Boolean-width of graphs*

B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle

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

Abstract. We introduce the graph parameter boolean-width, related to the number of different unions 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. Boolean-width is similar to rank-width, which is related to the number of $GF[2]$-sums (1+1=0) of neighborhoods instead of the Boolean-sums (1+1=1) used for boolean-width. For an n-vertex graph G given with a decomposition tree of boolean-width k we show how to solve Minimum Dominating Set, Maximum Independent Set and Minimum or Maximum Independent Dominating Set in time $O(n(n+2^{3k}k))$. We show that for any graph the square root of its boolean-width is never more than its rank-width. We also 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 tree-width, branch-width, clique-width and rank-width $\Theta(n)$. Moreover, any optimal rank-decomposition of such a graph will have boolean-width $\Theta(n)$, *i.e.* exponential in the optimal boolean-width.

1 Introduction

Width parameters of graphs, like tree-width, branch-width, clique-width and rank-width, are important in the theory of graph algorithms. Many NP-hard graph optimization problems have fixed-parameter tractable (FPT) algorithms when parameterized by these graph width parameters, see e.g. [14] for an overview. Such algorithms usually have two stages, a first stage computing the right decomposition of the input graph and a second stage solving the problem by a divide-and-conquer approach, or dynamic programming, along the decomposition. For practical applications we must look carefully at the runtimes as a function of the parameter. We may then have to concentrate on heuristic algorithms for the first stage, for example in the way done for tree-width as part of the TreewidthLIB project at University of Utrecht, see e.g. [19]. For the second stage we should carefully design algorithms for each separate problem. When comparing the usefullness of these width parameters, we first need to compare the values of the parameters on various graph classes, we secondly need good algorithms or fast heuristics for the first stage, and we thirdly need to compare the best runtimes for the second stage. In this paper we introduce a graph width parameter called boolean-width, and compare it to other parameters.

Firstly, we show that the boolean-width of a graph is never more than quadratic in its rank-width, which also constitutes a comparison with other parameters since the rank-width of a graph is known to never be larger than its clique-width, nor its branch-width, nor its tree-width plus one [22, 23, 25]. We also know that the boolean-width of a graph is never larger than its tree-width plus one [1]. On the other hand we show a class of graphs, the Hsu-grids, that have boolean-width bounded by k while they have rank-width (and thus also clique-width, branch-width and tree-width) exponential in k. See Figure 1 for a sketch of how the various parameters compare. Note that for any class of graphs we have

* Supported by the Norwegian Research Council, project PARALGO.

only three possibilities: either all five parameters are bounded (e.g. for trees) or none of them are bounded (e.g. for grids) or only clique-width, rank-width and boolean-width are bounded (e.g. for cliques).

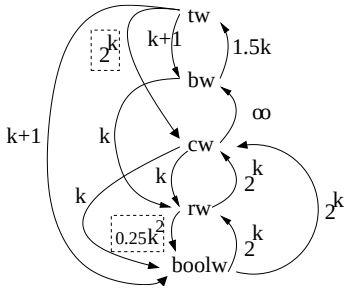

Fig. 1. Upper bounds tying parameters tw=tree-width, bw=branch-width, cw=clique-width, rw=rank-width and boolw=boolean-width. An arrow from P to Q labelled $f(k)$ means that any class of graphs having parameter P bounded by k will have parameter Q bounded by $O(f(k))$, and ∞ means that no such upper bound can be shown. Except for the labels in a box the bounds are known to be tight, meaning that there is a class of graphs for which the bound is $\Omega(f(k))$. For the box containing label 2^k a $\Omega(2^{k/2})$ bound is known [6].

Secondly, regarding first stage algorithms, since boolean-width is tied to rank-width, when parameterizing by the boolean-width of an input graph we get an FPT algorithm that computes an approximation of an optimal boolean-width decomposition, by applying either the algorithm of Hliněný and Oum [13] computing an optimal rank-decomposition or the approximation algorithm of Oum and Seymour [23]. We have also initiated research into heuristic algorithms for the first stage.

Thirdly, for the second stage, we concentrate in this paper on the Minimum Dominating Set (MDS) problem and show that given a decomposition of boolean-width k of an n-vertex graph we can solve MDS in time $O(n(n + 2^{3k}k))$. See Figure 2 for a comparison of the best runtimes for MDS when parameterized by other width parameters. Combining the information in Figures 1 and 2 we see that the runtime for MDS compares well to the other parameters. For example, clique-width is bounded for a class of graphs exactly when boolean-width is, but as we show in Section 3 boolean-width is never larger than clique-width, and therefore $O^*(2^{3boolw})$ is always better than $O^*(2^{4cw})$. We also show there exists graphs where cliquewidth is exponential in boolean-width and for these graphs $O^*(2^{3boolw})$ is exponentially better than $O^*(2^{4cw})$. In a companion paper [3] we similarly show that in time $O^*(2^{d\times q\times boolw^2})$, for problem-specific constants d and q, we can solve a large class of vertex subset and vertex partitioning problems.

	tree-width	branch-width	clique-width	rank-width	boolean-width
MDS	$O^*(2^{1.58tw})$[26]	$O^*(2^{2bw})$ [9]	$O^*(2^{4cw})$ [18]	$O^*(2^{0.75rw^2+O(rw)})$ [5, 10]	$O^*(2^{3boolw})$ [here]

Fig. 2. Runtimes achievable for Minimum Dominating Set using various parameters.

A main open problem is to approximate the boolean-width of a graph better than what we get by using the algorithm for rank-width [13]. Nevertheless, for many problems it could be advantageous to use boolean-width for the second stage regardless of which decomposition is given. In Figure 3 we illustrate the runtimes achievable by the best second-stage algorithm for the MDS problem using either the algorithm given in Section 4 of this pa-

per or the best runtime when parameterized by rankwidth, which are $O^\star(2^{0.75rw^2+O(rw)})$ algorithms in both [5] and [10]. The values in Figure 3 assume we are given a decomposition tree (T,δ) of rank-width rw and is based on the result from Section 3 that the boolean-width of (T,δ) lies between $\log rw$ and $\frac{1}{4}rw^2+\frac{5}{4}rw+\log rw$.

boolean-width of (T,δ)=	*using rank-width*	*using boolean-width*
$0.25rw^2+O(rw)$	$O^*(2^{0.75rw^2+O(rw)})$	$O^*(2^{0.75rw^2+O(rw)})$
rw	$O^*(2^{0.75rw^2+O(rw)})$	$O^*(2^{3rw})$
$\log rw$	$O^*(2^{0.75rw^2+O(rw)})$	$O^*(1)$

Fig. 3. Runtimes achievable for Minimum Dominating Set. Given a decomposition tree (T,δ) of rank-width rw, we know that the boolean-width of (T,δ) lies between $\log rw$ and $0.25rw^2+O(rw)$.

