# A Search for the Dominating-Set-Width

by

**Ruben Henri Meuwese**

Master Thesis Computing Science
ICA-3854671

Utrecht University

Supervisors:

Dr. E.J. van Leeuwen
Dr. J.M.M. van Rooij

Utrecht, 2020

*Dedicated to*
*Piet van Dijk and Henriette Smit*
*who never stopped wondering and learning about the world.*

## Abstract

In this master's thesis we provide a new width measurement definition on graphs. This width measurement determines the number of different dominating sets on a subgraph of a graph. Therefore we call it the dominating-set-width. With this width we create a fixed-parameter tractable dynamic programming algorithm that solves the minimum dominating set problem. Dominating-set-width is based on another width measurement, called boolean-width, which was introduced by Bui-Xuan et al. [6] along with an algorithm that solves the minimum dominating set problem. We will show that our algorithm has a faster run time than the one introduced by Bui-Xuan et al. [6]. Furthermore, this master's thesis provides a data structure for this new width, as well as an upper and lower-bound for the width when a graph is restricted to a graph class.

# Acknowledgements

I would like to take this opportunity to first and foremost thank my supervisors Erik Jan van Leeuwen en Johan van Rooij for giving me the chance to do this research project. They showed a lot of trust in me when things almost took a bad turn midway and stuck with me to the end.
Second, I would like to thank are Laurens Stoop, Rogier Wuijts and Roel Brouwer for allowing me to work in their office and providing a fantastic work environment.
Third, I would like to thank Shawn Pieters Kwiers and Stef van Gogh for helping me with the tiny details.
Finally, I wish to thank my family and friends who supported me during my studies and provided the necessary distractions that kept my spirits up.
A special thank goes to my girlfriend Robin Kramer for her patience with me during the final stages of this master's thesis.

# Contents

# 1 Introduction

In the study of algorithmic design for NP-hard problems there are many different fields of study that research fast theoretical ways to solve different NP-hard problems. One of these fields of study is the study of parameterized algorithms. This will be the focus of this master's thesis.

## 1.1 Parameterized Algorithms

Parameterized algorithms looks at other ways to express the complexity for a problem rather than the classical way. These algorithms add an extra parameter to the complexity, which provides more information about the difficulty of a problem, which in turn can create much faster algorithms. The run times of these algorithms can be expressed as $O(f(n,k))$ with $f$ a computable function, $n$ the complexity of the input size and $k$ the added parameter. There are many usages for these algorithms.

A subsection or subset of parameterized algorithms are the algorithms for fixed-parameter tractable problems, also known as FPT problems or just FPT algorithms. Fixed-parameter tractable problems are problems which can be solved in $O(f(k) \cdot p(n))$ time with $f$ a computable function and $p$ a polynomial. These run times are often noted by $O^*(f(k))$. One of the big differences between FPT algorithms and general parameterized algorithms is that FPT algorithms exclude run times like $O(n^k)$ or $O(k^n)$.

For more information we refer the reader to Cygan et al. [9], Downey and Fellows [11], Fomin and Kratsch [13] and Flum and Grohe [12].

## 1.2 Motivation of the Master's Thesis

We focus on the FPT algorithms which solve graph problems that are NP-hard. In graph problems the parameter of the FPT algorithm are often properties of a graph. The highest degree of all the vertices in the graph or the length of the longest shortest path between two vertices. The parameters that are related to the decomposition of the whole graph are referred to as a width parameter or just the width of a graph. We shall see that there is a variety of widths and that these are best suited to solve a variety of different problems.

The power of these widths lies in their size or value. If a width is significantly lower than the input size, then the FPT algorithm has a runtime that is practically polynomial. This way NP-hard graph problems become much more manageable.

## 1.3 Research Questions

The goal of this master's thesis is to find a good width parameter for solving the minimum dominating set problem.

**Definition 1.** *Let $G$ be the graph $(V,E)$. A subset $D \subseteq V(G)$ is a dominating set if every vertex $v \in V(G)$ is either in $D$ or is adjacent to a vertex in $D$. The minimum dominating set problem is defined as: Given a graph $G$ find the smallest possible set $D$ such that $D$ is a dominating set.*

We define a good width as a width that allows for a fast FPT algorithm and has a relatively lower value compared to other known widths. This width shall be called the dominating-set-width. The main research question is:

**Question:** What is the definition of the dominating-set-width?

The following questions will provide the necessary insights and arguments that will result in a good answer to the main research question:

**Question:** How fast can algorithms using dominating-set-width be when solving the minimum dominating set problem?
**Question:** Given a graph, what is its dominating-set-width?

## 1.4 Content Description

We start with some preliminaries about the general definition of a width for a FPT algorithm in section 2. We will also go over some known widths to give a better idea of what a width is.

In section 3 we dive into the details of one of the widths mentioned in section 2 called boolean-width. We will see that this width will serve as a stepping stone to find a definition for the dominating-set-width. This definition will be discussed in section 4. In the same section we will provide a new FPT algorithm that solves the minimum dominating set problem and prove its correctness.

In section 5 we will discuss the data structure necessary for running the new algorithm given in section 4.

And finally, in section 6 we will go over different graph classes and look for the upper-bound of the dominating-set-width value for each of these classes.

## 2 Preliminaries

In this section we have a look at how widths are defined in a general way and how they are used. We look at a selection of known width parameters and their relation to each other. After that we look at two widths in more detail: tree-width and boolean-width.

### 2.1 The Definition of a Width

Most widths we will mention in this section are formed by some measurement on a cut of a graph. Generally speaking a cut is nothing more than splitting a graph into two subgraphs. This can be done for example, by partitioning the edges into two sets. A measurement on these types of cuts can be almost anything. For example, it can be the number of vertices that are endpoints of edges on both side of the cut, this is called branch-width (Robertson et al. [17]). Another way to form a cut is by partitioning all the vertices of the graph into two sets. The measurements on a cut like this can again be almost anything. A simple measurement, for example, is the size of the maximum matching on the edges across the cut. This width measurement is called the maximum-matching-width (Vatshelle [23]). An example of a more complicated measurement is boolean-width (Vatshelle [23]).

Boolean-width uses an equivalence relation on the subsets of the vertices on one side of the cut. The number of equivalence classes is the measurement on the cut. In section 3 we will show that the equivalence classes formed by this definition have a relation to the number of different independent sets on a cut.

In general these different measurements on cuts are used in combination with a decomposition tree. A decomposition tree $(T, \delta)$ for a graph $G$ is a binary tree graph $T$ for which each leaf node represents a vertex or edge of $G$ depending on what type of cuts are used. This representation is done by creating a bijection $\delta$ from the leaf nodes to the vertices or edges. The tree graphs $T$ can either be rooted or not; in this thesis we assume every decomposition tree is rooted.

If we remove an edge in $T$, we split the tree into two parts, creating two trees. This creates two disjoint sets of leaves and thus two disjoint sets of vertices or edges via the bijection $\delta$, inducing a cut on which a width measurement can be used. This can be done for any edge in $T$ and with that the width of $T$ can be defined. The width of $T$ is the maximum width measure value taken over every cut induced by removing an edge in $T$. The width of a graph $G$ is the minimum width value over every possible decomposition tree $(T, \delta)$. To formalize this:

**Definition 2.** *Let $G$ be the graph $(V, E)$. A decomposition tree for $G$ is a pair $(T, \delta)$ such that:*

1. *$T$ is a rooted binary tree graph with root $r$.*
2. *$\delta$ is a bijection from the leaves of $T$ to $V(G)$ (or $E(G)$ depending on the width measurement).*

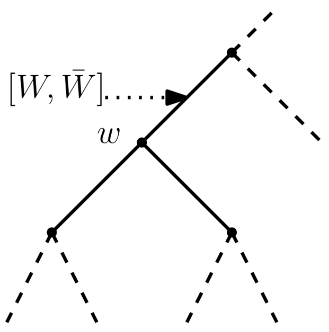

Figure 1: Example of a cut induced by the node $w$ of a decomposition tree.

**Definition 3.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree for $G$. We define $W \subseteq V(G)$ $(E(G))$ for any node $w$ as the set of vertices (edges) corresponding to the leaves in the subtree rooted at $w$ in $T$. The edge from $w$ to its parent node in $T$ represents a cut, denoted by $[W, \bar{W}]$ with $\bar{W} = V(G) - W$ $(\bar{W} = E(G) - W)$. The width of $(T, \delta)$ is the maximum width measure value taken over all the cuts $[W, \bar{W}]$.*

From now on, for the sake of notation and clarity, when we say "$w$ induces a cut $[W, \bar{W}]$" we mean the cut induced by the edge from $w$ to its parent node in $T$ as shown in Figure 1 .

**Definition 4.** *Let $G$ be the graph $(V, E)$. For any chosen width measure defined on a cut, we define the width of $G$ as the minimum width value of every decomposition tree $(T, \delta)$ of $G$.*

Decomposition trees with a low width value can be used to get fast dynamic programming algorithms, also known as DP algorithms. This is done by going from the leaf nodes to the root of the tree moving up one level at a time. The width of a cut will provide insight into which partial solutions for a subgraph associated to the cut have to be stored and which ones can be discarded. We will see a clear example of such an application of discarding partial solutions in Section 3 and 4.

One width that does not follow this idea of a decomposition tree is tree-width (Seymour and Thomas [21]). Tree-width constructs a new graph on which a measurement is defined that forms the value of the width. These new graphs, which are usually if not always tree graphs, have constraints limiting the different ways they can be constructed.

With tree-width the additional tree graph, called a tree decomposition - not to be confused with a decomposition tree - is constructed with something ofter referred to as bags. These bags are the nodes in the tree decomposition.

**Definition 5.** *Let $G$ be the graph $(V, E)$. A bag is a subset of $V(G)$. A tree decomposition of $G$ is a tree graph $(B, F)$ such that $B$ is a collection of bags and $F$ is a collection of edges between the bags such that:*

- *For every vertex $v \in V(G)$ all bags in $B$ containing $v$ form a connected subgraphs in the tree decomposition.*
- *For every edge $e \in E(G)$ there is a bag in $B$ such that both endpoints of $e$ or in that bag.*

*The tree-width of a tree decomposition is the size of the largest bag minus one. The tree-width of a graph is the minimum tree-width value over all possible tree decompositions.*

Although this may seem a bit weird at first glance, the takeaway from this width definition is that it shows how much a graph looks like a tree graph. The more a graph looks like a tree, the easier it becomes to run a dynamic programming algorithm on it, creating a faster algorithm.

## 2.2 Overview of Known Widths

The widths mentioned in the previous subsection are just some of the known widths. There are many more widths, we will only mention a few by name to give an impression of the landscape of all the widths. Most of the known widths will fall into one of the following three categories:

1. The width measure is a property of a new tree graph $T$ created with the vertices of G, i.e. tree-width (Seymour and Thomas [21]) and tree-depth (Iwata et al. [15])
2. The width is defined on a decomposition tree $(T, \delta)$ where the leaves of $T$ represent the vertices of $G$ and the internal nodes represent subsets of $V(G)$ creating cuts where a width measurement is used, i.e. boolean-width (Vatshelle [23]), clique-width (Courcelle et al. [8]), rank-width (Oum [16]) and maximal-matching-width (Vatshelle [23]).
3. The width is defined on a decomposition tree $(T, \delta)$ where the leaves of $T$ represent the edges of $G$ and the internal nodes represent subsets of $E(G)$ creating cuts where a width measurement is used, i.e. branch-width (Robertson et al. [17]).

The relation between widths has been the subject of a lot of research as well as the proofs for NP-hardness of computing the optimal width value of a graph. Here we show some of these results.

Figure 2 is taken from Bui-Xuan et al. [6] as well as the description beneath it. Both use different abbreviation notations for the width than in this thesis. Figure 2 shows proven relations between widths with respect to the upper-bound of each width.

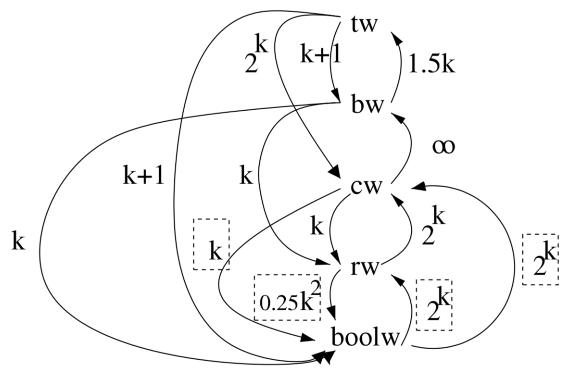

Figure 2: Upper bounds tying parameters $tw = tree-width$, $bw = branch-width$, $cw = clique-width$, $rw = rank - width$, and $boolw = boolean - width$. An arrow from $P$ to $Q$ labeled $f(k)$ means that any class of graphs having parameter $P$ bounded by $k$ will have parameter $Q$ bounded by $f(k)$ or $O(f(k))$, and $\infty$ means that no such upper bound can be shown. The results surrounded by a box are shown in Bui-Xuan et al. [6] . 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 $tw \rightarrow cw$ where an $\Omega(2k/2)$ bound is proven by Corneil and Rotics [7] , and $rw \rightarrow boolw$ where an $\Omega(k)$ bound is known (Theorem 2 of Bui-Xuan et al. [6]).

Ahn and Jeong [1] proved that computing the maximum matching width is NP-hard.

Seymour and Thomas [20] proved that solving a problem concerning routing of phone lines is NP-hard. This problem can be translated to calculating the optimal branch-width by generalizing the problem to a property of a graph, proving that computing branch-width is NP-hard.

Arnborg et al. [2] proved that computing the tree-width for a given graph is NP-hard. This was done by looking at a different decision problem: for a given value $k$, is $G$ a $k$-tree? If the answer is yes, then you would know a way to construct $G$ via an ordering on the vertices of $G$. This order can be used to make a tree decomposition with tree-width $k + 1$.

Bodlaender et al. [4] proved the NP-hardness for tree-depth. Although at the time it was known as the rank of a graph or rank-width and not tree-depth. Not to be confused with rank-width, which was mentioned above in the three categories and in Figure 2.

Sæther and Vatshelle [19] provide a simple criterium that proves computing a width parameter defined by a measurement on a cut is NP-hard. With this we can prove that the computation of the optimal boolean-width, rank-width and clique-width of a graph is NP-hard.

Vatshelle [23] (chapter 6) shows that for some widths, even when the optimal width value of a graph is known, constructing a decomposition tree that has the correct width value is NP-hard.

## 3 The Boolean-Width

In this section we will take a more in-depth look at boolean-width. Boolean-width will prove to be a good stepping stone for the search for the dominating-set-width.

