0 \leq j \leq k_b$ **do**
      $Tab_w[R_w][i + j] = Tab_w[R_w][i + j] + Tab_a[R_a][i] * Tab_b[R_b][j]$
    end for
  end if
end for

Theorem 6. *Given an n-vertex graph G and a decomposition tree (T, δ) of G, we can count the number of independent sets of G of any size in time $O(\alpha^2 n^2 k 2^{2k})$, where $k = boolw(T, \delta)$ and α is the size of the maximum independent set in G.*

6.3. Minimum Dominating Set

We want to solve the Minimum Dominating Set (Min DS) problem on a graph G by dynamic programming along a decomposition tree of G. The algorithm for Min DS is more complicated than the one given for Max IS, but its runtime as a function of boolean-width is only slightly higher. For a cut $\{A, \overline{A}\}$ note that, unlike the case of independent sets, a set S of vertices dominating A will include also vertices of $\overline{A}$ that dominate vertices of A "from the outside". This motivates the following definition.

Definition 10. Let G be a graph and $A \subseteq V(G)$. For $X \subseteq A$, $Y \subseteq \overline{A}$, if $A \setminus X \subseteq N(X \cup Y)$ we say that the pair (X, Y) dominates A.

Note that 'pair domination' behaves well w.r.t. the neighborhood equivalence classes.

Lemma 11. *Let G be a graph and $A \subseteq V(G)$. Let $X \subseteq A$, $Y, Y' \subseteq \overline{A}$, and $Y \equiv_{\overline{A}} Y'$. Then (X, Y) dominates A if and only if (X, Y') dominates A.*

Proof: Since (X, Y) dominates A we have $A \setminus X \subseteq N(X \cup Y)$. Since $Y \equiv_A Y'$ we have $N(Y) \setminus A = N(Y') \setminus A$. Then it follows that $A \setminus X \subseteq N(X \cup Y')$, meaning (X, Y') dominates A. □

We will index the table Tab_w at w by two sets: one representing the equivalence class of $\equiv_A$ that *partially* dominates A "from the inside", and one representing the equivalence class of $\equiv_{\overline{A}}$ that dominates the rest of A "from the outside".

Definition 11. The table Tab_w used for Min DS at a node w of T_r has index set $LR_{A_w} \times LR_{\overline{A_w}}$. For $R_w \in LR_{A_w}$ and $R_{\overline{w}} \in LR_{\overline{A_w}}$ the table should store

$$Tab_w[R_w][R_{\overline{w}}] = \min_{S \subseteq A_w} \{|S| : S \equiv_{A_w} R_w \text{ and } (S, R_{\overline{w}}) \text{ dominates } A_w\}$$

and ∞ if no such S exists.

Note that Tab_w has exactly $2^{2bool\text{-}dim(A_w)}$ entries. For every node w we assume that initially every entry of Tab_w is set to ∞. For a leaf l of T_r, we have $A_l = \{\delta(l)\}$. Note that $\equiv_{A_l}$ has only two equivalence classes: one containing $\emptyset$ and the other containing A_l. For $\equiv_{\overline{A_l}}$, we have the same situation: one class containing $\emptyset$ and the other containing $\overline{A_l}$. We initialize Tab_l brute-force. Let R be the representative of $[\overline{A_l}]_{\equiv_{\overline{A_l}}}$.

$Tab_l[\emptyset][\emptyset] = \infty$
$Tab_l[\{\delta(l)\}][\emptyset] = 1$
$Tab_l[\{\delta(l)\}][R] = 1$
$Tab_l[\emptyset][R] = 0$.

Let w be a node with two children a and b, and assume that Tab_a and Tab_b have been correctly computed. Note that each of them can have up to $2^{2boolw(T,\delta)}$ entries, and therefore a naive computation of Tab_w by looping over all pairs of entries in the children tables will result in a worst case runtime in $O(2^{4boolw(T,\delta)})$ multiplied by the time spent for finding the right entry of the parent table Tab_w that we want to update. Instead, in Algorithm 6 we apply Observation 2 to give an $O^*(2^{3boolw(T,\delta)})$ time algorithm by looping over only $2^{boolw(T,\delta)}$ entries in each table.

The following lemma will be useful in the correctness proof.

Algorithm 6 Combine step for Min DS at node w with children a, b

for all $R_w \in LR_{A_w}, R_{\overline{w}} \in LR_{\overline{A_w}}$ **do**
  initialize $Tab_w[R_w][R_{\overline{w}}] = \infty$
end for
for all $R_a \in LR_{A_a}, R_b \in LR_{A_b}, R_{\overline{w}} \in LR_{\overline{A_w}}$ **do**
  find the representative $R_{\overline{a}}$ of the class $[R_b \cup R_{\overline{w}}]_{\equiv_{\overline{A_a}}}$
  find the representative $R_{\overline{b}}$ of the class $[R_a \cup R_{\overline{w}}]_{\equiv_{\overline{A_b}}}$
  find the representative R_w of the class $[R_a \cup R_b]_{\equiv_{A_w}}$
  $Tab_w[R_w][R_{\overline{w}}] = min(Tab_w[R_w][R_{\overline{w}}], Tab_a[R_a][R_{\overline{a}}] + Tab_b[R_b][R_{\overline{b}}])$
end for

Lemma 12. *For a graph G, let A, B, W be a 3-partitioning of $V(G)$, and let $S_a \subseteq A, S_b \subseteq B$ and $S_w \subseteq W$. $(S_a, S_b \cup S_w)$ dominates A and $(S_b, S_a \cup S_w)$ dominates B iff $(S_a \cup S_b, S_w)$ dominates $A \cup B$.*

Proof: Let $S = S_a \cup S_b \cup S_w$. Clearly, $(S_a, S_b \cup S_w)$ dominates A iff $A \setminus S_a \subseteq N(S)$. Likewise, $(S_b, S_a \cup S_w)$ dominates B iff $B \setminus S_b \subseteq N(S)$. Therefore, $A \setminus S_a \subseteq N(S)$ and $B \setminus S_b \subseteq N(S)$ iff $A \cup B \setminus S_a \cup S_b \subseteq N(S)$ iff $(S_a \cup S_B, S_w)$ dominates $A \cup B$. □

Lemma 13. *The Combine step for Min DS is correct.*

Proof: Let node w have children a, b and assume Tab_a, Tab_b have been filled correctly. We show that after executing the Combine step in Algorithm 6 the table Tab_w is filled according to Definition 11. We first show for every $R_w \in LR_{A_w}$ and $R_{\overline{w}} \in LR_{\overline{A_w}}$ that if there is a set $S_w \equiv_{A_w} R_w$ such that $(S_w, R_{\overline{w}})$ dominates A_w, then $Tab_w[R][R_{\overline{w}}] \leq |S_w|$. Let $S_a = S_w \cap A_a$ and $S_b = S_w \cap A_b$. The algorithm loops over all triples of representatives: at some point it will check $(R_a, R_b, R_{\overline{w}})$, where R_a is the representative of $[S_a]_{\equiv_{A_a}}$ and R_b is the representative of $[S_b]_{\equiv_{A_b}}$. We know that $(S_a \cup S_b, R_{\overline{w}})$ dominates A_w so it follows from Lemma 12 that $(S_a, S_b \cup R_{\overline{w}})$ dominates A_a. Note that $R_{\overline{a}}$ as computed in the combine step is the representative of $[S_b \cup R_{\overline{w}}]_{\equiv_{\overline{A_a}}}$ so that it follows from Lemma 11 that $(S_a, R_{\overline{a}})$ dominates A_a. Hence, $Tab_a[R_a][R_{\overline{a}}] \leq |S_a|$. Arguing analogously we have that $Tab_b[R_b][R_{\overline{b}}] \leq |S_b|$. Thus, to conclude that $Tab_w[R_w][R_{\overline{w}}] \leq |S_a| + |S_b| = |S_w|$ all we need to show is that $R_w \equiv_{A_w} R_a \cup R_b$. By Observation 3 we have $R_a \equiv_{A_w} S_a$ and $R_b \equiv_{A_w} S_b$ so that $R_a \cup R_b \equiv_{A_w} S_a \cup S_b$. Since $S_w = S_a \cup S_b$ and we assumed $R_w \equiv_{A_w} S_w$ we therefore have $R_a \cup R_b \equiv_{A_w} R_w$ as desired.

To finish the correctness proof, we need to show that if $Tab_w[R_w][R_{\overline{w}}] = k$ then there exists $S_w \subseteq A_w$ with $|S_w| = k$ and $S_w \equiv_{A_w} R_w$ such that $(S_w, R_{\overline{w}})$ dominates A_w in G. For this note that, from the Combine step and assumed correctness of children tables, there must exist indices $R_a \in LR_{A_a}$ and $R_b \in LR_{A_b}$, with $S_a \equiv_{A_a} R_a$ and $S_b \equiv_{A_b} R_b$ such that $(S_a, R_{\overline{a}})$ dominates A_a, and $(S_b, R_{\overline{b}})$ dominates A_b, and $|S_a \cup S_b| = s$, and with $R_{\overline{a}}$ the representative of $[R_b \cup R_{\overline{w}}]_{\equiv_{\overline{A_a}}}$, and $R_{\overline{b}}$ the representative of $[R_a \cup R_{\overline{w}}]_{\equiv_{\overline{A_b}}}$. We claim that $S_a \cup S_b$ is the desired S_w. Since $(S_b \cup R_{\overline{w}}) \equiv_{\overline{A}_a} R_{\overline{a}}$ and $(S_a, R_{\overline{a}})$ dominates A_a it follows from Lemma 11 that $(S_a, S_b \cup R_{\overline{w}})$ dominates A_a. Likewise, $(S_b, S_a \cup R_{\overline{w}})$ dominates A_b. We deduce from Lemma 12 that $(S_a \cup S_b, R_{\overline{w}})$ dominates $A_a \cup A_b = A_w$. It remains to show that $S_a \cup S_b \equiv_{A_w} R_w$. By Observation 3 we have $R_a \equiv_{A_w} S_a$ and $R_b \equiv_{A_w} S_b$ so that $R_a \cup R_b \equiv_{A_w} S_a \cup S_b$. Since we assumed $R_a \cup R_b \equiv_{A_w} R_w$ we therefore have $S_a \cup S_b \equiv_{A_w} R_w$ as desired. □

Theorem 7. *Given an n-vertex graph G and a decomposition tree (T, δ) of G, the Minimum Dominating Set problem on G can be solved in time $O(n(n + ntc^2(T, \delta) \cdot k2^k + k^2 2^{3k}))$ where $k = boolw(T, \delta)$. The runtime can also be written $O(n^2 + nk2^{3k})$.*

Proof: We start by running, for all cuts $\{A, \overline{A}\}$ given by edges of T, the pre-processing routines described in Section 5, with Lemma 5 being replaced by Lemma 10. These operations take time $O(n \cdot ntc^2(T, \delta) \cdot k2^k)$ (see proof of Theorem 5).

We then perform the dynamic programming described in this section, subdividing an arbitrary edge of T by a new root node r to get T_r, initializing the table for every leaf of T_r, and traversing T_r in a bottom-up fashion filling the table for every internal node based on already filled tables of its children. At the root r we have $A_r = V(G)$ so that by induction on the rooted tree applying Lemma 9 the size of the maximum IS in G is found at the unique entry of Tab_r.

The combine step is executed $O(n)$ times and loops over $O(2^{3k})$ triplets of representatives. In each execution of this loop we must find the representative for $R_b \cup R_{\overline{w}}$, $R_a \cup R_{\overline{w}}$, and $R_a \cup R_b$. Each of the three is of size $O(k)$, so finding their representatives using the data structure of Lemma 8 takes $O(k)$ time (see Lemma 6). The runtime is therefore $O(n(n + ntc^2(T, \delta) \cdot k2^k + k2^{3k}))$, and also $O(n^2 + nk2^{3k})$ since $ntc(T, \delta) \leq 2^k$. □

6.4. Counting dominating sets

Counting the number of dominating sets in G of cardinality k for each $0 \leq k \leq n$ can be accomplished by a similar algorithm having runtime with an additional factor n^2. The table Tab_w should be indexed by $LR_{A_w} \times LR_{\overline{A_w}} \times \{0, 1, ..., n\}$ and store

$Tab_w[R_w][R_{\overline{w}}][k] = |\{S : S \subseteq A_w \text{ and } S \equiv_{A_w} R_w \text{ and } (S, R_{\overline{w}}) \text{ dominates } A_w \text{ and } |S| = k\}|$.

The initialization at a leaf l of T_r sets all entries to zero except (for R the representative of $[\overline{A_l}]_{\equiv_{\overline{A_l}}}$):

$Tab_l[\delta(l)][\emptyset][1] = 1$

$Tab_l[\delta(l)][R][1] = 1$

$Tab_l[\emptyset][R][0] = 1$

The Combine step is given in Algorithm 7. There are four things to consider for the correctness. All sets S we count have to be partial dominating sets, we must keep track of their sizes correctly, we must not leave out any such set and we must not count any such set twice. All these except not counting twice follow easily. Let us therefore argue that no dominating set is counted twice. We do this by induction on the decomposition tree from the leaves to the root. Assume for contradiction that there is an entry $Tab_w[R_w, R_{\overline{w}}]$ with some set S counted twice, while tables Tab_a and Tab_b at children of w are correct. The combine step loops over all triples $R_a, R_b, R_{\overline{w}}$ and $R_{\overline{w}}$ is used in the index of the update so $R_{\overline{w}}$ must have been the same in any update counting S. Note also that S uniquely defines the two representatives R_a and R_b (since the representative for $S \cap A_a$, respectively $S \cap A_b$ is unique), and S also uniquely defines the integers k_a and k_b. But then there is only a single triple $R_a, R_b, R_{\overline{w}}$ and unique integers k_a, k_b that could have resulted in an update of $Tab_w[R_w, R_{\overline{w}}]$ counting the set S so correctness follows.

Theorem 8. *Given an n-vertex graph G and a decomposition tree (T, δ) of G, we can count the number of dominating sets of G of any size in time $O(n^3 k 2^{3k})$, where $k = boolw(T, \delta)$.*

6.5. Independent Dominating Sets

Combining the requirements of independence and domination in the definition of tables and in the algorithm we can solve both the Minimum and Maximum Independent Dominating Set problems. Note for the runtime

Algorithm 7 Combine step for Counting number of dominating sets at node w with children a, b

for all $R_w \in LR_{A_w}, R_{\overline{w}} \in LR_{\overline{A_w}}, k \in [0, n]$ **do**
  initialize $Tab_w[R_w][R_{\overline{w}}][k] = 0$
end for
for all $R_a \in LR_{A_a}, R_b \in LR_{A_b}, R_{\overline{w}} \in LR_{\overline{A_w}}$ **do**
  find the representative $R_{\overline{a}}$ of the class $[R_b \cup R_{\overline{w}}]_{\equiv_{\overline{A_a}}}$
  find the representative $R_{\overline{b}}$ of the class $[R_a \cup R_{\overline{w}}]_{\equiv_{\overline{A_b}}}$
  find the representative R_w of the class $[R_a \cup R_b]_{\equiv_{A_w}}$
  for $k_a = 0$ to $k_a \leq n$ **do**
    for $k_b = 0$ to $k_b \leq n$ **do**
      $Tab_w[R_w][R_{\overline{w}}][k_a + k_b] += Tab_a[R_a][R_{\overline{a}}][k_a] \times Tab_b[R_b][R_{\overline{b}}][k_b]$
    end for
  end for
end for

given in Theorem 5 that $O(n(n + ntc^2(T, \delta) \cdot k2^k + k^2 2^{2k}))$ is bounded by $O(n^2 + nk2^{3k})$ since $ntc(T, \delta) \leq 2^k$.

Corollary 3. *Given an n-vertex graph G and a decomposition tree (T, δ) of G, we can solve the Minimum Independent Dominating Set and Maximum Independent Dominating Set problems on G in time $O(n^2 + nk2^{3k})$, where $k = boolw(T, \delta)$.*

6.6. Weighted cases

If the input graph G comes with a weight function on the vertices $w : V(G) \to \mathbb{R}$ we may wish to find the independent set with largest sum of weights, or the dominating set with smallest sum of weights. This can be accomplished in the same runtime as Max IS and Min DS and requires only a very small change to the algorithm. For $S \subseteq V(G)$ let $w(S) = \Sigma_{v \in S} w(v)$. The tables must store

For Max weighted IS:
$Tab_w[R] = \max_{S \subseteq A_w}\{w(S) : S \equiv_{A_w} R \text{ and } S \text{ an IS of } G\}$

For Min weighted DS:
$Tab_w[R][R'] = \min_{S \subseteq A_w}\{w(S) : S \equiv_{A_w} R \text{ and } (S, R') \text{ dominates } A_w\}$

and the algorithms remain the same. Likewise for finding an independent dominating set with smallest or largest weight.

7. Conclusion and Perspectives

Since the first introduction of boolean-width at IWPEC 2009 (essentially an extended abstract of this paper) several new results have appeared that we now summarize. Using the pre-processing routines described in Section 5 of this paper, algorithms with runtime $O^*(2^{c \cdot k^2})$ have been given for a large class of vertex subset and vertex partitioning problems (the so-called (σ, ρ)-problems and D_q-problems [43]) for problem specific constants c [1], given a decomposition tree of boolean-width k.

For several classes of perfect graphs, like interval graphs and permutation graphs, it has been shown that boolean-width is logarithmic and that a decomposition witnessing this can be found in polynomial time [4]. On the other hand rank-width, and hence the other main parameters, can on these graph classes have value proportional to the square root of the number of vertices. Additionally, for these graph classes the above-mentioned vertex subset and partitioning problems will have runtime $O^*(2^{c \cdot k})$, yielding the first polynomial-time algorithms for the weighted versions of all those problems on e.g. permutation graphs.

Recent results tie boolean-width nicely to tree-width and branch-width by showing that for any graph we have $boolw(G) \leq tw(G) + 1$ and $boolw(G) \leq bw(G)$ for $bw(G) \neq 0$ [1]. For a random graph G on n vertices it has been shown that whp $boolw(G) = \Theta(\log^2 n)$ [1], this in contrast to $rw(G) = tw(G) = bw(G) = cw(G) = ntc(G) = modw(G) = \Theta(n)$ [27, 29, 31]. Moreover, a decomposition tree witnessing the polylog boolean-width of a random graph can be found in polynomial time, so that we get quasi-polynomial time algorithms for the above-mentioned problems on input a random graph.

An important question concerns the practical applicability of boolean-width. The divide-and-conquer algorithms given here are practical and easy to implement. A heuristic for computing a decomposition tree of low boolean-width has been implemented and experiments made on the graphs in TreewidthLIB show that boolean-width could indeed have practical applicability [25].

There are many questions about boolean-width left unanswered. It is known that the boolean-width of a graph is smaller than its tree-width, branch-width and clique-width, but it is not clear how high the boolean-width can be as a function of its rank-width. Is boolean-width linear in rank-width, or subquadratic in rank-width, for every graph? It has been shown that a $k \times k$ grid has rank-width exactly $k-1$ [26]. We have seen that

its boolean-width lies between $\frac{1}{6}k$ (see Theorem 2) and $k+1$ (derived from the upper bound given by clique-width), but it would be nice to close this gap and find its exact value.

On the theoretical side it would be nice to improve on the $2^{2 \cdot boolw(G)}$-approximation to optimal boolean-width of Theorem 1 that applies the algorithm computing a decomposition tree of optimal rank-width of [22]. Note that the runtime of that approximation algorithm is FPT when parameterized by boolean-width of the input graph. The best we can hope for is an FPT algorithm computing optimal boolean-width, but any algorithm computing a decomposition tree of boolean-width polynomial in the optimal boolean-width would be nice. It seems such an algorithm will require some new techniques, as indicated by the tightness of Theorem 1 adressed in Lemma 3 and also the fact that *bool-dim* is not a submodular function [33]. The graphs of boolean-width at most one are exactly the graphs of rank-width at most one, *i.e.* the distance-hereditary graphs. What about the graphs of boolean-width at most $\log_2 3$, do they also have a nice characterization, and can they be recognized in polynomial time? More generally, is there an alternative characterization of the graphs of boolean-width at most $\log_2 k$ for any integer k, for example by a finite list of forbidden substructures, like minors for tree-width and vertex-minors for rank-width?

References

[1] I. Adler, B.-M. Bui-Xuan, Y. Rabinovich, G. Renault, J. A. Telle, and M. Vatshelle. On the boolean-width of a graph: structure and applications. In *Workshop on Graph-Theoretic Concepts in Computer Science (WG'10)*, volume 6410 of *LNCS*, pages 159–170, 2010.

[2] N. Alon and J. Spencer. *The probabilistic method*. Wiley-Interscience Series in Discrete Mathematics and Optimization, 2000.

[3] E. Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.

[4] R. Belmonte and M. Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *to appear in WG'2011.*

[5] H. Bodlaender and A. Koster. Treewidth Computations I Upper Bounds. Technical Report UU-CS-2008-032, Department of Information and Computing Sciences, Utrecht University, 2008.

[6] H. Bodlaender, E.-J. van Leeuwen, J. van Rooij, and M. Vatshelle. Faster algorithms on clique and branch decompositions. In *35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10)*, LNCS, pages 174–185, 2010.

[7] A. Brandstaedt and V. V. Lozin. On the linear structure and clique-width of bipartite permutation graphs. *Ars Combinatoria*, 67:719–734, 2003.

[8] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Feedback vertex set on graphs of low cliquewidth. In *20th International Workshop on Combinatorial Algorithms (IWOCA'09)*, volume 5874 of *LNCS*, pages 113–124, 2009.

[9] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discrete Applied Mathematics*, 158(7):809–819, 2010.

[10] P. Charbit, S Thomassé, and A. Yeo. The minimum feedback arc set problem is NP-hard for tournaments. *Combinatorics, Probability and Computing*, 16(1):1–4, 2007.

[11] D. Corneil and U. Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.

[12] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993.

[13] B. Courcelle, J. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[14] C. Damm, K. H. Kim, and F. W. Roush. On covering and rank problems for boolean matrices and their applications. In *5th Annual International Conference on Computing and Combinatorics (COCOON'99)*, volume 1627 of *LNCS*, pages 123–133, 1999.

[15] R. Downey and M. Fellows. *Parameterized Complexity*. Springer Verlag, 1999.

[16] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer Verlag, 2006.

[17] R. Ganian and P. Hliněný. On Parse Trees and Myhill-Nerode-type Tools for handling Graphs of Bounded Rank-width. *Discrete Applied Mathematics*, 158(7):851–867, 2010.

[18] J. Geelen, A. Gerards, and G. Whittle. Branch-width and well-quasi-ordering in matroids and graphs. *Journal of Combinatorial Theory, Series B*, 84(2):270–290, 2002.

[19] J. Goldman and G.-C. Rota. The number of subspaces of a vector space. *Recent Progress in Combinatorics*, pages 75–83, 1969.

[20] M. Golumbic and U. Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(3):423–443, 2000.

[21] M. Habib, C. Paul, and L. Viennot. Partition refinement techniques: An interesting algorithmic tool kit. *International Journal of Foundations of Computer Science*, 10(2):147–170, 1999.

[22] P. Hliněný and S. Oum. Finding branch-decompositions and rank-decompositions. *SIAM Journal on Computing*, 38(3):1012–1032, 2008. Abstract at *ESA'07*.

[23] P. Hliněný, S. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal*, 51(3):326–362, 2008.

[24] W.-L. Hsu. Decomposition of perfect graphs. *Journal of Combinatorial Theory, Series B*, 43(1):70–94, 1987.

[25] E.M. Hvidevold, S. Sharmin, J. A. Telle, and M. Vatshelle. Finding good decompositions for dynamic programming on dense graphs. `submitted. see http://www.ii.uib.no/~telle/bib/HSTV11.pdf`.

[26] V. Jelínek. The rank-width of the square grid. In *34rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'08)*, volume 5344 of *LNCS*, pages 230–239, 2008.

[27] Ö. Johansson. Clique-decomposition, NLC-decomposition and modular decomposition – Relatiohships and results for random graphs. *Congressus Numerantium*, 132:39–60, 1998.

[28] K. H. Kim. *Boolean matrix theory and its applications*. Marcel Dekker, 1982.

[29] T. Kloks and H. Bodlaender. Only few graphs have bounded treewidth. Technical Report UU-CS-92-35, Department of Information and Computing Sciences, Utrecht University, 1992.

[30] D. Kobler and U. Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2-3):197–221, 2003. Abstract at *SODA'01*.

[31] C. Lee, J. Lee, and S. Oum. Rank-width of random graphs. `submitted. see http://arxiv.org/abs/1001.0461`.

[32] H. X. Nguyen and P. Thiran. Active measurement for multiple link failures diagnosis in IP networks. In *5th Passive and Active Measurement Workshop*, volume 3015 of *LNCS*, pages 185–194, 2004.

[33] S. Oum. *private communication*.

[34] S. Oum. *Graphs of Bounded Rank-width*. PhD thesis, Princeton University, 2005.

[35] S. Oum. Rank-width is less than or equal to branch-width. *Journal of Graph Theory*, 57(3):239–244, 2008.

[36] S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

[37] P. Pattison and R. Breiger. Lattices and dimensional representations: matrix decompositions and ordering structures. *Social Networks*, 24(4):423–444, 2002.

[38] M. Rao. *Décomposition de graphes et algorithmes efficaces*. PhD thesis, Université Paul Verlaine, Metz, 2006.

[39] M. Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008.

[40] B. Reed. Tree-width and tangles: a new connectivity measure and some applications. In *Surveys in Combinatorics, Soc. Lecture Note Ser.* vol. 241*, Cambridge university Press*, pages 87–162. 1997.

[41] N. Robertson and P. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

[42] J. Rooij, H. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *17th Annual European Symposium on Algorithms (ESA'09)*, volume 5757 of *LNCS*, pages 566–577, 2009.

[43] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997.

[44] M. Zivkovic. Row space cardinalities. *Semigroup Forum*, 73(3):404–426, 2006.

Graph Classes with Structured Neighborhoods and Algorithmic Applications

Rémy Belmonte[a], Martin Vatshelle[a,*]

[a] *Department of Informatics, University of Bergen, P.O. Box 7803, N-5020 Bergen, Norway.*

Abstract

Given a graph in any of the following graph classes: trapezoid graphs, circular permutation graphs, convex graphs, Dilworth k graphs, k-polygon graphs, circular arc graphs and complements of k-degenerate graphs, we show how to compute decompositions with the number of d-neighborhoods bounded by a polynomial of the input size. Combined with results of Bui-Xuan, Telle and Vatshelle [1] this leads to polynomial time algorithms for a large class of locally checkable vertex subset and vertex partitioning problems on all of these graph classes. The boolean-width of a graph is related to the number of 1-neighbourhoods and our results imply that any of these graph classes have boolean-width $O(\log n)$.

1. Introduction

When solving graph problems by divide and conquer, we need to recursively divide the input graph G. A natural way to do this is to recursively partition the vertices of the graph in two parts. The resulting decomposition of G can be stored as a full binary tree whose leaves are in bijection with the n vertices of G, called *decomposition tree*. In a companion paper [1], Bui-Xuan, Telle and Vatshelle define for a given cut an equivalence relation on vertex subsets, namely d-neighborhood equivalence, for every fixed integer d. Further, they define nec_d as the maximum number of equivalence classes of the d-neighborhood equivalence relation over the cuts defined by a given

*Corresponding author. Tel: (+47) 55 58 42 00. Fax: (+47) 55 58 41 99.

Email addresses: remy.belmonte@uib.no (Rémy Belmonte), vatshelle@ii.uib.no (Martin Vatshelle)

decomposition tree. In this paper we give polynomial time algorithms for computing decomposition trees with nec_d polynomial in n for any graph belonging to one of the following classes: circular k-trapezoid graphs, k-polygon graphs, Dilworth k graphs, complement of k-degenerate graphs and convex graphs (see Group II of Figure 1). In [1], it is shown that given a graph G and a decomposition tree of G the large class of locally checkable vertex subset and vertex partitioning problems (LC-VSVP problems as defined in [2]) can be solved in time polynomial in n and nec_d. Combined with the results in this paper we get polynomial time algorithms solving any LC-VSVP problem on any graph class in Group I or II of Figure 1.

In a seminal paper by Courcelle, Makowski and Rotics [3], it was shown that every problem expressible in MSO_1 logic can be solved in linear time on graphs of bounded clique-width, i.e Group I of Figure 1. However, since e.g. the Maximum Clique problem is NP-hard on complements of planar graphs [4] (a subclass of co-5-degenerate graphs), we cannot expect to obtain such a strong result on all the graph classes in Group II of Figure 1. Instead our results imply polynomial time algorithms for the LC-VSVP problems, a subclass of the MSO_1 problems, via their relation to d-neighborhoods. This includes many problems related to independence, domination and homomorphism. Many of the LC-VSVP problems have been studied separately on many of the graph classes in Group II of Figure 1, see [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. A previous result closely related to ours is by Kratochvíl, Manuel and Miller in [16] who showed that a subset of the LC-VSVP problems is solvable in polynomial time on interval graphs.

The notion of d-neighborhood generalizes the classical notion of neighborhood, i.e. two subsets of a set A of vertices are 1-neighborhood equivalent with respect to the cut $\left(A, \overline{A}\right)$ if they have the same neighborhood in $\overline{A}$. Boolean-width of a graph G is a parameter introduced by Bui-Xuan, Telle and Vatshelle [17], defined as $\log_2(nec_1)$ of an optimal decomposition of G. In particular, they gave FPT algorithms for solving Maximum Weight Independent Set and Minimum Weight Dominating Set in $2^{O(k)} \cdot poly(n)$ time, assuming a decomposition tree of boolean-width k is given. As a corollary of the results in this paper, we get that all the classes in Group II of Figure 1 have boolean-width $O(\log n)$ and thus the two above algorithms are polynomial on these graph classes. To our knowledge, this is the first time an FPT algorithm is used to give polynomial time algorithms on a graph class where the parameter value is not bounded by a constant.