For an appropriate class of Hsu-grids we are able to show that *any* optimal rank-decomposition will have boolean-width exponential in the optimal boolean-width. This suggests that although we can solve NP-hard problems in polynomial time on Hsu-grids if we use boolean-width as the parameter (as we see in Figure 3) we would get exponential time if we used any of the other graph width parameters.

Finally, we remark that the use of Boolean-sums in the definiton of boolean-width (see Section 2) means a new application for the theory of Boolean matrices, *i.e.* matrices with Boolean entries, 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 [8, 17, 20, 24].

2 Boolean-width

When applying divide-and-conquer to a graph we first need to divide the graph. A common way to store this information is to use a *decomposition tree* and to evaluate decomposition trees using a *cut function*. The following formalism is standard in graph and matroid decompositions (see, *e.g.*, [11, 23, 25]).

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 $n=|V(G)|$ leaves, and δ is a bijection between the vertices of G and the leaves of T. For $A\subseteq V(G)$ let $\overline{A}$ denote the set $V(G)\setminus A$. Every edge of T defines a cut $\{A,\overline{A}\}$ of the graph, i.e. a partition of $V(G)$ in two parts, namely the two parts given, via δ, by the leaves of the two subtrees of T we get by removing the edge. Let $f:2^V\to\mathbb{R}$ be a symmetric function, i.e. $f(A)=f(\overline{A})$ for all $A\subseteq V(G)$, also called a* cut function. *The f-width of (T,δ) is the maximum value of $f(A)$, taken over all cuts $\{A,\overline{A}\}$ of G given by an edge uv of T. The f-width of G is the minimum f-width over all decomposition trees of G.*

The cuts $\{A,\overline{A}\}$ given by edges of the decomposition tree are used in the divide step of a divide-and-conquer approach. For the conquer step we solve the problem recursively, following the edges of the tree T (after choosing a root) in a bottom-up fashion, on the graphs induced by vertices of one side and of the other side of the cuts. In the combine step we must join solutions from the two sides, and this is usually the most costly and complicated operation. The question of what 'solutions' we should store to get an efficient combine step is related to what type of problem we are solving. Let us consider vertex

subset or vertex partitioning problems on graphs, and in particular Maximum Independent Set for simplicity[1]. For a cut $\{A, \overline{A}\}$ we note that if two independent sets $X \subseteq A$ and $X' \subseteq A$ have the same set of neighbors in $\overline{A}$ 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 the following equivalence relation on subsets of A will be useful.

Definition 2. *Let G be a graph and $A \subseteq V(G)$ a vertex subset of G. Two vertex subsets $X \subseteq A$ and $X' \subseteq A$ are neighbourhood equivalent w.r.t. A, denoted by $X \equiv_A X'$, if $\overline{A} \cap N(X) = \overline{A} \cap N(X')$.*

If for each class $[X]_{\equiv_A}$ we store the maximum independent set in $[X]_{\equiv_A}$, and similarly for each class $[Y]_{\equiv_{\overline{A}}}$ we store the maximum independent set in $[Y]_{\equiv_{\overline{A}}}$, then we can perform the combine step in time depending only on the number of such equivalence classes. The same argument can be made for a large class of vertex subset and partitioning problems. Thus, to solve these problems as fast as possible on general graphs by divide-and-conquer we need a decomposition tree minimizing the number of equivalence classes over each cut defined by the tree. This minimum value is given by the boolean-width of the graph.

Definition 3 (Boolean-width). *The cut-bool $: 2^{V(G)} \to \mathbb{R}$ function of a graph G is defined as*

$$\textit{cut-bool}(A) = \log_2 |\{S \subseteq \overline{A} : \ \exists X \subseteq A \wedge \ S = \overline{A} \cap \bigcup_{x \in X} N(x)\}|$$

It is known from boolean matrix theory that cut-bool is symmetric [17, Theorem 1.2.3]. Using Definition 1 with $f = $ cut-bool we define the boolean-width of a decomposition tree, denoted $boolw(T, \delta)$, and the boolean-width of a graph, denoted $boolw(G)$.

Note that we take the logarithm base 2 of the number of equivalence classes simply to ensure that $0 \leq boolw(G) \leq |V(G)|$, which will ease the comparison of boolean-width to other parameters. For a vertex subset A, the value of $\textit{cut-bool}(A)$ can also be seen as the logarithm in base 2 of the number of pairwise different vectors that are spanned, via Boolean sums (1+1=1), by the rows of the $A \times \overline{A}$ sub-matrix of the adjacency matrix of G.

3 Values of boolean-width compared to other graph width parameters

Due to lack of space, all proofs of this section have been moved to the appendix, except for Lemma 2, whose proof is straightforward.

In this section we compare boolean-width to tree-width tw, branch-width bw, clique-width cw and rank-width rw. For any graph, it holds that the rankwidth of the graph is essentially the smallest parameter among the four [22, 23, 25]: $rw \leq cw$ and $rw \leq bw \leq tw+1$ (unless $bw = 0$ and $rw = 1$). Accordingly, we focus on comparing boolean-width to rankwidth, and prove that $\log rw \leq boolw \leq \frac{1}{4}rw^2 + \frac{5}{4}rw + \log rw$ with the lower bound being tight to a constant multiplicative factor. We also know that the boolean-width of

[1] Minimum Dominating Set is the main example of this paper and solving it by divide-and-conquer is indeed more complicated than solving Maximum Independent Set. Nevertheless, the runtime of our algorithm for Minimum Dominating Set, after employing several tricks, will in fact have a runtime matching what we could get for Maximum Independent Set.

a graph is never larger than its tree-width plus one [1]. Furthermore, we also prove that $\log cw - 1 \leq boolw \leq cw$ with both bounds being tight to a constant multiplicative factor.

Rank-width was introduced in [21, 23] based on the $cut\text{-}rank : 2^{V(G)} \to \mathbb{N}$ function of a graph G, which is the rank over $GF[2]$ of the submatrix of the adjacency matrix of G having rows A and columns $\overline{A}$. To see the connection with boolean-width note that

$$cut\text{-}rank(A) = \log_2 |\{Y \subseteq \overline{A} : \ \exists X \subseteq A \wedge \ Y = \overline{A} \cap \bigwedge_{x \in X} N(x)\}|$$

Here $\bigwedge$ is the symmetric difference operator. Note that *cut-rank* is a symmetric function having integer values. Using Definition 1 with $f = cut\text{-}rank$ will define the rankwidth of a decomposition tree, denoted $rw(T, \delta)$, and the rankwidth of a graph, denoted $rw(G)$. We first investigate the relationship between the *cut-bool* and the *cut-rank* functions.