### 3.1 The Intuition of Boolean-Width

Boolean-width is a width definition on a cut in a graph as mentioned in section 2. The internal nodes of a decomposition tree represent subsets of the vertices of a graph and induce cuts in the graph. Boolean-width looks at the different ways vertices make connections across a cut. To be more specific, boolean-with takes a set of vertices on one side of a cut and looks at the vertices adjacent to this set across the cut. This set of vertices will be called the neighborhood of a set across the cut or sometimes just the neighborhood.

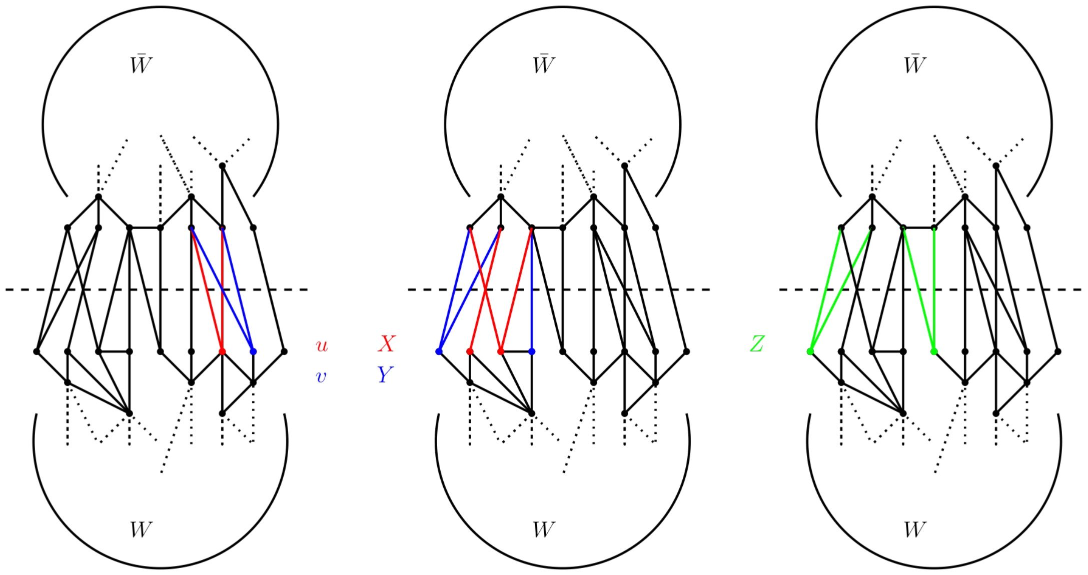

Figure 3: Example of a cut splitting graph $G$ into two subgraphs, $W$ and $\bar{W}$.

We are only interested in the vertices who have a neighborhood across the cut; that is why the other vertices and edges of $G$ in Figure 3 are not shown. The vertices $u$ and $v$ are adjacent to the same set of vertices across the cut in $\bar{W}$. We can say that $u$ and $v$ have the same neighborhood. We can extend this idea to subsets of vertices of the subgraph instead of just single vertices. For example, the sets of vertices $X$ and $Y$ are adjacent to the same set of vertices across the cut in $\bar{W}$. $Z$, however, is not adjacent to the same set of vertices across the cut. We can say that $X$ and $Y$ have the same neighborhood and $Z$ has a different neighborhood.

If we extend the subgraph $W$ shown in the example, every vertex in $\bar{W}$ adjacent to $X$ will be adjacent to $Y$, because they have the same neighborhood. With this observation we can ask ourselves the question: "How different are the sets of vertices $X$ and $Y$ from each other?". The answer to this question is the idea behind the definition of boolean-width. This definition will say that $X$ and $Y$ are not different at all, they are equivalent.

### 3.2 The Definition of Boolean-Width

Now we will formalize the intuition of boolean-width starting with the relation $u$ and $v$ have in Figure 3. This relation is an equivalence relation, thus it creates equivalence classes for which we can define a representative.

**Definition 6.** *Let $G$ be the graph $(V, E)$, $(T, \delta)$ a decomposition tree for $G$ and let $[W, \bar{W}]$ be the cut induced by a node $w$ in $(T, \delta)$. Vertices $u$ and $v$ in $W$ are called twins if $N(u) \cap \bar{W} = N(v) \cap \bar{W}$, with $N(u)$ the set of vertices adjacent to $u$. A twin class of $W$ is a subset of $W$ such that all vertices are twins of each other. The representative of a twin class in $[W, \bar{W}]$ is the lexicographically smallest element in the class.*

The lexicographically order is basically the formalization of putting vertices and subsets of $V(G)$ in alphabetical order. This is achieved by labeling every vertex of a graph. The order does not need to be the alphabetical order. Any order on the labels of the vertices will suffice as long as the order is a total order.

The twin class where $u$ and $v$ belong to shall be represented by $u$ as there are no other vertices on the cut which are twins of $u$ and $v$ and $u$ comes before $v$ in the alphabetical order and thus also in the lexicographically order. Now we move on to the relation $X$ and $Y$ have in Figure 3.

**Definition 7.** *Let $G$ be the graph $(V, E)$ and $W \subseteq V$. For all $X, Y \subseteq W$, $X$ and $Y$ are considered boolean-width equivalent if and only if*

$$N(X) \cap \bar{W} = N(Y) \cap \bar{W}$$

*Notation: $X \equiv_{bw} Y$*

Figure 4 shows the cut in Figure 3 in more detail and shows that the definition of boolean-width indeed gives us an answer the question: "How different are the sets of vertices $X$ and $Y$ from each other?". It states that $X$ and $Y$ belong to the same boolean-width equivalence class and just like the twin classes these boolean-width classes get a representative that uses the lexicographically order.

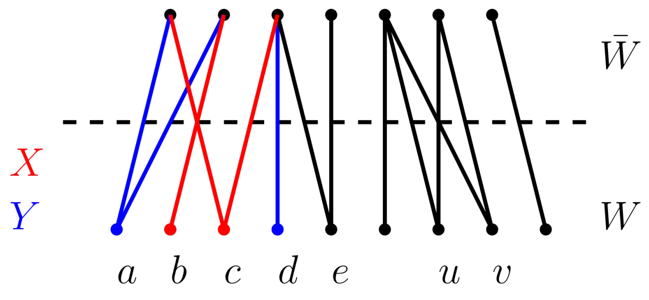

Figure 4: The cut in Figure 3.

**Definition 8.** *Let $G$ be the graph $(V, E)$ and $W \subseteq V$. We define a representative of a boolean-width class in $W$ as a set $I$ such that:*

1. *$X$ is in the boolean-width class represented by $I$ if and only if $X \equiv_{bw} I$*
2. *If $X \equiv_{bw} I$ then the following two statements are true:*
   (a) *If $I \equiv_{bw} X$, then $|I| \leq |X|$.*
   (b) *If $I \equiv_{bw} X$ and $|I| = |X|$, then $I$ is lexicographically smaller than $X$.*

For example, in Figure 4 the boolean-width class where $X$ and $Y$ belong to will be represented by the set $\{a, c\}$. From this point on when we speak about a boolean-width class $I$ we mean the boolean-width class represented by $I$.

Now we will formalize the definition of the boolean-width of a graph $G$. Observe in Figure 4 that every twin class on its own is also a boolean-width class. Furthermore, the union of the twin classes represented by $c$ and $d$ does not create a new boolean-width class. The set $\{c, d\}$ is boolean-width equivalent to $\{c\}$. The union of $c$ and $e$, however, does form a new boolean-width class represented by $\{c, e\}$. These examples show that the union of any number of twin classes can create a boolean-width class, but they might not all be different. This means that we get at most two to the power of number of twin classes in $W$ different boolean-width classes in $W$. For that reason the definition of the boolean-width of a cut $[W, \bar{W}]$ takes the $\log_2$ of the number of boolean-width classes.

**Definition 9.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. We define the boolean-width of a cut $[W, \bar{W}]$ induced by a node $w$ in $T$ as the $\log_2$ value of the number of boolean-width classes in the cut.*

**Definition 10.** *Let $G$ be the graph $(V,E)$ and $(T,\delta)$ a decomposition tree of $G$. We define the boolean-width of $(T,\delta)$, $bw(T,\delta)$, as the maximal boolean-width value taken over every cut $[W,\bar{W}]$ induced by a node $w$ in $(T,\delta)$.*

**Definition 11.** *Let $G$ be the graph $(V,E)$. We define the boolean-width of $G$, $bw(G)$, as the minimum boolean-width value taken over every decomposition tree $(T,\delta)$ of $G$.*

The representatives of boolean-width classes are very useful in a dynamic programming algorithm that calculates the maximum independent set, because these representatives have the following very nice property when it comes to independent sets.

**Definition 12.** *Let $G$ be the graph $(V,E)$. A subset $X \subseteq V(G)$ is an independent set if every vertex $v,w \in X$ are not adjacent to each other. The maximum independent set problem is defined as: Given a graph $G$ find the largest possible set $X$ such that $X$ is an independent set.*

**Lemma 1.** *Let $I$ and $\bar{I}$ be boolean-width classes in $W$ respectively $\bar{W}$ for a cut $[W,\bar{W}]$ in $G$. If $I \cup \bar{I}$ is an independent set on the cut, the bipartite subgraph of $G$ formed by the vertices on the border on both sides of the cut, then the union of every independent set $X \equiv_{bw} I$ with independent set $\bar{X} \equiv \bar{I}$ is an independent set in $G$.*

*Proof.* Say it is not true, thus there are independent sets $X \equiv_{bw} I$ and $\bar{X} \equiv \bar{I}$ such that $X \cup \bar{X}$ is not an independent set in $G$. Let $\bar{x} \in \bar{X}$ such that $X$ has a vertex who is adjacent to $\bar{x}$. $X \equiv_{bw} I$, thus $I$ is also adjacent to $\bar{x}$. Let $i \in I$ such that $\bar{x}$ is adjacent to $i$. $\bar{X} \equiv \bar{I}$, thus again $\bar{I}$ is als adjacent to $i$, which is a contradiction. □

From a dynamic programmer's point of view this means that the only independent sets that need to be stored as a partial solution for the subgraph $W$ are the smallest independent sets in each boolean-width class. In other words, there are at most the same number of boolean-width classes different independent sets on the cut $[W,\bar{W}]$ worth storing. That is why boolean-width can, and should be, called independent-set-width. For more details on the dynamic programming algorithm we refer the reader to Section 6 of Bui-Xuan et al. [6].

Further proof that boolean-width should be called independent-set-width is provided by an overview of the different run times of the algorithms that use different widths to solve the maximum independent set problem. Both theorems are taken from Chapter 5 of Vatshelle [23].

**Theorem 1** ([10] , [14], [22], [6])**.** *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 $O^*(2^k)$* | *module-width in $O^*(5^k)$* |
| *branch-width in $O^*(2.28^k)$* | *MM-width in $O^*(4^k)$* |
| *clique-width in $O^*(2^k)$* | *rank-width in $O^*(1.42^{k^2})$* |
| *boolean-width in $O^*(4^k)$* | *MIM-width in $O^*(n^{2k})$* |

Although the FPT algorithms that uses boolean-width can be slower than those of tree-width, branch-width, clique-width and rank-width, Vatshelle [23] chapter 4 proves that boolean-width has a lower value on any graph compared to the other widths. Tree-width, branch-width, clique-width are all linear in boolean-width and rank-width is polynomial in boolean-width.

Boolean-width can also be used to solve the minimum dominating set problem. This way boolean-width gives an upper-bound for the run time of the dynamic programming algorithm that uses the new width. We will look for an equivalence relation on a cut $[W,\bar{W}]$ such that we can store the partial solutions for $W$ side using less equivalence classes than boolean-width without losing any best solution. With this equivalence relation we will create an FPT DP algorithm that is faster than that of boolean-width.

**Theorem 2** ([18] , [5], [6] )**.** *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 $O^*(3^k)$* | *module-width in $O^*(8^k)$* |
| *branch-width in $O^*(3.69^k)$* | *MM-width in $O^*(8^k)$* |
| *clique-width in $O^*(4^k)$* | *rank-width in $O^*(1.69^{k^2})$* |
| *boolean-width in $O^*(8^k)$* | *MIM-width in $O^*(n^{3k})$* |

## 4 The Dominating-Set-Width

In this section we will find the definition of the dominating-set-width. Much like Section 3 we start by looking at the intuitive idea behind the dominating-set-width before we formalize everything. After that we introduce a new FPT DP algorithm.

### 4.1 The Intuition of Dominating-Set-Width

Our FPT DP algorithm should work by storing dominating sets as partial solutions for subgraphs of $G$. These subgraphs start out as a single vertex. We combine subgraphs into bigger subgraphs combining different partial solutions, until we end up with the whole graph. The order in which these subgraphs are combined play a vital role in how long it takes to run the DP algorithm which combines the different partial solutions. This order is based on the decomposition tree, just like the boolean-width.

In order to create this equivalence relation, we start by taking a look at ways vertices in $W$ can be dominated, illustrated by Figure 5.

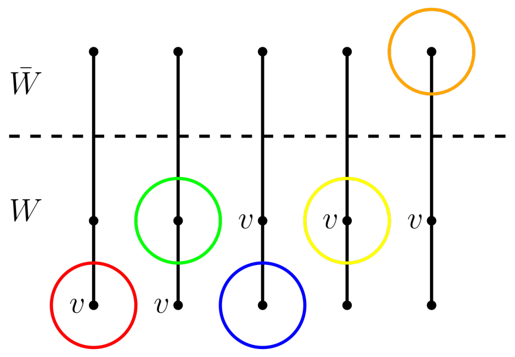

Figure 5: Different ways a vertex $v$ can be dominated.

For any vertex in $W$ we say it is on the border of the cut $[W, \bar{W}]$ if it is adjacent to a vertex in $\bar{W}$. A vertex in $W$ can either be on the border of the cut $[W, \bar{W}]$ or not. A vertex that is not on the border of $W$ can only be dominated by being inside the dominating set $D$ (red) or by a vertex in $W$ (green) or both. Vertices on the border of $W$ can be dominated by a vertex in $W$ (blue) or be inside the dominating set $D$ (yellow) or dominated by a vertex in $\bar{W}$ (orange) or a combination of the three.

Because of these different ways of dominating vertices in $W$, we do not just look at subsets of $W$ but also subsets of $\bar{W}$. Thus, a partial solution is a combination of sets, one set $X$ in $W$ and one set $\bar{X}$ in $\bar{W}$, which we call a pair. The equivalence relation will be on these pairs of sets.

Now we want to answer the question "When do we consider two pairs of sets, and thus two possible partial solutions, $(X, \bar{X})$ and $(Y, \bar{Y})$, equivalent?".