In the simple case of interval graphs and permutation graphs we show how

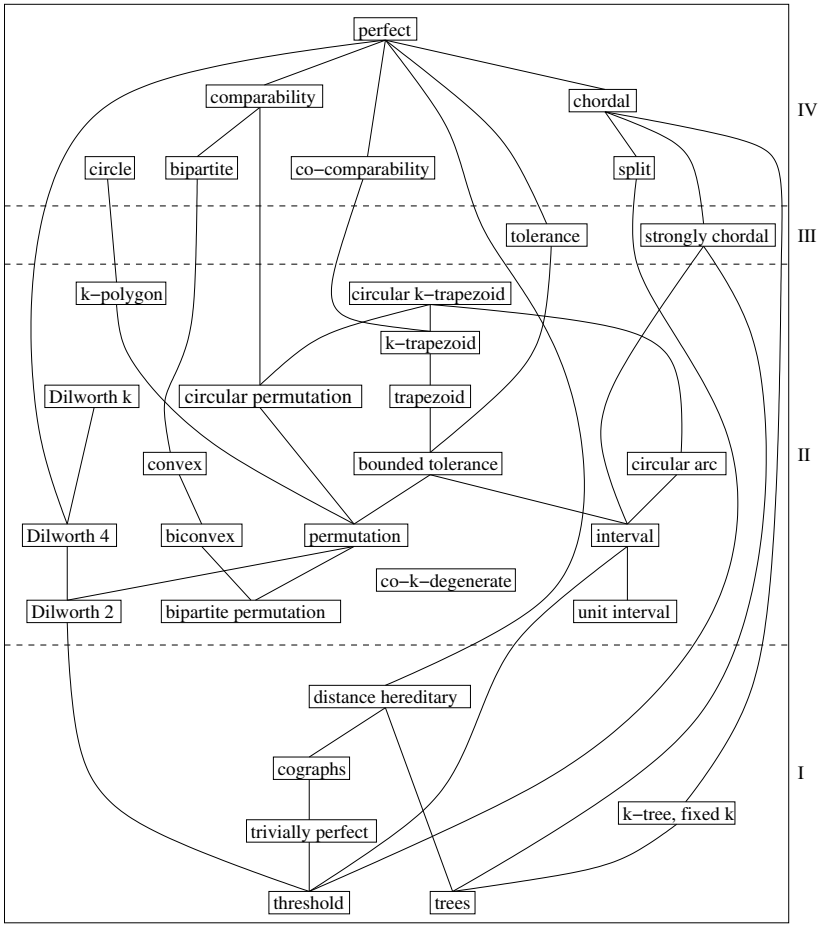


Figure 1: Inclusion diagram of some well-known graph classes.
(I) Graph classes where clique-width and boolean-width is bounded by a constant.
(II) Graph classes having decomposition trees with nec_d polynomial in n, boolean-width $O(\log n)$ and all LC-VSVP problems solvable in polynomial time (for k-trapezoid graphs assume an intersection model is given).
(III) It is unknown whether these classes have boolean-width $O(\log n)$.
(IV) Either boolean-width is not $O(\log n)$ or it is NP-hard to compute such decompositions.

to construct decompositions such that every cut defined by the decomposition has nested neighborhoods, i.e. the neighborhoods across the cut are totally ordered by inclusion. Bipartite graphs with nested neighborhoods are called chain graphs, and Yannakakis [18] showed that a bipartite graph is a chain graph if and only if the size of a maximum induced matching is 1. Twin free chain graphs, called Hsu graphs, are used in Hsu's generalized join [19]. We generalize the idea of a cut with nested neighborhoods by considering cuts inducing bipartite graphs with a constant size maximum induced matching. We use the fact that the size of a maximum induced matching across a cut bounds the number of d-neighborhood equivalence classes, to give polynomial upper bounds on the value of nec_d for all of the graph classes in Group II of Figure 1.

In Section 2, we start by introducing standard graph theoretic notions and terminology. We define d-neighborhoods and relate these to induced matchings. In section 3, we show that for interval graphs, circular arc graphs, permutation graphs, circular permutation graphs, trapezoid graphs, convex graphs, k-polygon graphs, Dilworth k graphs and complement of k-degenerate graphs we can in polynomial time compute decompositions where all cuts have the number of d-neighborhoods bounded by a polynomial of n. For k-trapezoid graphs and circular k-trapezoid graphs similar results are proven, but we must assume that an intersection model is given along with the input graph. In section 4, we show that for all the graph classes in Group II of Figure 1, except possibly Dilworth k graphs (for $k \geq 2$), our upper bounds are essentially tight. We do so by showing that these classes have rank-width $\Omega(\sqrt{n})$, which implies that none of these classes can have decompositions with nec_d at most $n^{o(1)}$. Moreover we show that for the graph classes in Group IV of Figure 1 we cannot find decompositions with nec_d bounded by a polynomial of n, unless $P = NP$. Finally in Section 5 we conclude and give some open problems.

2. Framework

All graphs considered in this paper are undirected, finite and simple. The *neighborhood* of a vertex u, denoted by $N(u)$, is the set of vertices u such that the edge (u, v) is in E and for a subset of vertices X we denote the union of the neighborhoods of vertices in X by $N(X) = \bigcup_{x \in X} N(x)$. Given a set $A \subseteq V$, we denote by $\overline{A}$ the *complement* of A in V, i.e. $V \setminus A$. We call a bipartition $\left(A, \overline{A}\right)$ of V a *cut* of G. We denote by $G[X]$ the subgraph of

G induced by X and $G[X, Y]$ the bipartite subgraph of G induced by those edges with one endpoint in X and the other in Y.

We now define formally the notion of decomposition tree. The choice of a decomposition tree greatly influences the running time of any algorithm using the decomposition tree. In order to choose the best decomposition tree, we evaluate a decomposition tree by using a cut function. The following formalism is referred to as *branch decomposition* of a cut function and is standard in graph and matroid theory, see e.g. [20, 21, 22].

Definition 1 (Decomposition tree). A decomposition tree of a graph G is a pair (T, δ) where T is a full binary tree (i.e. T rooted with every non-leaf having two children) and δ a bijection between the leaf set of T and the vertex set of G. For a node w of T let the subset of $V(G)$ in bijection δ with the leaves of the subtree of T rooted at w be denoted by V_w. We say the decomposition defines the cut $\left(V_w, \overline{V_w}\right)$.

Caterpillar decompositions are decompositions where the underlying tree is a path with one leaf added as neighbor of each internal node of the path. Many of our proofs will construct caterpillar decompositions. To describe a caterpillar decomposition of a graph G, we only give an ordering $v_1, \ldots, v_n$ of the vertices of G. To construct the caterpillar decomposition (T, δ) from an ordering, first construct a caterpillar T from a path $u_1, \ldots, u_n$ of length $|V|$. Then let δ be a mapping of v_1 to u_1, v_n to u_n and for all $i \in \{2, \ldots, n-1\}$, of v_i to a new leaf attached to u_i. Finally, subdivide any edge and root the decomposition at the newly added vertex.

One of the essential notions in [17] and [1] is that of a representative. For a cut $\left(A, \overline{A}\right)$ in a graph G a representative of a subset $X \subseteq A$ is a subset R having the same neighbors in $\overline{A}$ as X. Maximum induced matchings bound the maximal size of representatives needed to represent any neighborhood across a cut.

Definition 2 (Maximum induced matching over a cut). Let G be a graph, then we say a subset $M \subseteq V(G)$ of vertices is an induced matching in G if $G[M]$ is a disjoint union of edges. For $A \subseteq V(G)$ we define $mim(A)$ as the maximum number of edges in an induced matching in $G[A, \overline{A}]$.

To see the relation to representative note that no two subsets of $M \cap A$ has the same neighborhood in $\overline{A}$, hence all $2^{|M \cap A|}$ subsets have different neighborhoods in $\overline{A}$, and for the converse relation see Lemma 1. In the companion

paper [1] the notion of neighborhoods is generalized to d-neighborhoods, we now formally define the notion of d-neighborhood equivalence.

Definition 3 (d-neighborhood equivalence). Let d be a non-negative integer, G be a graph and $A \subseteq V(G)$. Two vertex subsets $X \subseteq A$ and $Y \subseteq A$ are d-neighbor equivalent with respect to A, denoted by $X \equiv_A^d Y$, if:

$$\forall v \in \overline{A} : \min(d, |X \cap N(v)|) = min(d, |Y \cap N(v)|).$$

Let $nec(\equiv_A^d)$ be the number of equivalence classes of $\equiv_A^d$. For (T, δ) a decomposition tree of G define $nec_d(T, \delta)$ as the maximum $nec(\equiv_{V_w}^d)$ and $nec(\equiv_{\overline{V_w}}^d)$ over all nodes w in $V(T)$. We refer to $nec(\equiv_A^d)$ as the number of d-neighborhoods of the cut $\left(A, \overline{A}\right)$ and to $nec_d(T, \delta)$ as the number of d-neighborhoods of (T, δ).

Definition 4 (Boolean-width). Let G be a graph and (T, δ) a decomposition of G. The boolean-width of (T, δ), denoted by $boolw\,(T, \delta)$, is defined as the maximum over all $w \in V(T)$ of $\log_2(nec\left(\equiv_{V_w}^1\right))$. The boolean-width of G, denoted by $boolw\,(G)$, is defined as the minimum $boolw\,(T, \delta)$ over all decompositions (T, δ) of G.

As a corollary of our results, we give logarithmic bounds on the boolean-width of the graph classes in Group II of Figure 1. Moreover, in Section 4 we use boolean-width to provide lower bounds on the value of nec_d on these graph classes.

In Section 3 we will show how to compute decompositions where the number of d-neighborhoods is polynomial in the size of the graph. To do this we will use the following connection between the number of d-neighborhoods and maximum induced matchings.

Lemma 1. *Let G be any graph and $A \subseteq V(G)$ any subset of the vertices. $mim\,(A) \leq k$ if and only if for every $S \subseteq A$, there is $R \subseteq S$ such that $N\,(R) \cap \overline{A} = N\,(S) \cap \overline{A}$ and $|R| \leq k$.*

Proof. First we prove $mim\,(A) \leq k \Rightarrow |R| \leq k$. Assume for contradiction that R is a minimal set such that $N\,(R) \cap \overline{A} = N\,(S) \cap \overline{A}$ and that $|R| > k$. Then for every $x \in R$ we know that $N\,(R \setminus x) \cap \overline{A} \neq N\,(R) \cap \overline{A}$. Let $y \in N\,(R) \cap \overline{A} \setminus N\,(R \setminus x) \cap \overline{A}$ be any private neighbour of x. Each such pair is an edge, i.e. $(x, y) \in E(G[A, \overline{A}])$, Let M contain all these vertices. Since

each y was a private neighbour of x we know that $N(y) \cap R = \{x\}$. Since we chose exactly one y for each $x \in R$ we know that $|M \cap \overline{A}| \leq |R|$, since each $x \in R$ has at least one neighbour in $M \cap \overline{A}$ we know that $G[M \cap A, M \cap \overline{A}]$ is 1-regular and hence$mim\,(A) > k$ leading to a contradiction.

For the other direction we will prove the converse namely, if $mim\,(A) > k \Rightarrow |R| > k$. Let M be an maximum induced matching over the cut $\left(A, \overline{A}\right)$, and $S = M \cap A$, then by assumption $|S| > k$, now we only need to show that for every strict subset $R \subset S$ we have $N\,(S) \cap \overline{A} \neq N\,(R) \cap \overline{A}$. Since M is an induced matching, there will be an edge $(u, v) \in M$ with $u \in S \setminus R$, but then by definition of an induced matching v has no neighbor in R and hence $N\,(R) \cap \overline{A} \neq N\,(S) \cap \overline{A}$. □

Lemma 2. *Let G be any graph and $A \subseteq V(G)$ any subset of vertices. Then $nec\left(\equiv_A^d\right) \leq n^{d \cdot mim(A)}$.*

Proof. First we prove the following:

Claim. For every subset $S \subseteq A$, there exist $R \subseteq S$ such that $R \equiv_A^d S$ and $|R| \leq mim\,(A) \cdot d$.

Proof of the claim. This proof is by induction on d and similar to that of [1, Lemma 5]. For $d = 1$, this follows from Lemma 1 by using $k = mim\,(A)$. Now, assume the induction hypothesis true up to $d-1$, then we show it true for d. Let $S' \subseteq S$ be an inclusion minimal set such that $N(S') \cap \overline{A} = N(S) \cap \overline{A}$ i.e. $S' \equiv_A^1 S$. Hence from Lemma 1 we have that $|S'| \leq mim\,(A)$. By induction hypothesis there exists $R' \subseteq (S \setminus S')$ such that $R' \equiv_A^{d-1} (S \setminus S')$ and $|R'| \leq mim\,(A) \cdot (d-1)$. Thus it is enough to show $R = R' \cup S' \equiv_A^d S$. We partition the nodes of $\overline{A}$ into (P, Q) such that $\forall v \in P$, we have $|N(v) \cap (S \setminus S')| = |N(v) \cap R'|$ and $\forall v \in Q$, we have $|N(v) \cap (S \setminus S')| \geq d-1$ and $|N(v) \cap R'| \geq d-1$. For every vertex $v \in P$, since $S \cap R' = \emptyset$ and $S' \subseteq S$, we know $|N(v) \cap S| = |N(v) \cap (S \setminus S')| + |N(v) \cap S'| = |N(v) \cap R'| + |N(v) \cap S'| = |N(v) \cap R|$. We have $N(S) = N(S')$ and since $d > 1$ we have $Q \subseteq N(S')$. For every vertex $v \in Q$, since $|N(v) \cap (S \setminus S')| \geq d-1$ we get $|N(v) \cap S| \geq d$ and since $|N(v) \cap R'| \geq d-1$ we get $|N(v) \cap R| \geq d$. Since (P, Q) is a partition we get $R \equiv_A^d S$ and $|R| \leq mim\,(A) \cdot d$, thus by induction the claim holds.

To bound the number of equivalence classes $nec\left(\equiv_A^d\right)$ we know from the claim that we only need to find the equivalence classes among the subsets of A of size at most $d \cdot mim\,(A)$. A trivial bound on number of subsets of A with size $d \cdot mim\,(A)$ gives us: $nec\left(\equiv_A^d\right) \leq |A|^{d \cdot mim(A)} \leq n^{d \cdot mim(A)}$. □

3. Finding Good Decompositions on Restricted Graph Classes

In this section we will show how to compute decomposition trees with $nec_d(T, \delta)$ bounded by a polynomial of n if the graph belongs to a certain graph-class. In almost all proofs, we construct a caterpillar decomposition by giving an ordering of the vertices of the graph, and then argue using Lemma 1 that for each cut of the decomposition the size of a maximum induced matching is bounded, and thus we can apply Lemma 2.

For all graph classes considered in this paper, except for the complements of k-degenerate graphs and Dilworth k graphs, we use definitions via a geometrical intersection model.

Definition 5 (Intersection Model). Let $\mathcal{F}$ be a family of nonempty sets. For a graph G, we say $\mathcal{F}$ is an intersection model of G if there exists a bijection φ from F to $V(G)$ such that two vertices $u, v \in V(G)$ are adjacent if and only if $\varphi(u)$ and $\varphi(v)$ intersect.

The sets in the intersection model usually consists of geometrical objects such as lines, circles or polygons.

3.1. Interval Graphs

An interval $I = \langle i, j \rangle$ is represented by an ordered pair of real numbers with $i < j$ and represent the set of real numbers $\{x : i < x < j\}$. Let $I_1 = (a, b)$ and $I_2 = (c, d)$ be two intervals, then I_1 intersects I_2 if and only if $a < d$ and $c < b$.

Definition 6 (Interval graph). A graph is an interval graph if it has an intersection model consisting of intervals.

Lemma 3. *Given an interval graph G and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^d$.*

Proof. Any interval graph has an intersection model where no interval starts or ends at the same point and we can find such an intersection model in linear time [23]. We build a caterpillar decomposition by sorting the vertices by the left endpoint of their corresponding intervals. Let us now consider any cut $(A, \overline{A})$ defined by the decomposition. We want to bound $mim(A)$ by applying Lemma 1. Thus we will show that for every set $S \subseteq A$, there is a set

R such that $N(R) \cap \overline{A} = N(S) \cap \overline{A}$ and $|R| \leq 1$. Let σ be the total ordering of the vertices of A sorted by their right endpoint. Since all left endpoints of intervals corresponding to vertices of A are to the left of all left endpoints of intervals corresponding to vertices of $\overline{A}$, two vertices $u \in A, u' \in \overline{A}$ are neighbors if and only if the right endpoint of u is to the right of the left endpoint of u'. Hence, for every pair of vertices $u, v \in A$ if $\sigma(u) \leq \sigma(v)$ then $N(u) \cap \overline{A} \subseteq N(v) \cap \overline{A}$. For every set $S \subseteq A$, let R contain the unique vertex of S whose interval has the rightmost right endpoint. We then have $N(R) \cap \overline{A} = N(S) \cap \overline{A}$ and $|R| \leq 1$. Therefore, by Lemma 1, $mim(A) \leq 1$ for all cuts. Then for any d, by Lemma 2 we have $nec_d(T, \delta) \leq n^d$. □

3.2. Circular-arc graphs

Circular-arc graphs is a natural generalization of interval graphs.

Definition 7 (Circular arc graph). A graph is a circular arc graph if it has an intersection model consisting of arcs of a circle.

Lemma 4. *Given a circular-arc graph G and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^{2d}$.*

Proof. We can compute the circular-arc intersection model of G in linear time [24]. Fix an arbitrary point p on the circle. We define the distance of an arc from p as follows: If the arc contains p, then the distance is 0, otherwise it is the minimum distance between p and any point of the arc. For any vertex u, we denote by arc_u the arc corresponding to u.

Build a caterpillar decomposition by totally ordering the vertices in order of increasing distance of their associated arc from p, breaking ties arbitrarily. Note that this decomposition can be computed in polynomial time. We now consider any cut $(A, \overline{A})$ of this decomposition. By construction, for every $x \in A, y \in \overline{A}$, the distance of arc_x from p is less than or equal to the distance of arc_y from p.

Now, we prove that for any set $S \subseteq A$, there exists a subset $S' \subseteq S$ such that $|S'| \leq 2$ and $N(S) \cap \overline{A} = N(S') \cap \overline{A}$. Let arc_l be the arc extending the furthest from p in clockwise direction and arc_r the arc extending the furthest from p in counter-clockwise direction and $S' = \{l, r\}$. In Figure 3.2 we get $arc_l = arc_a$ and $arc_r = arc_e$. Assume for contradiction that there exist $v \in \overline{A}$ such that $v \in N(S) \setminus N(S')$. Since arc_v does not intersect arc_l

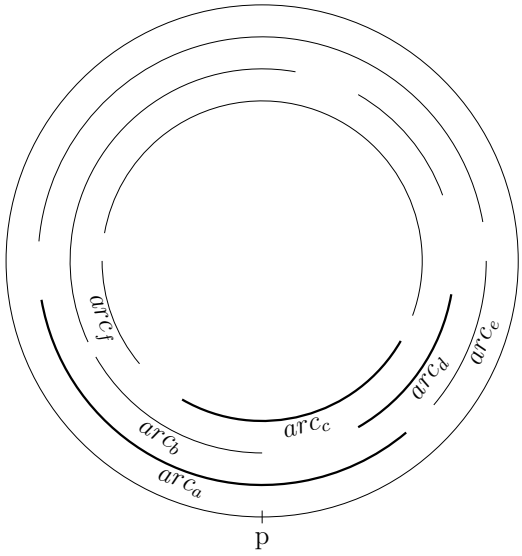


Figure 2: An intersection model of an circular-arc graph with 10 vertices. Let $A = \{a, b, c, d, e, f\}$. Note that A forms a cut of the decomposition since no other arcs are closer to p. Let $S = \{a, c, d\}$. Note the arcs of S are drawn in bold. We then get $arc_l = arc_a$ and $arc_r = arc_d$, thus $S' = \{a, d\} \equiv_A S$.

or arc_r and the distance of arc_v is larger than both arc_l and arc_r. The arc corresponding to the vertex in X overlapping with arc_v must extend further than either arc_l or arc_r leading to a contradiction.

For every set $S \subseteq A$, let R contain the vertices l and r as defined above. We then have $N(R) \cap \overline{A} = N(S) \cap \overline{A}$ and $|R| \leq 2$. Therefore, by Lemma 1, $mim(A) \leq 2$ for all cuts. Then for any d, by Lemma 2 we have $nec_d(T, \delta) \leq n^{2d}$. □

3.3. Permutation Graphs

Definition 8 (Permutation graph). Let L and U be two distinct infinite parallel lines. A graph is a permutation graph if it has an intersection model consisting of straight line-segments with one endpoint on L and one on U.

Lemma 5. *Given a permutation graph G and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^d$.*

Proof. We compute the permutation model of G in linear time [25]. We build a caterpillar decomposition by sorting the vertices by the upper endpoint of their corresponding line. Let us now consider a cut $(A, \overline{A})$ of the decomposition. Let σ be the total ordering of the vertices of A sorted by their lower endpoint, hence $\forall u, v \in A, \sigma(u) \leq \sigma(v)$ iff the lower endpoint of u is to the left of the lower endpoint of v. Since all upper endpoints of lines corresponding to vertices of A are to the left of all upper endpoints of lines corresponding to vertices of $\overline{A}$, two vertices $u \in A, u' \in \overline{A}$ are neighbors if and only if the lower endpoint of u is to the right of the lower endpoint of u'.

Hence for any set $S \subseteq A$ there exists $x \in S$ such that $N(S) \cap \overline{A} = N(x) \cap \overline{A}$, namely the vertex of S with the rightmost lower endpoint, by Lemma 1, $mim(A) \leq 1$ for all cuts. Then for any d, by Lemma 2 we have $nec_d(T, \delta) \leq n^d$. □

3.4. Circular Permutation Graphs

Definition 9 (Circular permutation graph). Let L and U be two parallel different circles on a cylinder. A graph is a circular permutation graph if it has an intersection model consisting of straight line-segments with one endpoint on L and one on U.

Lemma 6. *Given a circular permutation graph graph G and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^{2d}$.*

Proof. We compute the circular permutation model of G in linear time [26]. Let s_v be the line corresponding to the vertex v. We build a caterpillar decomposition using an ordering obtained by sorting the vertices by the endpoint on L of their corresponding lines in clockwise order starting from any point p. Let us now consider a cut $(A, \overline{A})$ of the decomposition. For any $S \subseteq A$ we show that we can find $S' \subseteq S$ such that $N(S) \cap \overline{A} = N(S') \cap \overline{A}$ and $|S'| \leq 2$. Let l (resp. r) be the line corresponding to the vertex $v \in S$ that extend the furthest from p in clockwise (resp. counter-clockwise) direction.

Let $S' = \{l, r\}$ and assume for contradiction that there exist $v \in \overline{A}$ such that $v \in N(S) \setminus N(S')$. Since s_v does not intersect s_l nor s_r and that the distance from p to the point of s_v on L is greater in clockwise (resp.

counter-clockwise) direction than the point of s_l (resp. s_r) on L we have that the distance from p to the point of s_v on U is greater in clockwise (resp. counter-clockwise) direction than the point of s_l (resp. s_r) on U.

For every set $S \subseteq A$, let R contain the vertices l and r as defined above. We then have $N(R) \cap \overline{A} = N(S) \cap \overline{A}$ and $|R| \leq 2$. Therefore, by Lemma 1, $mim(A) \leq 2$ for all cuts. Then for any d, by Lemma 2 we have $nec_d(T, \delta) \leq n^{2d}$. □

3.5. Trapezoid graphs

Let L and U be two infinite parallel lines. Let A and B be two non-crossing straight line-segments with one endpoint on L and the other on U. The finite area defined by the four lines L, U, A, B is called a trapezoid between L and U. Trapezoid graphs is a generalization of permutation graphs, i.e. using $A = B$ and a generalization of interval graphs, i.e. using $L = U$.

Definition 10 (Trapezoid graph). A graph is a trapezoid graph if it has an intersection model consisting of trapezoids between two parallel lines.

Lemma 7. *Given a trapezoid graph G and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^{2d}$.*

Proof. We compute the trapezoid intersection model of G in $O(n^2)$ time [27]. We build a caterpillar decomposition by sorting the vertices by the upper right corner of their corresponding trapezoid from left to right. Let us now consider a cut $(A, \overline{A})$ of the decomposition. We show that for any $S \subseteq A$, we can find a set $S' \subseteq S$ with $N(S) \cap \overline{A} = N(S') \cap \overline{A}$ and $|S'| \leq 2$: For the upper line (resp. lower), we take the the trapezoid u (resp. l) with the rightmost upper (resp. lower) right corner, we then set $S' = \{u, l\}$. Let us assume for contradiction that $\exists x \in \overline{A} : x \in N(S) \setminus N(S')$. The trapezoid of x must intersect some trapezoid of S on the upper or lower line. If it does not intersect u or l, then the whole trapzeoid of x is to the right of u and l. By construction of the decomposition, x would have been in A, thus $N(S) \cap \overline{A} = N(S') \cap \overline{A}$. For every set $S \subseteq A$, let R contain the vertices u and l as defined above. We then have $N(R) \cap \overline{A} = N(S) \cap \overline{A}$ and $|R| \leq 2$. Therefore, by Lemma 1, $mim(A) \leq 2$ for all cuts. Then for any d, by Lemma 2 we have $nec_d(T, \delta) \leq n^{2d}$. □

3.6. k-trapezoid graphs

k-trapezoid graphs are a natural generalization of trapezoid graphs and interval graphs in the sense that the 2-trapezoid graphs are exactly the trapezoid graphs and 1-trapezoid graphs are exactly interval graphs.

Definition 11 (k-trapezoid graphs). Let $L_1, \ldots, L_k$ be k parallel lines. In order to build a k-trapezoid, first choose two points s_i and e_i on each line such that $s_i < e_i$. Then, make two non-intersecting paths s and e by joining s_i to s_{i+1} and e_i to e_{i+1} respectively by straight lines for each $i \in \{1, \ldots, k-1\}$. A k-trapezoid is the polygon defined by s, e and the lines going from s_1 to e_1 and s_k to e_k in clockwise direction. A k-trapezoid graph is the intersection graph of k-trapezoids.

Note that k-trapezoid graphs are equivalent to comparability graphs of partial orders of interval dimension k [28]. Moreover, Yannakakis showed in [29] that deciding if a partial order of height 1 has dimension at most 3 is NP-complete. Therefore, recognizing k-trapezoid graphs for any fixed $k \geq 3$ is NP-complete. Additionally, by [30] the smallest integer k such that a given graph G is a k-trapezoid graph cannot be approximated within a factor better than $\sqrt{n}$.

Lemma 8. *Given a k-trapezoid graph G together with a k-trapezoid model of G and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^{kd}$.*

Proof. We build a caterpillar decomposition by sorting the vertices by the rightmost corner of their corresponding k-trapezoid. Let us now consider a cut $(A, \overline{A})$ of the decomposition. We show that for any $S \subseteq A$, we can find a set $S' \subseteq S$ with $N(S) \cap \overline{A} = N(S') \cap \overline{A}$ and $|S'| \leq k$: For each line i, we take the the k-trapezoid r_i, corresponding to a vertex of S, which contains the rightmost point on L_i. We set S' as the set of all r_i. Let us assume for contradiction that $\exists v \in \overline{A} : v \in N(S) \setminus N(S')$, let t_v be the trapezoid corresponding to v. Since v is in $\overline{A}$ there must exist some x such that t_v contains a point to the right of r_x on L_x. Also ther must exist some y, such that t_v intersects some trapezoid of S on L_y hence t_v contains a point to the left of the rightmost point of r_y on L_y. Since both t_v and r_y are contineous they have to intersect at some point between L_x and L_y. Thus $N(S) \cap \overline{A} = N(S') \cap \overline{A}$ hence $mim(A) \leq k$ then by Lemma 2 the lemma holds.

For every set $S \subseteq A$, let R contain all the vertices r_i as defined above. We then have $N(R) \cap \overline{A} = N(S) \cap \overline{A}$ and $|R| \leq k$. Therefore, by Lemma 1, $mim(A) \leq k$ for all cuts. Then for any d, by Lemma 2 we have $nec_d(T, \delta) \leq n^{kd}$. □

It is an interesting open problem whether one, given a k-trapezoid graph can build a decomposition tree having $nec_d(T, \delta) = n^{O(k)}$.

3.7. Circular k-trapezoid Graphs

Circular k-trapezoid graphs form a natural extension of the k-trapezoid graphs introduced in [31], see [32].

Definition 12 (Circular k-trapezoid graph). Let $C_1, \ldots, C_k$ be k circles on the surface of a cylinder, all orthogonal to its axis. In order to build a circular k-trapezoid, first choose two points s_i and e_i on each line. Then, make two non-intersecting paths s and e by joining s_i to s_{i+1} and e_i to e_{i+1} respectively by straight lines for each $i \in \{1, \ldots, k-1\}$. A circular k-trapezoid is the polygon defined by s, e and the arcs going from s_1 to e_1 and s_k to e_k in clockwise direction. A circular k-trapezoid graph is the intersection graph of circular k-trapezoids.

Lemma 9. *Given a circular k-trapezoid graph G and a circular k-trapezoid model and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^{2kd}$.*

Proof. Let p be an arbitrary point on C_1. We define the distance of a k-trapezoid from p as the minimum distance between p and any point of the arc of the k-trapezoid on C_1. For any vertex u, we denote by t_u the k-trapezoid corresponding to u and $arc_{u,i}$ the arc of C_i contained in t_u. Build a caterpillar decomposition by adding the vertices in order of increasing distance of their associated k-trapezoid from p, breaking ties arbitrarily. We now consider any cut $(A, \overline{A})$ of this decomposition.

By construction, for every $x \in A, y \in \overline{A}$, the distance of t_x from p is less than or equal to the distance of t_y from p. Now, we prove that for any set $S \subseteq A$, there exists a subset $S' \subseteq S$ such that $|S'| \leq 2 \cdot k$ and $S \equiv_A S'$. Let $r(S, i)$(resp. $l(S, i)$) be the vertex $v \in S$ such that t_v containins the extremal point of L_i in clockwise (respectively counter-clockwise) direction. Let $S' = \bigcup_{i \leq k} \{r(S, i), l(S, i)\}$, assume for contradiction that there $\exists v \in \overline{A}$: $v \in N(S) \setminus N(S')$. By construction of the decomposition $arc_{v,1}$ must contain

a point more extreme than the points on $arc_{l(S,1),1}$ and $arc_{r(S,1),1}$. Since $v \in n(S)$ there must exist a j such that, without loss of generality, $arc_{r(S,j),j}$ contains a more extreme point than the least extreme point of $arc_{v,j}$, but then $t_{r(S,j)}$ contains both a point less extreme and more extreme than t_v, hence they must intersect.

For every set $S \subseteq A$, let R contain all the vertices $r(S,i)$ and $l(S,i)$ as defined above. We then have $N(R) \cap \overline{A} = N(S) \cap \overline{A}$ and $|R| \leq 2k$. Therefore, by Lemma 1, $mim(A) \leq 2k$ for all cuts. Then for any d, by Lemma 2 we have $nec_d(T,\delta) \leq n^{2kd}$. □

3.8. Convex Graphs

Definition 13 (Convex graph). A graph $G = (V,E)$ is convex if G is bipartite with color classes X and Y and an ordering $x_1, \ldots, x_{|X|}$ of X such that for every vertex $u \in Y$ and $x_i, x_j \in N(u)$, we have for every vertex $x_t \in X$ that if $i < t < j$ then $x_t \in N(u)$, i.e. the vertices in $N[u]$ are consecutive in the ordering of X.

Lemma 10. *Given a convex graph G and any positive integer d, we can, in polynomial time, compute a decomposition tree (T,δ) of G having $nec_d(T,\delta) \leq n^d$.*

Proof. Since G is convex we can in polynomial time find a bipartition (X,Y) of V and σ_X an ordering of X such that for every vertex $u \in Y$ and $x, y \in N(u)$ [23]. Hence we have for every vertex $z \in X$ that if $\sigma_X(x) < \sigma_X(z) < \sigma_X(y)$ then $z \in N(u)$. We construct a total ordering σ of V from σ_X by keeping the ordering of vertices in X and for each vertex $v \in Y$ we insert v immediately after the last element of $N(v)$. We construct a caterpillar decomposition from the order σ.

Let us now consider a cut $(A, \overline{A})$ of the decomposition. We want to prove that for any subset S of A, there exists a set $S' \subseteq S$ such that $S' \equiv_A S$ and $|S'| \leq 1$. Note that by construction of σ, we have $\forall v \in Y \cap A, N(v) \cap \overline{A} = \emptyset$, hence we can assume $S' \subseteq X \cap S$.

Let $v_1, v_2, \ldots, v_t$ be the ordering of the vertices of $X \cap S$ induced by σ. Since all the vertices in $Y \cap \overline{A}$ appear later in σ than v_t, then we have for every vertex $v \in Y \cap \overline{A}$, either $v_t \in N(v)$ or $N(v) \cap S = \emptyset$. Moreover, note that $N(A \cap X) \cap \overline{A} = \emptyset$. Hence $N(\{v_t\}) \cap \overline{A} = N(S) \cap \overline{A}$.