Lemma 1. *[5] Let G be a graph and $A \subseteq V(G)$. Let $nss(A)$ be the number of spaces that are $GF[2]$-spanned by the rows (resp. columns) of the $A \times V(G) \setminus A$ submatrix of the adjacency matrix of G. Then, $\log cut\text{-}rank(A) \leq cut\text{-}bool(A) \leq \log nss(A)$. Moreover, it is well-known from linear algebra that $nss(A) \leq 2^{\frac{1}{4}cut\text{-}rank(A)^2 + \frac{5}{4}cut\text{-}rank(A)} cut\text{-}rank(A)$.*

This lemma can be derived from a reformulation of [5, Proposition 3.6], and a complete proof using the new terminology is also recalled in the appendix. We now prove that both bounds given in this lemma are tight. For the lower bound we recall the graphs used in the definition of Hsu's generalized join [15]. For all $k \geq 1$, the graph H_k is defined as the bipartite graph having color classes $A(H_k) = \{a_1, a_2, \ldots, a_{k+1}\}$ and $B(H_k) = \{b_1, b_2, \ldots, b_{k+1}\}$ such that $N(a_1) = \emptyset$ and $N(a_i) = \{b_1, b_2, \ldots, b_{i-1}\}$ for all $i \geq 2$ (an illustration is given in Figure 4 of the appendix). In these graphs, a union of neighborhoods of vertices of $A(H_k)$ is always of the form $\{b_1, b_2, \ldots, b_l\}$ with $1 \leq l \leq k$, hence,

Lemma 2. *For the above defined graph H_k, it holds that $cut\text{-}bool(A(H_k)) = \log k$ and $cut\text{-}rank(A(H_k)) = k$.*

For the tightness of the upper bound of Lemma 1 we now recall the graphs used in the characterization of rank-width given in [5]. For all $k \geq 1$, we denote by $[\![1, k]\!] = \{1, 2, \ldots, k\}$. The graph R_k is defined as a bipartite graph having color classes $A(R_k) = \{a_S, \ S \subseteq [\![1, k]\!]\}$ and $B(R_k) = \{b_S, \ S \subseteq [\![1, k]\!]\}$ such that a_S and b_T are adjacent if and only if $|S \cap T|$ is odd.

Lemma 3. *For the above defined graph R_k, it holds that $cut\text{-}bool(A(R_k)) = \log nss(A(R_k))$ and $cut\text{-}rank(A(R_k)) = k$.*

Since Lemma 1 holds for all edges of all decomposition trees, it is clear that for all graphs G we have $\log rw(G) \leq boolw(G) \leq \frac{1}{4}rw(G)^2 + \frac{5}{4}rw(G) + \log rw(G)$. We now address the tightness of this lower bound. We say that a cut $\{A, \overline{A}\}$ is *balanced* if $\frac{1}{3}|V(G)| \leq |A| \leq \frac{2}{3}|V(G)|$. In any decomposition tree of G, there always exists an edge of the tree which induces a balanced cut in the graph. We lift the tightness result on graph cuts given by Lemma 2 to the level of graph parameters in a standard way, by using the structure of a grid. The main idea is that any balanced cut of a grid will contain either a large part of some column of the grid, or it contains a large enough matching. We then add edges to the columns of the grid and fill each of them into a Hsu graph. The formal definition is given below while an illustration is given in Figure 4 of the appendix. Note that graphs with a similar definition have also been studied in relation with clique-width in a different context [2].

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 of 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.*

Lemma 4. *For large enough integers p and q, we have that $boolw(HG_{p,q}) \leq \min(2\log p, q)$ 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 optimal rank decomposition of $HG_{p,q}$ has boolean-width at least $\lfloor \frac{q}{6} \rfloor$.*

Notice that not only the lemma addresses the tightness of the lower bound on boolean-width as a function of rank-width, but also the additional stronger property that for a special class of Hsu-grids *any* optimal rank decomposition has boolean-width exponential in the optimal boolean-width.

Theorem 1. *For any decomposition tree (T, δ) and any graph G it holds that $\log rw(T,\delta) \leq boolw(T,\delta) \leq \frac{1}{4}rw(T,\delta)^2 + \frac{5}{4}rw(T,\delta) + \log rw(T,\delta)$ and $\log rw(G) \leq boolw(G) \leq \frac{1}{4}rw(G)^2 + \frac{5}{4}rw(G) + \log rw(G)$. Moreover, 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 $boolw(U_k) \geq \lfloor \frac{1}{6}rw(U_k) \rfloor - 1$.*

Remark 1. The inequalities about L_k is a direct application of Lemma 4 for well-chosen values of p and q. The graph U_k are standard $k \times k$ grids. We leave open the question whether $boolw(G) = O(rw(G))$.

Remark 2. Let (T, δ) and (T', δ') be such that $rw(G) = rw(T, \delta)$ and $OPT = boolw(G) = boolw(T', \delta')$. We then have from Theorem 1 that $boolw(T, \delta) \leq rw(T, \delta)^2 \leq rw(T', \delta')^2 \leq (2^{OPT})^2$. Hence, any optimal rank-width decomposition of G is also a 2^{2*OPT}-approximation of an optimal boolean-width decomposition of G. There is an FPT algorithm to compute an optimal rank-width decomposition of G in $O(f(rw(G)) \times |V(G)|^3)$ time [13].

One of the most important applications of rank-width is to approximate the clique-width $cw(G)$ of a graph by $\log(cw(G)+1) - 1 \leq rw(G) \leq cw(G)$ [23]. Although we have seen that the difference between rank-width and boolean-width can be quite large, we remark that, w.r.t. clique-width, boolean-width behaves similarly as rank-width, namely

Theorem 2. *For any graph G it holds that $\log cw(G) - 1 \leq boolw(G) \leq cw(G)$. Moreover, for every integer k higher than some constant, there are graphs L_k and U_k of clique-width at least k such that $boolw(L_k) \leq 2\log cw(L_k) + 4$ and $boolw(U_k) \geq \lfloor \frac{1}{6}cw(U_k) \rfloor - 1$.*

4 Algorithms

Given a decomposition tree (T, δ) of a graph G we will in this section show how to solve a problem on G by a divide-and-conquer (or dynamic programming) approach. We subdivide an arbitrary edge of T to get a new root node r, denoting by T_r the resulting rooted tree, and let the algorithm follow a bottom-up traversal of T_r. With each node w of T_r we associate a table data structure Tab_w, that will store optimal solutions to subproblems related to the cut $\{A, \overline{A}\}$ given by the edge between w and its parent. In Subsection 4.2 we will define the tables used and in particular give the details of the combine step. For the moment it suffices to say that the table indices will be related to the classes of the equivalence relation $\equiv_A$ of Definition 2. Firstly, in Subsection 4.1 we show how to enhance the decomposition tree with information needed to handle these equivalence classes.

