Boolean-width of graphs[☆]

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

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

Abstract

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

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

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

1. Introduction

Width parameters of graphs, like tree-width, branch-width, clique-width and rank-width, are important in the field of graph algorithms. Many NP-hard graph optimization problems have fixed-parameter tractable (FPT) algorithms when parameterized by these parameters (see [23] for a recent overview, and [15, 16] for extensive ones).

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

[☆]Supported by the Norwegian Research Council, project PARALGO.
[*]Corresponding author. Tel: (+47) 55 58 40 36. Fax: (+47) 55 58 41 99.
[1]We use O^* notation that hides polynomial factors.

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

This approximation for clique-width was achieved by introducing a new parameter, the so-called rank-width $rw(G)$, that is interesting in itself. Firstly, given an n-vertex graph a decomposition of optimal rank-width can be computed in time $O(f(rw(G))poly(n))$, for some polynomial function $poly$ and a function f at least exponential [22]. Secondly, rank-width is potentially much smaller than clique-width, tree-width and branch-width: for any graph G we have $\log cw(G) \leq rw(G) \leq cw(G)$, and $rw(G) \leq tw(G) + 1$ and $rw(G) \leq bw(G)$ [35], in contrast to clique-width where there exist graphs with $cw(G)$ in $2^{\Theta(tw(G))}$ [11]. A possible drawback of rank-width is that so far no well-known NP-hard problems are solvable in time $O^*(2^{c \cdot k})$ on a decomposition of rank-width k, *e.g.* for Minimum Dominating Set the fastest runtime is $O^*(2^{0.75k^2+O(k)})$ [9, 17]. However, note that for graphs having rank-width much smaller than clique-width and tree-width it will still be preferable to use rank-width for *e.g.* Minimum Dominating Set.

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

Our paper is organized as follows. In Section 2 we define boolean-width based on the number of unions of neighborhoods across the cuts given by a decomposition tree, and argue that it is a natural parameter if the goal is fast divide-and-conquer algorithms, at least for independence and domination problems. In Section 3 we compare boolean-width to other width parameters, and in particular to rank-width. We show that $\log rw(G) \leq boolw(G) \leq 0.25rw^2(G) + O(rw(G))$. This means that boolean-width is (constantly) bounded on the same classes of graphs as clique-width and rank-width, but for higher bounds the situation is different. For a class of graphs C say parameter P is logarithmic, resp. polylog, if the value of P for any n-vertex graph G in C is logarithmic in n, resp. polylog in n. For example, boolean-width is logarithmic on interval graphs and polylog on random graphs. Whenever P is logarithmic on C, resp. polylog on C, any algorithm with runtime $O^*(2^{c \cdot P(G)})$, resp. $O^*(2^{poly(P(G))})$, will on input a graph G in C have polynomial runtime, resp. quasi-polynomial runtime. From the results depicted in Figure 1 it follows that if any of tree-width, branch-width, clique-width or rank-width is polylog on a class of graphs then so is boolean-width, while we show in Section 3 that boolean-width is logarithmic on Hsu-grids but the other parameters

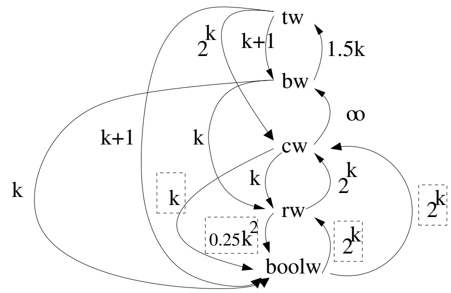

Figure 1: Upper bounds tying parameters tw=tree-width, bw=branch-width, cw=clique-width, rw=rank-width, and $boolw$=boolean-width. An arrow from P to Q labelled $f(k)$ means that any class of graphs having parameter P bounded by k will have parameter Q bounded by $f(k)$ or $O(f(k))$, and ∞ means that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

are not even polylog on Hsu-grids. Recent results showing that $boolw(G) \leq bw(G)$ [1] imply that if any of tree-width, branch-width, or clique-width is logarithmic on some graph class then so is boolean-width, but as the Hsu-grids show the converse is not always true.