For every set $S \subseteq A$, let R contain the vertex v_t as defined above. We then have $N(R) \cap \overline{A} = N(S) \cap \overline{A}$ and $|R| \leq 1$. Therefore, by Lemma 1,

$mim(A) \leq 1$ for all cuts. Then for any d, by Lemma 2 we have $nec_d(T, \delta) \leq n^d$. □

3.9. k-polygon graphs

Definition 14 (k-polygon graph). A k-polygon graph is the intersection graph of chords (straight lines between two points on distinct sides) of a convex k sided polygon.

Lemma 11. *Given a k-polygon graph G and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^{2kd}$.*

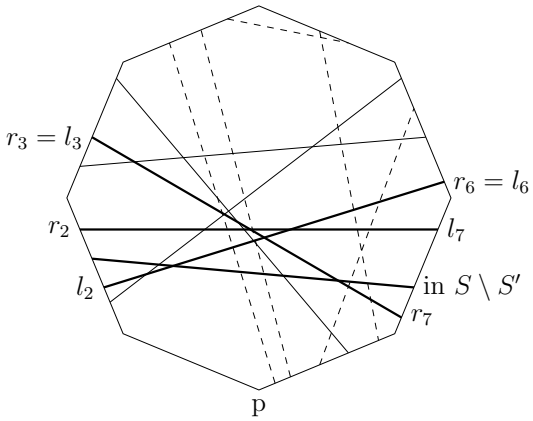


Figure 3: A 8-polygon graph with 12 vertices, the vertices in S are represented by bold lines, those in $A \setminus S$ by thin lines and those in $\overline{A}$ by dashed lines.

Proof. We compute the k-polygon intersection model of G in $O(4^k \cdot n^2)$ time [33]. Let p be an arbitrary corner of the k-polygon. We measure the distance of a point from p as the distance around the edge of the k-polygon in clockwise direction. We define the distance of a chord from p as the minimum distance of any endpoint of the chord from p. We build a caterpillar decomposition of G by ordering the vertices of G by increasing distance from p of their corresponding chords.

Consider any cut $(A, \overline{A})$ of the decomposition. We prove that for any set $S \subseteq A$, there exists a subset $S' \subseteq S$ such that $|S'| \leq 2k$ and $S' \equiv_A S$. We denote by t the maximum distance from p to a chord of any vertex in A. We can observe that, by construction of the decomposition, for every vertex u in A, if both endpoints of the chord corresponding to u are at distance at most t from p, then $N(u) \cap \overline{A} = \emptyset$. Now, we associate with each side of the k-polygon with an index $i \in \{1, ..., k\}$ ordered in clockwise direction starting from p. For each side we define $S_i \subseteq S$ as the set of vertices of which line has an endpoint on side i. Each vertex of G belongs to exactly 2 such sets. We also define for each side i such that $S_i \neq \emptyset$, the point l_i on side i as the endpoint of a coord corresponding to a vertex in S_i closest to p, likewise r_i is the endpoint of a coord on side i corresponding to a vertex in S_i furthest from p.

Let $S' = \bigcup_{i \leq k} \{l_i, r_i\}$, we claim that $S' \equiv_A S$. Let us assume for contradiction that there exists a vertex $x \in (N(S) \setminus N(S')) \cap \overline{A}$. Let c_x be the coord corresponding to x, p_a and p_b be the endpoints of c_x such that $t < p_a < p_b$.

Let $y \in S$ be a neighbor of x and c_y the coord corresponding to y. Let p_y be the endpoint of c_y such that $t < p_y$ and j the index of the side containing p_y, then we know since y is a neighbor of x that $p_a < p_y < p_b$. We also know by definition that $l_j < p_y < r_y$ Since no coord can have both endpoints on the same side we have either $a < j$ or $j < b$, if $a < j$ then $p_a < l_j < p_y < p_b$ hence c_x intersects the coord ending at l_j, likewise we can argue if $j < b$ leading to a contradiction.

For every set $S \subseteq A$, let R contain all the vertices l_i and r_i as defined above. We then have $N(R) \cap \overline{A} = N(S) \cap \overline{A}$ and $|R| \leq 2k$. Therefore, by Lemma 1, $mim(A) \leq 2k$ for all cuts. Then for any d, by Lemma 2 we have $nec_d(T, \delta) \leq n^{2kd}$. □

3.10. Dilworth k Graphs

Definition 15 (Dilworth k graph). Two vertices x and y are said to be comparable if either $N(y) \subseteq N[x]$ or $N(x) \subseteq N[y]$. The Dilworth number of a graph is the largest number of pairwise incomparable vertices of the graph. A graph is a Dilworth k graph if it has Dilworth number k.

Dilworth k graphs can be recognized in $O(k^2 \cdot n^2)$ time [34].

Lemma 12. *Given a Dilworth k graph G and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^{kd}$.*

Proof. Let us consider any cut $\left(A, \overline{A}\right)$ of G and $S \subseteq A$. We want to prove that there exist S' such that $|S'| \leq k$ and $N(S') \cap \overline{A} = N(S) \cap \overline{A}$. Let S' be an inclusion minimal subset of S such that $N(S') \cap \overline{A} = N(S) \cap \overline{A}$. If S' contains two vertices x and y such that $N(y) \subseteq N[x]$, then $S' \setminus \{y\}$ contradicts the minimality of S'. Since V cannot contain more than k pairwise incomparable vertices, $|S'| <= k$. Therefore $mim(A) \leq k$ and by applying Lemma 2 the lemma follows. □

3.11. Complements of k Degenerate Graphs

Definition 16 (k degenerate graph). A graph G is k-degenerate if there exists an elimination ordering $v_1, \ldots, v_n$ of the vertices of G such that $\forall i \in \{1, \ldots, n\}, |\{v_j : j > i \text{ and } v_j \in N(v_i)\}| \leq k$.

Lemma 13. *Given a graph G that is the complement of a k-degenerate graph, and any positive integer d, we can, in polynomial time, compute a decomposition tree (T, δ) of G having $nec_d(T, \delta) \leq n^d \cdot 2^{kd}$.*

Proof. We build a caterpillar decomposition of G using the elimination ordering induced by the k-degeneracy of $\overline{G}$. We consider a cut $(A, \overline{A})$ of the decomposition.

Note first that since $\overline{G}$ is k-degenerate, every vertex of A has at most k neighbors in $\overline{A}$. Therefore, in G every vertex of A has at least $|\overline{A}| - k$ neighbors in $\overline{A}$. Moreover every $S \subseteq A$ with size d sees at least $|\overline{A}| - kd$ vertices d times. There is at most n^d ways to choose S. For each such choice there is at most kd missing edges, hence at most 2^{kd} non-equivalent subsets containing S. □

3.12. Boolean-width

We say that a class of graphs $\mathcal{C}$ has boolean-width $O(f(n))$ if for every G in $\mathcal{C}$ we have $boolw(G) \in O(f(n))$. We now restate the results in this section in terms of boolean-width, using the fact that, for any graph G and any decomposition (T, δ) of G, we have $boolw(G) \leq boolw(T, \delta) = \log_2 nec_1(T, \delta) \leq \log_2 nec_d(T, \delta)$. Combining this fact with the results in this section we get the following.

Corollary 14. *The following graph classes all have boolean-width $O(\log n)$: interval graphs, circular arc graphs, permutation graphs, circular permutation graphs, trapezoid graphs, k-trapezoid graphs, circular k-trapezoid graphs, convex graphs, k-polygon graphs, Dilworth k graphs and complements of k-degenerate graphs.*

4. Lower bounds

In this section we provide lower bounds for $nec_d(T,\delta)$ of optimal decompositions for various classes of graphs. We do so by providing lower bounds for the boolean-width of these classes. Note that a lower bound on $boolw\,(T,\delta)$ also implies the same lower bound for $nec_1(T,\delta)$, which in turn give the same lower bound for $nec_d(T,\delta)$ for $d \geq 2$. However, the converse of this statement is not true for values of $d \geq 2$. We show that the upper bounds we gave in Section 4 are tight in two senses. We say that a class of graphs $\mathcal{C}$ has boolean-width $\Omega(f(n))$ if for any integers k and n there exists a graph $G \in \mathcal{C}$ with $|V(G) \geq n|$ having boolean-width larger than $k \times f(|V(G)|)$. Firstly, we show that all graph classes (except possibly Dilworth k graphs) in Group II of Figure 1 have boolean-width $\Omega(\log n)$. Secondly, we show that for all graph classes in Group IV of Figure 1, it is highly unlikely that they have boolean-width $O(\log n)$. We use the following result on the relation between boolean-width and some other width parameters.

Theorem 15 (Bui-Xuan, Telle, Vatshelle [17, 1])**.** *For any graph G and any decomposition (T,δ) of G, it holds that $\log rw\,(T,\delta) - 1 \leq \log cw(T,\delta) - 1 \leq boolw\,(T,\delta) \leq \log nec_d(T,\delta)$, where $boolw\,(T,\delta)\,, rw\,(T,\delta)$ and $cw(T,\delta)$ denote respectively the boolean-width, rank-width and clique-width of (T,δ).*

Hence if a graph class has rank-width or clique-width $\Omega(n^c)$ for some constant $c > 0$, then this graph class also has boolean-width $\Omega(\log n)$. We now give a lower bound for the rank-width of proper interval, bipartite permutation graphs and complement of grids, which implies the desired $\Omega(\log n)$ lower bound for boolean-width.

We call *Hsu-graph* a bipartite graph $H = (V,E)$ with $V = \{v_1, v_2, \ldots v_a\}$, $\{u_1, u_2, \ldots u_b\}$ and $v_i, u_j \in E(H)$ if and only if $i \leq j$. A *Hsu-join-chain* of length q and width p is constructed as follows. Let $\mathcal{F} = G_1, G_2, \ldots, G_q$ be a family of graphs, all on at least p vertices. For $j \in \{i, i+1\}$, let $S_j \subseteq V(G_j)$, $|S_j| = p$ and σ_j an ordering of S_j. Then, for every $1 \leq i \leq q-1$, let $G[S_i \cup S_{i+1}]$ be isomorphic to a Hsu-graph where we identify $\sigma_j(r)$ with v_r and $\sigma_{j+1}(r)$ with u_r.

Lemma 16. *If G is a HSU-join-chain of length q and width p where $q > 3p$ then $rw\,(G) \geq p/2$.*

Proof. Let $\mathcal{F} = G_1, G_2, \ldots, G_q$ be the family of graphs used to construct G. Without loss of generality, we assume $|V(G_i)| = p$ for $1 \leq i \leq q$. To show

that G has high rank-width, note that every decomposition tree of G contains a $(\frac{1}{3}, \frac{2}{3})$-balanced cut $\left(A, \overline{A}\right)$. We show that every such cut has *cut-rank* at least $p/2$. Assume for contradiction that $\textit{cut-rank}(A) < p/2$. We distinguish two cases:

Case 1: At least p graphs in $\mathcal{F}$ contains vertices from both A and $\overline{A}$. Then for every i and j with $|i - j| = 1$ such that G_i contains vertices from both A and $\overline{A}$ we know that there is an edge (u, v) with $u \in V(G_i)$ such that $(u, v) \in G[V(G_i), V(G_j)]$. There is at least p such edges with different pairs of i and j, and without loss of generality there is at least $p/2$ such edges where $min(i, j)$ is odd. These edges form an induced matching contradicting that $\textit{cut-rank}(A) < p/2$.

Case 2: At least $2p + 1$ graphs in $\mathcal{F}$ contains vertices all from the same side of the cut $\left(A, \overline{A}\right)$. Since the cut is balanced there must be at least one graph with all its vertices in A and one graph with all its vertices in $\overline{A}$. Let G_i and $G_{i'}$ be two such graphs of minimum distance, without loss of generality assume $i < i'$ and $V(G_i) \subseteq A$. Then for each $r \in \{1, 2, \ldots, p\}$ there must be a j such that $\sigma_j(r) \in A$ and $\sigma_{j+1}(r) \in \overline{A}$, let S be a set containing all such pairs of vertices and $H = G[S]$ the subgraph induced by these vertices.

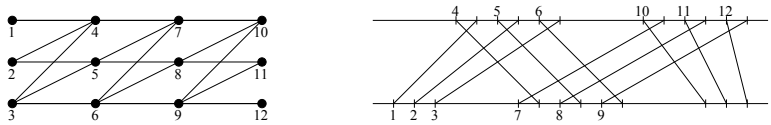
The graph H is formed by a collection of Hsu-graphs whose size sums up to at least p, there is an induced subgraph of H which is a collection of vertex disjoint Hsu-graphs whose size sum up to at least $2p/3$, namely partitioning the edges into 3 by $j \mod 3$ and picking the largest partition. This graph has rank-width at least $2p/3$ leading to a contradiction. □

We now describe two distinct families of Hsu-join-chains, one being a subclass of bipartite permutation graphs and the other a subclass of proper interval:

Corollary 17. *Bipartite permutation graphs have rank-width* $\Omega(\sqrt{n})$ *and boolean-width* $\Theta(\log n)$.

Proof. Let a *Hsu-stable-chain* of length q and width p be the Hsu-join-chain of length q and width p where for every i, G_i is a stable set of size p. Clearly, Hsu-stable-chains are bipartite permutation graphs (see Figure 4), and by Lemma 16 and Theorem 15 they have rank-width $\Theta(p)$ and boolean-width $\Theta(\log p)$. □

Recall that proper interval graphs are interval graphs admitting an interval model where all the intervals have same length.

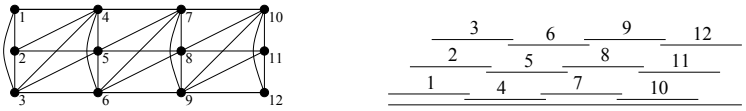


(a) (b)

Figure 4: (4×3) Hsu-stable chain (a) and its permutation representation (b).

Corollary 18. *Proper interval graphs have rank-width* $\Omega(\sqrt{n})$ *and boolean-width* $\Theta(\log n)$.

Proof. Let a *Hsu-Clique-chain* of length q and width p be the Hsu-join-chain of length q and width p where for every i, G_i is a clique of size p. Clearly, Hsu-clique-chains are proper interval graphs, and by Lemma 16 and Theorem 15 they have rank-width $\Theta(p)$ and boolean-width $\Theta(\log n)$. □



(a) (b)

Figure 5: (4×3) Hsu-Clique chain (a) and its proper interval representation (b).

Moreover, note that Jelínek showed in [35] that $\sqrt{n} \times \sqrt{n}$ grids have rank-width exactly $\sqrt{n} - 1$. Since the rank-width of a graph differs by at most one from the rank-width of its complement, then complement of grids have rank-width at least $\sqrt{n} - 2$. Since grids are 2-degenerate, then complement of k-degenerate graphs have rank-width $\Omega(\sqrt{n})$.

Finally, we can summarize these lower bounds as follows:

Lemma 19. *All graph classes in Group II of Figure 1 (except possibly Dilworth k graphs), have boolean-width* $\Omega(\log n)$.

Another interesting question to ask is whether there exist more graph classes having logarithmic boolean-width. The usual way to answer this question is by either showing how to construct a decomposition of small width, or by showing an infinite family of graphs of large width. For some graph classes it is possible to provide such examples of graphs having non-logarithmic boolean-width, like for the $q \times q$ grid. However, for other classes

of graphs, we do not know any example of infinite family of graphs having non-logarithmic boolean-width. We are nonetheless able to provide some lower bounds:

Lemma 20. *For all the classes in Group IV of Figure 1, either they do not have boolean-width $O(\log n)$, or such a decomposition cannot be computed in polynomial time unless $P = NP$.*

Proof. Note first that for all the classes of graphs in Group IV of Figure 1, Minimum Weight Dominating Set is NP-complete (see [36], [37] and [38]). Moreover, Minimum Weight Dominating Set can be solved in time $O(2^{3 \cdot boolw} \cdot poly(n))$ [17]. Assume now that there exists a class C in Group IV of Figure 1 having boolean-width $O(\log n)$ and where such decompositions can be computed in polynomial time. Then Minimum Weight Dominating Set can be computed in time $O(2^{O(logn)} \cdot poly(n))$ which is polynomial in n. Hence if a class of graphs on which Minimum Weight Dominating Set is NP-complete has boolean-width $O(\log n)$, then computing such decompositions is NP-hard. □

Note that this holds not only for Minimum Weight Dominating Set, but as long as there exists a problem which can be solved in $O(2^{O(boolw} \cdot poly(n))$ time. Finally, we can get stronger lower bounds by working under a stronger hypothesis. The *E*xponential *T*ime *H*ypothesis (ETH) states that there does not exists an algorithm for solving 3-Sat running in time $2^{o(n)}$. We can reformulate Lemma 20 as follows:

Lemma 21. *For all the classes in Group IV of Figure 1, either they do not have boolean-width $O(n^{o(1)})$, or such a decomposition cannot be computed in time $2^{n^{o(1)}}$, unless ETH fails.*

Proof. Assume for contradiction that there exists a class of graphs $\mathcal{C}$ in Group IV of Figure 1 for which a decomposition of boolean-width $n^{o(1)}$ can be computed in time $2^{n^{o(1)}}$. Recall that Minimum Weight Dominating Set is NP-complete on all the classes in Group IV of Figure 1. Hence, there is a polynomial time reduction from k-SAT to Minimum Weight Dominating Set on $\mathcal{C}$ such that from any instance I of k-SAT, a graph $G = (V, E)$ belonging to $\mathcal{C}$ can be built such that $n \leq |I|^c$, for some constant $c > 0$ and solving Minimum Weight Dominating Set on G implies a solution to k-SAT on I. Recall that Minimum Weight Dominating Set can be solved in $2^{3 \cdot boolw(G)} \cdot poly(n)$. Finally, since we assumed we could compute

a decomposition of boolean-width $n^{o(1)}$ in time $2^{n^{o(1)}}$, the instance I can be solved in $2^{3 \cdot n^{o(1)}} \cdot poly(n)$, which is equivalent to $2^{3 \cdot |I|^{o(c)}} \cdot poly(|I|^c)$. This would imply that we could solve the instance I in time $2^{o(|I|)}$. Hence the Lemma follows. □

This means for instance that if split graphs have boolean-width polylogarithmic in n, then it is NP-hard to compute a decomposition of split graphs having boolean-width within a factor $\log(n)$ of the optimum.

5. Conclusion

We have shown that all graph classes in Group II of Figure 1 admit a decomposition where nec_d is bounded by a polynomial of n, and we can compute such decompositions in polynomial time if the intersection is model is given. This answers an open question from [17]. The following theorem is the main motivation for our results.

Theorem 22 (Main theorem of [1])**.** *Let G be a graph given along with a decomposition tree (T, δ). For every LC-VSVP problem Π, there are constants d and q such that Π can be solved in time $O(n^4 \cdot q \cdot nec_d(T, \delta)^{3q})$.*

Combined with the results in Section 3 we get the following theorem:

Theorem 23. *Let $\mathcal{C}$ be one of the following graph classes: Dilworth k graphs, convex graphs, trapezoid graphs, circular permutation graphs, circular arc graphs or circular k-trapezoid graphs. Then, every LC-VSVP problem can be solved in polynomial time on $\mathcal{C}$.*

For the particular case of complement of k-degenerate, we gave a bound for $nec_d(T, \delta)$ of the form $2^{d \cdot k} \cdot n^d$, which implies the following:

Theorem 24. *Let G be the complement of a k-degenerate graph, given along with a decomposition tree (T, δ). Every LC-VSVP problem can be solved in time $2^{O(k)} \cdot poly(n)$.*

This means that every LC-VSVP problem can be solved in FPT time on a graph G when parameterized by the degeneracy of the complement of G, with single exponential dependence in the parameter. Finally, we leave open the question of whether the classes in Group III of Figure 1 have logarithmic boolean-width.

References

[1] B.-M. Bui-Xuan, J. A. Telle, M. Vatshelle, Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems, Theoretical Computer Science Submitted to this issue (?) (2012) ?

[2] J. A. Telle, A. Proskurowski, Algorithms for vertex partitioning problems on partial k-trees, SIAM Journal on Discrete Mathematics 10 (4) (1997) 529–550.

[3] B. Courcelle, J. A. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width, Theory of Computing Systems 33 (2) (2000) 125–150.

[4] M. R. Garey, D. S. Johnson, L. J. Stockmeyer, Some simplified np-complete graph problems, Theoretical Computer Science 1 (3) (1976) 237–267.

[5] A. Brandstädt, D. Kratsch, On the restriction of some np-complete graph problems to permutation graphs, in: Proceedings of FCT, Vol. 199 of Lecture Notes in Computer Science, Springer-Verlag, 1985, pp. 53–62.

[6] M.-S. Chang, Efficient algorithms for the domination problems on interval and circular-arc graphs, SIAM Journal on Computing 27 (6) (1998) 1671–1694.

[7] P. Damaschke, H. Müller, D. Kratsch, Domination in convex and chordal bipartite graphs, Information Processing Letters 36 (5) (1990) 231–236.

[8] J. Díaz, J. Nesetril, M. J. Serna, D. M. Thilikos, H-colorings of large degree graphs, in: Proceedings of EurAsia-ICT, Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 850–857.

[9] E. S. Elmallah, L. K. Stewart, Independence and domination in polygon graphs, Discrete Applied Mathematics 44 (1-3) (1993) 65–77.

[10] M. Farber, J. Keil, Domination in permutation graphs, Journal of Algorithms 6 (1985) 309–321.

[11] P. van't Hof, D. Paulusma, J. M. M. van Rooij, Computing role assignments of chordal graphs, Theoretical Computer Science 411 (40–42) (2010) 3601–3613.

[12] W.-L. Hsu, K.-H. Tsai, Linear time algorithms on circular-arc graphs, Information Processing Letters 40 (3) (1991) 123–129.

[13] Y. Liang, Dominations in trapezoid graphs, Information Processing Letters 52 (6) (1994) 309–315.

[14] C. Rhee, Y. Liang, S. Dhall, S. Lakshmivarahan, An $o(n + m)$-time algorithm for finding a minimum-weight dominating set in a permutation graph, SIAM Journal on Computing 25 (2) (1996) 404–419.

[15] K.-H. Tsai, W.-L. Hsu, Fast algorithms for the dominating set problem on permutation graphs, Algorithmica 9 (6) (1993) 601–614.

[16] J. Kratochvíl, P. D. Manuel, M. Miller, Generalized domination in chordal graphs, Nordic Journal of Computing 2 (1) (1995) 41–50.

[17] B.-M. Bui-Xuan, J. A. Telle, M. Vatshelle, Boolean-width of graphs, Theoretical Computer Science 412 (39) (2011) 5187–5204.

[18] M. Yannakakis, Node deletion problems on bipartite graphs, SIAM Journal on Computing 10 (2) (1981) 310–327.

[19] W.-L. Hsu, Decomposition of perfect graphs, Journal of Combinatorial Theory, Series B 43 (1) (1987) 70–94.

[20] J. Geelen, A. Gerards, G. Whittle, Branch-width and well-quasi-ordering in matroids and graphs, Journal of Combinatorial Theory, Series B 84 (2) (2002) 270–290.

[21] S.-i. Oum, P. D. Seymour, Approximating clique-width and branch-width, Journal of Combinatorial Theory, Series B 96 (4) (2006) 514–528.

[22] N. Robertson, P. D. Seymour, Graph minors. X. obstructions to tree-decomposition, Journal of Combinatorial Theory, Series B 52 (2) (1991) 153–190.

[23] K. S. Booth, G. S. Lueker, Testing for the consecutive ones property, interval graphs and graph planarity using pq-tree algorithms, Journal of Computer and System Sciences 13 (1976) 335–379.

[24] R. M. McConnell, Linear-time recognition of circular-arc graphs, Algorithmica 37 (2003) 93–147.

[25] D. Kratsch, R. M. Mcconnell, K. Mehlhorn, J. P. Spinrad, Certifying algorithms for recognizing interval graphs and permutation graphs (2003) 158–167.

[26] R. Sritharan, A linear time algorithm to recognize circular permutation graphs, Networks 27 (3) (1996) 171–174.

[27] T. H. Ma, J. P. Spinrad, On the 2-chain subgraph cover and related problems, Journal of Algorithms 17 (2) (1994) 251–268.

[28] H. L. Bodlaender, T. Kloks, D. Kratsch, H. Müller, Treewidth and minimum fill-in on d-trapezoid graphs, Journal of Graph Algorithms and Applications 2 (2).

[29] M. Yannakakis, The complexity of the partial order dimension problem, SIAM Journal on Algebraic and Discrete Methods 3 (3) (1982) 351–358.

[30] R. Hegde, K. Jain, The hardness of approximating poset dimension, Electronic Notes in Discrete Mathematics 29 (2007) 435–443.

[31] D. Kratsch, T. Kloks, H. Müller, Measuring the vulnerability for classes of intersection graphs, Discrete Applied Mathematics 77 (3) (1997) 259–270.

[32] Y.-l. Lin, Circular and circle trapezoid graphs, Journal of Science and Engineering Technology 2 (2) (2006) 11–17.

[33] E. S. Elmallah, L. K. Stewart, Polygon graph recognition, Journal of Algorithms 26 (1) (1998) 101–140.

[34] V. R. Stefan Felsner, J. Spinrad, Recognition algorithms for orders of small width and graphs of small dilworth number, Order 20 (4) (2003) 351–364.

[35] V. Jelínek, The rank-width of the square grid, in: 34rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'08), Vol. 5344 of LNCS, 2008, pp. 230–239.

[36] D. G. Corneil, Y. Perl, Clustering and domination in perfect graphs, Discrete Applied Mathematics 9 (1984) 27–40.

[37] M.-S. Chang, Weighted domination of cocomparability graphs, Discrete Applied Mathematics 80 (3) (1997) 135–148.

[38] J. M. Keil, The complexity of domination problems in circle graphs, Discrete Applied Mathematics 42 (1) (1993) 51–63.

Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems[☆]

Binh-Minh Bui-Xuan[a], Jan Arne Telle[b], Martin Vatshelle[b,*]

[a] *CNRS – Université Paris 6, France.*
[b] *Department of Informatics, University of Bergen, Norway.*

Abstract

Given a graph G we provide dynamic programming algorithms for many locally checkable vertex subset and vertex partitioning problems. Their runtime is polynomial in the number of equivalence classes of problem-specific equivalence relations on subsets of vertices, defined on a given decomposition tree of G. Using these algorithms all these problems become solvable in polynomial time for many well-known graph classes like interval graphs and permutation graphs (Belmonte and Vatshelle [1]). Given a decomposition of boolean-width k we show that the algorithms will have runtime $O(n^4 2^{O(k^2)})$, providing the first large class of problems solvable in fixed-parameter single-exponential time in boolean-width.

1. Introduction

When solving graph problems by divide-and-conquer or by dynamic programming we need to recursively divide the input graph G. A natural way to do this is to recursively partition the vertices of the graph in two parts. The resulting decomposition of G can be stored as a full binary tree whose leaves are in bijection with the n vertices of G. In this paper we assume that we are given such a decomposition tree of G and focus on fast dynamic programming algorithms for a large class of locally checkable vertex subset and vertex partitioning problems. Depending on the problem being solved

[☆]Supported by the Norwegian Research Council, project PARALGO.
[*]Corresponding author. Tel: (+47) 55 58 42 00. Fax: (+47) 55 58 41 99.
Email addresses: buixuan@lip6.fr (Binh-Minh Bui-Xuan), telle@ii.uib.no (Jan Arne Telle), vatshelle@ii.uib.no (Martin Vatshelle)

and the given decomposition tree we define equivalence relations on vertex subsets and give algorithms with runtime polynomial in n and the number of equivalence classes of these relations. In a companion paper by Belmonte and Vatshelle [1] it is shown that for many families of graphs, like permutation graphs, interval graphs, Dilworth k graphs, we can in polynomial time find a decomposition tree where the number of such equivalence classes is polynomial in n. Combined with the results in this paper we get on all those graph families polynomial-time algorithms solving the class of locally checkable vertex subset and vertex partitioning problems.

This class includes many well-known NP-hard problems related to domination, independence and homomorphism, and also their vertex weighted versions. For example, vertex subset problems like Perfect Code, Maximum Induced Matching, Minimum Perfect Dominating Set and in general existence and optimization problems over any of the vertex subset properties listed in Table 1. For fixed integer d problems like Minimum d-Dominating Set, Induced d-Regular Subgraph, Minimum Subgraph of Degrees $\geq d$, and d-Vertex Coloring. Also, the problem of deciding if the input graph has a partitioning of its vertex set into a fixed number q of sets each having a property listed in Table 1. For a fixed simple graph H also problems like H-Coloring, H-Homomorphism, H-Covering, H-Partial Covering, and H-Role Assignment, see Table 2, asking for a homomorphism, with some possible local constraints, from the input graph G to the target graph H.

These are optimization problems over locally checkable neighborhood conditions, as defined in Section 2. For example, in the Minimum d-Dominating Set problem we optimize over vertex subsets S of the input graph G such that any vertex not in S has at least d neighbors in S. To check this condition note that we must count S-neighbors up to d, but no further since it does not matter if a vertex has d neighbors in S or if it has more than d neighbors in S. In a bottom-up traversal of the decomposition tree T we solve the problem on induced subgraphs of G of increasing size. For the subgraph induced by $A \subseteq V(G)$ two subsets $X, Y \subseteq A$ will be *equivalent* w.r.t. the d-Dominating Set constraint if any vertex v not in A either has the same number of neighbors in X and Y, or at least d in each. The number of equivalence classes nec^d_A of vertex subsets will in this way depend on the value d as given by the problem, on the various vertex subsets A as given by the decomposition tree T, and on the bipartite graphs induced by edges having exactly one endpoint in A as given by the graph G. In Section 5 we give dynamic programming algorithms solving locally checkable vertex sub-

σ	ρ	d	Standard name	VP	$\exists$	MAX	MIN
$\mathbb{N}$	$\geq d$	d	d-Dominating set	3	P	P	NPC
$\{d\}$	$\mathbb{N}$	d+1	Induced d-Regular Subgraph	2	?	NPC	?
$\geq d$	$\mathbb{N}$	d	Subgraph of Min Degree $\geq d$	?	P	P	NPC
$\leq d$	$\mathbb{N}$	d+1	Induced Subg. of Max Degree $\leq d$	?	P	NPC	P
$\{0\}$	$\{0,1\}$	2	Strong Stable set or 2-Packing	4	P	NPC	P
$\{0\}$	$\{1\}$	2	Perfect Code or Efficient Dominating set	4	NPC	NPC	NPC
$\{0,1\}$	$\{0,1\}$	2	Total Nearly Perfect set	3	P	NPC	P
$\{0,1\}$	$\{1\}$	2	Weakly Perfect Dominating set	3	NPC	NPC	NPC
$\{1\}$	$\{1\}$	2	Total Perfect Dominating set	3	NPC	NPC	NPC
$\{1\}$	$\mathbb{N}$	2	Induced Matching	2	P	NPC	P
$\{1\}$	$\mathbb{N}^+$	2	Dominating Induced Matching	2	NPC	NPC	NPC
$\mathbb{N}$	$\{1\}$	2	Perfect Dominating set	2	P	P	NPC
$\{0\}$	$\mathbb{N}$	1	Independent set	3	P	NPC	P
$\mathbb{N}$	$\mathbb{N}^+$	1	Dominating set	3	P	P	NPC
$\{0\}$	$\mathbb{N}^+$	1	Independent Dominating set	3	P	NPC	NPC
$\mathbb{N}^+$	$\mathbb{N}^+$	1	Total Dominating set	2	P	P	NPC

Table 1: Some vertex subset properties expressible as (σ, ρ)-sets, with $\mathbb{N} = \{0, 1, ...\}$ and $\mathbb{N}^+ = \{1, 2, ...\}$. Column d shows that we must count up to d neighbors. Column VP shows the smallest k for which the question of partition into k such sets is NP-complete. Columns $\exists$, MAX and MIN show complexity of existence, maximization and minimization over such sets, with P, NPC and ? denoting Polytime, NP-Complete and unknown (to us).

set problems in time polynomial in all the nec_A^d, and in Section 6 we extend this result to vertex partitioning problems.