4.1 Computing representatives

We assume a total ordering on the vertex set of G which stays the same throughout the whole paper. If vertex u comes before vertex v in the ordering then we say u is *smaller* than v. Using this ordering we also denote that a vertex set X is lexicographically smaller than vertex set Y by $X \leq_{lex} Y$. Let $\{A, \overline{A}\}$ be a cut given by an edge of the decomposition tree. For each equivalence class of $\equiv_A$ we want to choose one vertex subset as a representative for that class. The representative set for a class will be the lexicographically smallest among the sets in the class with minimum cardinality. More formally we define for $A \subseteq V(G)$ the list LR_A of all representatives of $\equiv_A$.

Definition 5 (List of Representatives). *Given a graph G and $A \subseteq V(G)$ we define the list LR_A of representatives of $\equiv_A$ as the unique set of subsets of A satisfying:*

1) $\forall X \subseteq A, \exists R \in LR_A : R \equiv_A X$
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* $R \leq_{lex} X$

Note that such a list will contain exactly one element for each equivalence class of $\equiv_A$.

Lemma 5. *Let G be a graph and $A \subseteq V(G)$. Let R be an element of the list LR_A of representatives of $\equiv_A$, then for any $X, Y \subseteq R$ s.t. $X \neq Y$, we have $X \not\equiv_A Y$.*

Proof Assume for contradiction $\exists X, Y \subseteq R : X \neq Y$ and $X \equiv_A Y$. Without loss of generality we can assume that $\exists v \in X : v \notin Y$. Since $X \equiv_A Y$ we have $N(v) \cap \overline{A} \subseteq N(Y) \cap \overline{A}$. Hence we get $N(R \setminus \{v\}) \cap \overline{A} = N(R) \cap \overline{A}$, contradicting the fact that R is a set in its equivalence class with minimum cardinality. □

Corollary 1. *Given a graph G and $A \subseteq V(G)$, every element R of the list LR_A of representatives of $\equiv_A$ satisfies $|R| \leq$ cut-bool(A).*

Proof There are $2^{|R|}$ subsets of R, and by Lemma 5 they are all mutually non equivalent. This means they all have different neighbourhoods in $\overline{A}$. From Definition 3 we know that there are at most $2^{\text{cut-bool}(A)}$ neighbourhoods in $\overline{A}$ hence the corollary follows. □

We now describe an algorithm to compute LR_A. It will at the same time compute a list LNR_A containing $N(R) \cap \overline{A}$, for every element R of LR_A. These two lists will be linked together, in such a way that given an element N of LNR_A we can access in constant time the element R of LR_A such that $N = N(R) \cap \overline{A}$, and vice versa. To do this in time depending only on cut-bool(A) we will need the notion of twin classes.

Definition 6. *Let G be a graph and let $A \subseteq V(G)$ be a vertex subset. A subset $X \subseteq A$ is a* twin set of A *if, for every $z \in \overline{A}$ and pair of vertices $x, y \in X$, we have x adjacent to z if and only if y adjacent to z. A twin set X is a* twin class of A *if X is a maximal twin set. The set of all twin classes of A forms a partition of A, that we call the* twin class partition of A. *We denote by TC_A the set containing for each twin class of A the smallest vertex of the class.*

Note that u and v belong to the same twin class of A if and only if $\{u\} \equiv_A \{v\}$. One consequence is that $|TC_A| \leq 2^{cut\text{-}bool(A)}$. Our algorithm will handle the edges crossing a cut $\{A, \overline{A}\}$ by using the two vertex sets TC_A and $TC_{\overline{A}}$. As a pre-processing step, we will compute TC_A and $TC_{\overline{A}}$ associated to every $A \subseteq V(G)$ that will be needed in our principal dynamic programming algorithm as specified in the lemma below, whose proof is given in the appendix.

Lemma 6. *Let G be a graph and (T, δ) a decomposition tree of G. Then, in $O(n(n + 2^{2boolw(T,\delta)}))$ global runtime we can compute, for every edge uv of T the two vertex sets TC_A and $TC_{\overline{A}}$ for $\{A, \overline{A}\}$ being the 2-partition of $V(G)$ induced by the leaves of the trees we get by removing uv from T. In the same runtime, for every $v \in A$, resp. $\overline{A}$, we compute a pointer to the vertex u in TC_A, resp. $TC_{\overline{A}}$, such that u and v are in the same twin class of A, resp. $\overline{A}$.*

We now focus on a particular cut $\{A, \overline{A}\}$, induced by some edge of the decomposition tree of G. Our algorithms will use the bipartite graph H_A with color-classes TC_A and $TC_{\overline{A}}$ and containing all edges of G crossing the cut $\{A, \overline{A}\}$. The graph H_A can be built in $O(|TC_A| \times |TC_{\overline{A}}|) = O(2^{2cut\text{-}bool(A)})$ time.

Lemma 7. *Given H_A as defined above for any $A \subseteq V(G)$, we can in time $O(2^{3cut\text{-}bool(A)} cut\text{-}bool(A))$ compute the list of representatives LR_A and the sorted list LNR_A of neighborhoods of elements of LR_A.*

Proof We describe the algorithm. The lists LR_A and LNR_A are initially empty. We will use auxiliary lists $NextLevel$, initially empty, and $LastLevel$ which initially will contain the empty set as its single element. We then run the following nested loops.

```
while LastLevel != ∅ do
  for R in LastLevel do
    for every vertex v of TC_A do
      R' = R ∪ {v}
      compute N' = N_{H_A}(R')
      if R' ≢_A R and N' is not contained in LNR_A then
        add R' to LR_A and NextLevel, and add N' to LNR_A at proper position
      end if
    end for
  end for
  set LastLevel = NextLevel, and NextLevel = ∅
end while
```

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

Algorithm 1 Initialize datastructure used for finding representative R of $[X]_{\equiv_A}$

INPUT: Lists LR_A and LNR_A and bipartite graph H_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 neighbourhood $N_{H_A}(R')$ in LNR_A
    add a pointer from $M[R][v]$ to R_U
  end for
end for
OUTPUT: M

We now argue for the runtime. Let $k = \textit{cut-bool}(A)$. The three loops loop once for each pair of element R (of TC_A) and vertex v (of TC_A). The number of representatives are exactly 2^k, while $|TC_A| \leq 2^k$, hence at most $O(2^{2k})$ iterations in total. Inside the innermost for-loop we need to calculate the neighbourhood of R', from Corollary 1 we get $|R'| \leq k+1$. Since no node in H_A have degree more than 2^k we can find $N_{H_A}(R')$ in $O(k2^k)$ time. Then to see if $R' \equiv_A R$ we compare the two neighbourhoods in $O(2^k)$ time. Then we want to check if the neighbourhood is contained in the list LNR_A, hence we want LNR_A to be a sorted list, then searching only takes $O(k)$ steps, however for each step comparing two neighbourhoods can take $O(2^k)$ time. Inserting into the sorted list LNR_A takes $O(2^k)$, and in the other lists $O(1)$ time. This means all operations in the inner for-loop can be done in $O(k2^k)$ time, giving a total running-time of $O(k2^{3k})$. □

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 binary-search in the list LNR_A for the set $N(X) \cap \overline{A}$ in time $O(2^{\textit{cut-bool}(A)}\textit{cut-bool}(A))$, but we want to do this in $O(|X|)$ time. To accomplish this 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 $[R \cup \{v\}]_{\equiv_A}$. It will be stored as a two dimensional table, leading to a constant time lookup.