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

Sections 5 and 6 are devoted to algorithms solving NP-hard problems on an n-vertex graph in time $O(2^{c \cdot k} poly(n))$ when given a decomposition tree of boolean-width k. Since the goal is to allow practical implementations of these algorithms we strive for simple descriptions, small constants c and low polynomial functions $poly$. In Section 5 we give a pre-processing routine setting up a data structure that will allow runtime at the combine step to be a function of the boolean-width of the decomposition tree, rather than the number of vertices. In Section 6 we show how to apply dynamic programming on a decomposition tree while analysing runtime as a function of its boolean-width. We focus on the Maximum Independent Set problem where we get runtime $O(n^2 k 2^{2k})$ and Minimum Dominating Set with runtime $O(n^2 + nk2^{3k})$. The algorithms can be deduced from similar algorithms in [9], that appeared before the introduction of boolean-width. We give the algorithms here using the new and simpler terminology and show that they have better runtime due to faster pre-processing and better data structures. We also give algorithms to handle the vertex weighted cases, also for Max and Min Weight Independent Dominating Set in the same runtime, and finally the case of counting all independent sets and dominating sets of given size.

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

2. Boolean-width

We deal with simple, loopless, undirected graphs and denote by $\{u,v\}$ an edge between vertices u and v. The complement of a vertex subset A of a graph $G=(V(G),E(G))$ is denoted by $\overline{A}=V(G)\setminus A$. The neighborhood of a vertex x is denoted by $N(x)$ and for a subset of vertices X we denote the union of their neighborhoods by $N(X)=\bigcup_{x\in X}N(x)$. A subset of vertices $X\subseteq V(G)$ is an independent set if there is no edge in G between any pair from X. A set $X\subseteq V(G)$ of vertices is a dominating set of G if $X\cup N(X)=V(G)$. A cut of G is a 2-partition $\{A,\overline{A}\}$ of $V(G)$. Two vertices $x,x'\in A$ are twins across $\{A,\overline{A}\}$ if $N(x)\cap\overline{A}=N(x')\cap\overline{A}$. A vertex subset $X\subseteq A$ is a twin class of A if X is a maximal set of vertices all of whom are twins across $\{A,\overline{A}\}$. The twin classes of A form a partition of A. For disjoint vertex subsets A,B of G we denote by $G(A,B)$ the bipartite graph on vertex set $A\cup B$ and edge set $\{\{u,v\}: u\in A\wedge v\in B\wedge\{u,v\}\in E(G)\}$. We denote by M_G the adjacency matrix of G, and by $M_G(A,B)$ the submatrix of M_G with the rows indexed by A and the columns by B. To ensure uniqueness of certain algorithms, *e.g.* for computing representatives for vertex subsets, we assume a total ordering σ on the vertex set of G which stays the same throughout the entire paper. If vertex u comes before vertex v in the ordering then we say u is σ-smaller than v. For easy disambiguation, we usually refer to vertices of a graph and nodes of a tree.

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

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

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

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

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

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

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

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

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

In this section we compare boolean-width $boolw$ to tree-width tw, branch-width bw, clique-width cw and rank-width rw. For any graph G, it holds that the rank-width of the graph is essentially the smallest parameter among tw, bw, cw, rw [35, 36, 41]: we have $rw(G) \leq cw(G)$ and $rw(G) \leq bw(G) \leq tw(G) + 1$ for $bw(G) \neq 0$. Accordingly, we focus on comparing boolean-width to rank-width.

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

Definition 3. Let G be a graph and $A \subseteq V(G)$. Let $\triangle$ be the symmetric difference operator, that applied to a family of sets gives the elements appearing in an odd number of sets. Define the