Edges	d	Standard name	NP-Complete
$\mathbb{N}$	1	H-coloring H-homomorphism	H bipartite
$\mathbb{N}^+$	1	H-role assignment H-locally surj. hom.	H on 3 vertices or more
$\{1\}$	2	H-covering H-locally bij. hom.	open, e.g. H k-regular for $k \geq 3$
$\{0,1\}$	2	H-partial covering H-locally inj. hom.	open, Harder than H-covering

Table 2: Various homomorphism problems for fixed simple graph H. These are expressible as locally checkable vertex partitioning problems with the degree constraint matrix D_q being the adjacency matrix of H with 1-entries replaced by value in column Edges, 0-entries replaced by $\{0\}$, and $q = |V(H)|$. Column d shows that we must count up to d neighbors. NP-completeness known for fixed H having property listed in the last column [2], with dichotomy known for the first two rows.

As shown in the companion paper [1], for many families of intersection graphs, like convex graphs and trapezoid graphs, one can in polynomial time find a decomposition tree T such that nec_A^d is polynomial in n, for any subset A appearing as the leaves of a subtree of T, and any fixed value d. This implies polynomial-time algorithms for the locally checkable vertex subset and vertex partitioning problems on these families of intersection graphs. On the other hand, for graph families where at least one of these problems remains NP-hard we cannot expect the existence of decomposition trees with every nec_A^d polynomial in n. In such cases it is common to define a width parameter of graphs and apply the theory of fixed parameter algorithms. One can, for each value of d, define a width parameter of graphs that captures the minimum, over all decomposition trees T of a graph G, of the maximum nec_A^d for any $A \subseteq V(G)$ at the leaves of a subtree of T. For the value $d = 1$ the resulting width parameter is exactly $2^{boolw(G)}$, where $boolw(G)$ is the boolean-width of G. Moreover, we show in Section 7 that for any d and A we have $nec_A^d \leq (nec_A^1)^{d \times log_2(nec_A^1)}$. This implies that given a graph G and a decomposition tree of boolean-width $boolw$ our dynamic programming algorithms are single-exponential fixed-parameter tractable in $boolw$. For vertex subset problems the runtime becomes $O(n^4 2^{3d \times boolw^2})$ and for problems asking for a

partition of the vertex set into q sets the runtime becomes $O(n^4 2^{3qd \times boolw^2})$.

Width parameters of graphs have many applications in the field of graph algorithms and especially in Fixed Parameter Tractable (FPT) algorithmics, see *e.g.* Downey and Fellows [3], Flum and Grohe [4], and Hliněný et al. [5]. Since the locally checkable vertex subset and vertex partitioning problems are expressible in monadic second-order logic it follows from the well-known theorem of Courcelle and Makowsky [6] that they are fixed-parameter tractable when parameterized by either the tree-width, branch-width, clique-width, rank-width or boolean-width of the input graph. Although the runtime resulting from this theorem contains a highly exponential factor (tower of powers), the problems behave very well for tree-width tw and branch-width bw: Given a decomposition tree of tree-width tw, they can be solved in $O^*(2^{O(tw)})$ and $O^*(2^{O(bw)})$ time [7]. This is not the same situation for clique-width cw, where until now the best runtime contained a $O^*(2^{2^{poly(cw)}})$ double exponential factor [8]. Having small boolean-width is witnessed by a decomposition of the graph into cuts with few different unions of neighborhoods across the cut. This makes the decomposition natural to guide dynamic programming algorithms to solve problems, like Maximum Independent Set, where vertex sets having the same neighborhoods can be treated as equivalent [9]. In this paper we extend such an observation to the much larger class of locally checkable vertex subset and vertex partitioning problems. As mentioned above, the runtime of our algorithms expressed by boolean-width $boolw$ of the given decomposition tree is $O^*(2^{O(boolw^2)})$, which can be interpreted as $O^*(2^{O(cw^2)})$ since for the clique-width cw resulting from this decomposition tree we have $cw \geq boolw$ [9]. For clique-width this improves by an exponential factor the best previous runtimes [8] and it provides for the first time a large class of problems for which dynamic programming gives runtime single exponential in boolean-width. It implies quasi-polynomial algorithms solving all these problems on random graphs, since it has been shown that a random graph on n vertices where the edges are drawn with respect to a uniform distribution almost surely has boolean-width $\Theta(\log^2 n)$, and it is easy to find a decomposition tree witnessing it [10]. On an arbitrary graph G a decomposition of optimal boolean-width can be computed in time $O(2.52^n)$ [11]. Heuristic algorithms finding decompositions for boolean-width compare well with heuristics for tree-width, in particular for dense graphs [12], opening for the possibility of a practical application of the algorithms given here.

The paper is organized as follows. In Section 2 we define the class of

problems and the decomposition trees used. In Section 3 we give an intuitive description of our algorithms, using the Maximum Induced Matching problem as example. In Section 4 we give a pre-processing step computing representatives to be used as indices of the dynamic programming. In Section 5 we give algorithms for vertex subset problems and in Section 6 algorithms for vertex partitioning problems. In Section 7 we define boolean-width and show its relation to nec_A^d that allows us to express runtime of our algorithms as a function of boolean-width. We end in Section 8 with some conclusions and open problems.

2. Locally checkable problems and rooted decomposition trees

We deal with simple, undirected graphs. The complement of a vertex subset A of a graph $G = (V(G), E(G))$ is denoted by $\overline{A} = V(G) \setminus A$. The neighborhood of a vertex x is denoted by $N(x)$ and for a subset of vertices X we denote the union of their neighborhoods by $N(X) = \bigcup_{x \in X} N(x)$. We denote by $G[X]$ the graph induced by $X \subseteq V(G)$. To ensure uniqueness of certain algorithms, *e.g.* for computing representatives of the equivalence relations on vertex subsets, we assume a total ordering σ on the vertex set of G which stays the same throughout the entire paper. For easy disambiguation, we usually refer to vertices of a graph and nodes of a tree.

We want to solve graph problems using a divide-and-conquer approach. To this aim, we need to store the information on how to recursively divide the input graph instance. A standard way to do this is to use a decomposition tree, for our purposes a rooted tree.

Definition 1. A decomposition tree of a graph G is a pair (T, δ) where T is a full binary tree (i.e. T rooted with every non-leaf having two children) and δ a bijection between the leaf set of T and the vertex set of G. For a node a of T let the subset of $V(G)$ in bijection δ with the leaves of the subtree of T rooted at a be denoted by V_a.

We will be interested in the following problems as defined in [7].

Definition 2. Let σ and ρ be finite or co-finite subsets of natural numbers. A subset S of vertices of a graph G is a *sigma-rho set*, or simply (σ, ρ)-*set*, of G if

$$\forall v \in V(G) : |N(v) \cap S| \in \begin{cases} \sigma & \text{if } v \in S, \\ \rho & \text{if } v \in V(G) \setminus S. \end{cases}$$

Table 1 shows some classical vertex subset properties expressed as (σ, ρ)-sets. The class of *locally checkable vertex subset problems* consist of finding a minimum or maximum (σ,ρ)-set in an input graph G, possibly on vertex-weighted graphs. This includes many NP-hard problems as indicated in Table 1. For NP-completeness results see [13, 14, 15, 16, 17, 18].

Since σ and ρ are either finite or co-finite we can check locally if S is a (σ, ρ)-set by counting for each vertex v the number of S-neighbors only up to $d(\sigma, \rho)$, defined as follows.

Definition 3. Let $d(\mathbb{N}) = 0$. For every finite or co-finite set $\mu \subseteq \mathbb{N}$, let $d(\mu) = 1 + min(max_{x\in\mathbb{N}} x : x \in \mu, max_{x\in\mathbb{N}} x : x \notin \mu)$. Let $d(\sigma, \rho) = max(d(\sigma), d(\rho))$.

For example, $d(\{1\}) = 2$ which is one more than the largest number contained in the finite set $\{1\}$, while $d(\{1, 2, ...\}) = 1$ which is one more than the largest number not contained in the co-finite set $\{1, 2, ...\}$, and $d(\{1\}, \{1, 2, ...\}) = 2$ which is the maximum of $d(\{1\})$ and $d(\{1, 2, ...\})$.

We can also ask for a partition of $V(G)$ into q classes, with each class satisfying a certain (σ, ρ)-property, as follows.

Definition 4. A *degree constraint* matrix D_q is a q by q matrix with entries being finite or co-finite subsets of natural numbers. A D_q*-partition* in a graph G is a partition $\{V_1, V_2, ..., V_q\}$ of $V(G)$ such that for $1 \leq i, j \leq q$ we have $\forall v \in V_i : |N(v) \cap V_j| \in D_q[i, j]$.

The *locally checkable vertex partitioning problems* consist of deciding if G has a D_q partition. NP-hard problems fitting into this framework include the question of deciding if an input graph has a partition into q (σ, ρ)-sets, which is in most cases NP-complete for small values of q, see the column VP in Table 1. It also includes for any fixed graph H the homomorphism problems listed in Table 2. Let us mention that extending the algorithms we give here to handle also the case of finding an extremal value (maximum or minimum) of the cardinality of a vertex partition class over all D_q-partitions is straightforward.

3. Detailed example: Maximum Induced Matching

We first describe our algorithms intuitively, taking as our main example the vertex subset maximization problem over (σ, ρ)-sets with $\sigma = \{1\}$ and $\rho = \mathbb{N}$. We thus want to compute the cardinality of a maximum set of vertices

S such that in $G[S]$ all vertices have degree one, the so-called Maximum Induced Matching problem. In a bottom-up traversal of T we will solve the problem on induced subgraphs of increasing size, at node a of T storing information on partial solutions to the problem on $G[V_a]$ in a table Tab_a.

A partial solution will have two parts (S_a, Z_a) where

- $S_a \subseteq V_a$ such that in $G[S_a]$ all vertices (of S_a) have degree *at most one*
- $Z_a \subseteq \overline{V_a}$ such that in $G[S_a \cup Z_a]$ vertices of S_a have degree *exactly one*

We call (S_a, Z_a) a partial solution to the Max Induced Matching problem on $G[V_a]$, in which Z_a is a witness of a necessary, but not sufficient, condition for S_a to be extendible into an induced matching of G.

The number of partial solutions could be exponential in n but many of them are superfluous. If two subsets $Z_a, Y_a \subseteq \overline{V_a}$ have the property that for every $v \in V_a$ the vertex v has either zero neighbors in each of Z_a and Y_a, or exactly one neighbor in each, or at least two neighbors in each, then it is not hard to check that for the Maximum Induced Matching problem (S_a, Z_a) is a partial solution if and only if (S_a, Y_a) is a partial solution, and one of the two will be superfluous. This motivates the following equivalence relation on subsets of vertices, which applies to the general σ, ρ case using $d(\sigma, \rho)$-neighbor equivalence. For the Maximum Induced Matching problem we have $d(\{1\}, \mathbb{N}) = 2$.

Definition 5 (d-neighbor equivalence)**.** Let d be a non-negative integer, G be a graph and $A \subseteq V(G)$. Two vertex subsets $X \subseteq A$ and $Y \subseteq A$ are *d-neighbor equivalent* w.r.t. A, denoted by $X \equiv_A^d Y$, if:

$$\forall v \in \overline{A} : \min(d, |X \cap N(v)|) = min(d, |Y \cap N(v)|).$$

Let $nec(\equiv_A^d)$ be the number of equivalence classes of $\equiv_A^d$.

In the example above we had $Z_a \equiv_{\overline{V_a}}^2 Y_a$ and for fixed S_a we need to store a partial solution (S_a, Z_a) for at most one member of the equivalence class of Z_a. A similar thing applies to S_a, if $S_a \equiv_{V_a}^2 S'_a$ and both (S_a, Z_a) and (S'_a, Z_a) are partial solutions and $|S_a| \geq |S'_a|$ then (S'_a, Z_a) is superfluous since we are solving a maximization problem. In light of this we index the table Tab_a of partial solutions at node a of T by the Cartesian product of the two sets of

equivalence classes $\equiv^2_{V_a} \times \equiv^2_{\overline{V_a}}$ (or rather by representatives of these classes) and store the following information:

$$Tab_a[X][Y] \stackrel{\text{def}}{=} max_{S \subseteq V_a}\{|S| : S \equiv^2_{V_a} X \text{ and } G[S \cup Y] \text{ is 1-regular}\}$$

or minus ∞ if no such S exists. Note that (S, Y) is a partial solution for $G[V_a]$. In this way we contain the runtime by a function of the number of equivalence classes $nec(\equiv^2_{V_a})$ and $nec(\equiv^2_{\overline{V_a}})$.

It is instructive to consider in some detail the initialization of table entries at leaf a of T associated to vertex $\delta(a) = v \in V(G)$. In that case we have $V_a = \{v\}$ with two classes of $\equiv^2_{V_a}$ (assuming v has at least one neighbor) with representatives $\{v\}$ and $\emptyset$, we have $\overline{V_a} = V(G) \setminus \{v\}$ with three classes of $\equiv^2_{\overline{V_a}}$ (assuming v has at least two neighbors) with representatives R_0, R_1 and R_2. The class of R_i for $i = 0, 1$ consists of those subsets of $V(G) \setminus \{v\}$ containing i vertices of $N(v)$, and for $i = 2$ those containing ≥ 2 vertices of $N(v)$. According to the definition of table entries we initialize as follows.

- $Tab_a[\emptyset][R_0] = Tab_a[\emptyset][R_1] = Tab_a[\emptyset][R_2] = 0$
- $Tab_a[\{v\}][R_0] = Tab_a[\{v\}][R_2] = -\infty$
- $Tab_a[\{v\}][R_1] = 1$

At the root r of T we have $V_r = V(G)$ with a single equivalence class of $\equiv^2_{V_r}$, and we have $\overline{V_r} = \emptyset$ with a single equivalence class of $\equiv^2_{\overline{V_r}}$. The single entry of Tab_r will contain the cardinality of the largest $S \subseteq V(G)$ such that in $G[S]$ all vertices have degree 1, i.e. the solution to the Maximum Induced Matching problem.

At the combine step we have a node w of T with children a, b such that $V_w = V_a \cup V_b$ and we compute partial solutions to $G[V_a \cup V_b]$ based on already computed partial solutions to $G[V_a]$ and $G[V_b]$. For any dynamic programming on decomposition trees it is important to keep in mind the below observation, that follows directly from definitions.

Observation 1. *If in the tree T node w has children a and b then $\{V_a, V_b, \overline{V_w}\}$ forms a 3-partition of $V(G)$.*

Another crucial and easy observation is the coarsening of neighborhood equivalence classes when traversing from a child node a to its parent node w.

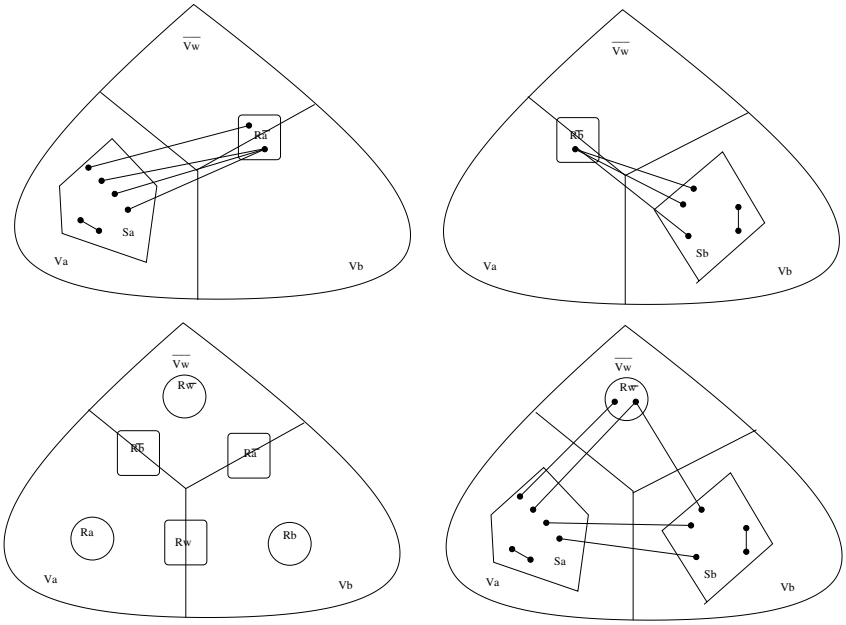


Figure 1: Solving Maximum Induced Matching by dynamic programming using the 2-neighbor equivalence relation on vertex subsets. At inner node w of T with children a, b we have partition $V_a, V_b, \overline{V_w}$ of $V(G)$. Bottom left shows three arbitrary representatives $R_a, R_b, R_{\overline{w}}$ of refined relations as circles, and as boxes the computed representatives $R_{\overline{a}}, R_{\overline{b}}, R_w$ of the resulting coarser relations. Top line shows partial solutions $(S_a, R_{\overline{a}})$ from $Tab_a[R_a][R_{\overline{a}}]$ and $(S_b, R_{\overline{b}})$ from $Tab_b[R_b][R_{\overline{b}}]$. Bottom right shows resulting partial solution $(S_a \cup S_b, R_{\overline{w}})$ from $Tab_w[R_w][R_{\overline{w}}]$.

Observation 2. *Let d be a non-negative integer, G be a graph with $V_a \subseteq V_w \subseteq V(G)$ and let $X, Y \subseteq V_a$. If $X \equiv^d_{V_a} Y$ then $X \equiv^d_{V_w} Y$.*

In particular we have that if $X \equiv^2_{V_a} Y$ or $X \equiv^2_{V_b} Y$ then $X \equiv^2_{V_w} Y$, so that $\equiv^2_{V_w}$ is a coarsening of the two relations $\equiv^2_{V_a}$ and $\equiv^2_{V_b}$. Likewise $\equiv^2_{\overline{V_a}}$ and $\equiv^2_{\overline{V_b}}$ are coarsenings of $\equiv^2_{\overline{V_w}}$. The algorithm will iterate over all triples of representatives $R_a, R_b, R_{\overline{w}}$ from the three most refined equivalence relations, $\equiv^2_{V_a}$, $\equiv^2_{V_b}$, $\equiv^2_{\overline{V_w}}$, compute the representatives of the resulting coarser relations: R_w of $\equiv^2_{V_w}$ (from R_a and R_b), $R_{\overline{a}}$ of $\equiv^2_{\overline{V_a}}$ (from R_b and $R_{\overline{w}}$), $R_{\overline{b}}$ of $\equiv^2_{\overline{V_b}}$ (from R_a and $R_{\overline{w}}$), and then update $Tab_w[R_w][R_{\overline{w}}]$ by $Tab[R_a][R_{\overline{a}}] + Tab[R_b][R_{\overline{b}}]$. For correctness it will be crucial that for any $S_a \equiv^2_{V_a} R_a$ and $S_b \equiv^2_{V_b} R_b$ the following holds: If in $G[S_a \cup R_b \cup R_{\overline{w}}]$ all vertices of S_a have degree 1 and in $G[S_b \cup R_a \cup R_{\overline{w}}]$ all vertices of S_b have degree 1, then in $G[S_a \cup S_b \cup R_{\overline{w}}]$ all vertices in $S_a \cup S_b$ will have degree 1. We refer to the formal description and correctness proof for details, see Figure 3.

The case of $\rho \neq \mathbb{N}$ is handled similarly. For example, consider Maximum Dominating Induced Matching for which it is NP-complete simply to decide if such a set exists. In this case we maximize over $\sigma = \{1\}, \rho = \mathbb{N}^+$ sets. Partial solutions (S_a, Z_a) must now also ensure that in $G[S_a \cup Z_a]$ vertices in $V_a \setminus S_a$ have degree at least one. The fact that $d(\mathbb{N}^+) = 1 < d(\{1\}) = 2$ does not matter, as the 1-neighbor equivalence relation is a coarsening of the 2-neighbor equivalence relation.

4. Computing representatives of *d*-neighbor equivalence classes

Before explaining the dynamic programming we show how to compute representatives of the $\equiv^d_{V_w}$ and $\equiv^d_{\overline{V_w}}$ relations used for indexing the table Tab_w at node w of the tree T. We denote by $rep^d_{V_w}(X)$ the representative of $X \subseteq V_w$ in the relation $\equiv^d_{V_w}$, and by $rep^d_{\overline{V_w}}(Y)$ the representative of $Y \subseteq \overline{V_w}$ in the relation $\equiv^d_{\overline{V_w}}$. For simplicity we define this using V_w instead of a generic subset A, but note that everything we say about $X \subseteq V_w$, $rep^d_{V_w}(X)$ and $\equiv^d_{V_w}$ will hold also for $rep^d_{\overline{V_w}}(Y)$, $Y \subseteq \overline{V_w}$ and $\equiv^d_{\overline{V_w}}$.

Definition 6 (representative)**.** We assume that a total ordering of the vertices of $V(G)$ is given. For every $X \subseteq V_w$, the representative $rep^d_{V_w}(X)$ is defined as the lexicographically smallest set $R \subseteq V_w$ such that $|R|$ is minimized and $R \equiv^d_{V_w} X$.

To check algorithmically if two subsets of V_w are equivalent w.r.t. V_w we use the d-neighborhood vector w.r.t. V_w defined as follows.

Definition 7 (d-neighborhood)**.** For $X \subseteq A \subseteq V(G)$, the d-neighborhood of X w.r.t. A, denoted $N_A^d(X)$, is a multiset of nodes from $\overline{A}$, such that, $\forall v \in \overline{A}$ the number of occurrences of v in $N_A^d(X)$ is equal to $\min\{|N(v) \cap X|, d\}$. Since we have assumed a fixed ordering of the vertices we will store $N_A^d(X)$ as an $|\overline{A}|$-vector over $\{0, 1, ..., d\}$.

Note that $X \equiv_A^d X'$ if and only if $N_A^d(X) = N_A^d(X')$.

Algorithm 1 List of representatives and their d-neighborhood

```
INPUT: Graph G, subset A ⊆ V(G) and integer d
Initialize LR_A, LNR_A, NextLevel to be empty
Initialize LastLevel = {∅}
while LastLevel != ∅ do
  for R in LastLevel do
    for every vertex v of A do
      R' = R ∪ {v}
      compute N' = N_A^d(R')
      if R' ≢_A^d R and N' is not contained in LNR_A then
        add R' to both LR_A and NextLevel
        add N' to LNR_A at the proper position
        add pointers between R' and N'
      end if
    end for
  end for
  set LastLevel = NextLevel, and NextLevel = ∅
end while
OUTPUT: LR_A and LNR_A
```

Lemma 1. *For any node w of T we can compute a list LR_{V_w} containing all representatives w.r.t. $\equiv_{V_w}^d$ in time $O(nec(\equiv_{V_w}^d) \cdot \log(nec(\equiv_{V_w}^d)) \cdot n^2)$.*

We also compute a data structure that given $X \subseteq V_w$, in time $O(\log(nec(\equiv_{V_w}^d)) \cdot |X| \cdot n)$ will allow us to find a pointer to $rep_{V_w}^d(X)$ in LR_{V_w}.

Proof. Algorithm 1 computes the list LR_{V_w} and the list LNR_{V_w} of all d-neighborhoods w.r.t. V_w of the members of LR_{V_w}. Before adding a representative R to the list LR_{V_w} we check if the list LNR_{V_w} contains the d-neighborhood $N_{V_w}^d(R)$. Therefore all elements of the list LR_{V_w} have different

d-neighbourhoods. All the representatives added to the list LR_{V_w} are also expanded by any of the vertices of V_w. Assume for contradiction that X is a minimal representative such that $N^d_{V_w}(X)$ is not in the list LNR_{V_w}. Then $\forall u \in X$ we have: $\forall Y \in LNR_{V_w} : X \setminus u \not\equiv Y$ since then $N^d_{V_w}(X)$ would have been added to LNR_{V_w}. Meaning that $N^d_{V_w}(X \setminus u)$ is not in LNR_{V_w} contradicting that X is minimal.

The total of number of representatives to be added to LR_{V_w} and number of d-neighborhoods added to LNR_{V_w} is $nec(\equiv^d_{V_w})$. The total number of possible representatives R' to be considered is $nec(\equiv^d_{V_w}) \cdot n$. Computing the union $R \cup \{v\}$ and the d-neighbouthood $N^d_{V_w}(R')$ can be done in $O(n)$ time by copying the d-neigborhood vector of R and updating the entries for vertices in $N(v) \cap \overline{V_w}$. If we realize the list LNR_{V_w} as a balanced binary search tree checking for containment can be done in $O(\log(nec(\equiv^d_{V_w})) \cdot n)$. Inserting into the list LR_{V_w} can be done in constant time. So in total the construction of LR_{V_w} and LNR_{V_w} takes time $O(nec(\equiv^d_{V_w}) \cdot \log(nec(\equiv^d_{V_w})) \cdot n^2)$.

Given a subset $X \subseteq V_w$ we can generate the d-neighborhood $N^d_{V_w}$ in $O(|X| \cdot n)$ time. Then we can binary search in the list LNR_{V_w} to find a pointer to the representative in time $O(\log(nec(\equiv^d_{V_w})) \cdot |X| \cdot n)$. □

5. Dynamic programming for vertex subset problems

We show in this section how to apply dynamic programming on a decomposition tree (T, δ) of a graph G to solve a (σ, ρ) locally checkable vertex subset optimization problem. Note that we do not assume any further information from the input of (T, δ) other than T being a tree with internal nodes of degree three and δ a bijection between its leaves and $V(G)$. As is customary, we let the algorithm follow a bottom-up traversal of T.

With each node w of T we associate a table data structure Tab_w that will store partial solutions to the problem we are solving. Note that we must satisfy the constraint imposed both by σ and by ρ and that we must account for the domination both 'from the inside', i.e. from V_w, and also the expectation 'from the outside', i.e. from $\overline{V_w}$. This motivates the following definitions.

Definition 8. Let G be a graph, $A \subseteq V(G)$, and $\mu \subseteq \mathbb{N}$. For $X \subseteq V(G)$, we say that X μ-dominates A if $\forall v \in A : |N(v) \cap X| \in \mu$. For $X \subseteq A$, $Y \subseteq \overline{A}$, the pair (X, Y) *σ, ρ-dominates* A if $(X \cup Y)$ σ-dominates X and $(X \cup Y)$ ρ-dominates $A \setminus X$.

Letting $d = d(\sigma, \rho)$ we note that for $Y \equiv^d_{\overline{A}} Y'$ we have $N^d_{\overline{A}}(Y) = N^d_{\overline{A}}(Y')$, i.e. the d-neighborhood of Y and Y' w.r.t. $\overline{A}$ are equal, see Definition 7. This proves the following lemma showing that σ, ρ-domination behaves well w.r.t. the $d(\sigma, \rho)$-neighbor equivalence relation.

Lemma 2. *Let G be a graph and $A \subseteq V(G)$ and σ, ρ finite or co-finite subsets of non-negative integers with $d(\sigma, \rho) = d$. Let $X \subseteq A$, $Y, Y' \subseteq \overline{A}$, and $Y \equiv^d_{\overline{A}} Y'$. Then (X, Y) σ, ρ-dominates A if and only if (X, Y') σ, ρ-dominates A.*

The index set of the table Tab_w at w will be $LR_{V_w} \times LR_{\overline{V_w}}$ and its contents is defined as follows.

Definition 9. Let *opt* stand for either function max or function min, depending on whether we are looking for a maximum or minimum (σ, ρ)-set, respectively. For every node w of T, for $X \subseteq V_w$ and $Y \subseteq \overline{V_w}$, let $R_X = rep^d_{V_w}(X)$ and $R_Y = rep^d_{\overline{V_w}}(Y)$. Let $d = d(\sigma, \rho)$. We define the contents of $Tab_w[R_X][R_Y]$ as:

$$Tab_w[R_X][R_Y] \stackrel{\text{def}}{=} \begin{cases} opt_{S \subseteq V_w}\{|S| : S \equiv^d_{V_w} R_X \text{ and } (S, R_Y) \ \sigma, \rho\text{-dominates } V_w\}, \\ -\infty \text{ if no such set } S \text{ exists and } opt = max, \\ +\infty \text{ if no such set } S \text{ exists and } opt = min. \end{cases}$$

Note that Tab_w has $nec(\equiv^d_{V_w}) \times nec(\equiv^d_{\overline{V_w}})$ entries, cf. Definition 5. At the root r of T the value of $Tab_r[rep^d_{V_r}(X)][\emptyset]$ (for all $X \subseteq V(G)$) will be the size of a maximum, resp. minimum, (σ, ρ)-set of G (cf. $V_r = V(G)$ and $\equiv^d_{V_r}$ has only one equivalence class).

For initialization, firstly, for every node w of T the value of every entry of Tab_w will be set to $+\infty$ or $-\infty$ depending on whether we are solving a minimization or maximization problem, respectively.

Updating the leaves: For a leaf l of T, we then perform the following update: letting $\delta(l) = v \in V(G)$, for every representative R w.r.t. $\equiv^d_{V(G)\setminus\{v\}}$, we set:

- If $|N(v) \cap R| \in \sigma$ then $Tab_l[\{v\}][R] = 1$.
- If $|N(v) \cap R| \in \rho$ then $Tab_l[\emptyset][R] = 0$.

Updating the internal nodes: In a bottom-up traversal of the tree T, for an inner node w of T with children a and b, the algorithm proceeds as follows.

- Loop over all triples $R_{\overline{w}} \in LR^d_{\overline{V_w}}$, $R_a \in LR^d_{V_a}$, $R_b \in LR^d_{V_b}$ and do:

  Compute $R_w = rep^d_{V_w}(R_a \cup R_b)$, $R_{\overline{a}} = rep^d_{\overline{V_a}}(R_b \cup R_{\overline{w}})$ and $R_{\overline{b}} = rep^d_{\overline{V_b}}(R_a \cup R_{\overline{w}})$

  Update $Tab_w[R_w][R_{\overline{w}}] = opt(Tab_w[R_w][R_{\overline{w}}], Tab_a[R_a][R_{\overline{a}}] + Tab_b[R_b][R_{\overline{b}}])$.

The following lemma will be useful in the correctness proof of this update.

Lemma 3. *For a graph G, let A, B, W be a 3-partitioning of $V(G)$, and let $S_a \subseteq A, S_b \subseteq B$ and $S_w \subseteq W$. $(S_a, S_b \cup S_w)$ σ, ρ-dominates A and $(S_b, S_a \cup S_w)$ σ, ρ-dominates B if and only if $(S_a \cup S_b, S_w)$ σ, ρ-dominates $A \cup B$.*

Proof. Let $S = S_a \cup S_b \cup S_w$. Consider $x \in S_a, x' \in A \setminus S_a$ and $y \in S_B, y' \in B \setminus S_b$. By Definition 8 $(S_a, S_b \cup S_w)$ σ, ρ-dominates A iff for every such x, x' we have $|N(x) \cap S| \in \sigma$ and $|N(x') \cap S| \in \rho$. Also, $(S_b, S_a \cup S_w)$ σ, ρ-dominates B iff for every such y, y' we have $|N(y) \cap S| \in \sigma$ and $|N(y') \cap S| \in \rho$.