Lemma 8. *Given H_A as defined above for any $A \subseteq V(G)$, we can in time $O(2^{3\textit{cut-bool}(A)}\textit{cut-bool}(A))$ compute a datastructure allowing, for any $X \subseteq A$, to access in $O(|X|)$ time the entry R of LR_A such that $X \equiv_A R$.*

Proof Let $k = \textit{cut-bool}(A)$. First we need to initialize the datastructure used for finding representatives using Agorithm 1. It goes through 2 for-loops, in total iterating $O(2^{2k})$ times. To find the neighbourhood of R' takes $O(2^k k)$ time. To search LNR_A for the neighbourhood takes $O(2^k k)$ time. All other operations are done in constant time, thus the runtime is $O(2^{3k}k)$.

Given $X \subseteq A$ we find the representative R of $[X]_{\equiv_A}$ as follows. Initially R will be empty. Then we iterate over all elements $u \in X$, first looking up $v \in TC_A$ such that u and v belong to the same twin class of A, and then replacing R by the representative of the class $[R \cup \{v\}]_{\equiv_A}$ (as given by the auxiliary data structure). □

4.2 Dynamic programming for dominating set

This section is based on the dynamic programming scheme used in [5] to give an algorithm for Minimum Dominating Set parameterized by rankwidth. For example, Lemma 11 is an adaptation from that paper to the current formalism parameterizing by boolean-width, and its proof has been moved to the appendix.

Recall that our algorithm will follow a bottom-up traversal of the tree T_r, computing at each node w of the tree a table Tab_w, that will store optimal solutions to subproblems related to the cut $\{A, \overline{A}\}$ given by the edge between w and its parent. If we were solving Maximum Independent Set then Tab_w would simply be indexed by the equivalence classes of $\equiv_A$. However, unlike the case of independent sets we note that a set of vertices D dominating A will include also vertices of $\overline{A}$ that dominate vertices of A 'from the outside'. This motivates the following definition.

Definition 7. *Let G be a graph and $A \subseteq V(G)$. For $X \subseteq A$, $Y \subseteq \overline{A}$, if $A \backslash X \subseteq N(X \cup Y)$ we say that the pair (X, Y) dominates A.*

The main idea for dealing with this complication is to index the table at w by two sets, one that represents the equivalence class of $D \cap A$ under $\equiv_A$ that dominates 'from the inside', and one that represents the equivalence class of $D \cap \overline{A}$ under $\equiv_{\overline{A}}$ that helps dominate the rest of A 'from the outside'. The subsequent lemma should indicate why this will work.

Lemma 9. *Let G be a graph and $A \subseteq V(G)$. For $X \subseteq A$, $Y, Y' \subseteq \overline{A}$, If (X, Y) dominates A and $Y \equiv_{\overline{A}} Y'$ then (X, Y') dominates A.*

Proof Since (X, Y) dominates A we have $A \setminus X \subseteq N(X \cup Y)$. Since $Y \equiv_{\overline{A}} Y'$ we have $N(Y) \cap A = N(Y') \cap A$. Then it follows that $A \setminus X \subseteq N(X \cup Y')$, meaning (X, Y') dominates A. □

For a node w of T_r we denote by $\{A_w, \overline{A_w}\}$, the cut given by the edge between w and its parent. In the previous subsection we saw how to compute for every node w of T_r the lists LR_{A_w} of representatives of $\equiv_{A_w}$ and $LR_{\overline{A_w}}$ of representatives of $\equiv_{\overline{A_w}}$.

Definition 8. *The two-dimensional table Tab_w will have 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 contents of $Tab_w[R_w][R_{\overline{w}}]$ after updating should be:*

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

Note that the table Tab_w will have $2^{2cut\text{-}bool(A_w)}$ entries. For every node w we assume that initially every entry Tab_w is set to ∞. For a leaf l of T_r, since $A_l = \{\delta(l)\}$, note that $\equiv_{A_l}$ has only two equivalence classes: one containing $\emptyset$ and the other containing A_l. For $\overline{A_l}$, we have the same situation with only two equivalence classes: one containing $\emptyset$ and the other containing $\overline{A_l}$. Therefore, we set $Tab_l[\emptyset][\emptyset] := \infty$, and $Tab_l[\{\delta(l)\}][\emptyset] := 1$ and $Tab_l[\{\delta(l)\}][R] := 1$ and $Tab_l[\emptyset][R] := 0$ (where R is the representative of $[\overline{A_l}]_{\equiv_{\overline{A_l}}}$) since the only of the four combinations that does not dominate A_l as in Definition 7 is $(\emptyset, \emptyset)$. Note that there would be a special case if $\delta(l)$ was an isolated vertex, but isolated vertices can easily be removed.

For the updating of internal nodes we have a node w with two children a and b and can assume that the tables Tab_a and Tab_b have been correctly computed. We need to

correctly compute the value of $Tab_w[R_w][R_{\overline{w}}]$ for each $R_w \in LR_{A_w}$ and $R_{\overline{w}} \in LR_{\overline{A_w}}$. Each table can have $2^{2boolw(T,\delta)}$ entries. Therefore, the number of pairs of entries, one from each of Tab_a and Tab_b, could be as much as $2^{4boolw(T,\delta)}$. Looping over all such pairs of entries we would in fact spend time $2^{5boolw(T,\delta)}$ since we would have to compute the right entry in Tab_w. Instead we achieve $2^{3boolw(T,\delta)}$ time by looping only over one half of the entries in each of the three tables, as follows:

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

Before proving correctness of this updating we need a small lemma.

Lemma 10. *For a graph G, let A, B, W be a 3-partitioning of $V(G)$, and let $S_a \subseteq A, S_b \subseteq B$ and $S_w \subseteq W$. $(S_a, S_b \cup S_w)$ dominates A and $(S_b, S_a \cup S_w)$ dominates B iff $(S_a \cup S_b, S_w)$ dominates $A \cup B$.*

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

Lemma 11. *The table at node w is updated correctly, namely for any representative $R_w \in LR_{A_w}$ and $R_{\overline{w}} \in LR_{\overline{A_w}}$, if $Tab_w[R_w][R_{\overline{w}}]$ is not ∞ then*

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

A proof of Lemma 11 is given in appendix.