$Y$ in $\bar{W}$ dominates a part of $W$, this part can overlap with $X$ in $W$. When this happens $Y$ provides useless domination. For example:

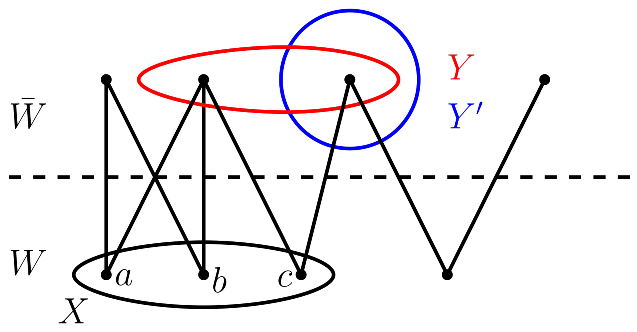

Figure 6: The sets $Y$ and $Y'$ help $X$ dominate $W$ in the same way.

In Figure 6 we see that $Y$ dominates all the vertices in $X$, which is unnecessary, and one vertex outside of $X$. $Y'$ on the other hand dominates only one vertex in $X$ and dominates the same vertex outside of $X$ that $Y$ dominates. From the point of view of $X$ there is no distinction between $Y$ and $Y'$ when it comes to the domination of $W$. Then why store both pairs as a partial solution? The answer is that we do not. When there is a pair $(X, Y')$ such that there is another pair $(X, Y)$ such that from the point of view of $X$ there is no distinction between $Y$ and $Y'$ we say $(X, Y')$ has become a redundant pair and we only store $(X, Y)$ as a partial solution. We will use this idea to discard as many partial solutions as possible.

Because we are interested in the different ways sets in $\bar{W}$ dominate across the cut we can restrain ourselves to the boolean-width classes of $\bar{W}$ that where introduced in Section 3. When we combine subgraphs we also need to know what sets in $W$ dominate in $\bar{W}$. Therefore we do the same with the sets in $W$ as with the sets in $\bar{W}$ and we use the boolean-width classes of $W$.

If we look back at Figure 6, we can see that there are sets boolean-width equivalent to $X = \{a, b, c\}$, namely $\{a, c\}$ and $\{b, c\}$. Every set boolean-width equivalent to $X$ contains $c$. This means that every set boolean-width equivalent to $X$ does not need any help with dominating $c$ from $\bar{W}$. This does not hold for $a$ and $b$. Sets boolean-width equivalent to $X$ can contain either $a$ or $b$ or both. This means that dominating $a$ and $b$ can be useful. However, even though $Y$ and $Y'$ do the same for $X$, they do not do the same for every set in the boolean-width class of $X$. Therefore we do not let $(X, Y)$ be equivalent to $(X, Y')$.

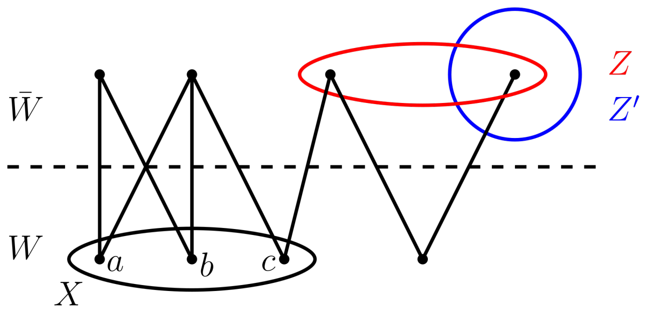

Figure 7: The sets $Z$ and $Z'$ help $X$ dominate $W$ in a different way.

In Figure 7, $Z$ and $Z'$ are helpful in the exact same way for every set in the boolean-width class of $X$. Both $Z$ and $Z'$ dominate the fourth vertex in $W$. $Z$ dominates $c$ which does not help any set boolean-width equivalent to $X$, while $Z'$ does not. Hence from the sets in the boolean-width class of $X$ points of view, $Z$ and $Z'$ help with dominating $W$ in the exact same way.

Therefore, we consider a width measurement in which $(X, Z)$ and $(X, Z')$ are considered equivalent whereas $(X, Y)$ and $(X, Y')$ are not.

## 4.2 The Definition of Dominating-Set-Width

With the intuition demonstrated above, we will now formalize the intuition into a definition for the dominating-set-width as well as a definition for the dominating-set-width classes.

**Definition 13.** *Let $G$ be the graph $(V, E)$, $(T, \delta)$ a decomposition tree of $G$ and $w$ is a node in $T$. Let $W \subseteq V$ be the set of vertices corresponding to the leaves in the subtree rooted at $w$ in $T$. For every boolean-width class in $W$ represented by $I$, let the center $C(I)$ of $I$ be the set vertices such that for every set $X$ boolean-width equivalent to $I$ it contains every vertex in the center.*

$$C(I) = \{\ x \mid \forall X \equiv_{bw} I \ :\ x \in X\ \} \tag{1}$$

In Figure 7 we can see that the center of the boolean-width class $\{a, c\}$ is $\{c\}$. Furthermore, we can see that the boolean-width class $\{a, c\}$ and $\{c\}$ have the same center.

**Definition 14.** *Let $G$ be the graph $(V, E)$, $(T, \delta)$ a tree decomposition of $G$ and $w$ is a node in $T$. Let $W \subseteq V$ be the set of vertices corresponding to the leaves in the subtree rooted at $w$ in $T$ and $\bar{W} = V(G) - W$. Let $\mathcal{D}_w$ be the set containing pairs of sets, one a subset of $W$ and one a subset of $\bar{W}$:*

$$\mathcal{D}_w = \{(X, \bar{X}) \mid X \subseteq W,\ \bar{X} \subseteq \bar{W}\ \}$$

*$(X, \bar{X})$ and $(Y, \bar{Y})$ are considered dominating-set-width equivalent if and only if points 1) and 2) are both true.*

1. $X \equiv_{bw} Y$
   *With $I$ the boolean-width class where $X$ and $Y$ belong to.*
2. $\big(N(\bar{X}) \cap W\big) - C(I) = \big(N(\bar{Y}) \cap W\big) - C(I)$

*Notation: $(X, \bar{X}) \equiv (Y, \bar{Y})$*

**Definition 15.** *Given a graph $G$ and $W \subseteq V(G)$ we define a representative of a dominating-set-width class in $W$ as a pair $(I, O)$ such that:*

1. *$(X, \bar{X})$ is in the dominating-set-width class represented by $(I, O)$ if and only if $(X, \bar{X}) \equiv (I, O)$*
2. *If $(X, \bar{X}) \equiv (I, O)$, then the following two points are true:*
   *(a) $|I| \leq |X|$ and if $|I| = |X|$, then $I$ is lexicographically smaller than $X$.*
   *(b) $|O| \leq |\bar{X}|$ and if $|O| = |\bar{X}|$, then $O$ is lexicographically smaller than $\bar{X}$.*

   *The lexicographical order is equal to the one used in boolean-width class representatives.*

From this point on, just as the boolean-width representatives, when we speak of a dominating-set-width class $(I, O)$ we mean the dominating-set-width class represented by $(I, O)$.

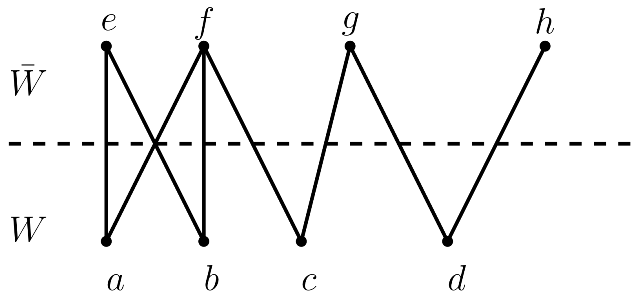

Figure 8: The cut from Figure 4, now with every vertex labeled.

The cut in Figure 8 has the following dominating-set-width classes:

For the boolean-width class $\emptyset$:
$(\emptyset, \bar{\emptyset})$ $(\emptyset, \{e\})$ $(\emptyset, \{f\})$ $(\emptyset, \{g\})$ $(\emptyset, \{h\})$ $(\emptyset, \{e, g\})$
For the boolean-width class $\{a\}$:
$(\{a\}, \bar{\emptyset})$ $(\{a\}, \{e\})$ $(\{a\}, \{f\})$ $(\{a\}, \{g\})$ $(\{a\}\{h\})$ $(\{a\}, \{e, g\})$
For the boolean-width class $\{c\}$:
$(\{c\}, \bar{\emptyset})$ $(\{c\}, \{e\})$ $(\{c\}, \{g\})$ $(\{c\}, \{e, g\})$
For the boolean-width class $\{a, c\}$:
$(\{a, c\}, \bar{\emptyset})$ $(\{a, c\}, \{e\})$ $(\{a, c\}, \{g\})$ $(\{a, c\}, \{e, g\})$
For the boolean-width class $\{d\}$:
$(\{d\}, \bar{\emptyset})$ $(\{d\}, \{e\})$ $(\{d\}, \{f\})$ $(\{d\}, \{g\})$ $(\{d\}, \{e, g\})$
For the boolean-width class $\{a, d\}$:
$(\{a, d\}, \bar{\emptyset})$ $(\{a, d\}, \{e\})$ $(\{a, d\}, \{f\})$ $(\{a, d\}, \{g\})$ $(\{a, d\}, \{e, g\})$

As we can see, we get that $(X, Z)$ and $(X, Z')$ from in Figure 7 are equivalent but $(X, Y)$ and $(X, Y')$ back in Figure 6 are not. We can also see that when two boolean-width classes $I$ and $I'$ have the same center they have the same combinations with respect to the boolean-width classes $O$.
Now we finalize the formalization of the dominating-set-width for a graph $G$.

**Definition 16.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. We define the dominating-set-width of a cut $[W, \bar{W}]$ induced by a node $w$ in $T$ as the* $\log_2$ *value of the number of dominating-set-width classes in the cut.*

**Definition 17.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. We define the dominating-set-width of $(T, \delta)$, $dsw(T, \delta)$, as the maximal dominating-set-width value taken over every cut $[W, \bar{W}]$ induced by a node $w$ in $(T, \delta)$.*

**Definition 18.** *Let $G$ be the graph $(V, E)$. We define the dominating-set-width of $G$, $dsw(G)$, as the minimum dominating-set-width value taken over every decomposition tree $(T, \delta)$ of $G$.*

Observe that at most the number of dominating-set-width classes equals the square of the number of boolean-width classes, thus $dsw(G) \leq 2 \cdot bw(G)$. Also observe that there are at least the same number of dominating-set-width classes as there are boolean-width classes, thus $bw(G) \leq dsw(G)$.

## 4.3 The New Algorithm

With the definition of the dominating-set-width we create a new DP algorithm that uses a combination step that looks like the combination step used by Bui-Xuan et al. [6]. For the rest of this section, $w$ is the parent node of $a$ and $b$ in the decomposition tree $(T, \delta)$ and $A$ and $B$ are subsets of $V(G)$ which are constructed in the same way as $W$ was in Definition 14 but now using $a$ and $b$ respectively. $I$ is a boolean-width class in $A$, $J$ in $B$ and $K$ in $W$. Furthermore, $(I, O)$ is a dominating-set-width class in $A$, $(J, Q)$ in $B$ and $(K, P)$ in $W$.

Let $Tab_w$ be a table where for each dominating-set-width class $(I, O)$ we store the size of a set $X$ such that $(X, O) \equiv (I, O)$ and $(X, O)$ dominates $W$. $(X, O)$ dominates $W$ means that $W \subseteq N[X] \cup N(O)$ with $N[X] = X \cup N(X)$. Let $Tab_a$ and $Tab_b$ be defined the same way with respect to $A$ and $B$. Our DP algorithm will initialize the tables $Tab_l$ for each leaf node $l$ in $T$ that minimizes the size stored at each index of $Tab_l$. Then it will loop over all the nodes $w$ of $T$ going from the bottom to the top and uses a new combination step to fill the tables $Tab_w$. Finally, it will fill $Tab_r$ at the root node $r$ of $T$ which has a single entry, $(\emptyset, \bar{\emptyset})$. This represents the size of a set $X$ such that $X$ dominates the whole graph $G$. The new combination step algorithm will minimize the set size stored at each index of $Tab_w$, thus it will minimize the set size stored at $Tab_r((\emptyset, \bar{\emptyset}))$ solving the minimum dominating set problem.

**Algorithm 1** Find the minimum dominating set for a graph $G$ given the decomposition tree $(T, \delta)$

```
1: for all leaf node $l$ in $T$ do
2:     $Tab_l(\emptyset, \bar{\emptyset}) = \infty$ (default value)
3:     $Tab_l(v, \bar{\emptyset}) = 1$
4:     $Tab_l(\emptyset, O) = 0$ with $O \neq \bar{\emptyset}$
5: for all $w$ of $T$, not a leaf, from bottom to top do
6:     Algorithm 2: Combine step at node $w$ with child notes $a$, $b$
```

**Algorithm 2** Combine step at node $w$ with child notes $a$, $b$

```
1: for all $(K, P)$ in $W$ do
2:     $Tab_w(K, P) = \infty$ (default value)
3: for all $I$ in $A$ and $J$ in $B$ do
4:     Find $K$ such that $K \equiv_{bw} I \cup J$
5:     for all $P$ such that $(K, P)$ is a dominating-set-width class representative in $W$ do
6:         Find $(I, O)$ such that $(I, J \cup P) \equiv (I, O)$
7:         Find $(J, Q)$ such that $(J, I \cup P) \equiv (J, Q)$
8:         $Tab_w(K, P) = \min \{Tab_w(K, P),\ Tab_a(I, O) + Tab_b(J, Q)\}$
```

To prove the correctness of Algorithm 1 we first prove two lemmas that will make the proof easier.

**Lemma 2.** *Let $K \equiv_{bw} I \cup J$ with $I$ a boolean-width class in $A$ and $J$ a boolean-width class in $B$. Then for every dominating-set-width class $(K, P)$ there are two dominating-set-width classes represented by $(I, O)$ and $(J, Q)$ such that for all $X \equiv_{bw} I$ and $Y \equiv_{bw} J$:*

1. $(X, Y \cup P) \equiv (I, O)$
2. $(Y, X \cup P) \equiv (J, Q)$

*Proof.* This proof is done by proving the following claim. The dominating-set-width classes $(I, O)$ and $(J, Q)$ such that $(I, J \cup P) \equiv (I, O)$ and $(J, I \cup P) \equiv (J, Q)$ are the ones we need. Note that these two classes are independent from $X$ and $Y$.
For $(X, Y \cup P)$ to be in equivalent to $(I, O)$ we need to prove two things:

1. $X \equiv_{bw} I$
2. $\big(N(Y \cup P) \cap A\big) - C(I) = \big(N(O) \cap A\big) - C(I)$.