Again, by Definition 8 $(S_a \cup S_B, S_w)$ σ, ρ-dominates $A \cup B$ iff for all $z \in S_a \cup S_b$ and $z' \in (A \cup B) \setminus (S_a \cup S_b)$ we have $|N(z) \cap S| \in \sigma$ and $|N(z') \cap S| \in \rho$, to finish the proof. □

Lemma 4. *The table of an inner node is updated correctly.*

Proof. Let node w have children a, b and assume Tab_a, Tab_b have been filled correctly. We show that after executing the update at node w the table Tab_w is filled according to Definition 9. We first show show that if $Tab_w[R_w][R_{\overline{w}}] = k$ then there exists $S_w \subseteq V_w$ with $|S_w| = k$ and $S_w \equiv^d_{V_w} R_w$ such that $(S_w, R_{\overline{w}})$ σ, ρ-dominates V_w in G. For this note that, based on the update step and assumed correctness of children tables, there must exist indices $R_a \in LR_{V_a}$ and $R_b \in LR_{V_b}$, with $S_a \equiv^d_{V_a} R_a$ and $S_b \equiv^d_{V_b} R_b$ such that $(S_a, R_{\overline{a}})$ σ, ρ-dominates V_a, and $(S_b, R_{\overline{b}})$ σ, ρ-dominates V_b, and $|S_a \cup S_b| = s$, and with $R_{\overline{a}} = rep^d_{\overline{V_a}}(R_b \cup R_{\overline{w}})$ and $R_{\overline{b}} = rep^d_{\overline{V_b}}(R_a \cup R_{\overline{w}})$. We claim that $S_a \cup S_b$ is the desired S_w. Since $(S_b \cup R_{\overline{w}}) \equiv^d_{\overline{V}_a} R_{\overline{a}}$ and $(S_a, R_{\overline{a}})$ σ, ρ-dominates V_a it follows from Lemma 2 that $(S_a, S_b \cup R_{\overline{w}})$ σ, ρ-dominates V_a. Likewise, $(S_b, S_a \cup R_{\overline{w}})$ σ, ρ-dominates V_b. We deduce from Lemma 3 that $(S_a \cup S_b, R_{\overline{w}})$ σ, ρ-dominates $V_a \cup V_b = V_w$. It remains to show that $S_a \cup S_b \equiv^d_{V_w} R_w$. By Observation 2 we have $R_a \equiv^d_{V_w} S_a$ and $R_b \equiv^d_{V_w} S_b$ so that $R_a \cup R_b \equiv^d_{V_w} S_a \cup S_b$. Since we assumed $R_a \cup R_b \equiv^d_{V_w} R_w$ we therefore have $S_a \cup S_b \equiv^d_{V_w} R_w$ as desired.

For the other direction, we need to show for every $R_w \in LR_{V_w}$ and $R_{\overline{w}} \in LR_{\overline{V_w}}$ that if there is a set $S_w \equiv^d_{V_w} R_w$ such that $(S_w, R_{\overline{w}})$ σ,ρ-dominates V_w, then after the update the value of $Tab_w[R_w][R_{\overline{w}}]$ is $\leq |S_w|$ if $opt = min$ and $\geq |S_w|$ if $opt = max$. Let $S_a = S_w \cap V_a$ and $S_b = S_w \cap V_b$. The algorithm loops over all triples of representatives: at some point it will check $(R_a, R_b, R_{\overline{w}})$, where $R_a = rep^d_{V_a}(S_a)$ and $R_b = rep^d_{V_b}(S_b)$. We know that $(S_a \cup S_b, R_{\overline{w}})$ σ,ρ-dominates V_w so it follows from Lemma 3 that $(S_a, S_b \cup R_{\overline{w}})$ σ,ρ-dominates V_a. Note that $R_{\overline{a}}$ as computed in the update satisfies $R_{\overline{a}} \equiv^d_{V_a} (S_b \cup R_{\overline{w}})$ so that it follows from Lemma 2 that $(S_a, R_{\overline{a}})$ σ,ρ-dominates V_a. Hence, after the update the value of $Tab_a[R_a][R_{\overline{a}}]$ will be $\leq |S_a|$ if $opt = min$ and $\geq |S_a|$ if $opt = max$. Arguing analogously we have that the value of $Tab_b[R_b][R_{\overline{b}}]$ will be $\leq |S_b|$ if $opt = min$ and $\geq |S_b|$ if $opt = max$. Thus, to conclude that the value of $Tab_w[R_w][R_{\overline{w}}]$ will be $\leq |S_a| + |S_b| = |S_w|$ if $opt = min$ and $\geq |S_a| + |S_b| = |S_w|$ if $opt = max$ all we need to show is that $R_w \equiv^d_{V_w} R_a \cup R_b$. By Observation 2 we have $R_a \equiv^d_{V_w} S_a$ and $R_b \equiv^d_{V_w} S_b$ so that $R_a \cup R_b \equiv^d_{V_w} S_a \cup S_b$. Since $S_w = S_a \cup S_b$ and we assumed $R_w \equiv^d_{V_w} S_w$ we therefore have $R_a \cup R_b \equiv^d_{V_w} R_w$ as desired. □

Theorem 1. *For every n-vertex graph G given along with a decomposition tree (T, δ), with $nec_d(T, \delta)$ the maximum $nec(\equiv^d_{V_w})$ and $nec(\equiv^d_{V_w})$ over all nodes w of this tree, any (σ,ρ)-vertex subset problem on G with $d = d(\sigma,\rho)$ can be solved in time $O(n^4 \cdot nec_d(T, \delta)^3)$.*

Proof. The correctness follows by structural induction on the tree T using Lemma 4 with the base case being the leaf initialization, so that at the root r of T the single index of the table Tab_r will contain the size of the optimal (σ,ρ) set in G. For complexity analysis, for every node w of T, we basically call the first computation of Lemma 1 once, then loop through every triplet $R_{\overline{w}}$, R_a, R_b of representatives, and there are at most $nec_d(T, \delta)^3$ such triplets. For each triplet we call the second computation of Lemma 1 three times, and since $|R_{\overline{w}}|, |R_a|, |R_b|$ and $\log(nec_d(T, \delta))$ all are at most n, we can perform the table update in $O(n^3)$ time. □

If the input graph G comes with a weight function on the vertices $w : V(G) \to \mathbb{R}$ we may wish to find the (σ,ρ) set with largest sum of weights, or with smallest sum of weights. This can be accomplished in the same runtime and requires only a very small change to the algorithm. For $S \subseteq V(G)$ let $w(S) = \Sigma_{v \in S} w(v)$. The tables must be defined to store the optimum value

over $w(S)$ rather than over $|S|$ and the algorithms remain the same apart from the leaf initialization.

6. Dynamic programming for vertex partitioning problems

We show in this section how to apply dynamic programming on a decomposition tree (T, δ) of a graph G to solve a locally checkable vertex partitioning problem defined by a degree constraint matrix D_q of finite and co-finite entries, see Definition 4. Let $d = d(D_q) = \max_{i,j} d(D_q[i,j])$.

Definition 10. Let G be a graph and let $A \subseteq V(G)$ be a vertex subset of G. Two q-tuples $(X_1, X_2, ..., X_q)$ and $(Y_1, Y_2, ..., Y_q)$ of subsets of A are d-neighbor equivalent w.r.t. A, denoted by $(X_1, X_2, ..., X_q) \equiv_A^{q,d} (Y_1, Y_2, ..., Y_q)$, if:

$$\forall i \forall v \in \overline{A} : \min(d, |N(v) \cap X_i|) = min(d, |N(v) \cap Y_i|)$$

Observation 3. *$(X_1, X_2, ..., X_q) \equiv_A^{q,d} (Y_1, Y_2, ..., Y_q)$ if and only if $\forall i X_i \equiv_A^d Y_i$. A consequence is that the number of equivalence classes of $\equiv_A^{q,d}$ is at most $nec(\equiv_A^d)$ to the power q.*

This Observation follows directly from Definitions 5 and 10. The dynamic programming algorithm will again follow a bottom-up traversal of T and maintain a table at each node of T with partial solutions. In the sequel we will define the values of Tab_w directly indexed by the equivalence classes. For this we need to first define representatives. For a node w of T, and $\mathcal{X} = (X_1, X_2, ..., X_q) : X_i \subseteq V_w$, we define $rep_{V_w}^{q,d}(\mathcal{X}) = (rep_{V_w}^d(X_1), rep_{V_w}^d(X_2), ..., rep_{V_w}^d(X_q))$.

Definition 11. Let G be a graph and $A \subseteq V(G)$. Let $\mathcal{X} = (X_1, X_2, ..., X_q) \in A^q$ and $\mathcal{Y} = (Y_1, Y_2, ..., Y_q) \in \overline{A}^q$. We say that $(\mathcal{X}, \mathcal{Y})$ D_q-dominates A if for all i, j we have that $(X_j \cup Y_j)$ $D_q[i,j]$-dominates X_i (cf. Definition 8).

Definition 12. For every node w of T, for every $\mathcal{X} = (X_1, X_2, ..., X_q) \in A^q$ and every $\mathcal{Y} = (Y_1, Y_2, ..., Y_q) \in \overline{A}^q$, let $\mathcal{R}_\mathcal{X} = rep_{V_w}^{q,d}(\mathcal{X})$ and $\mathcal{R}_\mathcal{Y} = rep_{V_w}^{q,d}(\mathcal{Y})$. We define the contents of $Tab_w[\mathcal{R}_\mathcal{X}][\mathcal{R}_\mathcal{Y}]$ as

$$Tab_w[\mathcal{R}_\mathcal{X}][\mathcal{R}_\mathcal{Y}] \stackrel{\text{def}}{=} \begin{cases} TRUE & \text{if } \exists \text{ partition } \mathcal{S} = (S_1, S_2, ..., S_q) \text{ of } V_w \text{ s.t:} \\ & \mathcal{S} \equiv_{V_w}^{q,d} \mathcal{R}_\mathcal{X} \text{ and } (\mathcal{S}, \mathcal{R}_\mathcal{Y}) \ D_q\text{-dominates } V_w \\ FALSE & \text{otherwise.} \end{cases}$$

It follows by definition that G has a D_q-partition if and only if some entry in the table at the root of T has value $TRUE$. The computation of the list of all representatives w.r.t. $\equiv^{q,d}_{V_w}$ is basically q times the one given in the previous section. The same situation holds for the computation of a representative from the input of a q-tuplet. Firstly, initialize all values in all tables to $FALSE$.

Updating the leaves: for a leaf l of T, like before, let $\delta(l) = v \in V(G)$ and let $A = \{v\}$. Firstly, there are q possible classes v could belong to in a q-partition of A (recall that empty sets are allowed). We call their representatives respectively $\mathcal{R}_{\mathcal{X}_1}$, $\mathcal{R}_{\mathcal{X}_2}$, ..., $\mathcal{R}_{\mathcal{X}_q}$. Secondly, for vertices in $B = V(G) \setminus \{v\}$ note that they are either neighbors of v or not. Hence we have at most $d+1$ choices (namely $0, 1, ..., d-1, \geq d$) for each of the q partition classes. (A consequence is that Tab_l has at most $q(d+1)^q$ entries.) For every representative $\mathcal{R}_{\mathcal{Y}} = (Y_1, Y_2, \ldots, Y_q)$ w.r.t. $\equiv^{q,d}_B$, we have that $(\mathcal{R}_{\mathcal{X}_i}, \mathcal{R}_{\mathcal{Y}})$ D_q-dominates $\{v\}$ if and only if $\forall j |N(l) \cap Y_j| \in D_q[i,j]$. Accordingly, we perform the following leaf update for every i and for every $\mathcal{R}_{\mathcal{Y}}$:

- $Tab_l[\mathcal{R}_{\mathcal{X}_i}][\mathcal{R}_{\mathcal{Y}}]$ is set to be $TRUE$ if and only if $\forall j$ $|N(v) \cap Y_j| \in D_q[i,j]$.

Updating the internal nodes: in the following, $\bigcup_q$ denotes the componentwise union of two q-tuples. For a node w with children a and b, the algorithm performs the following steps.

- Loop over all triples of representatives $\mathcal{R}_{\overline{w}}$ of $\equiv^{q,d}_{\overline{V_w}}$, $\mathcal{R}_{\overline{a}}$ of $\equiv^{q,d}_{\overline{V_a}}$, $\mathcal{R}_{\overline{b}}$ of $\equiv^{q,d}_{\overline{V_b}}$ and do:

  Compute $\mathcal{R}_w = rep^{q,d}_{V_w}(\mathcal{R}_a \bigcup_q \mathcal{R}_b)$, $\mathcal{R}_{\overline{a}} = rep^{q,d}_{\overline{V_a}}(\mathcal{R}_b \bigcup_q \mathcal{R}_{\overline{w}})$, $\mathcal{R}_{\overline{b}} = rep^{q,d}_{\overline{V_b}}(\mathcal{R}_a \bigcup_q \mathcal{R}_{\overline{w}})$

  If $Tab_w[\mathcal{R}_w][\mathcal{R}_{\overline{w}}] = FALSE$ then $Tab_w[\mathcal{R}_w][\mathcal{R}_{\overline{w}}] = Tab_a[\mathcal{R}_a][\mathcal{R}_{\overline{a}}] \wedge Tab_b[\mathcal{R}_b][\mathcal{R}_{\overline{b}}]$

Theorem 2. *For every n-vertex, m-edge graph G given along with a decomposition tree (T, δ) and an integer d. Deciding if G has a D_q-partition, with $d = \max_{i,j} d(D_q[i,j])$, can be solved in time $O(n^4 \cdot q \cdot nec_d(T,\delta)^{3q})$.*

Proof. The complexity analysis is very similar to the one given in Theorem 1, except we need to compute one reprsentative for each of the q parts, and uses

the bound in Lemma 3. The correctness proof follows the same style as the proof of Lemma 4, Some steps are not explained here because they were explained in Lemma 4.

For the correctness, let a, b be the children of w in T, assume Tab_a and Tab_b are correct.
($\Rightarrow$) For this direction of the proof we have that $Tab_w[\mathcal{R}_w][\mathcal{R}_{\overline{w}}] = TRUE$. Then there must exist some q-tuples $\mathcal{R}_a, \mathcal{R}_b$ such that $Tab_a[\mathcal{R}_a][\mathcal{R}_{\overline{a}}] = TRUE$ and we have $Tab_b[\mathcal{R}_b][\mathcal{R}_{\overline{b}}] = TRUE$, where $\mathcal{R}_{\overline{a}} = rep^d_{V_a}(\mathcal{R}_b \bigcup_q \mathcal{R}_{\overline{w}})$ and $\mathcal{R}_{\overline{b}} = rep^d_{V_b}(\mathcal{R}_a \bigcup_q \mathcal{R}_{\overline{w}})$. Hence there exists $\mathcal{S}_a$ partition of V_a and $\mathcal{S}_b$ partition of V_b such that $(\mathcal{S}_a, \mathcal{R}_{\overline{a}})$ D_q-dominates V_a $(\mathcal{S}_b, \mathcal{R}_{\overline{b}})$ D_q-dominates V_b. This means that $\forall i, j : (S_{a_j} \cup R_{\overline{a}_j})$ $D_q[i, j]$-dominates S_{a_i} and $\forall i, j : (S_{b_j} \cup R_{\overline{b}_j})$ $D_q[i, j]$-dominates S_{b_i}. It then follows that: $\forall i, j : (S_{a_j} \cup S_{b_j} \cup R_{\overline{w}_j})$ $D_q[i, j]$-dominates S_{a_i} and $\forall i, j : (S_{a_j} \cup S_{b_j} \cup R_{\overline{w}_j})$ $D_q[i, j]$-dominates S_{b_i}. It then follows that: $\forall i, j : (S_{w_j} \cup R_{\overline{w}_j})$ $D_q[i, j]$-dominates S_{w_i}. Which means $(\mathcal{S}, \mathcal{R}_{\overline{w}})$ D_q-dominates V_w.

($\Leftarrow$) For this direction of the proof we have that there exists a partition $\mathcal{S} = (S_1, ...S_q)$ of V_w such that: $(\mathcal{S}, \mathcal{R}_{\overline{w}})$ D_q-dominates V_w. This means that $\forall i, j : (S_{w_j} \cup R_{\overline{w}_j})$ $D_q[i, j]$-dominates S_{w_i}. Let $\mathcal{S}_a, \mathcal{S}_b$ be the componentwise intersection of $\mathcal{S}_w$ with V_a and V_b respectively. We then have: $\forall i, j : (S_{w_j} \cup R_{\overline{w}_j})$ $D_q[i, j]$-dominates S_{a_i} and $\forall i, j : (S_{w_j} \cup R_{\overline{w}_j})$ $D_q[i, j]$-dominates S_{b_i}. Hence $\forall i, j : (S_{a_j} \cup S_{b_j} \cup R_{\overline{w}_j})$ $D_q[i, j]$-dominates S_{a_i} and $\forall i, j : (S_{a_j} \cup S_{b_j} \cup R_{\overline{w}_j})$ $D_q[i, j]$-dominates S_{a_i}. Let $\mathcal{R}\overline{a} = rep^d_{V_a}(\mathcal{S}_b \bigcup_q \mathcal{R}_{\overline{w}})$ and $\mathcal{R}_{\overline{b}} = rep^d_{V_b}(\mathcal{S}_a \bigcup_q \mathcal{R}_{\overline{w}})$ then $\forall i, j : (S_{a_j} \cup R_{\overline{a}_j})$ $D_q[i, j]$-dominates S_{a_i} and $\forall i, j : (S_{b_j} \cup R_{\overline{b}_j})$ $D_q[i, j]$-dominates S_{b_i}. Let $\mathcal{R}_a = can^d_{V_a}(\mathcal{S}_a)$ and $\mathcal{R}_b = can^d_{V_b}(\mathcal{S}_b)$ then $Tab_a[\mathcal{R}_a][\mathcal{R}_{\overline{a}}] = TRUE$ and $Tab_b[\mathcal{R}_b][\mathcal{R}_{\overline{b}}] = TRUE$. Since the algorithm goes through all triples, it will at some point go through $(\mathcal{R}_a, \mathcal{R}_b, \mathcal{R}_{\overline{w}})$. And it will set $Tab_w[\mathcal{R}_w][\mathcal{R}_{\overline{w}}]$ to true, once it is true it will never change.

By induction all tables will be correct. □

7. Runtime expressed by boolean-width

We give an alternative definition of boolean-width, equivalent to the standard one [9].

Definition 13 (Boolean-width)**.** Let G be a graph and $A \subseteq V(G)$. The $bool\text{-}dim : 2^{V(G)} \to \mathbb{R}$ function of a graph G is defined as

$$bool\text{-}dim(A) = \log_2(nec(\equiv^1_A))$$

Let (T,δ) be a rooted decomposition tree of G. The boolean-width of (T,δ) is

$$boolw(T,\delta) = \max_{a\in V(T)} \{bool\text{-}dim(V_a)\}$$

The boolean-width of a graph G is the minimum boolean-width over all its rooted decomposition trees

$$boolw(G) = \min_{(T,\delta) \text{ of } G} \{boolw(T,\delta)\}$$

The classes of $\equiv_A^1$ are in a bijection with what is called the Boolean row space of the bipartite adjacency matrix of the graph on edges with exactly one endpoint in A, i.e. the set of vectors that are spanned via Boolean sum (1+1=1) by the rows of this matrix, see the monograph [19] on Boolean matrix theory. ¿From this bijection we get that the *bool-dim* function is symmetric, see [19, Theorem 1.2.3]. In particular, for any node w of T we have $nec(\equiv_{V_w}^1) = nec(\equiv_{\overline{V_w}}^1)$.

Lemma 5. *Let G be a graph and $A \subseteq V(G)$. Then, for every $X \subseteq A$, there is $R \subseteq X$ such that $R \equiv_A^d X$ and $|R| \leq d \cdot bool\text{-}dim(A)$. Moreover, $nec(\equiv_A^d) \leq 2^{d\cdot bool\text{-}dim(A)^2}$.*

Proof. We prove the first statement, namely bounding $|R|$ by induction on d. For $d \leq 1$ the lemma follows from Lemma 6 in [9]. Let $S \subseteq X$ be an inclusion minimal set such that $N(S) \cap \overline{A} = N(X) \cap \overline{A}$ e.g. $S \equiv_A^1 X$. Hence from this Lemma with $d = 1$ we have that $|S| \leq bool\text{-}dim(A)$. Assume the induction hypothesis true up to $d-1$, then we show it true for d. By induction hypothesis there exists $R' \subseteq (X \setminus S)$ such that $R' \equiv_A^{d-1} (X \setminus S)$ and $|R'| \leq bool\text{-}dim(A) \cdot (d-1)$. Thus it is enough to show $R = R' \cup S \equiv_A^d X$.

We partition the nodes of $\overline{A}$ into (P,Q) such that $\forall v \in P$, we have $|N(v) \cap (X \setminus S)| = |N(v) \cap R'|$ and $\forall v \in Q$, we have $|N(v) \cap (X \setminus S)| \geq d-1$ and $|N(v) \cap R'| \geq d-1$. For every vertex $v \in P$, since $S \cap R' = \emptyset$ and $S \subseteq X$, we know $|N(v) \cap X| = |N(v) \cap (X \setminus S)| + |N(v) \cap S| = |N(v) \cap R'| + |N(v) \cap S| = |N(v) \cap R|$. We have $N(X) = N(S)$ and since $d > 1$ we have $Q \subseteq N(S)$. For every vertex $v \in Q$, since $|N(v) \cap (X \setminus S)| \geq d-1$ we get $|N(v) \cap X| \geq d$ and since $|N(v) \cap R'| \geq d-1$ we get $|N(v) \cap R| \geq d$. Since (P,Q) is a partition we get $R \equiv_A^d X$ and $|R| \leq bool\text{-}dim(A) \cdot d$, thus by induction the lemma holds for all d.

To bound the number of equivalence classes $nec(\equiv_A^d)$ we know from the previous arguments that we only need to find the equivalence classes among the subsets of A of size at most $d \cdot bool\text{-}dim(A)$. Two vertices $x, x' \in A$ are

twins across $\{A, \overline{A}\}$ if $N(x) \cap \overline{A} = N(x') \cap \overline{A}$. Let H be obtained from the bipartite subgraph of G with color classes $A, \overline{A}$ after doing twin contraction of all twins. We know that every node of $V(H) \cap A$ has a unique neighborhood, hence $|V(H) \cap A| \leq 2^{bool\text{-}dim(A)}$. For any subset of A there is a multiset of $V(H) \cap A$ with the same d-neighbourhood, and a trivial bound on number of multisets of size $d \cdot bool\text{-}dim(A)$ of $V(H) \cap A$ gives us: $nec(\equiv_A^d) \leq 2^{d \cdot bool\text{-}dim(A)^2}$. □

Together with Theorem 1 we get the following.

Corollary 1. *For every graph G given along with a decomposition tree (T, δ) any (σ, ρ)-vertex subset problem on G with $d = d(\sigma, \rho)$ can be solved in time $O(n^4 \cdot q \cdot 2^{3d \cdot boolw(T,\delta)^2})$.*

Together with Theorem 2 and Observation 3 we get the following.

Corollary 2. *For every graph G given along with a decomposition tree (T, δ), deciding if G has a D_q-partition, with $d = \max_{i,j} d(D_q[i, j])$, can be done in time $O(n^4 \cdot 2^{3qd \cdot boolw(T,\delta)^2})$.*

8. Conclusions and Open Problems

The runtime of the algorithms given here for (σ, ρ)-problems and D_q-problems have the square of the boolean-width $boolw$ as a factor in the exponent, i.e. $O(boolw^2)$ in the exponent. For problems where $d = 1$ we can in fact improve this to a factor linear in the exponent [9], but that requires a special focus on these cases. We hope that also for the other problems (with any constant value of d) we could get runtimes with a better exponential factor, say $O(boolw \log boolw)$ in the exponent or maybe even linear. We must then improve the bound in Lemma 5. For the linear bound we must show that the number of d-neighborhood equivalence classes is no more than the number of 1-neighborhood equivalence classes raised to some function of d. This runtime question can also be formulated as a purely algebraic one. First generalize the concept of Boolean sums ($1 + 1 = 1$) to d-Boolean sums ($i + j = \min(i + j, d)$). For a Boolean matrix A let $R_d(A)$ be the set of vectors over $\{0, 1, ..., d\}$ that arise from all possible d-Boolean sums of rows of A. To get $O(boolw \log boolw)$ in the exponent it would suffice to show that there is a function f such that $|R_d(A)| \leq |R_1(A)|^{f(d) \log\log |R_1(A)|}$.

References

[1] R. Belmonte, M. Vatshelle, Graph classes with structured neighborhoods and algorithmic applications, Theoretical Computer Science Submitted to this issue (?) (2012) ?

[2] J. Fiala, J. Kratochvíl, Locally constrained graph homomorphisms - structure, complexity, and applications, Computer Science Review 2 (2008) 97–111.

[3] R. G. Downey, M. R. Fellows, Parameterized Complexity, Springer Verlag, 1999.

[4] J. Flum, M. Grohe, Parameterized Complexity Theory, Springer Verlag, 2006.

[5] P. Hliněný, S.-i. Oum, D. Seese, G. Gottlob, Width parameters beyond tree-width and their applications, The Computer Journal 51 (3) (2008) 326–362.

[6] B. Courcelle, J. A. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique width, Theory of Computing Systems 33 (1999) 125–150.

[7] J. A. Telle, A. Proskurowski, Algorithms for vertex partitioning problems on partial k-trees, SIAM Journal on Discrete Mathematics 10 (4) (1997) 529–550.

[8] M. U. Gerber, D. Kobler, Algorithms for vertex-partitioning problems on graphs with fixed clique-width, Theoretical Computer Science 299 (1-3) (2003) 719–734.

[9] B.-M. Bui-Xuan, J. A. Telle, M. Vatshelle, Boolean-width of graphs, Theoretical Computer Science 412 (39) (2011) 5187–5204.

[10] I. Adler, B.-M. Bui-Xuan, Y. Rabinovich, G. Renault, J. A. Telle, M. Vatshelle, On the boolean-width of a graph: Structure and applications, in: Proceedings of WG, 2010, pp. 159–170.

[11] M. Vatshelle, New width parameters of graphs, Ph.D. thesis, University of Bergen (2012).

[12] E. M. Hvidevold, S. Sharmin, J. A. Telle, M. Vatshelle, Finding good decompositions for dynamic programming on dense graphs, in: Proceedings of IPEC, 2011, pp. 219–231.

[13] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Co., 1978.

[14] J. Kratochvíl, Perfect Codes in General Graphs, Academia Praha, 1991.

[15] T. J. Schaefer, The complexity of satisfiability problems, Proceedings of STOC 10 (1978) 216–226.

[16] P. Heggernes, J. A. Telle, Partitioning graphs into generalized dominating sets, Nordic Journal of Computing 5 (2) (1998) 128–142.

[17] J. A. Telle, Vertex partitioning problems: Characterization, complexity and algorithms on partial k-trees, Ph.D. thesis, University of Oregon (1994).

[18] O. Amini, I. Sau, S. Saurabh, Parameterized complexity of the smallest degree-constrained subgraph problem, in: Proceedings of IWPEC, 2008, pp. 13–29.

[19] K. H. Kim, Boolean matrix theory and applications, Marcel Dekker, 1982.

Feedback Vertex Set on Graphs of Low Clique-width[☆]

Binh-Minh Bui-Xuan[a], Ondřej Suchý[b,1], Jan Arne Telle[a], Martin Vatshelle[a,*]

[a] *Department of Informatics, University of Bergen, Norway*
[b] *Universität des Saarlandes, Saarbrücken, Germany*

Abstract

The Feedback Vertex Set problem asks whether a graph contains q vertices meeting all its cycles. This is not a local property, in the sense that we cannot check if q vertices meet all cycles by looking only at their neighbors. Dynamic programming algorithms for problems based on non-local properties are usually more complicated. In this paper, given a graph G of clique-width cw and a cw-expression of G, we solve the Minimum Feedback Vertex Set problem in time $O(n^2 2^{O(cw \log cw)})$. Our algorithm applies dynamic programming on a so-called k-module decomposition of a graph, as defined by Rao [30], which is easily derivable from a k-expression of the graph. The related notion of module-width of a graph is tightly linked to both clique-width and NLC-width, and in this paper we give an alternative equivalent characterization of module-width.

Keywords: feedback vertex set, clique-width, graph algorithms
2000 MSC: 05C85, 68R10

1. Introduction

The problem of finding a minimum Feedback Vertex Set (FVS) in a graph, i.e. the smallest set of vertices whose removal results in a graph that has no cycles, has many applications. For example to optical networks [22], circuit testing, deadlock resolution, analyzing manufacturing processes and computational biology (see [10] and its bibliography). It is one of the classical NP-complete problems from the 1972 list of Karp [21] and has been extensively

[☆]Supported by the Norwegian Research Council, project PARALGO.
[*]Corresponding author.
Email addresses: buixuan@ii.uib.no (Binh-Minh Bui-Xuan), suchy@mmci.uni-saarland.de (Ondřej Suchý), telle@ii.uib.no (Jan Arne Telle), vatshelle@ii.uib.no (Martin Vatshelle)
[1]Supported by the DFG Cluster of Excellence on Multimodal Computing and Interaction, DFG project DARE (GU 1023/1-2) and by grant 1M0545 of the Czech Ministry of Education. Most of the work done while on Dep. of Applied Mathematics and Inst. for Theoretical Computer Science, Charles University, Prague, Czech Republic and while visiting the University of Bergen.

studied from many viewpoints, including linear programming [7], approximation algorithms [2, 13, 16, 22], exact algorithms [14] and parametrized complexity [6, 10, 17, 25, 29].

The minimum FVS problem is 2-approximable in polynomial time [1]. The fastest exact algorithm has runtime $O(1.7548^n)$ [14]. The fastest FPT (Fixed Parameter Tractable) algorithm when parametrized by the size q of the FVS has runtime $O(5^q qn^2)$ [6]. These algorithmic results are quite strong, but are not useful for cases of input graphs having a large number of vertices n, and a large minimum FVS q, if we want the actual smallest FVS. For such cases we may instead hope that the input graph has a bounded width parameter. For example, if G is a planar graph of treewidth tw then Kloks et al [23] give a dynamic programming algorithm solving minimum FVS on G in time $O(2^{O(tw \log tw)}n)$. Recently a randomized algorithm with runtime $O(3^{tw}n^{O(1)})$ for minimum FVS with one-sided error of probability at most 0.5 was developed [9]. It is still an open problem whether there is a deterministic $O(2^{O(tw)}n^{O(1)})$ algorithm for FVS, even though such algorithms exist for a large variety of NP-hard problems. However, for minimum FVS it would require a small breakthrough to get such an algorithm. One reason for this is that FVS is not a locally checkable property, in the sense that given q vertices we cannot check that they form an FVS simply by looking at their neighbors. One also has to consider paths between pairs of vertices. The same issue arises when the problem is parametrized by q the size of the FVS, but Dehne et al. [10] gave an $O(2^{O(q)}n)$ algorithm using the technique of iterative compression and the running time for this parameterization was improved in a long series of papers. In this paper we do not aim to prolong this series. Instead, we consider the problem on graphs of clique-width cw. This graph class encompasses large classes of graphs of unbounded treewidth, and for which powerful algorithmic results are known. Note that bounded clique-width does not imply bounded treewidth, hence we can not directly translate FPT algorithms parametrized by treewidth to FPT algorithms parametrized by clique-width. For instance, we have that any graph problem expressible in MSO_1-logic, as is the case with minimum FVS, is FPT when parametrized by clique-width (roughly, apply [20], then [27, Proposition 6.3], then [8]). Since FVS can be expressed in MSO_1-logic it follows that FVS is FPT parameterized by clique-width, however the running time will contain a tower of 2's, and we are not aware of any MSO_1 formulation which would lead to a tower with less than 4 levels. In this paper we are interested in as low exponential dependency on cw as possible, and for this we need to use a specially designed dynamic programming algorithm.