Theorem 3. *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 $O(n(n+2^{3boolw(T,\delta)}boolw(T,\delta)))$.*

Proof As a preprocessing step we compute the twin classes for all cuts induced by the edges of (T, δ) as described in Lemma 6. We then loop over all edges uv of T. Let $\{A, \overline{A}\}$ be the cut of G induced by the leaves of T after removing uv from T. We compute the graph H_A, as well as the lists LR_A, $LR_{\overline{A}}$, LNR_A, and $LNR_{\overline{A}}$ as described in Lemma 7, and also the datastructure for finding a representative of $[X]_{\equiv_A}$ and $[Y]_{\equiv_{\overline{A}}}$ as described in Lemma 8. After this loop we subdivide an arbitrary edge of T by a new root node r to get T_r. We then initialize the table Tab_l for every leaf l of T_r as described after Definition 8. Finally, we scan T_r in a bottom-up traversal and update the table Tab_w for every internal node w as described right before Lemma 10. After this, the optimum solution can be read at the (unique) entry $Tab_r[V(G)][\emptyset]$ of the table at the root of T_r.

The correctness follows from Lemma 11, when applied to $w = r$. The complexity analysis of the computation before setting the root r is a straightforward combination of those given in Lemmas 6, 7 and 8. After this, the initialization at every leaf of T_r takes $O(1)$ time. The update at every internal node w of T_r loops through $2^{3boolw(T,\delta)}$ triplets, and for each of them spend $O(boolw(T,\delta))$ time finding the three representatives and $O(1)$ time updating the value of $Tab_w[R_w][R_{\overline{w}}]$. □

Solving Maximum Independent Set (MIS) is simpler than solving Minimum Dominating Set. The table Tab_w at a node w will then be one-dimensional, indexed by the equivalence classes of $\equiv_{A_w}$, and will store the size of the maximum independent set in that class. In the combine step we loop over all pairs of representatives R_a from Tab_a and R_b from Tab_b and check if there are any edges between R_a and R_b. If not, then we look up the representative R_w of $[R_a \cup R_b]_{\equiv_{A_w}}$ and update $Tab_w[R_w]$ by the maximum of its old value and $Tab_a[R_a] + Tab_b[R_b]$. Combining these ideas we can solve both the Minimum and Maximum Independent Dominating Set problems. The runtimes will be dominated by the computation of representatives.

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

5 Conclusion and Perspectives

There are many questions about boolean-width left unanswered. The foremost concerns possibly its practical applicability. The divide-and-conquer algorithms given here are practical and easy to implement, but we need fast and good heuristics computing decomposition trees of low boolean-width. Research in this direction is underway.

On the theoretical side it would be nice to improve on the 2^{2*OPT}-approximation algorithm to an optimal boolean-width decomposition (c.f. Remark 2) we get by applying the algorithm computing an optimal rank-width decomposition [13]. Note that the runtime of that approximation algorithm is FPT when parameterized by boolean-width. The best we can hope for is an FPT algorithm computing a decomposition of optimal boolean-width, but any polynomial approximation would be nice.

When parameterizing by boolean-width we get FPT algorithms for any problem expressible in $MSOL_1$ [7]. For such problems we should carefully design algorithms as in this paper and [3] that aim for the lowest possible runtime dependency on boolean-width. The runtime of the algorithms in [3] can be greatly improved by a positive answer to the following question: first generalize the concept of Boolean sums (1+1=1) to d-Boolean-sums ($i + j = \min(i + j, d)$). For a Boolean matrix A let $R_d(A)$ be the set of vectors over $\{0, 1, ..., d\}$ that arise from all possible d-Boolean sums of rows of A. Is there a function f such that $|R_d(A)| \leq |R_1(A)|^{f(d)}$ (alternatively such that $|R_d(A)| \leq |R_1(A)|^{f(d) \log\log |R_1(A)|}$)?

The graphs of boolean-width at most one are exactly the graphs of rank-width one, *i.e.* the distance-hereditary graphs. What about the graphs of boolean-width at most two, do they also have a nice characterization? Is there a polynomial-time algorithm to decide if a graph has boolean-width at most two? More generally, is there an alternative characterization of the graphs of boolean-width at most k?

We do not know if the bound $boolw(G) \leq \frac{1}{4} rw(G)^2 + \frac{5}{4} rw(G) + \log rw(G)$ is tight to a multiplicative factor. For most well-known classes we should have $boolw = O(rw)$, but this needs to be investigated. Are there well-known graph classes where $boolw = O(\log rw)$? It has been shown that a $k \times k$ grid has rank-width k [16], and we have seen that its boolean-width lies between $\frac{1}{6}k$ (proof Theorem 1) and $k + 1$ (derived from its clique-width). What is the right value? All these questions should benefit from the connections between boolean-width and the field of Boolean matrix theory.

References

1. I. Adler, B.-M. Bui-Xuan, G. Renault, and M. Vatshelle. Boolean-width is less than or equal to branch-width. *manuscript in preparation*.
2. A. Brandstaedt and V. V. Lozin. On the linear structure and clique-width of bipartite permutation graphs. *Ars Combinatoria*, 67:719–734, 2003.
3. B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Fast algorithms for vertex subset and vertex partitioning problems on graphs of low boolean-width. *submitted manuscript*. `http://www.ii.uib.no/~telle/bib/BTV09II.pdf`.
4. B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Feedback vertex set on graphs of low cliquewidth. *to appear in Proceedings IWOCA 2009*. `http://www.ii.uib.no/~telle/bib/BTViwoca.pdf`.
5. B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *to appear in DAM: special issue of GROW*. `http://www.ii.uib.no/~telle/bib/BTV.pdf`.
6. D. Corneil and U. Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.
7. 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.
8. 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.
9. F. Dorn. Dynamic programming and fast matrix multiplication. In *14th Annual European Symposium (ESA'05)*, volume 4168 of *LNCS*, pages 280–291, 2006.
10. R. Ganian and P. Hliněný. On Parse Trees and Myhill-Nerode-type Tools for handling Graphs of Bounded Rank-width. *submitted manuscript*. `http://www.fi.muni.cz/~hlineny/Research/papers/MNtools-2.pdf`.
11. 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.
12. 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.
13. P. Hliněný and S. Oum. Finding branch-decompositions and rank-decompositions. *SIAM Journal on Computing*, 38(3):1012–1032, 2008. Abstract at *ESA'07*.
14. 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.
15. W.-L. Hsu. Decomposition of perfect graphs. *Journal of Combinatorial Theory, Series B*, 43(1):70–94, 1987.
16. 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.
17. K. H. Kim. *Boolean matrix theory and its applications*. Marcel Dekker, 1982.
18. 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*.
19. Hans L.Bodlaender and Arie M.C.A.Koster. Treewidth Computations I Upper Bounds. Technical Report UU-CS-2008-032, Department of Information and Computing Sciences, Utrecht University, 2008.
20. H. X. Nguyen and P. Thiran. Active measurement for multiple link failures diagnosis in IP networks. In *5th Passive and Active Measurement Workshop (PAM'04)*, volume 3015 of *LNCS*, pages 185–194, 2004.
21. S. Oum. *Graphs of Bounded Rank-width*. PhD thesis, Princeton University, 2005.
22. S. Oum. Rank-width is less than or equal to branch-width. *Journal of Graph Theory*, 57(3):239–244, 2008.
23. S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.
24. P. Pattison and R. Breiger. Lattices and dimensional representations: matrix decompositions and ordering structures. *Social Networks*, 24(4):423–444, 2002.
25. N. Robertson and P. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.
26. 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)*, 2009. *to appear*.