The first point is true by assumption. The second point can be proven by first showing that $N(Y \cup P) \cap A = N(J \cup P) \cap A$. By

$$\begin{aligned} N(Y \cup P) \cap A &= \big(N(Y) \cup N(P)\big) \cap A, \\ &= \big(N(Y) \cap A\big) \cup \big(N(P) \cap A\big), \\ &= \big(N(J) \cap A\big) \cup \big(N(P) \cap A\big) \ \big(\text{using that } Y \equiv_{bw} J\big), \\ &= \big(N(J) \cup N(P)\big) \cap A, \\ &= N(J \cup P) \cap A. \end{aligned} \tag{2}$$

From this follows $\big(N(Y \cup P) \cap A\big) - C(I) = \big(N(J \cup P) \cap A\big) - C(I)$. By construction of $(I, O)$ we know $\big(N(J \cup P) \cap A\big) - C(I) = \big(N(O) \cap A\big) - C(I)$, proving the second point.
We can do the exact same thing to prove $(Y, X \cup P) \equiv (J, Q)$. □

**Lemma 3.** *If $(X, O) \equiv (I, O)$ and $(X, O)$ dominates $A$, then for every $\bar{X} \subseteq \bar{A}$ such that $(X, \bar{X}) \equiv (I, O)$ the pair $(X, \bar{X})$ will also dominate $A$.*

*Proof.* If $(X, O) \equiv (I, O)$ and $(X, O)$ dominates $A$ then $A \subseteq \big(N(O) \cap A\big) \cup N[X]$ with $N[X] = X \cup N(X)$. $X \equiv_{bw} I$ which implies $C(I) \subseteq N[X]$, thus we also know $A \subseteq \big(\big(N(O) \cap A\big) - C(I)\big) \cup N[X]$. Let $\bar{X}$ be an arbitrary set such that $(X, \bar{X}) \equiv (I, O)$. Then $\big(N(\bar{X}) \cap A\big) - C(I) = \big(N(O) \cap A\big) - C(I)$, thus $A \subseteq \big(\big(N(\bar{X}) \cap A\big) - C(I)\big) \cup N[X]$ which implies $A \subseteq \big(N(\bar{X}) \cap A\big) \cup N[X]$ proving that $(X, \bar{X})$ will also dominate $A$. □

And finally we use one lemma from Bui-Xuan et al. [6] to prove the correctness of Algorithm 1.

**Lemma 4** (Lemma 12 of [6])**.** *For a graph $G$, let $A$, $B$, $\bar{W}$ be a 3-partitioning of $V(G)$, and let $X \subseteq A$, $Y \subseteq B$ and $P \subseteq \bar{W}$. The pair $(X, Y \cup P)$ dominates $A$ and $(Y, X \cup P)$ dominates $B$ iff the pair $(X \cup Y, P)$ dominates $A \cup B$.*

**Lemma 5.** *Algorithm 1 solves the minimum dominating set problem.*

*Proof.* To prove the correctness of Algorithm 1 we need to show that for every dominating-set width class represented by $(K, P)$ the size of the best partial solution equivalent to $(K, P)$ is stored at $Tab_w(K, P)$. This will be done by induction on the combination step done in Algorithm 2 . The induction hypothesis is:

$$\forall (K, P)\ Tab_w(K, P) := \min\{\ |X| \ \big|\ (X, P) \equiv (K, P) \textit{ and } (X, P) \textit{ dominates } W\}$$

Starting at the leave nodes of $(T, \delta)$, $L = \{v\}$. This is the initialization step. Leaves will always have three classes:

$$1.\ (\emptyset, \bar{\emptyset}) \quad 2.\ (v, \bar{\emptyset}) \quad 3. (\emptyset, O)$$

$O$ is a boolean-width class such that $O$ is connected to $v$. Note that $(v, O) \equiv (v, \bar{\emptyset})$. The values of table $Tab_l$ are set during the initialization step in Algorithm 1. $Tab_l(\emptyset, \bar{\emptyset})$ equals the default value, $\infty$, because $v$ is not dominated. $Tab_l(v, \bar{\emptyset}) = 1$ because the set $\{v\}$ is the minimum dominating set of $L$. $Tab_l(\emptyset, O) = 0$, because the empty set is the minimum dominating set of $L$ given that $v$ is dominated by $O$. All these partial solutions are the optimal partial solutions for each class. Hence the base case is true.

Assume that the tables of the child nodes $a$ and $b$ of $w$ have been filled correctly with respect to the induction hypothesis. Say there is a set $X$ such that $(X, \bar{X})$ dominates $W$ and this pair belongs to the dominating-set-width class represented by $(K, P)$.
Define:

- $X_a = X \cap A \equiv_{bw} I$
- $X_b = X \cap B \equiv_{bw} J$

With Lemma 2 we get the two dominating-set-width classes $(I, O)$ in $A$ and $(J, Q)$ in $B$ such that $(X_a,\ X_b \cup P) \equiv (I, O)$ and $(X_b, X_a \cup P) \equiv (J, Q)$. At some point Algorithm 2 will check the value of $Tab_a(I, O) + Tab_b(J, Q)$, because the construction of $(I, O)$ and $(J, Q)$ in the algorithm is the same as in Lemma 2. Tables $Tab_a$ and $Tab_b$ have been filled correctly, meaning $Tab_a(I, O) \leq |X_a|$ and $Tab_b(J, Q) \leq |X_b|$, thus $Tab_w(K, P) \leq |X|$.

To finish the proof we need to prove that when $Tab_w(K, P) = k$ there is a set $Y$ such that $|Y| = k$, $(Y, P) \equiv (K, P)$ and $(Y, P)$ dominates $W$. Let $(I, O)$ and $(J, Q)$ be the dominating-set-width classes such that $Tab_w(K, P) = Tab_a(I, O) + Tab_b(J, Q)$. Let $Y_a$ be a set such that $|Y_a| = Tab_a(I, O)$, $(Y_a, O) \equiv (I, O)$ and $(Y_a, O)$ dominates $A$. This set $Y_a$ exists, because $Tab_a$ was filled in correctly. Let $Y_b$ be defined the same way with respect to $(J, Q)$. $Y_a \cup Y_b$ is the set $Y$ we are looking for.

- $Tab_w(K, P) = Tab_a(I, O) + Tab_b(J, Q)$, thus $|Y_a \cup Y_b| = k$.
- $Y_a \equiv_{bw} I$, $Y_b \equiv_{bw} J$ and $K \equiv_{bw} I \cup J$, thus $(Y_a \cup Y_b, P) \equiv (K, P)$.
- $(Y_a \cup Y_b, P) \equiv (K, P)$, $Y_a \equiv_{bw} I$ and $Y_b \equiv_{bw} J$, thus $(Y_a, Y_b \cup P) \equiv (I, O)$, because of Lemma 2. $(Y_a, O)$ dominates $A$, thus $(Y_a, Y_b \cup P)$ dominates $A$, because of Lemma 3. In the same way, $(Y_b, Y_a \cup P) \equiv (J, Q)$ and $(Y_b, Y_a \cup P)$ dominates $B$. Thus $(Y_a \cup Y_b, P)$ dominates $A \cup B = W$, because of Lemma 4.

The algorithm ends at the root node $r$ of $T$. We just proved that $Tab_r$ will be filled in correctly, thus $Tab_r(\emptyset, \bar{\emptyset})$ is the size of the smallest set $X$ such that $X$ dominates the whole graph $G$. $\square$

We loop over each dominating-set-width class at least once, because each boolean-width class $K$ is combined at least once. Say that each boolean-width class $K$, such that there are less than $2^{bw(G)}$ number of dominating-set-width classes $(K, P)$, is only loped over once. Then the total number of iterations done in Algorithm 1 will be at most $2^{dsw(G)} + 2^{bw(G)}(2^{2 \cdot bw(G)} - 2^{bw(G)})$. Rewriting this upper-bound gives $2^{3 \cdot bw(T,\delta)} + 2^{bw(G)} - 2^{2 \cdot bw(G)}$ and because $2^{bw(G)} \leq 2^{2 \cdot bw(G)}$ we can conclude that the number of iterations done in Algorithm 1 is less than or equal to $2^{3 \cdot bw(T,\delta)}$. Thus if the dominating-set-width of the decomposition tree we use in Algorithm 1, is less than two times its boolean-width we will always have less than $2^{3 \cdot bw(T,\delta)}$ iterations. The rest of the steps can be done in $O(p(n))$ with $p$ a polynomial. We will look at these steps in more detail in section 5. Although this does not prove that Algorithm 1 has an asymptotically smaller run time than the algorithm that uses boolean-width, which was $O^*(2^{3 \cdot bw(T,\delta)})$, it does prove that our algorithm is faster than the algorithm that uses boolean-width, because we loop over less equivalence classes. In section section 6 we will see that there are decomposition trees which do have an asymptotically faster run time.

## 5 A Data Structure for Dominating-Set-Width

In this section we will construct a new data structure which is used in Algorithm 2. The new data structure allows quick access to the dominating-set-width class representative corresponding to a pair $(X, \bar{X})$. We start by going over the data structure used with the boolean-width. After that we will introduce algorithms that construct the final data structure.

### 5.1 The Data Structure of Boolean-Width

Just as how the definition of the boolean-width was a stepping stone for the definition of the dominating-set-width, so shall the data structure of the boolean-width be a stepping stone for the data structure for the dominating-set-width.

In Section 3 it was shown that twin class representatives are the building blocks for the boolean-width class representatives, so we need to know who they are for each cut of a decomposition tree. Once we know what the representative for each twin class is, we can move on to constructing the boolean-width class representatives for each cut of a decomposition tree. Finally, we can construct a matrix that will grand us quick access to the boolean-width class representative corresponding to a set $X$. First we need to formalize the different lists we just described.

**Definition 19.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. We define the list of twin class representatives $TC_w$ as the unique family $LR_w \subseteq W$ satisfying:*

$$\forall v \subseteq W \ \exists x \in LR_w \text{ such that } v \text{ is in the twin class represented by } x$$

**Definition 20.** *For a decomposition tree $(T, \delta)$ define $ntc(T, \delta)$ as the maximal number of twin classes $|TC_w|$ or $|TC_{\bar{w}}|$ taken over all the cuts induced by nodes in $(T, \delta)$.*

**Definition 21** (Definition 8 of [6]). *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. We define the list of boolean-width class representatives $LR_w$ as the unique family $LR_w \subseteq 2^{TC_w}$ satisfying:*

$$\forall X \subseteq W \ \exists I \in LR_w \text{ such that } I \equiv_{bw} X$$

*Furthermore, let $LNR_w$ be a list represented as a self-balancing binary search tree containing the neighborhoods of members of $LR_w$ in $[TC_w, TC_{\bar{w}}]$.*

$$LNR_w = \{N(I) \cap TC_{\bar{w}} \mid I \in LR_w\} \tag{3}$$

Bui-Xuan et al. [6] provide algorithms and lemmas which allow us to actually construct these different lists and search trees. From this point on, let $n$ be the number of vertices of a graph $G$.

**Lemma 6** (Lemma 10 of [6]). *Let $G$ be the $n$-vertex graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. For every cut $[W, \bar{W}]$ induced by a node $w$ in $(T, \delta)$ we compute the two vertex sets $TC_w$ and $TC_{\bar{w}}$ associated to the cut $[W, \bar{W}]$ in $O(n(n + ntc^2(T, \delta)))$ time. In the same time we compute for every $v \in W$ a pointer to $x \in TC_w$ for $v$ and $x$ being in the same twin class of $W$, and similarly for every $v \in \bar{W}$.*

**Lemma 7** (Lemma 7 of [6]). *Let $G$ be the $n$-vertex graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. Assume the pre-processing described in Lemma 6 has been done. Then, for every cut $[W, \bar{W}]$ induces by a node $w$ in $(T, \delta)$, we compute the list of representatives $LR_w$, the self-balancing binary search tree $LNR_w$ and pointers such that $I \in LR_w$ and $N \in LNR_w$ point to each other if and only if $N = N(I) \cap W$ in $O(n \cdot ntc^2(T, \delta) \cdot bw(T, \delta) \cdot 2^{bw(T,\delta)})$ time.*

**Lemma 8** (Lemma 8 of [6]). *Let $G$ be the $n$-vertex graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. Assume the pre-processing described in Lemmas 6 and 7 has been done. Then, for every cut $[W, \bar{W}]$ induced by a node $w$ in $(T, \delta)$, we compute a data structure $M_w$ in $O(n \cdot ntc^2(T, \delta) \cdot bw(T, \delta) \cdot 2^{bw(T,\delta)})$ time, allowing, for any $X \subseteq W$, to access in $O(|X|)$ time the entry $I$ of $LR_w$ such that $X \equiv_{bw} I$.*

**Lemma 9** (Lemma 6 of [6]). *Let $G$ be the graph $(V, E)$, $W \subseteq V(G)$ and $I \in LR_w$. We have $|I| \leq k$, with $k$ the boolean-width of the cut $[W, \bar{W}]$.*

**Corollary 1.** *From Lemma 5.1 we can directly conclude that for all $I \in LR_w$, $|C(I)| \leq k$ with $k$ the boolean-width of the cut $[W, \bar{W}]$, because by definition $C(I) \subseteq I$.*

For more details on the data structure used for boolean-width we refer the reader to Section 5 of Bui-Xuan et al. [6]. The most important part to take from this paper for this section is: when we construct the data structure mentioned in Lemma 8, the time it takes to find the correct boolean-width class representative given a set $X \subseteq W$ is $O(|X|)$. Using Lemma , this query time of Lemma 8 becomes $O(bw(T, \delta))$ if $X$ is a boolean-width class representative or a finite union of boolean-width class representatives.

## 5.2 Algorithms for the New Data Structure

With the basis of the data structure of the boolean-width established, we move on to the algorithms that will produce the data structure of the dominating-set-width. First we need to know the different centers of each boolean-width class on each cut of a decomposition tree. Once these centers are known we can combine boolean-width classes to create dominating-set-width classes. Finally, similarly to the boolean-width, we can construct a matrix that will grant us quick access to the dominating-set-width class representative corresponding to a pair $(X, \bar{X})$.

### 5.2.1 Constructing the Centers

We start by introducing a new definition about twins on a cut, which will help with calculating the centers of each boolean-width class.

**Definition 22.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. A twin class represented by $x \in TC_w$ is called a single twin if the corresponding twin class contain a single element.*

In Lemma 6, pointers are created for every $v \in W$ to the twin class representatives $x \in TC_w$ for every $w$ in $T$. At the moment a pointer is created we can check if a twin class contains more than one element. Say we create a pointer from $v$ to $x$. If $v \neq x$ then we know the twin class represented by $x$ contains more than one element. With this extra step we adjust Lemma 6 to also keep track of all the single twins in every cut.