The complement of a FVS is a vertex subset inducing a forest, and solving such tree-like problems using dynamic programming based on clique-width are usually more complicated than dynamic programming based on treewidth. An $O^*(2^{cw^2 \log cw})$ algorithm was given in [5], and an $O^*(2^{cw^2})$ algorithm was given in [15] (this algorithm also works for the lower parameter rank-width). The first algorithm is an extension of the treewidth algorithm with the key observation that we only need to consider cw^2 components of the complement of FVS. The second improves this by cleverly reducing the number of considered components

to cw.

In this paper we give an $O^*(2^{cw \log cw})$ algorithm. To this end, we use the trick of considering only cw components, but we also make use of a technique taking into account an "expectation from the outside" during the bottom-up computation of the dynamic programming. This technique was introduced in [3] where it was used for finding a minimum dominating set along a so-called H-join decomposition, and later also for FVS in [15]. Roughly, when operating on some reduced instance – *e.g.*, some subgraph $G[A]$ induced by vertex subset A – we not only compute solutions – *e.g.*, FVS, dominating sets, etc. – depending on $G[A]$, but also those solutions satisfying specific constraints depending on $G[V(G) \setminus A]$. As opposed to classical dynamic programming, a consequence of "expecting from the outside" will be that we no longer partition the set of possible solutions (of $G[A]$) into equivalence classes with a one-to-one correspondence between such classes and indices of the table (data-structure for dynamic programming). Instead, each possible solution now can influence several indices of the table.

The exponential dependency on clique-width of our algorithm matches asymptotically the current best known algorithms based on treewidth. More precisely, our algorithm finds a minimum FVS on a graph G of clique-width cw in time $O(2^{O(cw \log cw)} n^2)$, when given a cw-expression of G which is a decomposition of the graph showing that it has clique-width cw.

Clique-width is related to the notion of NLC-width of a graph [12] with which it shares most properties but we have chosen to use clique-width in this paper simply because that notion is more well known. More specifically, our algorithm applies dynamic programming on a so-called k-module decomposition of a graph, as defined by Rao [30], which is easily derivable from a k-expression of the graph. The related notion of module-width of a graph is tightly linked to both clique-width and NLC-width, and in this paper we give an alternative equivalent characterization of module-width.

2. Framework

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. Consider the following unifying decomposition framework for several decomposition schemes. A binary tree is a rooted tree where every internal node has exactly two children.

Definition 2.1 (Decomposition tree)**.** A *rooted decomposition tree* of a graph G is a pair (T, δ) where T is a binary tree having $n = |V(G)|$ leaves and δ is a bijection between the vertices of G and the leaves of T.

Roughly, trees with their leaves in a bijection with the vertices of G are important for techniques like divide-and-conquer or dynamic programming since they show how to "divide" the graph instance into several sub-instances and recurse. Clearly, any tree with the right number of leaves and a bijection can be considered as a decomposition tree. Then, a common technique to select those that are more suited for some task is to use an evaluating function.

Definition 2.2 (Decomposition and width parameters)**.** Let G be a graph, $f : 2^{V(G)} \to \mathbb{R}$ a function assigning a non-negative real value to subsets of $V(G)$, and (T, δ) a rooted decomposition tree of G. For every node u of T, let V_u denote the vertex subset of G induced by the leaves of the subtree of T rooted at u. The *f-width of* (T, δ) is the maximum value of $f(V_u)$, taken over every node u of T. An *optimal f-decomposition* of G is a rooted decomposition tree of G having minimum f-width. The *f-width of G* is the f-width of an optimal f-decomposition of G.

If f is also required to be symmetric, namely that $f(V_u) = f(V(G) \setminus V_u)$ for every V_u, then the above framework, up to unrooting the tree T and setting $f(V(G)) = f(\emptyset) = 0$, is equivalent to the one developed for the study of branch decomposition of symmetric and submodular functions (see, *e.g.*, [27, Section 2] for a short and recent introduction). This includes the branch-width [31], rank-width [27], and boolean-width [4] decompositions of graphs. On the other hand, rooted decomposition trees as defined here can be used for situations where the symmetry does not occur, for instance with a branch-like decomposition of a submodular function that is not necessarily symmetric, a clique-width or NLC-width expression, or a so-called k-module decomposition as will be presented below.

For an efficient complexity analysis of the algorithm that will be described in Section 4, we will be interested in the following definition of f-width, so-called module-width in [26, 32].

Definition 2.3. Let G be a graph and let $X \subseteq V(G)$ be a vertex subset. A subset $A \subseteq X$ is a *twin set of X* if, for every $z \in V(G) \setminus X$ and pair of vertices $x, y \in A$, we have x adjacent to z if and only if y adjacent to z. A twin set A is a *twin-class of X* if A is maximal. The set of all twin-classes of X forms a partition of X, that we call the *twin-class partition of X*.

Definition 2.4 (Module-width)**.** The function $\mu_G : 2^{V(G)} \to \mathbb{N}$ is defined such that $\mu_G(X)$ is the number of twin-classes of X in the graph G. The *module-width decompositions and parameters* of G refer to those of Definition 2.2 when $f = \mu_G$. The μ_G-width of G will be called the *module-width* of G and denoted by $\mu w(G)$.

The above terminology of module-width is according to the name given to an equivalent notion that was mentioned in [26, last two pages] and formalized in [32, Section 6.1.2]. Indeed, one can use a similar decomposition framework, so-called k-module decomposition, in order to result in the same parameter as follows.

Definition 2.5. ([26, 32]) Let G be a graph. A vertex subset $X \subseteq V(G)$ is a *k-module* if there exists a partition of X into k twin sets. G is a *k-module decomposable graph* if there is a rooted decomposition tree (T, δ) such that every vertex subset of G that is induced by the leaves of some subtree of T is also a k-module of G. The *module-width* of G is the minimum integer k such that G is k-module decomposable.

Definitions 2.4 and 2.5 both lead to the same notion of module-width thanks to the following simple observations. Firstly, if X is a k-module, then it is also a $(k+1)$-module as long as $k+1 \leq |X|$. Secondly, the minimum number k such that X is a k-module is exactly $\mu_G(X)$.

Clique-width and NLC-width expressions are constructions of a graph using logic operations. For a proper introduction to clique-width and NLC-width refer to [8, 12]. The underlying graphs of clique-width and NLC-width expressions are rooted trees where every internal node has at most two children and where the leaves are in a bijection with the vertices of the graph. This, up to contracting one child nodes, can be seen as a rooted decomposition tree. The clique-width $cw(G)$ and the NLC-width $nlc\text{-}w(G)$ of a graph G are parameters of G having powerful algorithmic properties. For instance, we have that any graph problem expressible in MSO_1-logic is FPT when parametrized by one of these two parameters (roughly, apply [20], then [27, Proposition 6.3], then [8]). They are closely linked to module-width by the following property.

Theorem 2.6. *([32, Theorem 6.6]) We have for any graph G that*

$$\mu w(G) \leq nlc\text{-}w(G) \leq cw(G) \leq 2\mu w(G).$$

We now give an alternative viewpoint of these module-width decompositions, that will link module-width to the so-called H-join decomposition framework [3] in an unexpected way.

Definition 2.7. Let H be a bipartite graph with color classes V_1 and V_2, thus $V(H) = V_1 \cup V_2$. Let G be a graph and $X \subseteq V(G)$ a subset of its vertices. We say that G is an *H-join across the ordered cut* $(X, V(G) \setminus X)$ if there exists a partition of X with set of classes $\mathcal{P}$ and a partition of $V(G) \setminus X$ with set of classes $\mathcal{Q}$, and injective functions $f_1 : \mathcal{P} \to V_1$ and $f_2 : \mathcal{Q} \to V_2$, such that for any $x \in X$ and $y \in V(G) \setminus X$ we have x adjacent to y in G if and only if x belongs to a class P_i of $\mathcal{P}$ and y to a class Q_j of $\mathcal{Q}$ with $f_1(P_i)$ adjacent to $f_2(Q_j)$ in H.

We will abusively refer to ordered cuts simply by cuts. Twins in a bipartite graph are vertices in the same color class having exactly the same neighborhood. A *twin contraction* is the deletion of a vertex when it has a twin. Notice that H-joins are insensitive to twin contractions: if H' is obtained from H by a twin contraction then G is an H-join across some cut if and only if G is an H'-join across the same cut. Note also that we do allow a twin-free bipartite graph to have one isolated vertex in each color class. We model the joining in module-width decompositions by using the following graph.

Definition 2.8. For a positive integer k we define a bipartite graph Y_k having for each integer i of $\{1, 2, \ldots, k\}$ a vertex $a_i \in A$ and having for each subset S of $\{1, 2, \ldots, k\}$ a vertex $b_S \in B$, with $V(Y_k) = A \cup B$. This gives k vertices in A and 2^k vertices in B. A vertex a_i is adjacent to a vertex b_S if and only if $i \in S$.

Lemma 2.9. *Let k be an integer, let H be a bipartite graph over color classes $V_1 \cup V_2$ with $|V_1| \leq k$. Then, applying successive twin contractions in H until*

stability will always result in a graph that is isomorphic to an induced subgraph of Y_k.

Proof. Just give an arbitrary ordering over the vertices of $V_1 = (v_1, v_2, \ldots, v_l)$, and map them to the l first vertices $a_1, a_2, \ldots, a_l$ of Y_k, respectively (note that $l \leq k$ by hypothesis). Then, for every vertex $u \in V_2$ of H, let $S = \{i : v_i \in N(u)\}$, and map u to vertex b_S of Y_k. Hence, H is an induced subgraph of Y_k. Now, applying twin contractions on a subgraph of Y_k will always result in another induced subgraph of Y_k. □

Corollary 2.10. *The function μ_G of Definition 2.4 is exactly equal to the function η_G defined for all $X \subseteq V(G)$ by*

$$\eta_G(X) = \min\{k : G \text{ is a } Y_k\text{-join across the cut } (X, V(G) \setminus X)\}.$$

Proof. In Definition 2.7 of an H-join across $(X, V(G) \setminus X)$, if we consider as joining partition of X the twin-class partition of X, then H is a bipartite graph having exactly $\mu_G(X)$ vertices on one of its color class. Lemma 2.9 then allows to conclude. □

3. Computing the twin-classes

In the next section we will give a dynamic programming algorithm to solve the feedback vertex set problem on an input made by an n-vertex m-edge graph G and one of its rooted decomposition tree (T, δ). Note that the underlying graph of a clique-width expression of G is a rooted tree where each internal node has at most two children, and the leaves are in a bijection with the vertices of G. Contracting the internal nodes having one child will result in a rooted decomposition tree of G. Moreover, it can also be obtained from the proof of Theorem 2.6 that the module-width of this rooted decomposition tree is at most the clique-width of the clique-width expression. Consequently, if the input to our algorithm is the graph G and a clique-width k expression of G, we can transform them in a straightforward manner to an input made of G and one of its rooted decomposition tree of module-width at most the value of k.

For every internal node u of T with V_u being the vertex subset of G induced by the subtree of T rooted at u, we will need to compute the twin-classes of V_u as mentioned in the definition of μ_G in Definition 2.4. In this section, we will describe how to perform such a computation for every internal node u of T, in global running time $O(n^2)$.

We will use the so-called partition refinement algorithmic technique (refer to, *e.g.*, [18, 28] for details). Partitions will be represented by double-linked lists. A *refinement operation* of a partition $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_k)$ of V_u using $A \subseteq V_u$ as pivot is the act of splitting every Q_i into $Q_i \cap A$ and $Q_i \setminus A$. The output of a refinement operation can be of two types. It can be made of one partition of V_u which is the result of removing all empty sets from $(Q_1 \cap A, Q_1 \setminus A, Q_2 \cap A, Q_2 \setminus A, \ldots, Q_k \cap A, Q_k \setminus A)$. We refer to these as one-to-one refinements. It can also be composed of two partitions (one of A and one of $V_u \setminus A$) which

result from removing all empty sets from $(Q_1 \cap A, Q_2 \cap A, \ldots, Q_k \cap A)$ and $(Q_1 \setminus A, Q_2 \setminus A, \ldots, Q_k \setminus A)$. We refer to these as one-to-two refinements. With the appropriate data structure, all these types of refinement operations can be implemented to run in $O(|A|)$ time for each operation (refer to, *e.g.*, [18] for details).

A simple way to compute the twin-class partition of V_u is to initialize $\mathcal{Q} = (V_u)$ and, for every vertex $z \in V(G) \setminus V_u$, perform an one-to-one refinement of $\mathcal{Q}$ using the neighborhood $N(z)$ of z as pivot. The correctness follows directly from the definition of twin-classes. This computation would have $O(m)$ runtime for each internal node u of T, hence a global $O(nm)$ runtime.

The main idea to reduce this runtime is to observe that, in the above operations, we can use $N(z) \cap V_u$ as pivot instead of $N(z)$ (for every $z \in V(G) \setminus V_u$) without modifying the refined partition of each step. However, the sum over every possible V_u and $z \in V(G) \setminus V_u$ of the value $|N(z) \cap V_u|$ might still be large. We will observe a second fact. For a partition $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_k)$ of X and a subset $Y \subseteq X$, we denote by $\mathcal{Q}[Y]$ the partition of Y which results from removing all empty sets from $(Q_1 \cap Y, Q_2 \cap Y, \ldots, Q_k \cap Y)$.

Remark 3.1. *Let w be an internal node of T with children a and b. Let V_w, V_a, and V_b be the vertex subsets of G induced by the leaves of the subtrees of T rooted at w, a, and b, respectively. Let $\mathcal{Q}_w = (Q_w(1), Q_w(2), \ldots, Q_w(h_w))$ be the twin-class partition of V_w. Then, initializing $\mathcal{Q} = \mathcal{Q}_w[V_a]$ and refining $\mathcal{Q}$ using $N(z) \cap V_a$ as pivot for all $z \in V_b$ will result to the twin-class partition of V_a.*

Basically, the algorithmic difference given by the remark is that we can now be restricted to $z \in V_b$ instead of using all $z \in V(G) \setminus V_a$ as before. The main point is that the sum over every possible V_a and $z \in V_b$ of the value $|N(z) \cap V_a|$ will be at most twice the value $n + m$ (every edge of G appears at most twice in the sum). We now implement Remark 3.1.

First of all, the bottleneck of using $N(z) \cap V_a$ as pivot will be that, unlike the case with $N(z)$ which can be read simply in the adjacency list of G, we will need to compute $N(z) \cap V_a$ for every possible V_a and z. We do this as a preprocessing step as follows.

We prepare the tree T as described in [19] so that afterwards we can, given two leaves x and y of T, compute the lowest common ancestor w of x and y in T in $O(1)$ time. This can also be done in such a way that, if a and b denote the children of w, then we can in $O(1)$ time decide whether x is a descendent of a or it is a descendent of b. Then, for every internal node w of the tree T, with children a and b, we initialize two tables $N_w^{b \to a}$ and $N_w^{a \to b}$ that will contain, for every vertex z in V_b (resp. V_a), the neighborhood of z in V_a (resp. V_b). Now, we scan through every edge xy of G and compute the lowest common ancestor w of x and y, as well as the children a and b of w such that x is a descendent of a, and finally add x to $N_w^{b \to a}[y]$ and y to $N_w^{a \to b}[x]$. Clearly, after scanning all edges of G, we have that $N_w^{b \to a}[z] = N(z) \cap V_a$ for all w, a, b, and z. This preprocessing takes $O(m)$ time.

We come to the proper computation of the twin-class partitions. The twin-class partition associated to the root of T only has one class, which is $V(G)$. Suppose that we have computed the twin-class partition $\mathcal{Q}_w$ of an internal node w having children a and b. This partition $\mathcal{Q}_w$ is stored in a double-linked list w.r.t. the data structure used for partition refinement. Basically, the following operations can operate directly on this data structure, if we allow ourselves to modify the double-linked list. However, the information on the twin-classes of V_w would then be lost. For this reason, before continuing, we duplicate the data structure of $\mathcal{Q}_w$ so that we store the twin-classes of V_w in a private place of node w. Then, we can compute $\mathcal{Q}_w[V_a]$ and $\mathcal{Q}_w[V_b]$ simply by performing an one-to-two refinement of $\mathcal{Q}_w$ using either V_a or V_b as pivot (cf. $V_b = V_w \setminus V_a$) for each w. Duplication and refinement using V_a (or V_b) as pivot take $O(n)$ time for every node w, hence an $O(n^2)$ global runtime.

We then initialize $\mathcal{Q} = \mathcal{Q}_w[V_a]$ and, for every entry z of the table $N_w^{b\to a}$, refine $\mathcal{Q}$ using $N_w^{b\to a}[z]$ as pivot. As mentioned before, the main point of all these procedures is that the sum of the size of all possible pivots will now be at most twice the value $n + m$. Hence, the global runtime of this step is in $O(n + m)$. We deduce the following lemma, whose proof is straightforward. Recall that from the input of a clique-width expression of G, we can derive a rooted decomposition tree simply by contracting all internal nodes having one child in the underlying graph of the clique-width expression. The module-width of this decomposition tree is at most the clique-width of the expression.

Lemma 3.2. *Given a graph G and either (T, δ) a rooted decomposition tree of G, or a clique-width expression tree of G. Then in $O(n^2)$ global runtime we can compute and store, for every internal node u of T with V_u being the vertex subset of G induced by the leaves of the subtree of T rooted at u, the partition of V_u into its twin-classes $Q_u(1), Q_u(2), \ldots, Q_u(h_u)$.*

4. Solving the Feedback Vertex Set Problem

Definition 4.1. A *feedback vertex set* of a graph G is a subset S of the vertices of G with $G[V(G) \setminus S]$ a forest. A *forest inducing set (FI-set)* of a graph G is a subset of vertices S with $G[S]$ a forest.

Fact 4.2. *If S is a FI-set of maximum cardinality then $V(G) - S$ is a feedback vertex set of minimum cardinality.*

We give dynamic programming algorithms that given a graph G and a rooted decomposition tree (T, δ) of G will find the size of a minimum Feedback Vertex Set of G, by computing the size of a maximum FI-set in G. Recall that in Section 3 V_a, V_b, V_w were the vertex subsets corresponding to subtrees rooted at nodes a, b, w of T. For simplicity we adopt $A = V_a, B = V_b, W = V_w$ for the rest of this section. The runtime of the algorithm will be expressed as a function of $\mu_G(A)$, i.e. the number of twin-classes of such vertex subsets $A = V_a$.

4.1. Definition of Tables

For $A \subseteq V(G)$ let $\mathcal{TC}_A = \{TC_A^1, TC_A^2, ...TC_A^k\}$ be the twin-classes of A, using $k = \mu w(V(G))$ and allowing some empty classes. The indices of table Tab_A will consist of pairs $(\mathcal{P}, \mathcal{C})$ where $\mathcal{P}$ is a partition of $\mathcal{TC}_A$ and $\mathcal{C}$ is a partition of a subset of $\mathcal{TC}_A$. We denote the classes of $\mathcal{P}$ by $Q_A^0, Q_A^1, R_A^0, R_A^1, ..., R_A^k$, allowing some empty classes. $\mathcal{C}$ will be a partition of $\mathcal{TC}_A \setminus (Q_A^0 \cup R_A^0)$ and a coarsening of $R_A^1, R_A^2, ..., R_A^k$ in the sense that two twin-classes both in R_A^i, for some $1 \leq i \leq k$, must belong to the same class in $\mathcal{C}$.

Before defining the contents of the table formally, let us briefly give some intuition. An index $(\mathcal{P}, \mathcal{C})$ will store a largest FI-set $S \subseteq A$ satisfying certain properties, e.g. where Q_A^0 and Q_A^1 are those twin-classes containing exactly zero and one vertex from S and the remaining classes of $\mathcal{P}$ consist of twin-classes containing at least two vertices of S. Among these, the twin-classes of R_A^0 are exactly those containing vertices that should not get *any* further FI-set neighbors as we progress up to the root of the decomposition tree. The partition of the remaining twin-classes into $R_A^1, R_A^2, ..., R_A^k$ is part of the 'expectation from the outside'. As we progress up the path to the root two nodes receive a new common FI-neighbor if and only if they are in twin-classes belonging to the same R_A^i. The partition class $\mathcal{C}$ is also part of the expectation and tells us about connected components of FI-sets.

Definition 4.3. For every partition $\mathcal{P}$ of $\mathcal{TC}_A$ into $k+3$ parts $Q_A^0, Q_A^1, R_A^0, R_A^1, ..., R$ (allowing empty classes) and every partition $\mathcal{C}$ of $\mathcal{TC}_A \setminus (Q_A^0 \cup R_A^0)$ that is a coarsening of $R_A^1, R_A^2, ..., R_A^k$, we have an index $(\mathcal{P}, \mathcal{C})$ in Tab_A. The contents of $\mathsf{Tab}_A[\mathcal{P}, \mathcal{C}]$ will be a vertex subset $S \subseteq A$ of maximum cardinality among all $S \subseteq A$ satisfying the pair $(\mathcal{P}, \mathcal{C})$, where such S is said to *satisfy* $(\mathcal{P}, \mathcal{C})$ if

1. $Q_A^0 = \{TC_A^i : |TC_A^i \cap S| = 0\}$
2. $Q_A^1 = \{TC_A^i : |TC_A^i \cap S| = 1\}$
3. The graph $G(\mathcal{P}, S)$ is a forest, where $G(\mathcal{P}, S)$ is constructed from $G[S]$ by adding new intermediate vertices $\{v_1, v_2, ..., v_k\}$, and edges from v_i to all vertices in twin-classes belonging to R_A^i, for $1 \leq i \leq k$.
4. Two vertices $u \in TC_A^i \cap S$ and $v \in TC_A^j \cap S$ with TC_A^i, TC_A^j not in R_A^0 belong to the same connected component of $G(\mathcal{P}, S)$ if and only if TC_A^i, TC_A^j are in the same class of $\mathcal{C}$.

If no set $S \subseteq A$ satisfying $(\mathcal{P}, \mathcal{C})$ exists then the contents of $\mathsf{Tab}_A[\mathcal{P}, \mathcal{C}]$ should be $\sharp$.

4.2. The dynamic programming algorithm

We are now ready to describe the algorithm computing a maximum FI-set of G. The algorithm starts by initializing all table entries of all tables of the tree T to $\sharp$.

At any leaf a of the tree T we have $A = \{\delta(a)\}$ and $\mathcal{TC}_A = \{\{\delta(a)\}\}$. Two entries of the table Tab_a will be updated to something other than $\sharp$ corresponding to the two choices for a set satisfying an index $(\mathcal{P}, \mathcal{C})$, namely $S = \{\delta(a)\}$

and $S = \emptyset$. The first choice gives $\mathsf{Tab}_a[\mathcal{P}, \mathcal{C}] = \{\delta(a)\}$ for $\mathcal{P}$ having empty classes except $Q_A^1 = \{\{\delta(a)\}\}$ and $\mathcal{C} = \{\{\delta(a)\}\}$. The second choice gives $\mathsf{Tab}_a[\mathcal{P}, \mathcal{C}] = \emptyset$ for $\mathcal{P}$ having empty classes except $Q_A^0 = \{\{\delta(a)\}\}$ and $\mathcal{C}$ the partition of the empty set.

In a bottom-up traversal of the tree T, when reaching an internal node w having children a and b we do the following dynamic programming:

For all index triples $(\mathcal{P}_A, \mathcal{C}_A), (\mathcal{P}_B, \mathcal{C}_B), (\mathcal{P}_W, \mathcal{C}_W)$
  If $\mathsf{Tab}_a[\mathcal{P}_A, \mathcal{C}_A] = S_A$ and $\mathsf{Tab}_b[\mathcal{P}_B, \mathcal{C}_B] = S_B$ (i.e. not $\sharp$ entries)
    and $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$
    and $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W] = \sharp$ or $|\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W]| < |S_A \cup S_B|$
  Then update $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W] := S_A \cup S_B$

After the bottom-up traversal filling all tables output the entry $\mathsf{Tab}_{root}[\mathcal{P}, \mathcal{C}]$ at the root of T, for $\mathcal{P}$ having empty classes except $R_{root}^0 = \{V(G)\}$ and $\mathcal{C}$ the partition of the empty set.

5. Correctness and timing

Let us start by noting that at the root of T we have $\mathcal{TC}_{root} = \{V(G)\}$ and the value of $\mathsf{Tab}_{root}[\mathcal{P}, \mathcal{C}]$ for the partition where $R_{root}^0 = \{V(G)\}$ and $\mathcal{C}$ the partition of the empty set will by Definition 4.3 be a maximum FI-set of G. A central part of the correctness argument is to show that the FI-set stored in a table entry will induce an acyclic graph. This will be done by contradiction, showing that a cycle in one graph can be replaced in another graph by a walk starting and ending in the same vertex and using some edge exactly once, and then applying the following easy observation.

Lemma 5.1. *Consider a graph H containing a walk starting and ending in the same vertex. If some edge uv appears an odd number of times in the walk then the subgraph H' of H induced by the edges in the walk contains a cycle.*

Proof. Note that H' is a connected graph. We first show that it remains connected also after removing the edge uv. Removing uv breaks the walk up in subwalks of three types, from u to u, from v to v and from u to v. Since uv is used an odd number of times and the original walk started and ended in same node, both u and v are used as endpoints an odd number of times. Therefore one of the subwalks will go from u to v showing that u and v are in the same connected component even after removing uv. Since adding a new edge to a connected graph will give a graph with a cycle the graph H' must have contained a cycle. □

Lemma 5.2. *The dynamic programming algorithm will correctly fill all tables.*

Proof. The lemma is proved by bottom-up induction on the tree T. Leaves of T are correctly updated since we try both subsets of nodes as FI-sets for the unique indices that they satisfy. Consider an internal node w of T with

children a, b and assume inductively that Tab_a and Tab_b are correct. We show that Tab_w is then updated correctly. Recall that A, B, W are the vertex subsets corresponding to subtrees rooted at a, b, w.

For any index $(\mathcal{P}_W, \mathcal{C}_W)$ we must show that if there is a set satisfying $(\mathcal{P}_W, \mathcal{C}_W)$ then $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W]$ is not equal $\sharp$ and that, if $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W] = S_W$ then $S_W \subseteq W$ is a largest set satisfying $(\mathcal{P}_W, \mathcal{C}_W)$. First, note that in the latter case we know S_W satisfies $(\mathcal{P}_W, \mathcal{C}_W)$ as this was checked in the algorithm.

Thus, assume for contradiction that there is a set $F_W \subseteq W$ satisfying $(\mathcal{P}_W, \mathcal{C}_W)$ and that we have either $|F_W| > |\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W]|$ or we have $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W]$ $\sharp$. From $\mathcal{P}_W, \mathcal{C}_W, F_W$ we first construct $(\mathcal{P}_A, \mathcal{C}_A), (\mathcal{P}_B, \mathcal{C}_B)$ such that $F_W = F_A \cup F_B$ and F_A satisfies $(\mathcal{P}_A, \mathcal{C}_A)$ and F_B satisfies $(\mathcal{P}_B, \mathcal{C}_B)$. Note that we will not be using $\mathcal{C}_W$ as the pair $\mathcal{P}_W, F_W$ uniquely defines $\mathcal{C}_W$. Let $F_A = F_W \cap A$ and $F_B = F_W \cap B$.

We now construct $\mathcal{P}_A = Q^0_A, Q^1_A, R^0_A, R^1_A, ..., R^k_A$. We set $Q^0_A = \{TC^i_A : |TC^i_A \cap F_A| = 0\}$ and $Q^1_A = \{TC^i_A : |TC^i_A \cap F_A| = 1\}$. Recall that $\mathcal{TC}_W$ is a coarsening of $\mathcal{TC}_A \cup \mathcal{TC}_B$, so that a class of $\mathcal{TC}_W$ is the union of some classes of $\mathcal{TC}_A$ and some of $\mathcal{TC}_B$. For any R^i_W with $i > 0$ the twin-classes of $\mathcal{TC}_A$ that are subsets of R^i_W, but have not been assigned to $Q^0_A \cup Q^1_A$, will be assigned to R^i_A.

A twin-class TC^h_A of $\mathcal{TC}_A$ that has not been assigned yet (neither to Q^0_A, Q^1_A nor any R^i_A) must be a subset of R^0_W and must have at least two vertices in F_A. If TC^h_A does not have a neighbor in F_B then it is assigned to R^0_A. If TC^h_A does have a neighbor in F_B then since $F_A \cup F_B$ contains no cycle all vertices in TC^h_A have a single neighbor v_h in F_B common to them all. We arbitrarily pick indices $j \in \{1, 2, ..., k\}$ with R^j_A still empty (since $k = \mu w(V(G))$ we can find enough such j). We then assign remaining twin-classes using these indices such that two remaining twin-classes TC^h_A and TC^q_A both belong to the same R^j_A if and only if they have the same common neighbor in F_B. All remaining R^i_A should be empty. This completes the construction of $\mathcal{P}_A$. For $\mathcal{P}_B$ we do the analogous construction.

Claim 5.3. *The graph $G(\mathcal{P}_A, F_A)$ is isomorphic (respecting twin-classes) to the subgraph of $G(\mathcal{P}_W, F_A \cup F_B)$ induced on edges with either both endpoints in F_A or exactly one endpoint in a twin-class belonging to R^i_A for some i. Same holds for $G(\mathcal{P}_B, F_B)$.*

Proof. Note that $G(\mathcal{P}_A, F_A)$ and $G(\mathcal{P}_W, F_A \cup F_B)$ clearly induce the same graph on F_A. All remaining edges of $G(\mathcal{P}_A, F_A)$ are, by definition of $G(\mathcal{P}_A, F_A)$, accounted for by noting that two vertices in some twin-class of R^i_A have a common neighbor (not in A) if and only if their twin-classes belong to the same R^i_A for $i > 0$. But then they then have a common neighbor also in $G(\mathcal{P}_W, F_A \cup F_B)$, since by construction R^i_A is one of two types: either all twin-classes in R^i_A are subsets of some twin-class in R^i_W, in which case these vertices have a common neighbor in $G(\mathcal{P}_W, F_A \cup F_B)$, or all vertices of all twin-classes in R^i_A have a common neighbor in F_B. Since we are only showing that $G(\mathcal{P}_A, F_A)$

is (isomorphic to, while respecting twin-classes) a *subgraph* of $G(\mathcal{P}_W, F_A \cup F_B)$ this suffices. □

We now construct $\mathcal{C}_A$, following Definition 4.3. Consider the graph $G(\mathcal{P}_A, F_A)$ constructed from $G[F_A]$ by adding new vertices $\{v_1, v_2, ..., v_k\}$, and edges from v_i to all vertices in all twin-classes belonging to R^i_A, for $1 \leq i \leq k$. We define $\mathcal{C}_A$ to be the partition of $\mathcal{TC}_A \setminus (Q^0_A \cup R^0_A)$ such that TC^i_A, TC^j_A not in $(Q^0_A \cup R^0_A)$ are in the same class of $\mathcal{C}_A$ if and only if two vertices $u \in TC^i_A \cap F_A$ and $v \in TC^j_A \cap F_A$ belong to the same connected component of $G(\mathcal{P}_A, F_A)$. For $\mathcal{C}_B$ we do the analogous construction. We are thus done with construction of indices $(\mathcal{P}_A, \mathcal{C}_A)$ and $(\mathcal{P}_B, \mathcal{C}_B)$ in Tab_a and Tab_b.