Appendix

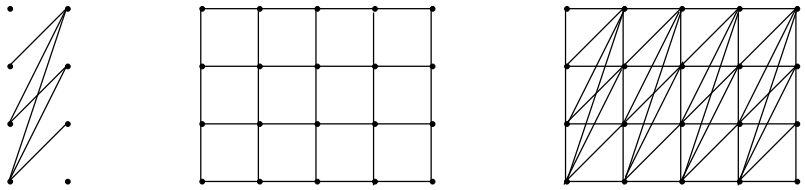

Fig. 4. The Hsu's graph H_3, the 4×5 grid, and the Hsu-grid $HG_{4,5}$.

Proof of Lemma 1.
This proof is an adaptation of the one given in [5, Proposition 3.6] to the terminology used in this paper. Let M be the $A \times V(G) \setminus A$ submatrix of the adjacency matrix of G. Let $\{a_1, a_2, \ldots, a_{\textit{cut-rank}(A)}\}$ be a set of vertices of A whose corresponding rows in M define a basis of M. Then, it is clear from definition that $N(a_1)$, $N(a_2)$, ..., $N(a_{\textit{cut-rank}(A)})$ are pairwise distinct, and from that the first inequality follows, namely $\textit{cut-rank}(A) \leq 2^{\textit{cut-bool}(A)}$.

Now let $X \subseteq A$. We define $R_X \subseteq A$ by the following algorithm.

Initialize $R_X \leftarrow \emptyset$ and $S \leftarrow \emptyset$
For every vertex $v \in A$ do:
  Let $T = N(v) \setminus A$
  If $T \subseteq N(X)$ and $T \setminus S \neq \emptyset$ then add v to R_X and add all vertices in T to S.

Then, the rows in M which correspond to the vertices of R_X are $GF[2]$-independent. Hence, every union of neighbourhood of vertices of A can be associated with (at least) one space that is $GF[2]$-spanned by some rows of M. This implies $2^{\textit{cut-bool}(A)} \leq nss_G(A)$.

For the last inequality, we notice that

$$nss_G(A) \leq \sum_{i=1}^{\textit{cut-rank}(A)} \binom{\textit{cut-rank}(A)}{i}_2, \text{ where } \binom{n}{m}_q = \prod_{i=1}^{m} \frac{1-q^{n-i+1}}{1-q^i}.$$

This is because $\binom{n}{m}_q$, which is known under the name of the q-binomial coefficient of n and m, is exactly the number of different subspaces of dimension m of a given space of dimension n over a finite field of q elements (roughly, $\frac{1-q^{n-i+1}}{1-q^i}$ is the number of choices of an i^{th} vector that is linearly independent from the previously chosen ones). Now let $a(\textit{cut-rank}(A)) = \sum_{i=1}^{\textit{cut-rank}(A)} \binom{\textit{cut-rank}(A)}{i}_2$. In order to conclude we can use the $q-$analog of Pascal triangles: $\binom{n}{m}_q = 2^m \binom{n-1}{m}_q + \binom{n-1}{m-1}_q$, for all $m \leq n$, with the convention that $\binom{n}{m}_q = 0$ if $m < 0$ or $m > n$. From this we firstly have that the highest number among $\binom{n}{m}_q$, for all $0 \leq m \leq n$, is when $m = \lceil \frac{n}{2} \rceil$. Therefore, $a(n) \leq n \times b(n)$ with $b(n) = \binom{n}{\lceil \frac{n}{2} \rceil}_q$. Finally, still using the q-analog of Pascal triangles, one can check that $b(n) \leq \left(2^{\lceil \frac{n}{2} \rceil} + 1\right) \times b(n-1) \leq 2^{\frac{1}{4}n^2 + \frac{5}{4}n}$. □

Proof of Lemma 3.

For convenience, we begin with a useful claim.

Claim 3.1. *Let $k \geq 1$ be an integer. We keep the notations used in the definition of R_k. Let $X \subseteq \{a_S,\ S \subseteq [\![1,k]\!]\}$ be such that $a_S, a_T \in X \Rightarrow a_{S \Delta T} \in X$. Then, we have $\forall S \subseteq [\![1,k]\!],\ \ N(a_S) \subseteq N(X) \Rightarrow a_S \in X$.*

Proof. Let $M(R_k)$ denote the bipartite adjacency matrix of R_k. It is a straightforward exercise to check that the rows in $M(R_k)$ which correspond to the vertices of X form a $GF[2]$-space of dimension at most k (and also hereafter we will always assume that this exercise is implicit whenever such a vertex subset X is involved). Besides, in general, a vertex subset X leads to a subspace equal to the closure under Δ (symmetric difference) of the neighborhoods of vertices in X, i.e. of the rows in $M(R_k)$ corresponding to X. For R_k, as opposed for example to Hsu's graph H_k, this closure of neighborhoods never goes outside the neighborhoods of R_k, because of the following fact, whose proof is a straightforward parity check.

Fact: $N(a_S) \Delta N(a_T) = N(a_{S \Delta T})$

We denote the dimension of X by $dim(X)$, and prove the claim by induction on $p = dim(X)$. If $p = 1$ then X contains only one vertex, say $X = \{a_T\}$. If $S \setminus T \neq \emptyset$, then any element $i \in S \setminus T$ would lead to $b_{\{i\}}$ being a neighbour of a_S but not a_T, contradicting that $N(a_S) \subseteq N(X)$. If $S \subsetneq T$, then any pair (i,j) with $i \in S$ and $j \in T \setminus S$ would lead to $b_{\{i,j\}}$ being a neighbour of a_S but not a_T, contradicting that $N(a_S) \subseteq N(X)$. Hence, $S = T$, which in particular means $a_S \in X$.