**Lemma 10.** *Let $G$ be the $n$-vertex graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. For every cut $[W, \bar{W}]$ induced by a node $w$ in $(T, \delta)$ we compute the two vertex sets $TC_w$ and $TC_{\bar{w}}$ associated to the cut $[W, \bar{W}]$ in $O(n(n + ntc^2(T, \delta)))$ time. In the same time we compute for every $v \in W$ a pointer to $x \in TC_w$ for $v$ and $x$ being in the same twin class of $W$, and similarly for every $v \in \bar{W}$ and we know which elements in $TC_w$ and $TC_{\bar{w}}$ are single twins.*

These single twins are the building blocks of the centers of each boolean-width class.

**Lemma 11.** *Let $G$ be the graph $(V, E)$ and $W \subseteq V$. For every boolean-width class $I$ in $W$ the center $C(I)$ only contains single twins.*

*Proof.* Say it is not true, then $C(I)$ contains a vertex $v$ who is not a single twin. Let $X \equiv_{bw} I$ such that $v \in X$. We can replace $v$ with a twin $u \neq v$ to get a new set. $X' = (X - v) \cup u$. Observe that $X' \equiv_{bw}$, thus $v \notin C(I)$ which is a contradiction. □

Now we can construct a list that contains the center for each boolean width class for every cut $[W, \bar{W}]$ induced by a node $w$ in $(T, \delta)$.

**Lemma 12.** *Let $G$ be the $n$-vertex graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. Assume the pre-processing described in Lemmas 7, 8 and 10 has been done. For every cut $[W, \bar{W}]$ induced by a node $w$ in $(T, \delta)$ we compute a list $LC_w$ containing for each $I \in LR_w$ its center $C(I)$ by running Algorithm 3 in $O(n \cdot ntc^2(T, \delta) \cdot bw(T, \delta) \cdot 2^{bw(T, \delta)})$ time.*

*Proof.* The idea of the algorithm is that we build the centers incrementally by adding the vertices in $TC_w$ one by one. After the first $n$ vertices $v \in TC_w$, denoted by $TC_w^n$, have been added, all the boolean-width classes $I \in CR$ are only using the first $n$ twin classes. The centers are the intersection of sets in $I$ who only use the first $n$ twins denoted by $LC_w^n$.

$$LC_w^n(I) = \bigcap_{\substack{X \subseteq 2^{TC_w^n} \\ X \equiv_{bw} I}} X = \{ x \mid \forall X \subseteq 2^{TC_w^n}\ X \equiv_{bw} I \ : \ x \in X \} \tag{4}$$

Now we will prove the correctness by induction for a single cut. Let $TC_w^n$ be the first $n$ twin classes of $TC_w$ and let $LC_w^n(I)$ be defined as in equation 4. At the first iteration, $CR = \{\emptyset\}$, and we add the first twin class $t_1$ resulting in $CR = \{\emptyset,\ \{t_1\}\}$ with $C(\emptyset) = \emptyset$ and $C(t_1) = \{t_1\}$ if and only if $t_1$ is a single, otherwise $C(t_1) = \emptyset$. The base case is correct.

Assume everything is correct up to $t_n$. $CR$ consists of boolean-width classes who have a member that is a subset of $2^{TC_w^n}$. Now we add $t_{n+1}$. Let $X = \{t_i, \cdots, t_{n+1}\} \equiv_{bw} I'$ for some $i \leq n$ such that $X = \{t_i, \cdots, t_m\} \cup \{t_{n+1}\}$ with $m \leq n$ and $\{t_i, \cdots, t_m\} \equiv_{bw} I \in CR$. At some point we combine $I$ with $t_{n+1}$ to get $X$ and we calculate $LC_w^{n+1}(I') = LC_w^{n+1}(I') \cap \big(LC_w^n(I) \cup \{t_{n+1}\}\big)$ or $LC_w^{n+1}(I') = LC_w^{n+1}(I') \cap LC_w^n(I)$ depending on if $t_{n+1}$ is a single twin or not because of Lemma 11. This is done for every arbitrary set $X$ of the form $\{t_i, \cdots, t_m\} \cup \{t_{n+1}\}$, thus $LC_w^{n+1}$ is calculated correctly.

On to the run time analysis. In each iteration in the loop of Algorithm 3 calculating $X$ takes constant time. Calculating $I'$ takes $O(k)$ time, with $k$ the boolean-width value of the cut $[W, \bar{W}]$, because of Lemma 5.1. Checking if $I' \in CR$ takes $O(|TC_w|k)$ time by representing $CR$ as a self-balancing binary search tree, just like $LNR_w$. Then adding $I'$ if necessary takes the same amount of time, $O(|TC_w|k)$. We only need to calculate one intersection in each iteration which takes $O(|TC_w|)$ time, thus each iteration will take $O(|TC_w|k)$ time. In the worst case scenario $|CR| = 2^k$, thus at most there are $O(|TC_w|2^k)$ number of iterations. We loop over $O(n)$ cuts in $(T, \delta)$ giving a total run time of $O(n \cdot ntc^2(T, \delta) \cdot bw(T, \delta) \cdot 2^{bw(T,\delta)})$. □

**Algorithm 3** Calculating $C(I)$ for each $I \in LR_w$ in the list $LC_w$

```
1: for all $I \in LR_w$ do
2:   $LC_w(I) = I$
3: Let $CR = \{\emptyset\}$
4: for all $v \in TC_w$ do
5:   for all $I \in CR$ do
6:     Let $X = I \cup \{v\}$
7:     Let $I' \equiv_{bw} X$
8:     if $I' \notin CR$ then
9:       Add $I'$ to $CR$
10:    if $v$ is a single twin then
11:      $LC_w(I') = LC_w(I') \cap \big(LC_w(I) \cup \{v\}\big)$
12:    else
13:      $LC_w(I') = LC_w(I') \cap LC_w(I)$
```

### 5.2.2 Constructing the Dominating-set-width Classes

Now that $LC_w$ is constructed, we can continue with the algorithm for constructing the dominating-set-width class representatives.

**Definition 23.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. We define $LDR_w$ as the list of dominating-set-width class representatives as the unique family $LDR_w \subseteq 2^{TC_w \times TC_{\bar{w}}}$ satisfying:*

$$\forall (X, \bar{X}) \in \mathcal{D}_w \ \exists (I, O) \in LDR_w \text{ such that } (X, \bar{X}) \equiv (I, O)$$

**Definition 24.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. Let $[W, \bar{W}]$ be the cut induces by a node $w$ in $(T, \delta)$ and $LDR_w$ the list of dominating-set-width representatives. For every $I \in LR_w$ we define $LD(I)_w$ as a list represented as a self-balancing binary search tree of unique areas dominated by sets in $\bar{W}$ but with the center of $I$ removed:*

$$LD(I)_w \ = \ \{ \ D \ \big| \ \exists (I, O) \in LDR_w \ : \ D = N(O) \cap W - C(I) \ \}$$

**Lemma 13.** *Let $G$ be the $n$-vertex graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. Assume the pre-processing described in Lemmas 7, 8, 10 and 12 has been done. For every cut $[W, \bar{W}]$ induced by a node $w$ in $(T, \delta)$ we compute the lists $LDR_w$ and $LD(I)_w$ for every $I \in LR_w$ by running Algorithm 4 in $O(n \cdot ntc(T, \delta) \cdot bw(T, \delta) \cdot 2^{2 \cdot bw(T, \delta)})$ time.*

*Proof.* The algorithm creates every possible combination of every boolean-width class in $W$ with every boolean-width class in $\bar{W}$. Each combination is tested to see if this combination creates a new dominating-set-width class in line 4. $LR_w$ and $LR_{\bar{W}}$ are sorted according to the lexicographical order due to the way $LR_w$ is constructed which is just like the centers. We fix $I$ before looping over all possible $O$, thus the pair $(I, O)$ for which $D \notin LD(I)_w$, and thus is added to $LD(I)_w$, is a dominating-set-width class representative according to Definition 23.

On to the run time analysis. In each iteration in the loops of Algorithm 4 calculating $D$ takes $O(|TC_w|)$ time. Checking if $D \in LD(I)_w$ takes $O(|TC_w| k)$ time, with $k$ the boolean-width value of the cut $[W, \bar{W}]$, because $LD(I)_w$ is a self-balancing binary search tree. Adding $D$ if necessary takes the same amount of time, $O(|TC_w| k)$. The rest of the steps take $O(1)$ time, thus each iteration will take $O(|TC_w| k)$ time. We loop over $O(2^{2k})$ boolean-width classes in each cut and $O(n)$ cuts in $(T, \delta)$ giving a total run time of $O(n \cdot ntc(T, \delta) \cdot bw(T, \delta) \cdot 2^{2 \cdot bw(T, \delta)})$ □

**Algorithm 4** Calculating $LDR_w$ for a node $w$ in $T$

```
1: for all $I \in LR_w$ do
2:   for all $O \in LR_{\bar{w}}$ do
3:     $D \ = \ N(O) \cap W - C(I)$
4:     if $D \notin LD(I)_w$ then
5:       Add $D$ to $LD(I)_w$ at the correct place
6:       Add $(I, O)$ to $LDR_w$
7:       Add a pointers from $D$ to $(I, O)$
```

### 5.2.3 Constructing the Query Matrix

Finally, we can create the data structure that allows for quick access to the dominating-set-width class representative corresponding to a pair $(X, \bar{X})$.

**Lemma 14.** *Let $G$ be the $n$-vertex graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. Assume the pre-processing described in Lemmas 7, 8, 10, 12 and 13 has been done. For every cut $[W, \bar{W}]$ induced by a node $w$ in $(T, \delta)$ we compute a data structure $MD_w$ by running Algorithm 5 in $O(n \cdot ntc^2(T, \delta) \cdot bw(T, \delta) \cdot 2^{bw(T, \delta)})$ time, allowing, for any $(X, \bar{X}) \in \mathcal{D}_w$, to access in $O(\max\{|X|, \ |\bar{X}|\})$ time the entry $(I, O)$ of $LDR_w$ such that $(X, \bar{X}) \equiv (I, O)$ by running Algorithm 6.*

The intuition for the proof of the correctness of Algorithm 5 follows directly from the observation that there is a bijection between the list $LD(I)_w$ and the dominating-set-width classes $(I, O)$.

*Proof.* We will prove the correctness of both algorithms for a single cut. We build the dominating-set-width class representative of a pair $(X, \bar{X}) \in \mathcal{D}_w$ by finding the boolean-width class $I$ such that $I \equiv_{bw} X$, then much like the data structure created to quickly find the correct boolean-width class we add the vertices of $\bar{X}$ one by one. This way the rest of the proof is analogous to Lemma 8 of Bui-Xuan et al. [6] (Lemma 8 in this section). There is just one thing different.

The difference is that instead of exploiting the bijection between the elements of $LR_w$ and $LNR_w$ we exploit the bijection between the elements of $LDR_w$ and the lists $LD(I)_w$ for the different $I \in LR_w$. This bijection is created in line 7 of Algorithm 4. An example of a $DM_w$ matrix is given in Figure 9

On to the run time analysis. In each iteration in the loop of Algorithm 5 calculating $D$ takes $O(|TC_w|)$ time. This can achieved by finding the boolean-width class of $O \cup \{v\}$ in $O(k)$ time, with $k$ the boolean-width value of cut the $[W, \bar{W}]$, then finding $N(O \cup \{v\}) \cap W \in LNR_{\bar{w}}$ in $O(1)$ time and then subtracting $C(I)$ in $O(|TC_w|)$ time. Once $D$ is calculated $(I, O')$ can be found in $O(1)$ time, because of the bijection mentioned above. The rest of the steps also take $O(1)$ time. We loop over $O(2^k)$ boolean-width classes and $(|TC_w|)$ twin classes in each cut and $O(n)$ cuts in $(T, \delta)$ giving a total run time of $O(n \cdot ntc^2(T, \delta) \cdot 2^{bw(T,\delta)})$.

For the run time analysis of Algorithm 6 we find $I$ in $O(|X|)$ time. Each iteration in the loop of Algorithm 6 takes $O(1)$ time and there are $O(|\bar{X}|)$ iterations. □

**Algorithm 5** Calculating $MD_w$ for a node $w$ in $T$

```
1: for all $v \in TC_{\bar{w}}$ do
2:   for all $(I, O) \in LDR_w$ do
3:     Let $D = N(O \cup \{v\}) \cap W - C(I)$
4:     Let $(I, O')$ be the dominating-set-width class $D$ points to in $LD(I)_w$.
5:     Add a pointer from $MD[(I, O)][v]$ to $(I, O')$
```

**Algorithm 6** Find the dominating-set-width class representative $(I, O)$ for a pair $(X, \bar{X}) \in \mathcal{D}_w$

```
1: Let $I$ be the boolean-width class of $X$
2: Let $O = \emptyset$
3: for all $v \in \bar{X}$ do
4:   $(I, O) = MD[(I, O)][v]$
```

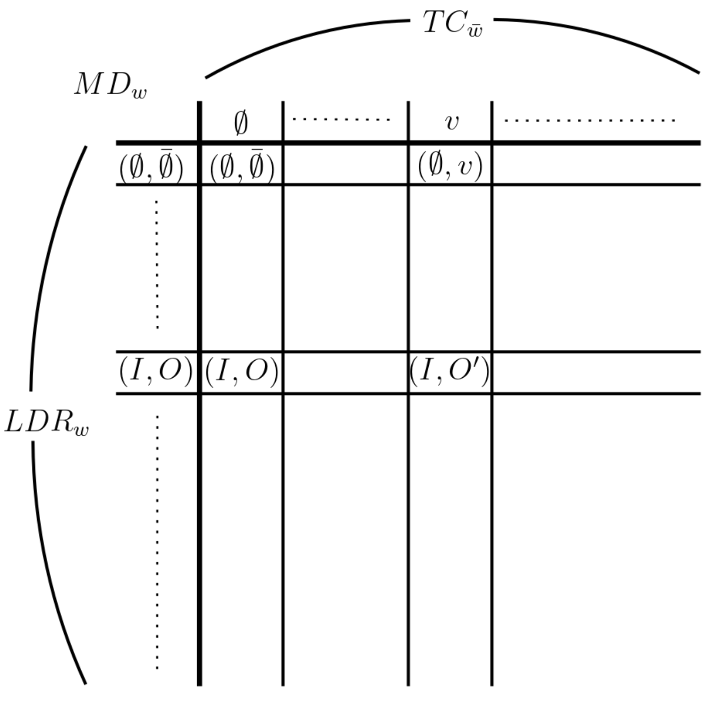

Figure 9: An example of a $MD_w$ matrix, here $(I, O') \equiv (I, O \cup v)$.