Claim 5.4. *F_A satisfies $(\mathcal{P}_A, \mathcal{C}_A)$ and F_B satisfies $(\mathcal{P}_B, \mathcal{C}_B)$.*

Proof. We give the argument for F_A only since the argument for F_B is symmetric. It is obvious from the construction that F_A will satisfy the two first constraints in Definition 4.3 for $(\mathcal{P}_A, \mathcal{C}_A)$. By Claim 5.3 and the fact that $F_A \cup F_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$ the graph $G(\mathcal{P}_A, F_A)$ is a forest so that it satisfies the third constraint. By construction of $\mathcal{C}_A$ it is clear that F_A satisfies the fourth constraint. □

Based on the inductive assumption that Tab_a and Tab_b are correct we know that since F_A satisfies $(\mathcal{P}_A, \mathcal{C}_A)$ and F_B satisfies $(\mathcal{P}_B, \mathcal{C}_B)$ we have $\mathsf{Tab}_a[\mathcal{P}_A, \mathcal{C}_A] = S_A$ for some largest $S_A \subseteq A$ satisfying $(\mathcal{P}_A, \mathcal{C}_A)$, $\mathsf{Tab}_a[\mathcal{P}_B, \mathcal{C}_B] = S_B$ for some largest $S_B \subseteq B$ satisfying $(\mathcal{P}_B, \mathcal{C}_B)$, and thus $|S_A| \geq |F_A|$, and $|S_B| \geq |F_B|$. Consider what happens when the algorithm considers the triple $(\mathcal{P}_A, \mathcal{C}_A)$, $(\mathcal{P}_B, \mathcal{C}_B)$, $(\mathcal{P}_W, \mathcal{C}_W)$. If $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$ then we are guaranteed that $|\mathsf{Tab}[\mathcal{P}_W, \mathcal{C}_W]|$ $|S_A \cup S_B| \geq |F_A \cup F_B| = |F_W|$ which will establish the contradiction.

Thus, it remains only to show that $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$, as in Definition 4.3. From Definition 4.3 we get that if TC^i_A belongs to Q^0_A or Q^1_A then $|S_A \cap TC^i_A| = |F_A \cap TC^i_A|$, same holds for B. A twin-class TC^i_W is a union of some twin-classes from $\mathcal{TC}_A$ and $\mathcal{TC}_B$. If TC^i_W belongs to Q^0_W or Q^1_W then it is a union of twin-classes belonging to Q^0_A, Q^1_A, Q^0_B or Q^1_B and hence $|(S_A \cup S_B) \cap TC^i_W| = |F_W \cap TC^i_W|$. Therefore by construction $S_A \cup S_B$ satisfies the first two constraints of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$. The following claims will be useful to show that the third constraint is satisfied.

Claim 5.5. *For any $i \geq 1$ all vertices in any non-empty twin-class of R^i_A have, in the graph $G(\mathcal{P}_W, S_A \cup S_B)$, exactly one neighbor not in A, which is common to them all. Same for R^i_B.*

Proof. We first show that no vertex x of a twin-class that is a subset of R^i_A can have more than one neighbor outside A in $G(\mathcal{P}_W, S_A \cup S_B)$. We do this by a case analysis showing that I: it cannot have any neighbor in an R^j_B-class, II: it cannot have two neighbors in Q^1_B-classes, III: it cannot have two new intermediate neighbors, and IV: it cannot have one intermediate neighbor and one neighbor in a Q^1_B-class.

Since by Claim 5.4 F_A, S_A satisfy $(\mathcal{P}_A, \mathcal{C}_A)$ and also F_B, S_B satisfy $(\mathcal{P}_B, \mathcal{C}_B)$ all twin-classes of R_A^i contain at least two vertices of S_A and of F_A, and R_B^j (for $j > 0$ an index of a non-empty R_B^j) contain at least two vertices of S_B and of F_B. Therefore, since all vertices in a twin-class of $\mathcal{TC}_A$ have the same neighbors outside A case I must hold since otherwise we would have a 4-cycle in $G(\mathcal{P}_W, F_A \cup F_B)$, contradicting that $F_A \cup F_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$. Likewise all twin-classes of Q_B^i contain one vertex of both S_B and of F_B. Therefore, case II must hold since otherwise we would again have a 4-cycle in $G(\mathcal{P}_W, F_A \cup F_B)$. By Definition 4.3 no vertex in the graph $G(\mathcal{P}_W, S_A \cup S_B)$ has more than one intermediate neighbor so case III holds. For case IV note first that the only way vertex x in a twin-class of R_A^i can have an intermediate neighbor in $G(\mathcal{P}_W, S_A \cup S_B)$ is if its twin-class is a subset of R_W^j for some $j > 0$. But then the vertex of F_A in the same twin-class will also have an intermediate neighbor in $G(\mathcal{P}_W, F_A \cup F_B)$. Since all twin-classes of Q_B^i contain one vertex of both S_B and of F_B we conclude that case IV must hold since otherwise we would again have a 4-cycle in $G(\mathcal{P}_W, F_A \cup F_B)$.

We now show that every vertex in any twin-class of R_A^i has a common neighbor outside A in $G(\mathcal{P}_W, S_A \cup S_B)$. Firstly, every vertex whose twin-class is a subset of R_W^j for $j > 0$ has a common intermediate neighbor. Secondly, for a vertex whose twin-class is a subset of R_W^0 any neighbor it has outside A must be in S_B. Any two vertices of F_A in twin-classes of R_A^i have in $G(\mathcal{P}_A, F_A)$ a common neighbor, by Claim 5.4. By Claim 5.3 these two vertices of F_A have also in $G(\mathcal{P}_W, F_A \cup F_B)$ a common neighbor, which must be in F_B since R_A^i is a subset of R_W^0. If any two vertices x, y in F_A have a common neighbor in F_B then any two vertices in S_A from the same twin-classes as x, y must have a common neighbor in F_B. This concludes the proof. □

Claim 5.6. *The graph $G(\mathcal{P}_A, S_A)$ is isomorphic (respecting twin-classes) to the subgraph of $G(\mathcal{P}_W, S_A \cup S_B)$ induced on edges with either both endpoints in S_A or exactly one endpoint in a twin-class belonging to R_A^i for some i. Same holds for $G(\mathcal{P}_B, S_B)$.*

Proof. Note that $G(\mathcal{P}_A, S_A)$ and $G(\mathcal{P}_W, S_A \cup S_B)$ clearly induce the same graph on S_A. For the edges with exactly one endpoint in a twin-class belonging to R_A^i Claim 5.5 implies that such an edge exists in both graphs or none of the two graphs. No other edges exist in $G(\mathcal{P}_A, S_A)$. □

To show that $S_A \cup S_B$ satisfies the third constraint of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$ we show that $G(\mathcal{P}_W, S_A \cup S_B)$ is a forest. We will prove this by contradiction, showing that if we have a cycle Ψ in $G(\mathcal{P}_W, S_A \cup S_B)$ then we also have a walk Ψ' containing a cycle in $G(\mathcal{P}_W, F_W)$ (which we know is a forest). We assume that the cycle Ψ is induced and break it into parts uniquely as follows (after first uniquely choosing parts of type 1 below and then of type 2 below the rest of the cycle will uniquely be of types 3 and 4 below):

1. maximal paths starting and ending in S_A containing at least one edge of $G[S_A]$ and otherwise only containing edges with exactly one endpoint in

a twin-class belonging to R_A^i for some i.
2. maximal paths starting and ending in S_B containing at least one edge of $G[S_B]$ and otherwise only containing edges with exactly one endpoint in a twin-class belonging to R_B^i for some i.
3. crossings from S_A to S_B directly by one edge.
4. crossings using one intermediate new vertex v_i.

We assume additionally that the cycle Ψ has the smallest number of parts over all induced cycles. Each part starts and ends in a vertex of $S_A \cup S_B$ and these endpoints are called *special* vertices.

Claim 5.7. *Each twin-class of $\mathcal{TC}_A$ and $\mathcal{TC}_B$ contains at most one special vertex.*

Proof. Assume for contradiction that some twin-class contains two special vertices x, y. This twin-class must be some R_A^i (or R_B^i) since these are the only classes containing more than one vertex of S_A (or S_B). Any special vertex x in S_A (resp S_B) has two neighbors x_1, x_2 in the cycle Ψ. These two neighbors cannot both be in S_A (resp S_B), since x would then not be a special vertex. By Claim 5.5, x, y have exactly one neighbor not in A, thus wlog we can assume $x_1 = y_1 \notin A$ so that the cycle Ψ has consecutive vertices x, x_1, y. Their other neighbor(s) x_2, y_2 in the cycle are in S_A and thus x, y are not special vertices. □

Claim 5.8. *Ψ cannot have only one part.*

Proof. Note that Ψ must have at least two parts since if it had only one part this part would have to be of type 1 (or type 2) and be a cycle in which case Claim 5.6 would imply that $G(\mathcal{P}_A, S_A)$ (or $G(\mathcal{P}_B, S_B)$) had a cycle, and thus not satisfy $(\mathcal{P}_A, \mathcal{C}_A)$ (or $(\mathcal{P}_B, \mathcal{C}_B)$) contradicting Claim 5.4. □

We are ready to start constructing Ψ'. First, we choose for each special vertex $v \in S_A$ (resp S_B) of Ψ an arbitrary vertex of F_A (resp F_B) from the same twin-class as v. By Claim 5.7 we have thus chosen at most one vertex from each twin-class. We then replace each part of Ψ in $G(\mathcal{P}_W, S_A \cup S_B)$ by a path in $G(\mathcal{P}_W, F_W)$ between the two chosen vertices and call the resulting graph Ψ'.

Claim 5.9. *Parts of type 1 (resp 2) can be replaced by paths in $G(\mathcal{P}_W, F_A \cup F_B)$ containing at least one edge of $G[F_A]$ and otherwise only containing edges with exactly one endpoint in a twin-class belonging to R_A^i for some i (resp for type 2, replacing A by B).*

Proof. By Claim 5.6 a part of type 1 in Ψ, i.e. a path in $G(\mathcal{P}_W, S_A \cup S_B)$, gives a path in $G(\mathcal{P}_A, S_A)$. Since both S_A and F_A satisfy $(\mathcal{P}_A, \mathcal{C}_A)$ this gives a path in $G(\mathcal{P}_A, F_A)$. This means that for any part of type 1 between special vertices u, v there is a path in $G(\mathcal{P}_A, F_A)$ between the vertices $u'v'$ chosen from the same twin-classes as u, v, having edges in $G[F_A]$ and edges with exactly one endpoint in a twin-class belonging to R_A^i. We now show that there is such a

path in $G(\mathcal{P}_A, F_A)$ between u', v' containing at least one edge of $G[F_A]$. Note that if u, v both belong to the same R^i_A then u, v have a common neighbor in $G(\mathcal{P}_W, S_A \cup S_B)$ and Ψ would have only one part and we apply Claim 5.8. Thus u', v' are in twin-classes belonging to different classes of the partition $\mathcal{P}_A$. Any path in $G(\mathcal{P}_A, F_A)$ between vertices in twin-classes belonging to different classes of $\mathcal{P}_A$ must contain an edge of $G[F_A]$.

Let us now argue that the existence of such a path from u' to v' in $G(\mathcal{P}_A, F_A)$ implies the existence of a similar path in $G(\mathcal{P}_W, F_A \cup F_B)$. Firstly, on F_A the induced subgraphs are the same in both graphs. Secondly, by construction of $\mathcal{P}_A$ from $\mathcal{P}_W$ any two vertices of F_A having in $G(\mathcal{P}_A, F_A)$ a common neighbor outside F_A also have in $G(\mathcal{P}_W, F_A \cup F_B)$ a common neighbor outside F_A.

We can similarly argue for parts of type 2. □

Parts of type 3 are easy to replace since if there is an edge in $G(\mathcal{P}_W, S_A \cup S_B)$ from a vertex in a twin-class TC^i_A to a vertex in a twin-class TC^j_B then in the graph $G(\mathcal{P}_W, F_W)$ any vertex in TC^i_A will be connected to all vertices in TC^j_B. Parts of type 4 are also easy to replace: such a part goes from a vertex in a twin-class TC^i_A via an intermediate new vertex to a vertex in twin-class TC^j_B with both twin-classes belonging to the same R^k_W and thus since F_W satisfies $\mathcal{P}_W, \mathcal{C}_W$ the same edges are present in $G(\mathcal{P}_W, F_W)$. This finishes the construction of Ψ' from Ψ.

Note that the subgraph Ψ' must be connected since we started with a cycle Ψ, then first replaced each special vertex v by some v' and then replaced paths of the cycle between special vertices u, v by paths connecting u' and v'. Ψ' is therefore a walk starting and ending in the same vertex and that is why Lemma 5.1 is useful to show that Ψ' contains a cycle.

Claim 5.10. *If Ψ contains an edge from a special vertex u in a twin-class of Q^1_A (resp Q^1_B) to a vertex w not in A nor in a twin-class of R^i_B (resp not in B nor in a twin-class of R^i_A) then Ψ' contains a cycle.*

Proof. Note that for $u \in Q^1_A$ there are only two choices for w: it can either belong to a twin-class of Q^1_B or it can be an intermediate vertex between u and a special vertex of B. The special vertex u of Ψ is replaced in Ψ' by u' in the same twin-class and no other special vertex of Ψ is replaced by u'. Similarly, if $w \in Q^1_B$ then it is replaced by a vertex w' from the same twin-class and it is the only such vertex. Otherwise we let $w' = w$. We will argue that the edge $u'w'$ appears exactly once in Ψ', and the statement will follow from Lemma 5.1. The edge $u'w'$ does not belong to $G[F_A]$ nor to $G[F_B]$ and neither u' nor w' belongs to some R^i_A or R^i_B. Therefore, by Claim 5.9 parts of type 1 or 2 in Ψ will never be replaced by a path containing the edge $u'w'$. A part of type 3 will be replaced by a single edge, and a part of type 4 by a path on two edges, from A to B. Two parts of type 3 cannot both be replaced by the same edge $u'w'$ since then the cycle Ψ would have contained the edge uw twice contradicting the fact that Ψ was chosen to be simple. Similarly, two parts of type 4 cannot both be replaced by a path containing the edge $u'w'$ since then the cycle Ψ would have

contained the edge from u to its intermediate neighbor twice contradicting the fact that Ψ was chosen to be simple. Similarly we can argue for $u \in Q_B^1$. □

Claim 5.11. *If all edges of Ψ belong either to $G[S_A]$ or $G[S_B]$ or have at least one endpoint in R_A^i or R_B^i then Ψ' contains a cycle.*

Proof. By Claim 5.5 a vertex u in R_A^i (resp R_B^i) has a single neighbor outside B so that if the cycle Ψ contains a vertex u in a twin-class of R_A^i (resp R_B^i) then Ψ must contain an edge uv of $G[S_A]$ ($G[S_B]$). Thus Ψ has at least one part of type 1 or 2.

We now argue that the condition in the claim implies that the parts of Ψ must alternate between being: of type 1 (containing at least one edge of $G[S_A]$), then crossing parts of type 3 or 4 (containing no edges of $G[S_A]$ or $G[S_B]$), then of type 2 (containing at least one edge of $G[S_B]$), then again crossings, and so forth. Consider wlog a part of type 1 ending in a special vertex $u \in S_A$. The other part with special vertex u cannot be of type 1 or 2 since parts of type 1 are maximal and parts of type 2 have special vertices in S_B. We need to show that going around the cycle Ψ from u we must encounter a part of type 2 before we encounter a part of type 1 again (i.e. they alternate). When going around the cycle Ψ from u there are two cases, either we encounter a vertex in a twin-class of some R_B^i before encountering an edge of $G[S_A]$, or not. In the former case we would next encounter an edge of $G[S_B]$ by the observation at the start of the proof of this claim and would be done. In the latter case the condition in the claim implies that every edge of Ψ between u and the occurrence of the edge of $G[S_A]$ would have an endpoint in R_A^i, which would contradict the maximality of the type 1 part ending in u. We conclude that the parts alternate as described above.

Parts of type 3 and 4 in Ψ are replaced in Ψ' by crossings containing no edges of $G[A]$ or $G[B]$ and by Claim 5.9 parts of type 1 (resp 2) are replaced in Ψ' by paths containing at least one edge of $G[A]$ and no edge of $G[B]$ (resp at least one edge of $G[B]$ and no edge of $G[A]$). Call an edge of Ψ' with both endpoints in $G[A]$ or both in $G[B]$ a one-sided edge. Firstly, if there is some one-sided edge uv of Ψ' that appears only once in the walk defined by Ψ' then by Lemma 5.1 the walk Ψ' contains a cycle. Otherwise, take a one-sided edge uv such that there are two appearances of it in the walk Ψ' with no other one-sided edge appearing twice in the part of the walk between these two appearances of uv. Note that there must be at least one special vertex between the two occurences.

As the types alternate, the subgraph induced by the edges between these two uses of uv in the walk Ψ' must contain a one-sided edge yz (on the other side). We therefore have a walk in Ψ' starting and ending in vertex u containing an edge yz used exactly once, so that Ψ' contains a cycle by Lemma 5.1. □

If Ψ does not fulfill the condition of Claim 5.11 then it contains an edge having one endpoint in a twin-class of Q_A^1 (resp Q_B^1) and the other endpoint not in A (resp not in B) and not in a twin-class of R_B^i (resp R_A^i). But then Ψ fulfills Claim 5.10. Thus the existence of a cycle Ψ in $G(\mathcal{P}_W, S_A \cup S_B)$ implies

a cycle in $G(\mathcal{P}_W, F_W)$ either by Claim 5.10 or 5.11, contradicting the fact that F_W satisfies $(\mathcal{P}_W, \mathcal{C}_W)$. Thus, we have shown that $S_A \cup S_B$ satisfies the third constraint of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$. We now show that it satisfies also the fourth constraint.

The argument for the fourth constraint is similar to the one for the third constraint but a bit simpler since we only need to replace paths by connected graphs and not cycles by connected graphs containing a cycle.

We show there exists a path Γ in $G(\mathcal{P}_W, S_A \cup S_B)$ from a vertex in $TC^i_W \notin R^0_W$ to a vertex in $TC^j_W \notin R^0_W$ if and only if there exists a path Γ' in $G(\mathcal{P}_W, F_W)$ from a vertex in TC^i_W to a vertex in TC^j_W. Since F_W satisfies the fourth constraint of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$, showing this will imply that also $S_A \cup S_B$ satisfies it. Given Γ we construct Γ' as follows. Break Γ into parts of 4 types in such a way that the endpoints of a part contains no vertex from R^0_W.

1. paths having edges in A only
2. paths having edges in B only
3. a single edge from S_A to S_B
4. a path of length two from S_A or S_B to S_A or S_B via a new vertex v_i

This can be done since vertices of R^0_W have no crossing edges. First replace each vertex $v \in TC^p_A$ (resp. $v \in TC^p_B$) that is the endpoint of a part of Γ by an arbitrary vertex v' of F_W in TC^p_A (resp. TC^p_B). A part (of type 1) with endpoints u, v containing only edges from A is a path also in $G(\mathcal{P}_A, S_A)$, and since both S_A and F_A satisfy $(\mathcal{P}_A, \mathcal{C}_A)$ such a part can be replaced by a path between u', v' in $G(\mathcal{P}_A, F_A)$, and since by Claim 5.3 $G(\mathcal{P}_A, F_A)$ is a subgraph of $G(\mathcal{P}_W, F_W)$ it can be replaced by a path between u', v' in $G(\mathcal{P}_W, F_W)$. The same holds for parts (of type 2) with endpoints u, v containing only edges from B. A part of type 3 is easy to replace: edges from S_A to S_B are replicated in F_W since all vertices in a twin-class have the same neighbors on the other side. A part of type 4 is also easily replaced since they go from a vertex in a twin-class TC^j_A or TC^j_B to a vertex in a twin-class TC^p_A or TC^p_B both in R^i_W and thus in $G(\mathcal{P}_W, F_W)$ all vertices of these two twin-classes will be connected by such a path of length two via new vertex v_i. This concludes the construction of Γ' and shows that there is a path between u', v' in $G(\mathcal{P}_W, F_W)$. The opposite direction, given Γ' constructing Γ, is done in an analogous manner. Thus $S_A \cup S_B$ satisfies the fourth constraint of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$.

We have shown that $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$. Therefore we cannot have that $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W] = \sharp$ and, since $|S_A \cup S_B| = |S_A| + |S_B| \geq |F_A| + |F_B| = |F_W|$, we cannot have $|F_W| > |\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W]|$, finishing the proof. □

Theorem 5.12. *Given either a rooted decomposition tree (T, δ) of module-width k of a graph G, or a k-expression of a graph G of clique-width at most k, we can in $O(k^{5k} n^2)$ steps solve the Minimum Feedback Vertex Set problem on G.*

Proof. Consider first the case of input being a rooted decomposition tree. By Lemma 3.2 we can compute twin-classes for all nodes of the tree in time $O(n^2)$. Note that for any node a of the tree T the number of twin-classes of V_a is at

most k. By Definition 4.3 and Lemma 5.2 the maximum value over all entries in the table at the root of our dynamic programming algorithm will correctly solve the problem.

The tables are indexed by $\mathcal{P}$, an ordered partition and $\mathcal{C}$ an unordered partition of a subset defined by $\mathcal{P}$. There are $O(k^{k+3})$ ordered partitions of $k+3$ elements. The number of unordered partitions of k elements is bounded by the number of ordered partitions. Note that the number of unordered partitions is so much smaller than the number of ordered partitions that it will cancel all factors polynomial in k, hence the size of the tables are $O(k^{2k})$. For the runtime, the bottleneck is the inner node update procedure which loops over all triples of table indexes $(\mathcal{P}_A, \mathcal{C}_A), (\mathcal{P}_B, \mathcal{C}_B), (\mathcal{P}_W, \mathcal{C}_W)$ and check if the union the two elements $S_A = \mathsf{Tab}_a[\mathcal{P}_A, \mathcal{C}_A]$ and $S_B = \mathsf{Tab}_b[\mathcal{P}_B, \mathcal{C}_B]$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$. This gives a total of $O(k^{6k})$ iterations, however $\mathcal{C}_W$ is uniquely defined by the other 5 elements hence only $O(k^{5k})$ iterations are needed. To check if $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$ we first make the union in $O(n)$ time and then check the four constraints of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$. The two first constraints are checked in $O(n)$ time, building the graph $G(\mathcal{P}_W, S_A \cup S_B)$ and checking if it is a forest is straight forward to do in $O(m)$ time. However if the two graphs $G[S_A]$ and $G[S_B]$ are stored from the previous step, then one can build the graph in $O(k^2 n)$ time, and since a forest has at most $n-1$ edges also check if $G(\mathcal{P}_W, S_A \cup S_B)$ is a forest. Checking that the connected components match $\mathcal{C}_W$ is also done in $O(n)$ time as long as the graph is a forest. In total the combine step is $O(k^{5k} n)$. The preprocessing is done in $O(n^2)$ time, initialization is done in $O(n \times k^{2k})$ time. Filling the tables requires n combine steps, hence the total running time is $O(n^2 k^{5k})$.

Note that within the same runtime we could instead have taken as input a k-expression of a graph G of clique-width at most k. This is since by Theorem 2.6 the module-width of G is no larger than the clique-width of G, and from the k-expression we easily derive a rooted decomposition tree of module-width at most k. □

6. Conclusion

The Feedback Vertex Set problem has a non-local property that does not lend itself to easy dynamic programming. Using the technique of 'expectation from the outside' in the definition of tables of the dynamic programming, and not the standard technique of partitioning the solution space into equivalence classes, we have given an FPT algorithm for FVS parametrized by the clique-width of a given decomposition. The exponential runtime of this algorithm matches the runtime of the best known deterministic algorithm when parametrizing by treewidth. Note that many graph classes have unbounded treewidth and bounded clique-width but the opposite cannot occur. It is already an open problem to solve FVS deterministically in exponential time $O^*(2^{O(tw)})$, so maybe $O^*(2^{O(cw)})$ is too much to hope for. Boolean-width and rank-width are parameters bounded on the same graph classes as clique-width, but their values can be exponentially smaller than clique-width. The best runtime known

for FVS on graphs of rank-width rw is $O(2^{5rw^2}rw^3n)$ [15], and from this we can deduce an algorithm with runtime $O(2^{5(2^{2bw})+3bw}n)$ for graphs of boolean-width bw. Can we get runtime $O^*(2^{O(bw^2)})$ for boolean-width?

[1] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics* 12:289–297, 1999.

[2] R. Bar-Yehuda, D. Geiger, J. Naor, and R. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal on Computing* 27:942–959, 1998.

[3] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rank-width. *Discrete Applied Mathematics* 158(7):809–819, 2010. .

[4] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Fast FPT algorithms for vertex subset and vertex partitioning problems using neighborhood unions. `http://arxiv.org/abs/0903.4796`

[5] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Feedback Vertex Set on Graphs of Low Cliquewidth. In *Proceedings of IWOCA'09*, LNCS 5874, pages 113–124, 2009.

[6] J. Chen, F. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved Algorithms for the Feedback Vertex Set Problems. In *Proceedings of WADS'07*, LNCS 4619, pages 422–433, 2007.

[7] F. Chudak, M. Goemans, D. Hochbaum, and D. Williamson. A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Operations Research Letters* 22:111–118, 1998.

[8] B. Courcelle, J.A. Makowsky, U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[9] M. Cygan, J. Nederlof, Ma. Pilipczuk, Mi. Pilipczuk, J. van Rooij and J. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. to appear in *Proceedings of FOCS'11*

[10] F. Dehne, M. Fellows, M. Langston, F. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT Algorithm for the Undirected Feedback Vertex Set Problem. In *Proceedings of COCOON'05*, LNCS 3595, pages 859–869, 2005.

[11] R.Downey and M.Fellows. *Parameterized Complexity*, Springer-Verlag (1999)

[12] W. Espelage, F. Gurski, E. Wanke. How to Solve NP-hard Graph Problems on Clique-Width Bounded Graphs in Polynomial Time. In *Proceedings of WG'01*, LNCS 2204, pages 117–128, 2001.

[13] G. Even, J. Naor, B. Schieber, and L. Zosin. Approximating minimum subset feedback sets in undirected graphs with applications. *SIAM Journal on Discrete Mathematics* 13:255–267, 2000.

[14] F. Fomin, S. Gaspers, A. Pyatkin, and I. Razgon. On the Minimum Feedback Vertex Set Problem: Exact and Enumeration Algorithms. *Algorithmica*, 52:293–307, 2008.

[15] R. Ganian and P. Hliněný. On Parse Trees and Myhill-Nerode-type Tools for handling Graphs of Bounded Rank-width. *Discrete Applied Mathematics* 158(7):851–867, 2010.

[16] M. Goemans and D. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica* 18(1):37–59, 1998.

[17] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences* 72(8):1386–1396, 2006.

[18] M. Habib, C. Paul, L. Viennot. Partition Refinement Techniques: An Interesting Algorithmic Tool Kit. *International Journal of Foundations on Computer Science* 10(2):147–170, 1999.

[19] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[20] P. Hliněný, S. Oum. Finding branch-decompositions and rank-decompositions. *SIAM Journal on Computing* 38(3):1012–1032, 2008.

[21] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[22] J. Kleinberg and A. Kumar. Wavelength conversion in optical networks. *Journal of Algorithms* 38:25–50, 2001.

[23] T. Kloks, C. Lee, J. Liu. New Algorithms for k-Face Cover, k-Feedback Vertex Set, and k-Disjoint Cycles on Plane and Planar Graphs. In *Proceedings of WG'02*, LNCS 2573, pages 282–295, 2002.

[24] D. Kobler, U. Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics* 126(2-3): 197–221, 2003.

[25] A. Koutsonas and D. Thilikos. Planar Feedback Vertex Set and Face Cover: Combinatorial Bounds and Subexponential Algorithms. In *Proceedings of WG'08*, LNCS 5344, pages 254–274, 2008.

[26] J.-M. Lanlignel. Autour de la décomposition en coupe. *Ph. D. thesis,* Université Montpellier II, 2001.

[27] S. Oum, P. Seymour. Approximating clique-width and branch-width. *J. Combin.Theory Ser. B* 96(4):514–528, 2006.

[28] R. Paige, R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing* 16(6):973–989, 1987.

[29] V. Raman, S. Saurabh, and C. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms* 2(3):403–415, 2006.

[30] M. Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics* 308(24):6157–6165, 2008.

[31] N. Robertson, P. Seymour. Graph minors X: Obstructions to tree-decomposition. *Journal on Combinatorial Theory Series B,* 52:153–190, 1991.

[32] M. Rao. Décompositions de graphes et algorithmes efficaces. *Ph. D. thesis,* Université Paul Verlaine, Metz, 2006.

Finding Good Decompositions for Dynamic Programming on Dense Graphs[☆]

Eivind Magnus HVIDEVOLD[a], Sadia SHARMIN[a], Jan Arne TELLE[a], Martin VATSHELLE[a,]

[a] *Department of Informatics, University of Bergen, Norway.*

Abstract

It is well-known that for graphs with high edge density the tree-width is always high while the clique-width can be low. Boolean-width is a new parameter that is never higher than tree-width or clique-width and can in fact be as small as logarithmic in clique-width. Boolean-width is defined using a decomposition tree by evaluating the number of neighborhoods across the resulting cuts of the graph. Several NP-hard problems can be solved efficiently by dynamic programming when given a decomposition of boolean-width k, e.g. Max Weight Independent Set in time $O(n^2k2^{2k})$ and Min Weight Dominating Set in time $O(n^2 + nk2^{3k})$. Finding decompositions of low boolean-width is therefore of practical interest. There is evidence that computing boolean-width is hard, while the existence of a useful approximation algorithm is still open. In this paper we introduce and study a heuristic algorithm that finds a reasonably good decomposition to be used for dynamic programming based on boolean-width. On a set of graphs of practical relevance, specifically graphs in TreewidthLIB, the best known upper bound on their tree-width is compared to the upper bound on their boolean-width given by our heuristic. For the large majority of the graphs on which we made the tests, the tree-width bound is at least twice as big as the boolean-width bound, and boolean-width compares better the higher the edge density. This means that, for problems like Dominating Set, using boolean-width should outperform dynamic programming by tree-width, at least for graphs of edge density above a certain bound. In view of the amount of previous work on heuristics for tree-width these results indicate that boolean-width could in the future outperform tree-width in practice for a large class of graphs and problems.

[☆]Supported by the Norwegian Research Council, project PARALGO.

Email address: Sadia.Sharmin@uib.no (Sadia SHARMIN)

1. Introduction

Many NP-hard graph problems become polynomial-time solvable when restricted to graphs of bounded tree-width or bounded clique-width. These algorithms usually have two stages, a first stage finding a decomposition of width k of the input graph, and a second stage of dynamic programming along the decomposition. The dynamic programming is typically exponential in k, e.g. given a decomposition of tree-width k it solves Maximum Weight Independent set in time $O(n2^k)$ and Minimum Weight Dominating set in time $O(n3^k k^2)$ [20]. It is therefore important to have fast algorithms for the first stage, i.e. to find decompositions of small width. For clique-width such algorithms are not known, apart from the 2^{OPT} approximation achieved through rank-width [13]. For tree-width there is an $O(f(n)2^{O(k^3)})$ algorithm for finding a decomposition of tree-width k, if it exists [3]. This algorithm is not practical [17], but much work has been done on finding decompositions of low tree-width in practical settings, see the overviews [5, 4]. The web site TreewidthLIB [19] has been established to provide a benchmark and to join the efforts of people working in experimental settings to solve graph problems using tree-width and branch-width [12, 16]. This includes problems from computational biology [18, 21, 22], constraint satisfaction [9, 11], and probabilistic networks [15]. However, tree-width and branch-width are unsuitable for non-sparse graphs, as a decomposition of tree-width or branch-width k means the graph has $O(k^2 n)$ edges. Clique-width, on the other hand, can be low for dense graphs, but so far no experimental study has been done for clique-width or similar notions. To our knowledge this paper is the first case of an experimental study on computing a notion of width that works also for non-sparse graphs.