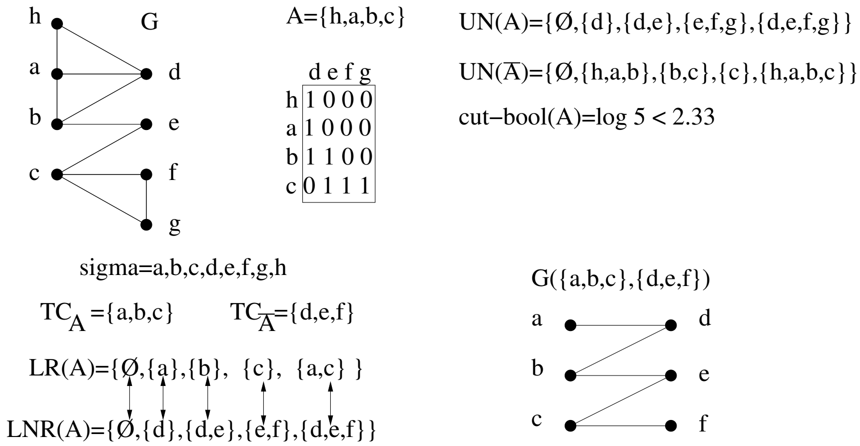

Figure 2: Example graph G and $A \subseteq V(G)$, with submatrix $M_G(A, \overline{A})$, unions of neighborhoods $U(A)$ and $U(\overline{A})$ with *bool-dim*(A) = *log*$_2|U(A)| < 2.33$, as defined in Section 2. Note that *cut-rank*(A) = *log*$_2|D(A)| = 3$, as defined in Section 3. Vertex ordering sigma yields twin class representatives TC_A and $TC_{\overline{A}}$, and the list of $\equiv_A$ representatives LR_A with pointers to list of their neighbors LNR_A, as defined in Section 5. Note that $\{h, b, c\}$ induces the largest independent set in G among the 7 subsets of A having $\equiv_A$ representative $\{a, c\}$. The graph $G(TC_A, TC_{\overline{A}})$ captures the essential information across the cut $\{A, \overline{A}\}$.

set of symmetric differences of neighborhoods of A across the cut $\{A, \overline{A}\}$ as

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

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

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

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

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

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

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

To prove the second inequality we first bijectively map $U(A)$ to some family $\mathcal{F} \subseteq 2^A$, then, we injectively map $\mathcal{F}$ to $DS(A)$. Combining these we get $|U(A)| = |\mathcal{F}| \leq |DS(A)|$, and the desired inequality follows. We let

$\mathcal{F} = \{R_X : \ \exists X \subseteq A$ such that R_X is the output of the below algorithm on input $X\}$.

Initialize $R_X \leftarrow \emptyset$ and $N_X \leftarrow \emptyset$;
For $v \in A$ taken in order σ on $V(G)$ do:
Let $W = N(v) \cap \overline{A}$;
If $W \subseteq N(X) \cap \overline{A}$ and $W \setminus N_X \neq \emptyset$ then add v to R_X and add all vertices in W to N_X.

Since the algorithm manipulates $N(X) \cap \overline{A}$ but not X, it is clear for all $X, X' \subseteq A$ that if $N(X) \cap \overline{A} = N(X') \cap \overline{A}$ then $R_X = R_{X'}$. Besides, at the end of the algorithm $N(R_X) \cap \overline{A} = N_X = N(X) \cap \overline{A}$ (the first equality is an invariant, and for the second note that the algorithm loops through all $v \in X \subseteq A$). In other words, if $R_X = R_{X'}$ then $N(X) \cap \overline{A} = N(X') \cap \overline{A}$. Hence, there is a bijection between $U(A)$ and $\mathcal{F}$. We now prove that the function $f : \mathcal{F} \to DS(A)$ given by $f(R_X) = \Delta closure(\{N(x) \cap \overline{A} : x \in R_X\})$ is injective, where $\Delta closure(\mathcal{G})$ is the unique smallest family containing $\mathcal{G}$ that is closed under the symmetric difference of its members. Let $R_X \in \mathcal{F}$ and $R_{X'} \in \mathcal{F}$ such that $R_X \neq R_{X'}$. Then we know from above that $N(X) \cap \overline{A} \neq N(X') \cap \overline{A}$, and hence $N(R_X) \cap \overline{A} \neq N(R_{X'}) \cap \overline{A}$. Therefore, $\Delta closure(\{N(x) \cap \overline{A} : x \in R_X\}) \neq \Delta closure(\{N(x) \cap \overline{A} : x \in R_{X'}\})$, to conclude the proof of the second inequality in the lemma.