We can apply Lemma 5.1 again to show that the run time of line 1 of Algorithm 6 is $O(k)$, with $k$ the boolean-width value of cut $[W, \bar{W}]$, when $X$ is a boolean-width representative. Following the same reasoning, there are $O(k)$ iterations in the loop of Algorithm 6 when $\bar{X}$ is a boolean-width representative, thus we can find the representative of the pair $(I, O)$ when $I$ and $O$ are boolean-width representatives or finite union of boolean-width representatives in $O(k)$ time.

### 5.2.4 Summary

We created a data structure for the dominating-set-width that can be used by Algorithm 1. This was accomplished by creating the temporary lists, $TC_w$, $TC_{\bar{w}}$, $LR_{\bar{w}}$, $LNR_w$, $LNR_{\bar{w}}$, $LC_w$ and $LD(I)_w$ and the permanent lists $LR_w$ and $LDR_w$ as well as two matrices $M_w$ and $MD_w$. These matrices allow quick access to the correct representative in $LR_w$ and $LDR_w$ when given $X$ and $(X, \bar{X})$ respectively.

**Corollary 2.** *Let $G$ be the $n$-vertex graph and $(T, \delta)$ a decomposition tree of $G$. Assume the pre-processing described in Lemmas 7, 8, 10, 12, 13 and 14 has been done. We can solve the minimum dominating set problem with Algorithm 1 which has a run time of $O(n \cdot bw(T, \delta) \cdot 2^{3 \cdot bw(T,\delta)})$.*

**Theorem 3.** *Let $G$ be the $n$-vertex graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. By doing the pre-processing described in Lemmas 7, 8, 10, 12, 13 and 14 we can solve the minimum dominating set problem in $O\big(n(n + ntc(T, \delta) \cdot bw(T, \delta) \cdot 2^{2 \cdot bw(T,\delta)} + bw(T, \delta) \cdot 2^{3 \cdot bw(T,\delta)})\big)$ which can also be expressed as $O\big(n^2 + n \cdot bw(T, \delta) \cdot 2^{3 \cdot bw(T,\delta)}\big)$, because $ntc(T, \delta) \leq 2^{bw(T,\delta)}$.*

Up to this point the run time of every algorithm has been expressed by $n$, $ntc(T, \delta)$ and $bw(T, \delta)$. Obviously, we would like to change this to $n$, $ntc(T, \delta)$ and $dsw(T, \delta)$. This is not an easy task, because we have no relation between $bw(T, \delta)$ and $dsw(T, \delta)$ other than $dsw(T, \delta) \leq 2 \cdot bw(T, \delta)$ and an observation about the number of iterations done in Algorithm 1. Thus we can't give an expression for these run times using only $n$, $ntc(T, \delta)$ and $dsw(T, \delta)$ that gives the same run time or a better one at this point. However, in the next section we will proof that we can change the expression of the run times for certain graph classes.

## 6 Graph Class Analysis

In this section we will prove an upper and lower-bounds for the dominating-set-width. After that, we will look at the size of the dominating-set-width for graphs restricted to graph classes. We finish the section with some theory about the upper-bound of the dominating-set-width for general graphs.

### 6.1 The Bounds of Dominating-Set-Width

Belmonte and Vatshelle [3] proved for a number of different graph classes a upper-bounds for their boolean-width value, illustrated by Figure 10 which was taken from Belmonte and Vatshelle [3]. From the definition of the dominating-set-width we saw that the dominating-set-width is linear in boolean-width. Because of this relation all upper-bounds given in Belmonte and Vatshelle [3] are also upper-bounds for the dominating-set-width.

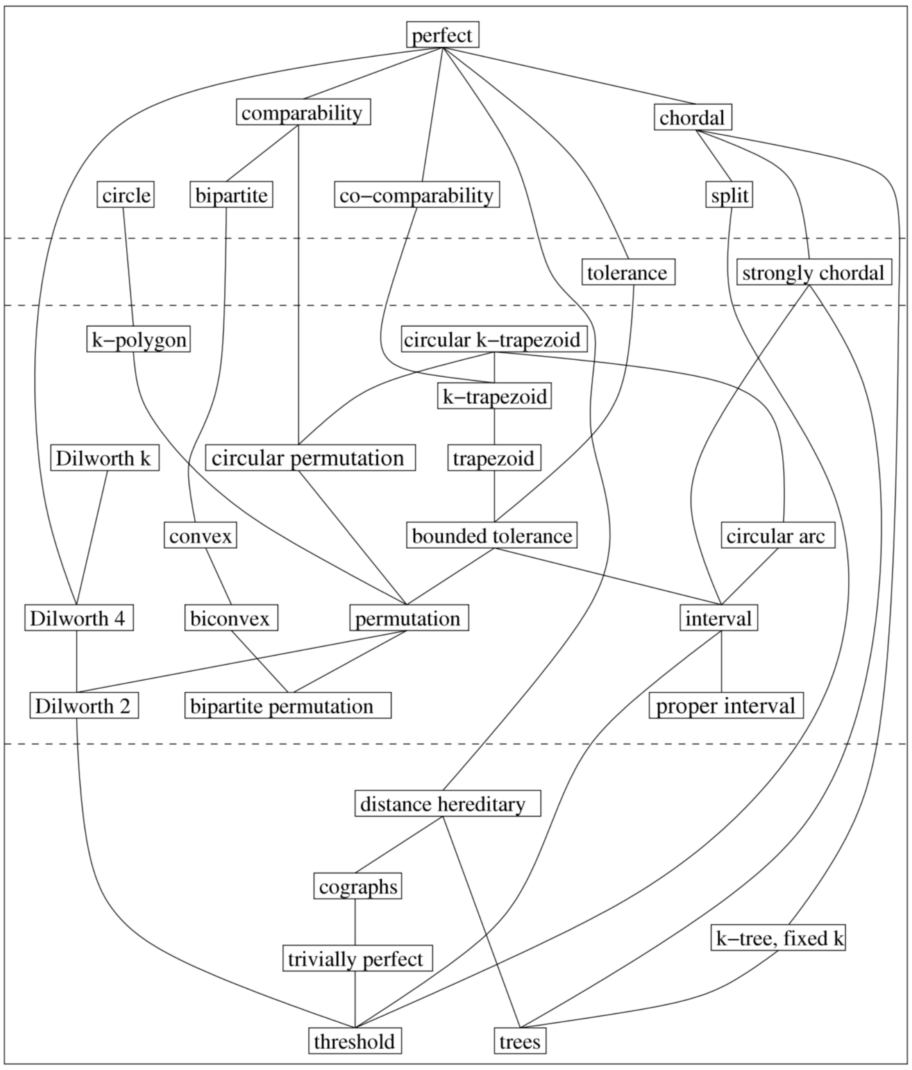

Figure 10: Inclusion diagram of some well-known graph classes. $(I)$ Graph classes where clique-width and boolean-width are bounded by a constant. $(II)$ Graph classes having decomposition trees with boolean-width $O(\log n)$. $(III)$ It is unknown whether these classes have boolean-width $O(\log n)$. $(IV)$ Either boolean-width is not $O(\log n)$ or it is NP-hard to compute such decompositions.

With the upper-bound done, we move on to a better lower-bound than the one described in Section 4. This lower-bound shall be proven to be a tight bound. But first we will prove a lemma that shows that all centers are boolean-width class representatives.

**Lemma 15.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. For all boolean-width classes $I \in LR_w$ in $W$, $C(I) \in LR_w$.*

*Proof.* Say it is not true. Let $S$ be the boolean-width class such that $S \equiv_{bw} C(I)$ and thus $S \neq C(I)$. Observe that $(I - C(I)) \cup S \equiv_{bw} I$, thus there is a set equivalent to $I$ such that it does not contain $C(I)$ which is a contradiction with the definition of $C(I)$. □

Now we can prove a lower bound of the dominating-set-width for general graphs, which will be a tight lower-bound.

**Lemma 16.** *For every graph $G$ we have that $3^{bw(G)} \leq 2^{dsw(G)}$.*

*Proof.* We know $C(I)$ is a boolean-width class representative. Let $I$ be a boolean-width class such that $C(I) = I$ and $|I| = 1$ on the cut $[W, \bar{W}]$. If we ignore this boolean-width class by removing $I$ from the border we still have at least $2^{k-1}$ boolean-width classes left in $W$. The same is true for $\bar{W}$, because both sides of a cut have the same amount of boolean-width classes. Therefore we get at least $2^{k-1}$ different neighborhoods in $W$ adjacent to sets in $\bar{W}$. This means that the least amount of different dominating-set-width classes created for $I$ is $2^{k-1}$. The same argument can be used to show that when $C(I) = I$ and $|I| = i$ we get at least $2^{k-i}$ dominating-set-width classes for $I$. The total amount of dominating-set-width classes created in the cut $[W, \bar{W}]$ is therefore bounded by:

$$\Sigma_{i=0}^{k} \ |\{ \ I \mid |C(I)| = i\}| \cdot 2^{k-i} \tag{5}$$

Observe that a bigger center results in a lower amount of dominating-set-width classes. We create the least amount of dominating-set-width classes when every center is unique and every possible size of a center is available. Say two boolean-width classes $I$ and $J$, with $N(I) \cap \bar{W} \subset N(J) \cap \bar{W}$, have the same center then the same amount dominating-set-width classes are created for $I$ and $J$. This is more or equal to the scenario when $J$ has a bigger center. The size of each center is bounded by the corresponding boolean-width class representative, which is bounded by the boolean-width of the decomposition tree $(T, \delta)$. This observation will lower bound given in Equation 5:

$$\Sigma_{i=0}^{k} \binom{k}{i} \cdot 2^{k-i} \tag{6}$$

Applying the Binomium of Newton shows that this equation is equal to $3^k$. □

We will now prove that this lower-bound is tight. This is done by showing that tree graphs and grid graphs both have a decomposition tree $(T, \delta)$ such that $3^{bw(T,\delta)} = 2^{dsw(T,\delta)}$.

### 6.1.1 Tree graphs

**Lemma 17.** *Tree graphs have a dominating-set-width value of* $\log_2(3)$.

*Proof.* The boolean-width value of a tree graph $G$ is one. Trees have a decomposition tree $(T, \delta)$ such that every cut induced by a node $w$ has two boolean-width classes, $\emptyset$ and $r$, where $r$ is the root of the subtree $W$. This is done by starting with an interior node $r$ which only has leaves as child vertices, then adding these leaf vertices one by one. After this is done, $r$ can been seen as a leaf vertex in $G$ by compressing all its child notes into itself. This process can be repeated until we have added every vertex. Figure 11 illustrates this process.

- For the boolean-width class $\emptyset$ the dominating-set-width classes are $(\emptyset, \bar{\emptyset})$ and $(\emptyset, \bar{r})$.
- For the boolean-width class $r$ the dominating-set-width class is $(r, \bar{\emptyset})$.

We end up with three dominating-set-width classes for each cut in the decomposition tree $(T, \delta)$ giving trees a dominating-set-width value of $\log_2(3)$. □

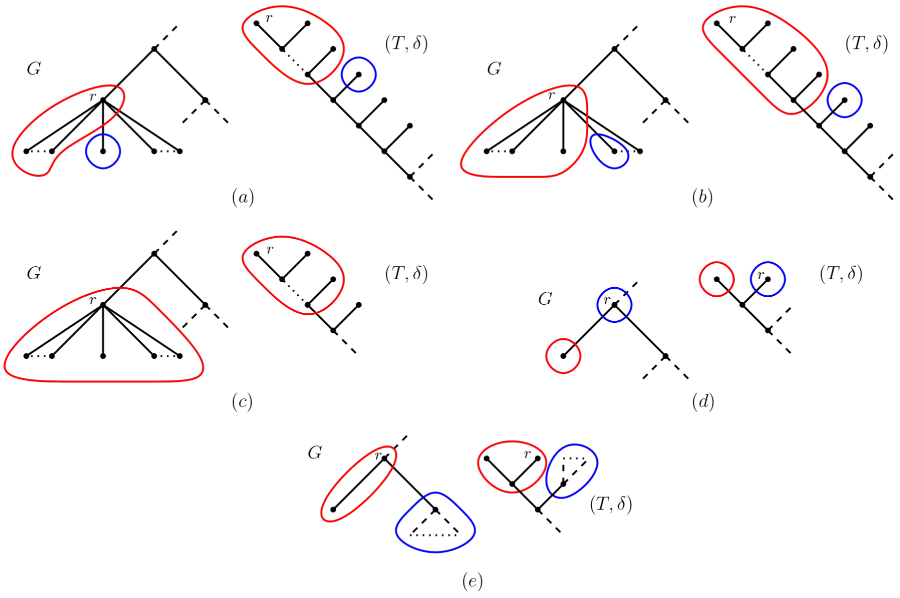

Figure 11: Set $A$ is always in red and set $B$ is always in blue. These steps show how we can combine each leaf node of a single internal node of a tree, steps $(a)$ to $(c)$. The set $A$ in step $(c)$ can be seen and handled as if it where a leaf node like in image $(d)$. We repeat these steps again one level higher in the tree as we see in step $(d)$ and $(e)$.

### 6.1.2 Grid graphs

**Lemma 18.** *All $n \times m$ grid graphs $G$ with $n < m$ have a decomposition tree $(T,\delta)$ such that $3^{bw(T,\delta)} = 2^{dsw(T,\delta)}$.*

*Proof.* Let $G$ be a $n \times m$ grid graph $(V,E)$ with $n < m$. We construct a decomposition tree by creating columns of size $n$ and add the columns one by one, going from one side to the other. This way we get at most $2^n$ boolean-width classes in each cut of $(T,\delta)$. For every cut $[W,\bar{W}]$ each boolean-width class $I$ in $W$, $C(I) = I$. Each vertex in $\bar{W}$ who is adjacent to $I$ is only adjacent to $I$ and nothing else in $W$. Therefore every boolean-width class $O$ in $\bar{W}$ which is adjacent to a part of $I$ can be replaced by a boolean-width class $Q$ which is not adjacent to any part of $I$, thus $(N(Q) \cap W) - I = N(Q) \cap W$. Figure 12 shows an example of such $O$ and $Q$.