Boolean-width is a recently introduced graph parameter motivated by algorithms [8]. It is defined by a decomposition tree that minimizes the number of different unions of neighbourhoods across resulting cuts of the graph. This decomposition is natural to solve problems where vertex sets having the same neighborhoods across the cuts can be treated as equivalent. This includes problems related to Independent Set, Dominating Set, Perfect Code, Induced k-Bounded Degree Subgraph, H-Homomorphism, H-Covering, H-Role Assignment etc [1]. Similarly to treewidth, dynamic programming algorithms to solve these problems using boolean-width employ a table at each node of the decomposition tree, to store solutions to partial problems. In contrast to treewidth, the dynamic programming for boolean-width involves a non-negligible pre-processing phase computing indices of the tables, the so-called 'representatives'. Regardless, the total runtimes are in many cases close to those for treewidth, e.g. given a decomposi-

tion of boolean-width k Max Weight Independent Set is solved in time $O(n^2k2^{2k})$ and Min Weight Dominating Set in time $O(n^2 + nk2^{3k})$ [8]. These boolean-width-based algorithms are straightforward and have been implemented in Java, without much effort, using only the description in [8]. Let us compare dynamic programming based on tree-width versus boolean-width, to solve Independent Set and Dominating Set, with focus on exponential factors. For Independent Set the exponential factor in the runtimes are 2^{tw} versus 2^{2boolw}, given decompositions of treewidth tw or boolean-width $boolw$, and boolean-width becomes preferable when $tw > 2boolw$. For Dominating Set the exponential factor in the runtime is 3^{tw} versus 2^{3boolw} and the cutoff is a bit lower, i.e. when $tw \geq 1.9boolw$.

It is known that boolean-width is never higher than tree-width or clique-width and it can be as low as logarithmic in clique-width [8]. For example, any interval graph or permutation graph has boolean-width $O(logn)$ [2] while there exist such graphs of clique-width $\Omega(\sqrt{n})$ and tree-width $\Omega(n)$. Also, a random graph with constant edge probability will almost surely have boolean-width $\Theta(\log^2 n)$ [1] but linear clique-width and tree-width. While these theoretical results favor boolean-width over tree-width, the cutoff $tw \geq 2boolw$ that we arrived at above applies when we are given a decomposition of treewidth tw or boolean-width $boolw$, as the output of a first stage algorithm. It is unknown if computing boolean-width is FPT or W-hard. In this paper we give a heuristic for the first stage, taking as input a graph G and finding a decomposition of G having reasonably low boolean-width. We tried various heuristics and present the one with best performance, which is a local search algorithm where the search for new solutions is based on interweaving between greedy choices and random choices. Theoretical evidence that random choices are useful for boolean-width, at least for random graphs, comes from the analysis of [1] showing that any decomposition of a random graph is expected to be a decomposition of relatively low boolean-width. On a set of graphs of practical relevance, specifically graphs in TreewidthLIB, the best known upper bound on their tree-width is compared to the upper bound on their boolean-width given by our heuristic. For 78% of those graphs in TreewidthLIB where both tree-width and boolean-width upper bounds were encountered, the tree-width bound is at least twice the boolean-width bound, thus meeting the $tw \geq 2boolw$ bound mentioned above. A drawback of tree-width is that it is always high when edge density is high. In contrast, boolean-width is typically low for dense graphs and our experiments show that within reasonable time we can find decompositions witnessing this. Our results indicate that, for problems like Dominating Set, using boolean-width will outperform dynamic programming by tree-width, at least for graphs of edge density above a certain bound. In view of

the amount of previous work on heuristics for tree-width we expect that further work on boolean-width heuristics will substantially increase the class of graphs for which boolean-width outperforms tree-width, also for other problems besides Independent Set and Dominating Set.

The rest of the paper is organized as follows. In Section 2 we define partial and full decomposition trees and boolean-width. In Section 3 we describe the heuristic finding a decomposition of low boolean-width. In Section 4 we describe the experimental results on graphs in TreewidthLIB, and also on small grid graphs. In Section 5 we draw some conclusions.

2. Boolean-width

We consider undirected graphs $G = (V, E)$ without loops. We denote the neighborhood of a vertex v by $N(v)$ and the union of neighborhoods of a vertex subset A by $N(A) = \cup_{v \in A} N(v)$. The complement of $A \subseteq V$ is denoted by $\bar{A} = V \setminus A$ and we call $(A, \bar{A})$ a cut of G. A partition of a set S consists of non-empty and disjoint subsets of S whose union is S. We follow custom by referring to vertices of a graph and nodes of a tree.

Definition 1 (Full and partial decomposition trees). A partial decomposition tree of a graph $G = (V, E)$ is a pair (T, δ), where T is a full binary tree and δ is a mapping from the nodes of T to non-empty subsets of V, satisfying the following: if x is the root of T then $\delta(x) = V$ and if nodes y and z of T are children of a node x then $(\delta(y), \delta(z))$ is a partition of $\delta(x)$. If a subtree of T rooted at x has $|\delta(x)|$ leaves then it is called a full decomposition subtree. If T has $|V|$ leaves then (T, δ) is called a full decomposition tree.

Note that in a partial decomposition tree (T, δ) of a graph G, if L is the set of leaves of T then $\{\delta(x) : x \in L\}$ is a partition of V. Hence in a full decomposition tree there will for each vertex v of G be a unique leaf x of T with $\delta(x) = \{v\}$. Likewise for each vertex of $\delta(x)$ in a full decomposition subtree rooted at x.

Definition 2 (Unions of neighborhoods and boolean-width). Let (T, δ) be a partial decomposition tree of a graph G. Let $V(T)$ be the nodes of T. Every node $x \in V(T)$ defines a cut $(\delta(x), \overline{\delta(x)})$ of G. The set of unions of neighborhoods of subsets of A across the cut $(A, \overline{A})$ is $UN(A) = \{N(X) \cap \overline{A} : X \subseteq A\}$. The boolean-width of (T, δ) is

$$boolw(T, \delta) = \max_{x \in V(T)} \{log_2 |UN(\delta(x))|\}$$

The boolean-width of a graph G is the minimum boolean-width over all its full decomposition trees $boolw(G) = \min_{\text{full } (T,\delta) \text{ of } G} \{boolw(T, \delta)\}$.

Note that $UN(A)$ are the subsets of $\overline{A}$ for which there exists an $X \subseteq A$ with $N(X) \cap \overline{A}$ being that subset, so we always have $\emptyset \in UN(A)$. It is known from boolean matrix theory [14] that $|UN(A)| = |UN(\overline{A})|$ and this is sometimes used by our code. Let us consider some examples. If $|UN(A)| = 2$ then the set of edges crossing the cut $(A, \overline{A})$ induce a complete bipartite graph. If the set of edges crossing the cut $(A, \overline{A})$ induce a perfect matching of G then $|UN(A)| = 2^{|V/2|}$. In the definition of boolean-width we take the logarithm base 2 of $|UN(A)|$ which ensures that $0 \leq boolw(G) \leq |V|$. If a graph has boolean-width one then it has a full decomposition tree such that, for every cut defined by a node of the tree, the edges crossing the cut, if any, induce a complete bipartite graph. From this it follows that the graphs of boolean-width one are exactly the distance-hereditary graphs [7].

Definition 3 (Split). A split of a set P is a partition into two subsets A and B, with the constraint that $min\{|A|, |B|\} \geq \frac{1}{3}|P|$.

3. Heuristic Algorithm

We present a local search heuristic that given a graph G computes a full decomposition tree of G. The search for new solutions in the space of candidate solutions is based on a fine balance between greedy choices and random choices. The heuristic, given in Algorithm 1, runs for a pre-defined length of time and then returns the best full decomposition found. Each heuristic pass iterates over all decomposition nodes of the current partial decomposition tree, including the children created by this heuristic pass. A newly created tree node always starts out as a leaf node, which δ maps to a set of vertices of G that may be larger than one. We keep track of the best full decomposition subtrees found for each $P \subseteq V$ encountered so far and call it $Best(P)$.

3.1. Greedy Initialization

Step 1 of Algorithm 1 greedily generates a full decomposition tree, to serve as the starting tree for the local search in Step 2. The greedy initialization starts with T containing a single node x (as both root and leaf) with $\delta(x) = V$ and repeatedly calls the **Split** subroutine until we get a full decomposition tree. The **Split**(P) subroutine returns a split (A, B) of P and is given in Algorithm 2. Starting with

Algorithm 1 : Generate a full decomposition of a given graph

Input: a graph G
Output: a full decomposition tree (T, δ) of G
Step 1: /*Greedily generate initial full decomposition tree*/
Initialize T with $V(T) = \{root\}$, $\delta(root) = V$
while $\exists$ leaf x of T with $|\delta(x)| > 1$
(A, B) = **Split**$(\delta(x))$;
Add leaves y and z as children of x with $\delta(y) = A$ and $\delta(z) = B$
for all $x \in V(T)$ store $Best(\delta(x))$, the subtree rooted at x
Step 2: /*Local Search for better trees*/
for fixed amount of time **do**
TryToImproveSubtree$(root)$
if (T, δ) is a full decomposition tree **then** $Best(V)$ = (T, δ)
return $Best(V)$

A being a random half of the vertices of P (unless P=V), it adds new vertices to A one by one in a greedy fashion while minimizing $|UN(A)|$ and $|UN(P \setminus A)|$, and returns the best split found along the way complying with the split constraint. The call of **Split**(V) at the root sets the initial conditions for the later splits and for this root-case we start with $A = \emptyset$, rather than a random half of the vertices, to allow the full benefit of the greedy choices. The local search in **TryToImproveSubtree** will for leaves of the current tree make calls to **Split**(P) but not for $P = V$, since the root of T will never again become a leaf and instead the **RandomSwap** subroutine described in the next subsection will be applied to the root.

The objective function optimized locally in **Split** is $|UN(A)|$, the number of unions of neighborhoods of A, which directly relates to boolean-width, see Definition 2. The computation of $|UN(A)|$ is done in a separate subroutine called **UN**(A) given in Algorithm 3. This subroutine starts by restricting from the cut $(A, \overline{A})$ to the subsets of vertices (S_1, S_2) having an edge going across the cut $(A, \overline{A})$. The list LN is used to accumulate the set $UN(A)$ in a straightforward way. Correctness is easy to show by induction on $|S_1|$. Early termination of the **UN**(A) subroutine is not shown in Algorithm 3 but is done if it is determined that $|LN|$ is too large for the cut $(A, \overline{A})$ to be interesting.

3.2. Local Search

The local search used to improve the current decomposition tree is initiated at the root of the tree T, in Step 2 of Algorithm 1. In the subroutine **TryToImprove-**

Algorithm 2 : **Split**(P)

Input: Set of vertices $P \subseteq V$.
Output: a partition (A,B) of P s.t. $min\{|A|, |B|\} \geq \frac{1}{3}|P|$.
if $P = V$ **then** $A_1 \leftarrow \emptyset$
else $A_1 \leftarrow$ random half of the vertices in P
$i = 1$
while $|P \setminus A_i| \geq \frac{1}{3}|P|$ **do**
  find $x \in P \setminus A_i$ s.t. $max\{\mathbf{UN}(A_i \cup \{x\}), \mathbf{UN}((P \setminus A_i) \setminus \{x\})\}$ is minimized.
  $A_{i+1} = A_i \cup \{x\}$.
  $i = i + 1$.
end while
find i such that $max\{\mathbf{UN}(A_i), \mathbf{UN}(P \setminus A_i)\}$ is minimized and $|A_i| \geq \frac{1}{3}|P|$.
return (A_i,$P \setminus A_i$).

Subtree(x), given in Algorithm 4, x is a node of the current partial decomposition tree (T, δ) and the goal is to improve the subtree of T rooted at x. That subroutine has four main parts.

(1) if x leaf then find candidate for split of its subset

(2) if x non-leaf then find candidate for swap of its two children subsets

(3) conditionally update (T, δ)

(4) for each child of x either use stored subtree or recurse

For (1) we use the **Split** subroutine described earlier. For (2) we use the **Random-Swap**(A,B) subroutine given in Algorithm 5 that randomly swaps vertices between A and B while complying with the split constraint. At the very onset of the local search, the current (T, δ) is the full decomposition tree found by the greedy initialization. However, the current decomposition tree ceases to be full as soon as the split given by **RandomSwap**$(\delta(y), \delta(z))$ in (2) is a good one and (3) updates (T, δ) so that y and z become leaves. If the new $\delta(y)$ is a subset of vertices for which a full decomposition subtree has never been stored, or the stored one is not good enough, then in (4) a recursive call is made to **TryToImproveSubtree**(y), with y a leaf of the current tree. If in that recursive call the split found in (1) is not good then in (3) we will return with y a leaf of the current (T, δ) having $|\delta(y)| > 1$, which explains the if-statement at the very end of Algorithm 1.

Algorithm 3 : **UN**(A)

Input: Set of vertices $A \subseteq V$.
Output: $|UN(A)|$, the number of unions of neighborhoods of the cut $(A, \overline{A})$
if $|UN(A)|$ has already been computed **return** the stored value
$S_1 = \{v \in A : \exists u \in \overline{A} \wedge (u, v) \in E\}$
$S_2 = \{v \in \overline{A} : \exists u \in A \wedge (u, v) \in E\}$
$LN \leftarrow \{\emptyset\}$ /$*$neighborhood set accumulator$*$/
for all $u \in S_1$ **do**
  for all $Y \in LN$ **do**
    $X \leftarrow (N(u) \cap S_2) \cup Y$
    if $X \notin LN$ **then** add X to LN
return The number of elements in LN

Note that the local improvements made in the local search are based on randomly swapping vertices between $\delta(y)$ and $\delta(z)$ for two nodes y and z with the same parent. As usual in local search, there is a fine balance to trying new splits versus sticking with old splits. The goal is to neither get stuck in local minima nor to swap so many nodes that we re-randomize completely and don't get a hill-climbing effect. Note in (4) that we store for each subset P of vertices encountered so far the best found full decomposition subtree $Best(P)$. The decision of when to try new splits and when to use the old splits is tied to the boolean-width of the best subtrees, and to the upper bound on boolean-width of G given by $Best(V)$.

3.3. Discussion and Implementation Details

We made our implementations in Java. Subsets of vertices are stored as bitvectors of length n, i.e. the number of vertices in the graph. We expect most of the subsets we store to be of size at least $\frac{n}{2}$ so this is an efficient way to store subsets. We also limited the boolean-width to 31, i.e. $|UN(A)| \leq 2^{31}$, but none of the graphs tested reached this limit. The bottleneck is rather the memory available on our machines. Let us explain. Our implementation of subroutine **UN**(A) uses memory proportional to $n * |UN(A)|$ bits. Since $|UN(A)| \leq 2^{min(|A|,|\overline{A}|)}$ the 'boolean-width ≤ 31' becomes a bottleneck only if the graph has at least 64 vertices. In that case the implementation is handling a list of neighborhoods of size $64 * 2^{31}$ bits which is 16 GB of memory and that is more memory than our desktop had. It is part of future research to find memory efficient methods to compute $|UN(A)|$.

Algorithm 4 : **TryToImproveSubtree**(x)

Input: a node x of T with $|\delta(x)| > 1$

(1) if x is a leaf **then** (A,B) = **Split**($\delta(x)$)

(2) else

Let y and z be the children of the node x.

(A, B)=**RandomSwap**($\delta(y), \delta(z)$)

(3) if $max\{\mathbf{UN}(A), \mathbf{UN}(B)\} < boolw(Best(V))$

then Set y and z as new leaf children of x with $\delta(y) = A$ and $\delta(z) = B$

else if x is still a leaf **then return** /* in case we came from (1) */

(4) if $max\{\mathbf{UN}(\delta(y)), \mathbf{UN}(\delta(z))\} < boolw(Best(V))$ **then**

for $w \in \{y, z\}$

if $boolw(Best(V)) > boolw(Best(\delta(w)))$

then use root of $Best(\delta(w))$ as w.

else if $|\delta(w) > 1|$ call **TryToImproveSubtree**(w)

if the subtree T_x rooted at x is a full subtree of $\delta(x)$

then update $Best(\delta(x))$ to T_x

As described, we are currently storing the best full decompositions of subtrees. Since bitvectors are easy to compare they are stored in a binary search tree for quick look-up. Storing all these solutions eats up memory, and for some big graphs this is the limiting factor. In the future we will consider more advanced schemes for storing the partial solutions encountered. In particular one should throw out elements that are no longer below the upper bound.

The search for new solutions in the space of candidate solutions is based on a fine balance between greedy choices and random choices, a balance that was arrived at mainly through experimentation. This appears e.g. in the choice of letting the **Split** subroutine start with a random half of the nodes on one side before trying vertices one-by-one in the more costly greedy stage. Similarly for the fully random choice of swapping in subroutine **RandomSwap**, and in the conditional tests in (3) and (4) of **TryToImproveSubtree**.

Although not specified in the pseudocode, for small subtrees we just return an arbitrary one, since if $|\delta(x)| \leq boolw(Best(V))$ then any full subtree at x will have boolean-width at most $boolw(Best(V))$. The **Split**(P) subroutine given in Algorithm 2 could be stopped as soon as a subset A_i with low $|UN(A_i)|$ and $|UN(P \setminus A_i)|$ values has been found. It is not clear that this is always better and currently it is not done. There are many calls of $\mathbf{UN}(A)$ for many subsets A that

Algorithm 5 : **RandomSwap**($\delta(y), \delta(z)$)

Input: $\delta(y), \delta(z) \subseteq V$ for sibling nodes y and z of T.
Output: split (A, B) of $\delta(y) \cup \delta(z)$.
Let x be the parent of y and z.
choose randomly i in $0..(|\delta(y)| - \frac{|\delta(x)|}{3})$ and j in $0..(|\delta(z)| - \frac{|\delta(x)|}{3})$.
choose randomly $M_i \subset \delta(y)$ and $M_j \subset \delta(z)$ with $|M_i| = i$ and $|M_j| = j$.
$A = (\delta(y) \setminus M_i) \cup M_j$
$B = (\delta(z) \setminus M_j) \cup M_i$
return (A, B).

only differ in a few vertices. A possible improvement is to store the sets of unions of neighborhoods $UN(A)$ and use these e.g. when computing $UN(A \cup \{v\})$ for a single added vertex v, allthough it is not clear how to do this efficiently. The **UN**(A) subroutine given in Algorithm 3 does not recompute known values, but otherwise it may seem naive. It forms the inner loop of the heuristic and it is the bottleneck for running on graphs with many vertices. We tried different approaches such as randomly sampling subsets to approximate $|UN(A)|$ and exploiting a correlation between the degree of a vertex and its contribution to $|UN(A)|$. These tests led to only insignificant improvements so for the moment we kept the naive algorithm. There are other, similar, improvements to **UN**(A) that can be attempted, and although they may not asymptotically improve the running-time of the heuristic they could potentially be of big help.

The balance between trying new splits and sticking to old splits is guided by the conditional test in (3) of Algorithm 4. We did try imposing stronger conditions in order to arrive at better splits sooner, but only minor improvements were seen, and only in some cases.

The heuristic ran for a predefined amount of time for each graph but there are several ways of experimenting with the stopping criteria, for example based on the size of the input graph, or on the fraction of time since an improved tree was last found.

4. Experimental Results

All presented results have been carried out on a Linux machine with 2.33 GHz Intel Core 2Duo CPU E6550 and 2 GB RAM. Our aim was not fast benchmark results, but to explore heuristics for finding decompositions of low boolean-width. TreewidthLIB is an online depository containing a collection of 710 graphs, to

be used as a benchmark for the comparison of algorithms computing treewidth. TreewidthLIB provides selected instance graphs, for which computing the tree-width is relevant, originating from applications like probabilistic networks, vertex coloring, frequency assignment and protein structures [5]. We ran our heuristic on the graphs in TreewidthLIB.

TreewidthLIB contains 710 graphs. For 482 graphs a tree-width bound is given in TreewidthLIB, and for 426 graphs we give a boolean-width bound using our heuristic. For the comparison we concentrate on the 300 graphs for which we have a bound on both tree-width and boolean-width, but let us first discuss the remaining 410 graphs. Among these 410 graphs, there are 126 having only a boolean-width bound, 182 having only a tree-width bound, and 102 having neither. Among the 182 graphs having only a tree-width bound there are some in a graph format not supported by our implementation, but for the majority of these graphs our heuristic simply timed out already at the greedy initialization stage. Note that for these 182 graphs, if we were given the decomposition of low tree-width k, we could easily have produced a decomposition of boolean-width at most k, using the $O(nk^2)$ algorithm which can be deduced from [1].

We now summarize our findings for the 300 graphs having both a tree-width bound and a boolean-width bound. Firstly, the boolean-width bound is always better than the tree-width bound, with the ratio of the tree-width bound divided by the boolean-width bound ranging from 1.15 to 29, with an average of 3.13. Not surprisingly, the ratio increased with higher edge density. In Fig.1 we have plotted this ratio against the edge density of the graphs for a total of 300 graphs. The trend line shows the growth of ratio with edge density.

Our heuristic algorithm starts with greedily finding a full decomposition tree giving an Initial Bound on boolean-width and then improves this bound iteratively. In the experiments we kept track of the decrease in the boolean-width over time. In Fig. 2 and Fig. 3 the upper bounds on boolean-width, i.e. the values of $boolw(BEST(V))$, are shown as they decrease over time, for the two graphs called *eil51.tsp* (V=51 and E=140) and *miles1500* (V=128,E=5198). For the graph *eil51.tsp* the Initial Bound was 9.1 after less than a second, then at the 'knee' of the curve before the improvement decays we found a Fast Bound of 6.2 after 4 seconds, and finally the Best Bound of 5.8 was found after 124 seconds. For each graph, we can likewise speak of three bounds: i) the Initial Bound given by the greedy initialization, ii) a Fast Bound found at the 'knee' of the curve, and iii) the Best Bound found possibly after a long runtime.

In Table 1 we summarize results for 8 selected graphs having a good variety of number of vertices V, edge density $density$, Time in seconds to find Initial Bound,

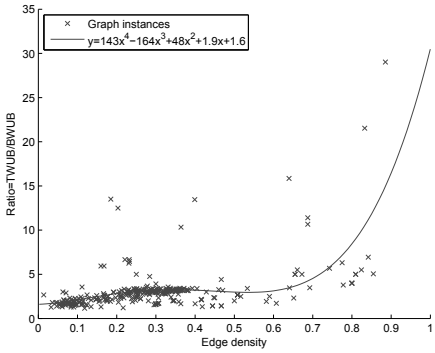


Figure 1: Ratio (treewidth divided by boolean-width) versus edge density in all the 300 graphs for which heuristically computed upper bounds are known.

Fast Bound, and Best Bound on boolean-width, its best known treewidth upper bound TWUB, and Ratio=TWUB/BWUB(Best Bound). The graphs are sorted by this Ratio. The *miles1500* graph is translated from the Stanford GraphBase. The *zeroin.i.1* and *mulsol.i.5* graphs originate from the 2nd DIMACS implementation challenge [10] and are generated from a register allocation problem based on real code. The *queen8_12* also comes from the DIMACS[10] graph coloring problems and is an example of n-queens puzzle. The graph *1awd* is from the field of computational biology with each vertex representing a single side chain and each edge representing the existence of a pairwise interaction between the two side chains. The graph *celar06-wpp* is a frequency assignment instance. The graph *BN_28* originates from Bayesian Network from evaluation of probabilistic inference systems at UAI 2006. The graph *eil51.tsp* is a Delauney triangulation of a traveling salesman problem.

4.1. Small grid graphs

We also ran our heuristic on graphs corresponding to the $n \times n$ grid. However, for square grids the current implementation of $\mathbf{UN}(A)$ is too memory-intensive and we had to limit the size to $n \leq 9$. These are sparse graphs having tree-width n and the upper bound we find on boolean-width is below this. See Figure 4. The boolean-width of square $n \times n$ grids is a topic we are investigating and our current guess is that the optimal upper bound, holding for all n, is about $0.8 * n$. If this is correct, the value computed by the heuristic is close to optimal, which

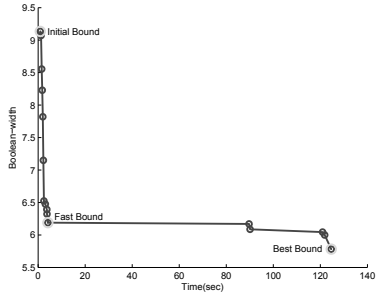


Figure 2: Improvement of boolean-width upper bound as the local search progresses over time, for the graph *eil51.tsp* (V=51,E=140)

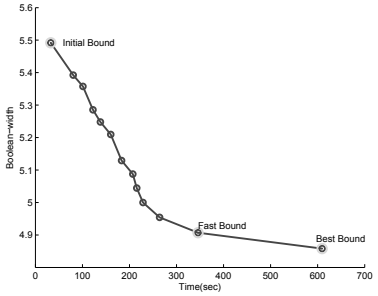


Figure 3: Improvement of boolean-width upper bound as the local search progresses over time, for the graph *miles1500* (V=128,E=5198)

Table 1: Results for selected graphs

		Edge	Initial Bound		Fast Bound		Best Bound			
Graph name	V	*density*	BWUB	Time(s)	BWUB	Time(s)	BWUB	Time(s)	TWUB	Ratio
miles1500	128	0.64	5.5	32.6	4.9	345.7	4.8	609.6	77	15.85
zeroin.i.1	211	0.19	4.0	74.1	3.8	116.2	3.7	168.0	50	13.51
mulsol.i.5	186	0.23	6.4	55.3	5.4	130.0	4.9	365.2	31	6.25
queen8_12	96	0.30	16.7	3055	16.7	3055	16.7	3055	65	3.91
1awd	89	0.27	13.3	67.5	11.1	521.1	10.8	702.9	38	3.52
celar06-wpp	34	0.28	4.5	0.1	3.2	0.8	3.0	4.8	11	3.37
BN_28	24	0.18	3.3	0.02	2.3	0.05	2.0	0.3	5	2.50
eil51.tsp	51	0.11	9.1	0.9	6.2	4.1	5.8	124.6	9	1.55

is somewhat interesting as it is our understanding that the heuristics for finding decompositions of low tree-width do not perform well on grid graphs.

5. Conclusion

We presented the first experimental study on computing a notion of width that works also for non-sparse graphs, based on the boolean-width parameter. Experiments with the graphs in TreewidthLIB show the strength of boolean-width versus tree-width, in a practical setting, in particular for graphs of edge density above a certain value. For more examples of real-world graphs of high edge density and high tree-width we could also look beyond the TreewidthLIB library. There are a number of open problems related to boolean-width heuristics and some have already been discussed in subsection 3.3. Firstly, we need a fast heuristic that directly constructs a reasonable upper bound on the boolean-width for any graph,

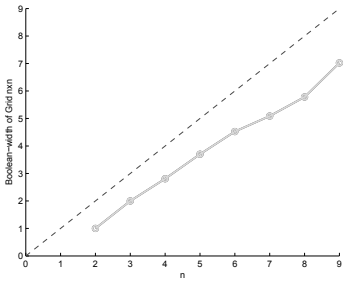


Figure 4: Upper-bound on boolean-width, as computed by our heuristic, for the $n \times n$ grid, with n ranging from 2 to 9. Tree-width is given by the dotted line $x = y$.

regardless of how big the graph is or what its edge density is. The main issue will be to give a fast heuristic for the computation of a good upper bound on $|UN(A)|$. Secondly, we need to consider heuristics for computing lower bounds on boolean-width, just as it has been done for tree-width [6]. Thirdly, we should explore pre-processing to simplify the graph instances, again this has been done extensively for tree-width [4]. These problems are of interest since our results indicate that using boolean-width could in the future outperform the use of tree-width in practice for a large class of graphs and problems.

References

[1] I. Adler, B. M. Bui-Xuan, Y. Rabinovich, G. Renault, J. A. Telle, and M. Vatshelle. On the boolean-width of a graph: Structure and applications. *Proceedings of the 36th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2010*, pages 159–170, 2010.

[2] R. Belmonte and M. Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2011*, 2011. see full version `www.ii.uib.no/~martinv/Papers/LogBoolw.pdf`.

[3] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[4] Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In Fedor V. Fomin, editor, *Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2006*, pages 1 – 14. Springer Verlag, LNCS, vol. 4271, 2006.

[5] Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208:259–275, 2010.

[6] Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations II. lower bounds. Technical Report UU-CS-2010-022, Department of Information and Computing Sciences, Utrecht University, Utrecht, the Netherlands, 2010. Accepted for publication in Information and Computation.

[7] A. Brandstadt. personal communication.

[8] B. M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Boolean-width of graphs (to appear). *Theoretical Computer Science*, 2011. see full version `www.ii.uib.no/~telle/bib/listofpub/BTV11.pdf`.

[9] Hubie Chen. Quantified constraint satisfaction and bounded treewidth. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 17th European Conference on Artificial Intelligence, ECAI 2004*, pages 161–165, 2004.

[10] The second DIMACS implementation challenge: NP-Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability. See http://dimacs.rutgers.edu/Challenges/, 1992–1993.

[11] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Acta Informatica*, 124:243–282, 2000.

[12] Illya V. Hicks, Arie M. C. A. Koster, and Elif Kolotoğlu. Branch and tree decomposition techniques for discrete optimization. In J. Cole Smith, editor, *TutORials 2005*, INFORMS Tutorials in Operations Research Series, chapter 1, pages 1–29. INFORMS Annual Meeting, 2005.

[13] P. Hliněný and S. Oum. Finding branch-decomposition and rank-decomposition. *SIAM Journal on Computing*, 38:1012–1032, 2008.

[14] K. H. Kim. Boolean matrix theory and its applications. 1982. Marcel Dekker.

[15] S. J. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *The Journal of the Royal Statistical Society. Series B (Methodological)*, 50:157–224, 1988.

[16] Arnold Overwijk, Eelko Penninkx, and Hans L. Bodlaender. A local search algorithm for branchwidth. *Proceedings of the 37th Conference on Current Trends in Theory and Practive of Computer Science, SOFSEM 2011*, pages 444–454, 2011.

[17] Hein Röhrig. Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1998.

[18] Y. Song, C. Liu, R. Malmberg, F. Pan, and L. Cai. Tree decomposition based fast search of RNA structures including pseudoknots in genomes. In *Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference, CSB'05*, pages 223–234, 2005.

[19] Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib, 2004– ....

[20] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Proceedings of the 17th Annual European Symposium on Algorithms, ESA 2009*, pages 566–577. Springer Verlag, Lecture Notes in Computer Science, vol. 5757, 2009.

[21] Jizhen Zhao, Dongsheng Che, and Liming Cai. Comparative pathway annotation with protein-DNA interaction and operon information via graph tree decomposition. In *Proceedings of Pacific Symposium on Biocomputing, PSB 2007*, volume 12, pages 496–507, 2007.

[22] Jizhen Zhao, Russell L. Malmberg, and Liming Cai. Rapid ab initio prediction of RNA pseudoknots via graph tree decomposition. *Journal of Mathematical Biology*, 56(1–2):145–159, 2008.