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

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

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

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

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

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

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

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

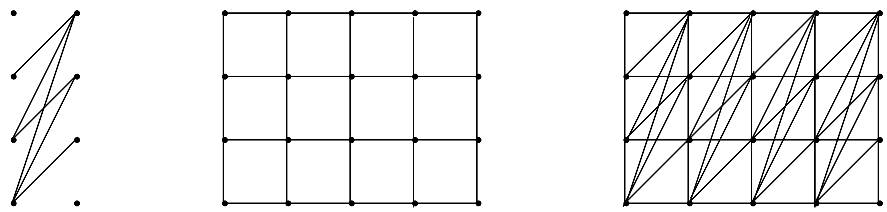

Figure 3: The Hsu graph H_3, the 4×5 grid, and the Hsu-grid $HG_{4,5}$.

illustration is given in Figure 3). We consider the cut $\{A_k, B_k\}$. A union of neighborhoods of vertices of A_k is always of the form $\{b_1, b_2, \ldots, b_l\}$, and as a consequence,

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

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

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

We begin with a useful lemma.

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

Proof: We distinguish two self-exclusive cases.

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

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

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

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

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

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

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

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

To prove the last statement of Lemma 3, we note that any horizontal decomposition tree of $HG_{p,q}$ has rank-width $2q$, and therefore the rank-width of $HG_{p,q}$ is at most $2q < p/4$. We consider a decomposition tree (T, δ) of $HG_{p,q}$ of optimal rank-width, and an edge $\{u, v\}$ of T inducing a balanced cut $\{A, \overline{A}\}$ in $HG_{p,q}$. From Lemma 2 and the fact that the rank-width of $HG_{p,q}$ is at most $2q < p/4$, $HG_{p,q}(A, \overline{A})$ has a $q/6$-matching as induced subgraph. Therefore, the value of $bool\text{-}dim(A)$ is at least $q/6$, and the boolean-width of (T, δ) is at least $q/6$. □

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

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

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

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

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

Proof: For a proper introduction to clique-width refer to [12]. We will in fact not address directly clique-width, but a closely related parameter called module-width [39], whose definition is based on rooted binary trees and twin classes of a subset of vertices. Let (T, δ) be a decomposition tree of G. Let T_r be the rooted binary tree we get by subdividing any edge of T for a root r. The module-width of (T_r, δ), denoted $modw(T_r, \delta)$, is the maximum, over all nodes a of T_r, of the number of twin classes of A_a^r. Note that $\overline{A_a^r}$ is not used in this definition and thus the choice of rooting is important. The module-width of G, denoted $modw(G)$, is the minimum module-width taken over every decomposition tree (T, δ) of G and over the subdivision of every edge e of T to obtain a rooted tree T_r [39].

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

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

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

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

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

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

We prove in this section that the quadratic upper bound on *bool-dim* as a function of *cut-rank* from Lemma 1 is tight. More precisely, we exhibit a graph G and $A \subseteq V(G)$ where $|U(A)| =$

$|DS(A)|$, leading to *bool-dim*$(A) = \log_2 |DS(A)| = \Theta(\textit{cut-rank}^2(A))$. Note that $U(A)$ is in a bijection with the Boolean space spanned over the Boolean algebra by the rows of $M_G(A, \overline{A})$. The question of the possible cardinalities of the Boolean space of a given $\{0,1\}$-matrix has been studied by several researchers, see [44] and the bibliography therein. Recall that