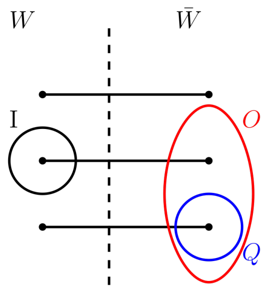

Figure 12

For each boolean-width class $I$ in $W$ there are $2^{n-|I|}$ different boolean-width classes $Q$ in $\bar{W}$ who are not adjacent to a part of $I$. Thus there are $2^{n-|I|}$ dominating-set-width classes created for this boolean-width class. The total amount of dominated-set-width classes created for each column can be calculated by summing over each possible size of $I$ and each possible combination of that size, $\binom{n}{|I|}$.

$$\Sigma^n_{|I|=0}\binom{n}{|I|}2^{n-|I|} = \Sigma^n_{|I|=0}\binom{n}{|I|}2^{n-|I|}\cdot 1^{|I|} = 3^n \text{ (applying the Binomium of Newton)} \quad (7)$$

Adding the columns one by one going from one side to the other results in the same amount of dominating-set-width classes, because we get the exact same amount of boolean-width classes on each side of the cut and they are constructed the same way. ☐

## 6.2 Second Run Time Analysis

At the end of Section 4 it was shown that Algorithm 1 has a run time of $O^*(2^{3 \cdot bw(T,\delta)})$. With the analysis done on tree and grid graphs, we can show that for these graphs we can get a run time of $O^*(2^{bw(T,\delta)}2^{dsw(T,\delta)})$.

**Lemma 19.** *Let $G$ be a $n$-vertex tree graph $(V,E)$ with decomposition tree $(T,\delta)$ described in Lemma 17, then Algorithm 1 has a run time of $O^*(2^{bw(T,\delta)}2^{dsw(T,\delta)})$.*

*Proof.* The proof is given by showing that there are at most $O(2^{bw(T,\delta)}2^{dsw(T,\delta)})$ numbers of iterations done in Algorithm 2. The rest of the run time analysis remains the same. Let $w$ be the parent node of $a$ and $b$ in the decomposition tree $(T,\delta)$ and $A$ and $B$ are subsets of $V(G)$ who are constructed in the same way as $W$ was in Definition 14 but now using $a$ respectively $b$. Both $A$ and $B$ have two boolean-width classes, $\emptyset_a$ and $r_a$ and $\emptyset_b$ and $r_b$ respectively. Without loss of generality assume that the root node of $A$ is a parent node for vertices in $B$, thus $r_a = r_w$, illustrated by Figure 13.

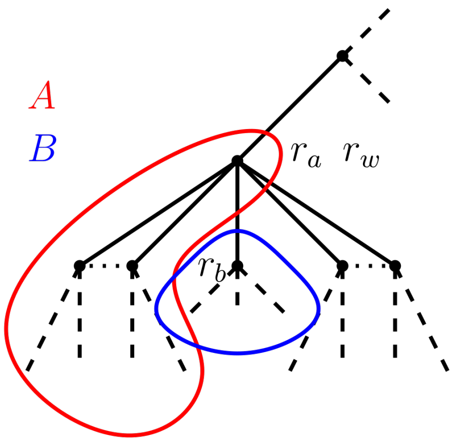

Figure 13: Sets represented by internal nodes of the decomposition tree $(T,\delta)$ described in Lemma 17.

$\emptyset_a$ and $\emptyset_b$ combined is boolean-width equivalent to $\emptyset_w$, just like $\emptyset_a$ combined with $r_b$. $r_a$ and $\emptyset_b$ combined is boolean-width equivalent to $r_w$ just as $r_a$ combined with $r_b$ is. This means that we loop two times over each the dominating-set-width class in $W$, thus at most $O(2^{bw(T,\delta)}2^{dsw(T,\delta)})$ numbers of iterations are done in Algorithm 2. □

**Lemma 20.** *Let $G$ be a $n \times m$ grid graph $(V,E)$ with $n < m$ with decomposition tree $(T,\delta)$ described in Lemma 20, then Algorithm 1 has a run time of $O^*(2^{bw(T,\delta)}2^{dsw(T,\delta)})$.*

*Proof.* The proof is given the same way as with the tree graphs. Observe that when a column is created every boolean-width class $K$ is the combination of two unique boolean-width classes of $A$ and $B$. Thus we loop over every dominating-set-width class in $W$ exactly once. When we combine two columns there are $2^{bw(T,\delta)}$ possible combinations to get every boolean-width class $K$. This means that we loop over every dominating-set-width class in $W$ a $2^{bw(T,\delta)}$ number of times, thus there are at most $O(2^{bw(T,\delta)}2^{dsw(T,\delta)})$ numbers of iterations done in Algorithm 2. □

Grid graphs have a linear boolean-width in $n$ according to Vatshelle [23]. Combined with Lemma 20 we get a nice result.

**Corollary 3.** *Let $G$ be a $n \times m$ grid graph $(V,E)$ with $n < m$ we can solve the minimum dominating set problem in $O^*(2^{bw(G)}2^{dsw(G)})$.*

## 6.3 The Dominating-Set-Width for General Graphs

There are many more graph classes for which we would like to answer the following question:

**Question:** For a given graph class $\mathcal{C}$ which of the following cases is true:

1. $\forall G \in \mathcal{C} \ : \ dsw(G) < 2 \cdot bw(G)$.
2. $\exists G \in \mathcal{C} \ : \ dsw(G) = 2 \cdot bw(G)$

Up to this point there are no more graph classes for which we can answer this question. We did however came up with some theory about the dominating-set-width for general graphs. We have proven a relation between the number of dominating-set-width classes on a cut and the presence of a single twin on the same cut.

**Lemma 21.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. Let $I \in LR_w$ be such that $C(I) \neq \emptyset$ and define $N_I := N(I) \cap TC_{\bar{w}}$. Then there is a vertex $y \in TC_{\bar{w}}$ such that $y$ is only adjacent to vertices in $C(I)$ and no other vertex in a set equivalent to $I$. In other words:*

$$\forall I \in LR_w \ \exists y \in TC_{\bar{w}} \ : \ N(y) \cap \bigcup_{X \equiv I} X \subseteq C(I) \tag{8}$$

*Proof.* Say it is not true. Then for every vertex in $N_I$ take an adjacent vertex that is in $W$ and who is in $\bigcup_{X \equiv I} X$ but not in $C(I)$. These vertices exists under the assumption that the lemma is not true. Call the set of all these vertices $X'$. $N(X') \cap TC_{\bar{w}} = N_I$, thus $X' \equiv_{bw} I$, but $X'$ does not contain any vertex in $C(I)$, which is a contradiction. □

**Lemma 22.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. Let $I \in LR_w$ such that $C(I) \neq \emptyset$, define $N_I := N(I) \cap TC_{\bar{w}}$ and let $y$ be a vertex in $N_I$ described in Lemma 21. Then $N(y) \cap W$ must contain a vertex $x$ such that $N(x) \cap \bar{W} \not\subseteq N_I$ or $(N(y) \cap W) - C(I) = \emptyset$.*

*Proof.* If $N(y) \cap W$ does not contain a vertex $x$ such that $N(x) \cap \bar{W} \not\subseteq N_I$ then every $x$ must be in $C(I)$, because of Lemma 21, thus $\big(N(y) \cap W\big) - C(I) = \emptyset$. If $x \notin C(I)$ and $N(x) \cap \bar{W} \subseteq N_I$ then $y$ is connected to a vertex in $\bigcup_{X \equiv I} X - C(I)$ which is a contradiction with the construction of $y$. □

To illustrate these two lemmas Figure 14 shows a cut with and $I$, $N_I$ and $C(I)$ which in this example is equal to $I$. $y_1$ is only adjacent to the center of $I$ and $y_2$ is adjacent to the center of $I$ but also to a vertex $x$ that is not in any set boolean-width equivalent to $I$. This $x$ is the vertex $x$ described in Lemma 22.

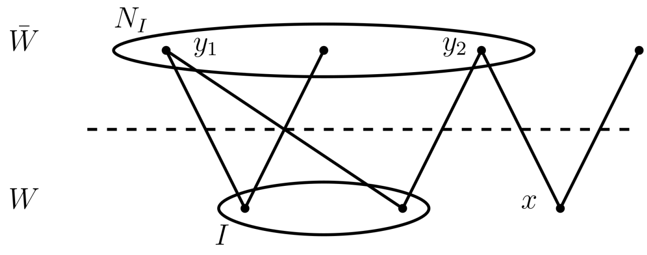

Figure 14: Illustration of Lemma 21 and 22.

**Lemma 23.** *Let $G$ be the graph $(V, E)$ and $(T, \delta)$ a decomposition tree of $G$. If there is a boolean-width class $I \in LR_w$ such that $C(I) \neq \emptyset$ then the dominating-set-width value of the cut $[W, \bar{W}]$ is less than two times the boolean-width value of $[W, \bar{W}]$.*

*Proof.* Let $I$ be a boolean-width class in $W$ such that $C(I) \neq \emptyset$, let $y$ be defined as in Lemma 21 and $X = \bigcup\{x\}$ with $x$ defined as in Lemma 22 with respect to $y$. Observe that $X = \big(N(y) \cap W\big) - C(I)$. If $X = \emptyset$ then $(I, y) \equiv (I, \bar{\emptyset})$ and we are done. Let $X \neq \emptyset$. We define two boolean-width classes in $\bar{W}$:

- $O$ is the boolean-width representative of $N(X) \cap \bar{W} - N_I$.
- $Q$ is the boolean-width representative of $\big(N(X) \cap \bar{W} - N_I\big) \cup \{y\}$.

$O$ is not adjacent to $C(I)$ and $Q$ is, thus we can see that $O \not\equiv_{bw} Q$. Figure 15 shows an example of such a $O$ and $Q$ for the cut on Figure 14 .

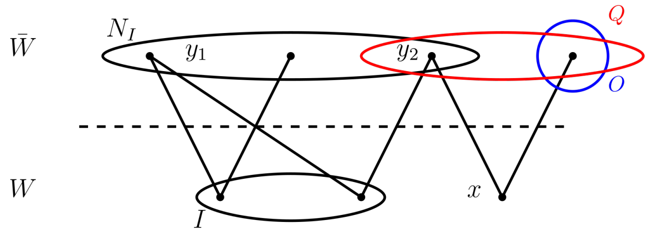

Figure 15

We will prove that $(I, O) \equiv (I, Q)$. To prove this we need to prove two points:

1. $I \equiv_{bw} I$
2. $\big(N(O) \cap W\big) - C(I) = \big(N(Q) \cap W\big) - C(I)$.

The first point is true. The second point can be proven by first showing that $N(Q) \cap W = \big(N(O) \cap W\big) \cup \big(N(y) \cap W\big)$. By

$$\begin{aligned} N(Q) \cap W &= N\big((N(X) \cap \bar{W} - N_I) \cup \{y\}\big) \cap W \ \big(\text{because } (N(X) \cap \bar{W} - N_I) \cup \{y\} \equiv_{bw} Q\big), \\ &= \big(N(N(X) \cap \bar{W} - N_I) \cup N(y)\big) \cap W, \\ &= \big(N(N(X) \cap \bar{W} - N_I) \cap W\big) \cup \big(N(y) \cap W\big), \\ &= \big(N(O) \cap W\big) \cup \big(N(y) \cap W\big) \ \big(\text{because } N(X) \cap \bar{W} - N_I \equiv_{bw} O\big). \end{aligned} \tag{9}$$

If we remove $C(I)$ from $N(Q) \cap W$ and use the equality of $X$ given above we get $\big(N(Q) \cap W\big) - C(I) = \big((N(O) \cap W) - C(I)\big) \cup \big((N(y) \cap W) - C(I)\big) = \big((N(O) \cap W) - C(I)\big) \cup X$. Let $x$ be an element from $X$ then $N(x) \cap \bar{W} \not\subseteq N_I$, thus $x$ is adjacent to vertices in $O$. And $x \notin C(I)$, thus $x \in \big(N(O) \cap W\big) - C(I)$. We can conclude that $\big((N(O) \cap W) - C(I)\big) \cup X = \big(N(O) \cap W\big) - C(I)$. This proves the second point and thus that $(I, O) \equiv (I, Q)$. □

With Lemma 23 we can prove the last lemma which shows the relation between the number of dominating-set-width classes on a cut and the presence of a single twin on the cut.

**Lemma 24.** *Let $G$ be the graph $(V, E)$. If for every tree decomposition $(T, \delta)$ of $G$ the list $TC_w$ contains a single twin with non empty center for every $w \in T$, then $dsw(G) < 2 \cdot bw(G)$.*

*Proof.* Let $W \subset V(G)$ and $TC_w$ has a single twin $v$ such that $C(v) \neq \emptyset$. Then because of Lemma 23 we know the dominating-set-width value of the cut $[W, \bar{W}]$ is less than two times the boolean-width value of $[W, \bar{W}]$. If this is true for every $w \in T$ for every decomposition tree $(T, \delta)$ then it is also true for the decomposition tree such that $bw(T, \delta) = bw(G)$ resulting in $dsw(G) < 2 \cdot bw(G)$. □

**Corollary 4.** *Let $G$ be a graph of the graph class $\mathcal{C}$. If for every decomposition tree $(T, \delta)$ the list $TC_w$ contains a single twin with a non empty center for every $w$ in $T$, then we can say that the dominating-set-width of the class $\mathcal{C}$ is strictly less then two times the boolean-width value of the class.*

## 6.4 Linear Dominating-Set-Width

We might be able to improve the result of the run time analysis by proving that the linear version of dominating-set-width has a faster run time than that of the linear boolean-width. The linear version of dominating-set-width and boolean-width is defined exactly the same way as the normal version, with the only difference being that the only decomposition trees allowed are caterpillar tree graphs.

**Definition 25.** *Let $G$ be the graph $(V, E)$. We define the linear boolean-width of $G$, $lbw(G)$, as the minimum boolean-width value taken over every caterpillar decomposition tree $(T, \delta)$ of $G$.*

**Definition 26.** *Let $G$ be the graph $(V, E)$. We define the linear dominating-set-width of $G$, $ldsw(G)$, as the minimum dominating-set-width value taken over every caterpillar decomposition tree $(T, \delta)$ of $G$.*

Linear boolean-width a run time of $O^*(2^{2 \cdot lbw(T,\delta)})$. In order to beat this we need a run time of $O^*(2^{ldsw(T,\delta)})$. In order to prove this run time, we need to show that there is a small enough bound on the number of combinations of $K$ in Algorithm 2. Unfortunately this is not true in the general case.

**Lemma 25.** *Let $G$ be the graph $(V, E)$. Let $A$, $B$ and $W$ be defined the same way as in Section 4. Assume that $B$ is a singleton set and $J \neq \emptyset$ is a boolean-width class in $B$. If $J$ is adjacent to every other vertex on the border of $\bar{A}$ and no others, then for every boolean-width class $I$ in $A$ we have $J \equiv_{bw} I \cup J$ in $W$ and the center of the boolean-width class where $J$ belongs to is $\emptyset$.*

*Proof.* $J$ is adjacent to every other vertex on the border of $\bar{A}$. Thus $N(J) \cap \bar{W}$ is every vertex on the border of $\bar{W}$. This means that adding $I$ to $J$ will not change a thing with respect to the set of vertices we are adjacent to across the cut. In other words, for every boolean-width class $I$ in $A$ we have $N(J) \cap \bar{W} = N(I \cup J) \cap \bar{W}$. Figure 16 shows a cut with such a boolean-width class $J$. Just as in previous figures of cuts only the edges across the cut are shown not the rest of the graph. The dotted line show that this type of cut can be of any size. For the second part, let $I$ be the boolean-width class in $A$ such that $N(I) \cap \bar{A}$ is every vertex of the border of $\bar{A}$. Then $I \equiv_{bw} J$ in $W$, because $J$ is adjacent to every other vertex on the border of $\bar{A}$ but no other one. Thus we have two disjoint sets in $W$ who are equivalent, resulting in an empty center. □

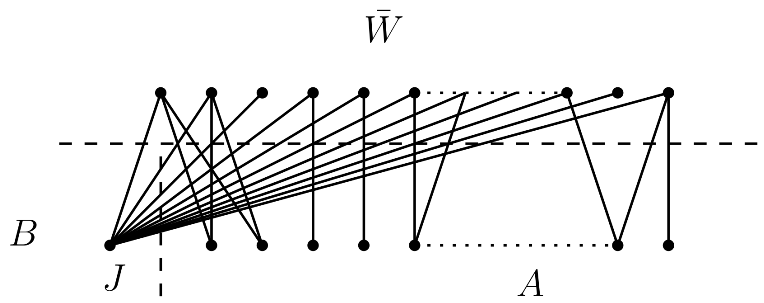

Figure 16: An example of a boolean-width class $J$ discribed in Lemma 25.

This lemma shows that there exists cuts such that Algorithm 2 will loop over $2^{2k}$ dominating-set-width classes where $k$ is the boolean-width of the cut. Because there are $2^k$ number of different combinations of boolean-width classes $I$ and $J$ that result in the same boolean-width class $K$ which has an empty center, thus there are $2^k$ differents sets $P$ such that $(K, P)$ is a dominating-set-width class. This can even happen when all other boolean-width classes do not have empty centers. Resulting in a run time slower than $O^*(2^{ldsw(T,\delta)})$.

# 7 Conclusion

In this master's thesis we have created a definition for the dominating-set-width and showed that it solves the minimum dominating set problem faster than boolean-width. It does not solve the minimum dominating set problem asymptotically faster for general graphs, but we have proven that there are graph classes for which it does solve the minimum dominating set problem asymptotically faster.

## 7.1 Discussion and Future Research

We would have liked to have proven more bounds for different graph classes. Unfortunately, analyzing graph classes proved to be a bigger challenge than originally thought. Every class has its own unique structure and properties, making it hard to come up with a general approach.
Although we are happy with the current definition of dominating-set-with, there is always room for improvement. We have some thoughts about improving the width that may lead to an algorithm that has an asymptotically faster tun time regardless of the graph class.

### 7.1.1 More Graph Class Analysis

Other graph classes that show potential for benefiting greatly of the dominating-set-width are planar graphs and $k$-tree graphs. For planar graphs creating a decomposition tree by using a sweep line shows that each cut has a number of single twins, which lowers the value of the dominating-set-width by quite a bit. However, it is unclear at this point how much this is. $k$-trees are built by adding vertices to the graph one by one. This order can be used to make a decomposition tree by creating a caterpillar tree graph that adds the vertices in the reversed order given with the $k$-tree.

### 7.1.2 An Improvement For The Dominating Set Widht

During the search for the definition of the dominating-set-width we came up with another definition that showed potential and which might be an improvement on the current definition. A small improvement, but an improvement nonetheless. The idea is to extend the centers of boolean-width classes. Let $D(X, O) = N(O) \cap W - X$ for a subset $W$ of $V(G)$. If for a dominating-set-width class $(I, O)$ in $W$ we have that for all $X \equiv_{bw} I$ there is another dominating-set-width class $(I, O')$ such that $D(X, O) = D(X, O')$ one could argue that the dominating-set-width class $(I, O)$ can be discarded. For example, Figure 17 shows a simple cut which has a boolean-width value of 7. This cut does contain single twins who have a non empty center, thus we know by Lemma 23 that the dominating-set-width value of this cut is less than two times its boolean-width value.

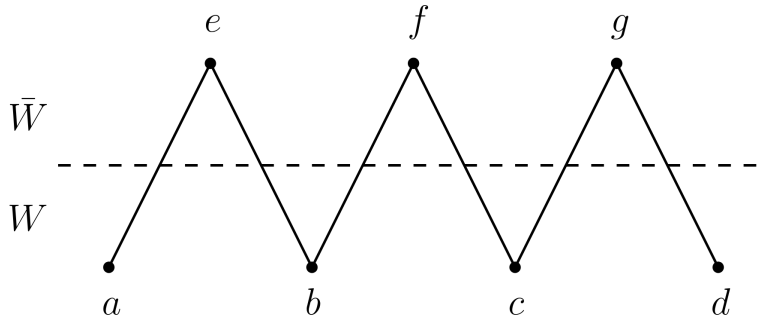

Figure 17

The boolean-width class $\{a, c\}$ has an empty center, because $\{a, c\} \equiv_{bw} \{b, d\}$. This means that there are 7 dominating-set-width classes of the form $(\{a, c\}, O)$ with $O$ a boolean-width class in $\bar{W}$. If we look closer to the dominating-set-width class $(\{a, c\}, \{f\})$ we can see that we may have created a class that would store a partial solution that we don't need. The elements in $(\{a, c\}, \{f\})$ are:

- $(\{a, c\}, \{f\})$
- $(\{b, d\}, \{f\})$
- $(\{a, b, d\}, \{f\})$
- $(\{b, c, d\}, \{f\})$
- $(\{b, c\}, \{f\})$
- $(\{a, b, c\}, \{f\})$
- $(\{a, c, d\}, \{f\})$
- $(\{a, b, c, d\}, \{f\})$

Now for every pair in this class we look at the help they receive from $\{f\}$, by calculating $(N(f) \cap W) - X)$ for every $X \equiv_{bw} \{a, c\}$. This gives the following results:

- $(\{a, c\}, \{f\})$ receives help dominating $b$ from $f$
- $(\{a, c, d\}, \{f\})$ receives help dominating $b$ from $f$
- $(\{b, d\}, \{f\})$ receives help dominating $c$ from $f$
- $(\{a, b, d\}, \{f\})$ receives help dominating $c$ from $f$
- $(\{b, c\}, \{f\})$ receives no help from $f$
- $(\{a, b, c\}, \{f\})$ receives no help from $f$
- $(\{b, c, d\}, \{f\})$ receives no help from $f$
- $(\{a, b, c, d\}, \{f\})$ receives no help from $f$

Every pair that receives receives no help from $f$ could just as well get paired up with the boolean-width class $\bar{\emptyset}$ in $\bar{W}$. $\{a, c\}$ and $\{a, c, d\}$ both receive help dominating $b$ from $f$ but both could have gotten that from $e$. $\{b, d\}$ and $\{a, b, d\}$ both receive help dominating $c$ from $f$ but bot could have gotten that from $g$. As we can see every pair in the dominating-set-width class $(\{\underline{a}, c\}, \{f\})$ could get the exact same help with dominating $W$ from a different boolean-width class in $\bar{W}$. With a different equivalence relation we could get equivalence classes in such a way that:

- $(\{a, c\}, \{f\})$ and $(\{a, c, d\}, \{f\})$ are in the same class as $(\{a, c\}, \{e\})$ and $(\{a, c, d\}, \{e\})$
- $(\{b, d\}, \{f\})$ and $(\{a, b, d\}, \{f\})$ are in the same class as $(\{b, d\}, \{g\})$ and $(\{a, b, d\}, \{g\})$
- $(\{b, c\}, \{f\})$, $(\{a, b, c\}, \{f\})$, $(\{b, c, d\}, \{f\})$ and $(\{a, b, c, d\}, \{f\})$ are in the same class as $(\{b, c\}, \bar{\emptyset})$, $(\{a, b, c\}, \bar{\emptyset})$,$(\{b, c, d\}, \bar{\emptyset})$

This way the dominating-set-width class $(\{a, c\}, \{f\})$ is no longer needed and we do not lose any partial solution. However, we can not use Algorithm 1. The lemmas that prove the correctness of the algorithm fail if we use this new equivalence relation to make equivalence classes. It would be interesting to see if we can create a FPT DP algorithm that uses this new equivalence relation, as it is clear that this creates less equivalence classes than the current definition of dominating-set-width.

# Bibliography

[1] Kwangjun Ahn and Jisu Jeong. “Computing the maximum matching width is NP-hard”. In: *CoRR* abs/1710.05117 (2017). arXiv: 1710.05117. URL: http://arxiv.org/abs/1710.05117.

[2] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. “Complexity of finding embeddings in a k-tree”. In: *SIAM Journal of Discrete Mathematics* 8.2 (1987), pp. 277–284. ISSN: 0196-5212.

[3] Rémy Belmonte and Martin Vatshelle. “Graph classes with structured neighborhoods and algorithmic applications”. In: *Theor. Comput. Sci.* 511 (2013), pp. 54–65. DOI: 10.1016/j.tcs.2013.01.011. URL: https://doi.org/10.1016/j.tcs.2013.01.011.

[4] Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza. “Rankings of Graphs”. In: *SIAM J. Discrete Math.* 11.1 (1998), pp. 168–181. DOI: 10.1137/S0895480195282550. URL: https://doi.org/10.1137/S0895480195282550.

[5] Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. “Faster Algorithms on Branch and Clique Decompositions”. In: *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings.* Ed. by Petr Hlinený and Antonın Kucera. Vol. 6281. Lecture Notes in Computer Science. Springer, 2010, pp. 174–185. ISBN: 978-3-642-15154-5. DOI: 10.1007/978-3-642-15155-2_17. URL: https://doi.org/10.1007/978-3-642-15155-2%5C_17.

[6] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. “Boolean-width of graphs”. In: *Theor. Comput. Sci.* 412.39 (2011), pp. 5187–5204. DOI: 10.1016/j.tcs.2011.05.022. URL: https://doi.org/10.1016/j.tcs.2011.05.022.

[7] Derek G. Corneil and Udi Rotics. “On the Relationship between Clique-Width and Treewidth”. In: *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings.* Ed. by Andreas Brandstädt and Van Bang Le. Vol. 2204. Lecture Notes in Computer Science. Springer, 2001, pp. 78–90. ISBN: 3-540-42707-4. DOI: 10.1007/3-540-45477-2_9. URL: https://doi.org/10.1007/3-540-45477-2%5C_9.

[8] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. “Handle-Rewriting Hypergraph Grammars”. In: *J. Comput. Syst. Sci.* 46.2 (1993), pp. 218–270. DOI: 10.1016/0022-0000(93)90004-G. URL: https://doi.org/10.1016/0022-0000(93)90004-G.

[9] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. ISBN: 978-3-319-21274-6. DOI: 10.1007/978-3-319-21275-3. URL: https://doi.org/10.1007/978-3-319-21275-3.

[10] Frederic Dorn. “Dynamic Programming and Fast Matrix Multiplication”. In: *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings.* Ed. by Yossi Azar and Thomas Erlebach. Vol. 4168. Lecture Notes in Computer Science. Springer, 2006, pp. 280–291. ISBN: 3-540-38875-3. DOI: 10.1007/11841036_27. URL: https://doi.org/10.1007/11841036%5C_27.

[11] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013. ISBN: 978-1-4471-5558-4. DOI: 10.1007/978-1-4471-5559-1. URL: https://doi.org/10.1007/978-1-4471-5559-1.

[12] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. ISBN: 978-3-540-29952-3. DOI: 10.1007/3-540-29953-X. URL: https://doi.org/10.1007/3-540-29953-X.

[13] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. ISBN: 978-3-642-16532-0. DOI: 10.1007/978-3-642-16533-7. URL: https://doi.org/10.1007/978-3-642-16533-7.

[14] Frank Gurski. “A comparison of two approaches for polynomial time algorithms computing basic graph parameters”. In: *CoRR* abs/0806.4073 (2008). arXiv: `0806.4073`. URL: `http://arxiv.org/abs/0806.4073`.

[15] Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. “On the Power of Tree-Depth for Fully Polynomial FPT Algorithms”. In: *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*. Ed. by Rolf Niedermeier and Brigitte Vallée. Vol. 96. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 41:1–41:14. ISBN: 978-3-95977-062-0. DOI: `10.4230/LIPIcs.STACS.2018.41`. URL: `https://doi.org/10.4230/LIPIcs.STACS.2018.41`.

[16] Sang-Il Oum. “Graphs of Bounded Rank-Width”. PhD thesis. USA: Princeton University, 2005.

[17] Neil Robertson and Paul D. Seymour. “Graph minors. X. Obstructions to tree-decomposition”. In: *J. Comb. Theory, Ser. B* 52.2 (1991), pp. 153–190. DOI: `10.1016/0095-8956(91)90061-N`. URL: `https://doi.org/10.1016/0095-8956(91)90061-N`.

[18] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. “Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution”. In: *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*. Ed. by Amos Fiat and Peter Sanders. Vol. 5757. Lecture Notes in Computer Science. Springer, 2009, pp. 566–577. ISBN: 978-3-642-04127-3. DOI: `10.1007/978-3-642-04128-0\_51`. URL: `https://doi.org/10.1007/978-3-642-04128-0%5C_51`.

[19] Sigve Hortemo Sæther and Martin Vatshelle. “Hardness of computing width parameters based on branch decompositions over the vertex set”. In: *Electronic Notes in Discrete Mathematics* 49 (2015), pp. 301–308. DOI: `10.1016/j.endm.2015.06.041`. URL: `https://doi.org/10.1016/j.endm.2015.06.041`.

[20] Paul D. Seymour and Robin Thomas. “Call Routing and the Ratcatcher”. In: *Combinatorica* 14.2 (1994), pp. 217–241. DOI: `10.1007/BF01215352`. URL: `https://doi.org/10.1007/BF01215352`.

[21] Paul D. Seymour and Robin Thomas. “Graph Searching and a Min-Max Theorem for Tree-Width”. In: *J. Comb. Theory, Ser. B* 58.1 (1993), pp. 22–33. DOI: `10.1006/jctb.1993.1027`. URL: `https://doi.org/10.1006/jctb.1993.1027`.

[22] Jan Arne Telle. “Vertex Partitioning Problems: Characterization, Com- plexity and Algorithms on Partial k-Trees”. PhD thesis. University of Oregon, 1994.

[23] Martin Vatshelle. “New width parameters of graphs”. PhD thesis. University of Bergen, 2012.