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

We now want to prove that $\mathcal{F}' = \{N(a_Z) : a_Z \in X'\}$ is closed under the symmetric difference of its members. Pick $N(a_Z)$ and $N(a_{Z'})$ therein: both of them belong to $\mathcal{F}$, so from Observation 1 and the fact $\mathcal{F}$ is closed under symmetric difference, we deduce that $N(a_{Z\Delta Z'}) \in \mathcal{F}$, in other words $a_{Z\Delta Z'} \in X$. Besides, note that we can also write $X' = \{a_Z \in X,\ |W\cap Z| \text{ is even}\}$ and it is clear that if both $|W\cap Z|$ and $|W\cap Z'|$ are even, then $|W\cap(Z\Delta Z')|$ is even. Hence, $\mathcal{F}'$ is closed under the symmetric difference of its members. We also check that $dim(\mathcal{F}') \leq p-1$. Indeed, $\mathcal{F}' \subseteq \mathcal{F}$, so we only need to show that $\Delta closure(\mathcal{F})$ properly contains $\Delta closure(\mathcal{F}')$. We consider a_T: clearly $N(a_T) \in \mathcal{F}$. Recall that $b_W \in N(a_T)\setminus N(a_S)$ and that $\mathcal{F}' = \{N(a_Z) : a_Z \in X \wedge b_W \notin N(a_Z)\}$. Therefore every member $N(a_{Z_0}) \in \Delta closure(\mathcal{F}')$ is such that $b_W \notin N(a_{Z_0})$. Since $b_W \in N(a_T)$, we deduce $N(a_T) \notin \Delta closure(\mathcal{F}')$. Hence, $dim(\mathcal{F}') \leq p-1$. Now, we can conclude by applying the inductive hypothesis on $\mathcal{F}'$. □

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

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

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

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

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

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

□

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

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

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

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

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

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

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

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

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

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

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

1) $\forall X \subseteq A, \exists R \in LR_A$ such that $R \equiv_A X$

2) $\forall R \in LR_A$, if $R \equiv_A X$ then $|R| \leq |X|$

3) $\forall R \in LR_A$, if $R \equiv_A X$ and $|R| = |X|$ then R lexicographically σ-smaller than X.

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

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

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

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

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

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

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

Let us first argue for the correctness of the algorithm. The first iteration of the while-loop will set $\{v\}$ as representative, for every $v \in TC_A$, and there exist no other representatives of size 1 in LR_A. The algorithm computes all representatives of size i before it moves on to those of size $i+1$. LastLevel will contain all representatives of size i while NextLevel will contain all representatives of size $i+1$ found so far. Every representative will be expanded by every possible node and checked against all previously found representatives. The only thing left to prove is that any representative R can be written as $R' \cup \{v\}$ for some representative R'. Assume for contradiction that no R' exists such that $R = R' \cup \{v\}$. Then let v be the lexicographically largest element of R, then

Algorithm 1 List of representatives and their neighborhood

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

$R \setminus \{v\}$ can not be a representative so let R'' be the representative of $[R \setminus \{v\}]_{\equiv_A}$. We know that $R'' \cup \{v\} \equiv_A R$, we know that $|R'' \cup \{v\}| \leq |R|$ and that $R'' \cup \{v\}$ comes before R in a lexicographical ordering contradicting that R is a representative.

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

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

Lemma 8. *Let G be an n-vertex graph and (T, δ) a decomposition tree of G. Assume the pre-processing described in Lemmata 5 and 7 has been done. Then, in time $O(n \cdot ntc^2(T,\delta) \cdot boolw(T,\delta) \cdot 2^{boolw(T,\delta)})$ we compute for every cut $\{A, \overline{A}\}$ associated to an edge of T a datastructure allowing, for any $X \subseteq A$, to access in $O(|X|)$ time the entry R of LR_A such that $X \equiv_A R$.*

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

Initialize M to a two dimensional table with $|LR_A| \times |TC_A|$ elements.
for every vertex v of TC_A **do**
  for R in LR_A **do**
    $R' = R \cup \{v\}$
    find R_U in LR_A that is linked to the neighborhood $N(R') \cap TC_{\overline{A}}$ in LNR_A
    add a pointer from $M[R][v]$ to R_U
  end for
end for

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

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

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

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

6. Dynamic programming for fast runtime by boolean-width

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

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

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

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

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

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

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

6.1. Maximum Independent Set

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

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

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

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

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

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

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

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

the class $[R_a \cup R_b]_{\equiv_{A_w}}$ is at least $|I_a| + |I_b| = |I_w|$. It remains to show that this representative is R_w. By Observation 3 we have $R_a \equiv_{A_w} I_a$ and $R_b \equiv_{A_w} I_b$ so that $R_a \cup R_b \equiv_{A_w} I_a \cup I_b$. Since $I_w = I_a \cup I_b$ and we assumed $R_w \equiv_{A_w} I_w$ we therefore have $R_a \cup R_b \equiv_{A_w} R_w$ as desired.

To finish the correctness proof, we need to show that if $Tab_w[R_w] = k$ then there exists $I_w \subseteq A_w$ with $|I_w| = k$ and $I_w \equiv_{A_w} R_w$ and I_w an IS in G. For this, note that the Combine step increases the value of $Tab_w[R_w]$ only if there exist indices $R_a \in LR_{A_a}$ and $R_b \in LR_{A_b}$ such that $R_a \cup R_b$ is an IS in $G(R_a, R_b)$, and $R_a \cup R_b \equiv_{A_w} R_w$, and $Tab_a[R_a] = k_a$, and $Tab_b[R_b] = k_b$, and $k_a + k_b = k$. Since Tab_a, Tab_b are filled correctly we have two independent sets I_a, I_b in G with $R_a \equiv_{A_a} I_a$ and $R_b \equiv_{A_b} I_b$ and $|I_a| = k_a$ and $|I_b| = k_b$. We claim that $I_a \cup I_b$ is the desired I_w. Since $R_a \cup R_b$ is an IS of $G(R_a, R_b)$ it is clear that $I_a \cup I_b$ is an IS in G of size $k_a + k_b = k$. It remains to show that $I_a \cup I_b \equiv_{A_w} R_w$. By Observation 3 we have $R_a \equiv_{A_w} I_a$ and $R_b \equiv_{A_w} I_b$ so that $R_a \cup R_b \equiv_{A_w} I_a \cup I_b$. Since we assumed $R_a \cup R_b \equiv_{A_w} R_w$ we therefore have $I_a \cup I_b \equiv_{A_w} R_w$ as desired. □

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

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

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

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

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

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

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

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

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

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

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

We now analyse the time complexity of the global computation. First we have to run the algorithm mentioned in Claim 10.1, which takes $O(n^2)$ time. For the rest, note that $|R_a| \leq ntc(T,\delta)$

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

6.2. Counting independent sets

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

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

The initialization at a leaf l of T_r should be:

$Tab_l[\delta(l)][0] = 0$
$Tab_l[\delta(l)][1] = 1$
$Tab_l[\emptyset][0] = 1$
$Tab_l[\emptyset][1] = 0$

The combine step is given in Algorithm 5. Note that two families F_a and F_b of vertex subsets, taken from two disjoint sets of vertices, can be combined into $|F_a| * |F_b|$ larger vertex subsets. Note also that in the inner loop of the combine step $k_a, k_b \leq \alpha$. The proof of correctness and runtime remains otherwise much the same.

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

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

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

6.3. Minimum Dominating Set

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

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

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

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

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

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

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

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

and ∞ if no such S exists.

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

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

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

The following lemma will be useful in the correctness proof.

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

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

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

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

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

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

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

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

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

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

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

6.4. Counting dominating sets

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

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

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

$Tab_l[\delta(l)][\emptyset][1] = 1$
$Tab_l[\delta(l)][R][1] = 1$
$Tab_l[\emptyset][R][0] = 1$

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

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

6.5. Independent Dominating Sets

Combining the requirements of independence and domination in the definition of tables and in the algorithm we can solve both the Minimum and Maximum Independent Dominating Set problems. Note for the runtime given in Theorem 5 that $O(n(n + ntc^2(T,\delta) \cdot k2^k + k^2 2^{2k}))$ is bounded by $O(n^2 + nk2^{3k})$ since $ntc(T,\delta) \leq 2^k$.

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

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

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

6.6. Weighted cases

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

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

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

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

7. Conclusion and Perspectives

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

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

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

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

There are many questions about boolean-width left unanswered. It is known that the boolean-width of a graph is smaller than its tree-width, branch-width and clique-width, but it is not clear how high the boolean-width can be as a function of its rank-width. Is boolean-width linear in rank-width, or subquadratic in rank-width, for every graph? It has been shown that a $k \times k$ grid has rank-width exactly $k-1$ [26]. We have seen that its boolean-width lies between $\frac{1}{6}k$ (see Theorem 2) and $k+1$ (derived from the upper bound given by clique-width), but it would be nice to close this gap and find its exact value.

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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