We now assume that the claim is true upto dimension $p - 1 \geq 1$, and consider X such that $dim(X) = p$. In particular X contains at least two vertices. Let a_T be a vertex in X such that $a_S \neq a_T$. 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 $j \in T$. In any case, we have that $b_W \in N(a_T) \setminus N(a_S)$. Let $X' = \{a_Z \in X,\ b_W \notin N(a_Z)\}$. We will prove that $N(a_S) \subseteq N(X')$. Indeed, if $b_Z \in N(a_S) \setminus N(X')$, then $b_{W \Delta Z} \in N(a_S) \setminus N(X')$. But then, pick any basis in the subspace corresponding to X which does not contain the row corresponding to a_T, and express this row as a sum of vectors in the basis. Let p be the number of vectors among these which correspond to vertices in $X \setminus X'$. By checking the column corresponding to b_W we deduce that p is odd. By checking the two columns corresponding to b_Z and $b_{W \Delta Z}$ we deduce that p is even (sum of two odd numbers, which both exist since $N(a_S) \subseteq N(X)$). This is a contradiction and thus $N(a_S) \subseteq N(X')$. Finally, we also have that $a_Z, a_{Z'} \in X' \Rightarrow a_{Z \Delta Z'} \in X'$. Indeed, since $X' \subseteq X$, we have that $a_{Z \Delta Z'} \in X$. Besides, note that $X' = \{a_Z \in X,\ |W \cap Z| \text{ is even}\}$ and it is clear that if both $|W \cap Z|$ and $|W \cap Z'|$ are even, then $|W \cap (Z \Delta Z')|$ is even. Now, we can conclude by applying the inductive hypothesis on X', which is of dimension lesser than $p - 1$. □

We now prove the lemma by claiming a stronger fact. Let any subset Y of vertices of $B(R_k)$ be associated with a subset $f(Y)$ of vertices of $A(R_k)$ as $f(Y) = \{a_S,\ b_S \notin Y\}$. Such a set $f(Y)$ can also be seen as a set of rows of $M(R_k)$, the bipartite adjacency matrix of R_k. Then, f is a bijection from the set $UN(R_k)$ of all unions of neighborhoods of some vertices of $A(R_k)$ to the set of all vector spaces spanned by some rows of $M(R_k)$.

Indeed, f is a well-defined function over the subsets of vertices of $B(R_k)$. It is also clear that f is injective. We first look at the image of $UN(R_k)$ by f. Let $Y \in UN(R_k)$ and let $X \subseteq \{a_S,\ S \subseteq [\![1,k]\!]\}$ be such that $Y = N(X)$. Let $a_S, a_T \in f(Y)$. By definition,

neither b_S nor b_T belong to Y. 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, $a_{S\Delta T} \in f(Y)$. In other words, the rows of $M(R_k)$ which correspond to the vertices of $f(Y)$ form a vector space over $GF[2]$.

In order to conclude, we only need to prove that, for every $X \subseteq \{a_S,\ S \subseteq [\![1,k]\!]\}$ such that $a_S, a_T \in X \Rightarrow a_{S\Delta T} \in X$, there exists $Y \in UN(R_k)$ such that $f(Y) = X$. Let us consider such a set X, and define $Y = \{b_S,\ a_S \notin X\}$. It is clear that $f(Y) = X$, and so the only thing left to prove is that $Y \in UN(R_k)$. More precisely, let $X' = f(N(X))$, we will prove that $Y = N(X')$.

- Let $b_S \in N(X')$. Then, there exists $a_T \in X'$ such that $|S \cap T|$ is odd. By definition of $X' = f(N(X))$, if $a_T \in X'$ then we know that $b_T \notin N(X)$. Since $S \cap T$ is odd (hence a_S and b_T 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. Applying the definition of $X' = f(N(X))$, we obtain that, for every $b_T \notin N(X)$, $|S \cap T|$ is even. In other words, for every $b_T \notin N(X)$, $b_T \notin N(a_S)$. Therefore, $N(a_S) \subseteq N(X)$. The above Claim 3.1 then applies and yields $a_S \in X$. This, by definition of Y, means $b_S \notin Y$. Hence, $Y \subseteq N(X')$.

□

Proof of Lemma 4.

Let m/n denote $\lfloor \frac{m}{n} \rfloor$. We begin with a useful claim.

Claim 4.1. *Let $p \geq 2$ and $q \geq 2$. Let $\{A, B\}$ be a balanced cut of the Hsu-grid $HG_{p,q}$, and let H be the bipartite graph containing all edges of $HG_{p,q}$ crossing $\{A, B\}$. Then, either the cut-rank of A is at least $p/4$, or H 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 $\{A, B\}$
- 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 $v_{i,j+1} \in B$. Therefore, there are at least $p/4$ row indices i's for which there exists j such that $v_{i,j} \in A$ and $v_{i,j+1} \in B$ 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 H 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 $\{A, B\}$ we have that there are at least $q/3$ columns each containing at least one vertex of B. Then, for each such column j we can find an edge $(v_{i,j}, v_{i+1,j})$ crossing $\{A, B\}$. Choosing one such edge every two columns will lead to a $q/6$ matching that is an induced subgraph of H. □

To prove the lemma, we will focus on two types of decomposition trees. A star is a tree having one and only one internal node, the so-called center of the star. The 2-leaf comb is the 2-leaf star. The n-leaf comb is obtained by appending one leaf of the 2-leaf comb to the center of an $(n-1)$-leaf comb; the center of the n-leaf comb is the center of the

former comb. In a tree, *stretching* an internal node n is the action of adding a new node n' adjacent to n, and removing at least one neighbour of old node n to be neighbour of new node n'.

The vertical super-decomposition tree of the Hsu-grid $HG_{p,q}$ is obtained by appending every leaf of a q-leaf comb to the center of a new p-leaf star, unrooting the obtained graph, and subsequently mapping the j^{th} leaf set of the q different p-leaf stars to the set of vertices of $HG_{p,q}$ standing on the j^{th} column in $V(HG_{p,q})$. A *vertical decomposition tree* is obtained by consecutively stretching all internal nodes of degree more than 3 in the vertical super-decomposition tree. The notion of a *horizontal (super-)decomposition tree* is defined similarly when swapping the roles of p and q, and that of columns and rows. Notice from $p \geq 3$ and $q \geq 3$ that both vertical and horizontal decomposition tree are indeed decomposition tree (in particular their underlying trees are subcubic).

We now come to the actual proof of the lemma. It is straightforward to check that the boolean-width of any vertical decomposition tree of $HG_{p,q}$ is at most $2\log p$ and the boolean-width of any horizontal decomposition tree of $HG_{p,q}$ is at most q. Therefore, $boolw(HG_{p,q}) \leq \min(2\log p, q)$. Besides, it follows directly from the above Claim 4.1 that $rw(HG_{p,q}) \geq \min(p/4, q/6)$.

To prove the third and last claim, we first notice that any horizontal decomposition tree of $HG_{p,q}$ has rankwidth $2q$, and therefore the rank-width of $HG_{p,q}$ is at most $2q < p/4$. We now consider an optimal rank decomposition of $HG_{p,q}$. Let uv be an edge of the decomposition which induces a balanced cut $\{A, B\}$ in $HG_{p,q}$. Let H be the bipartite graph containing all edges of $HG_{p,q}$ crossing $\{A, B\}$. From the above Claim 4.1 and the fact that the rank-width of $HG_{p,q}$ is at most $2q < p/4$, H has a $q/6$-matching as induced subgraph. Therefore, the boolean-cut of A is at least $q/6$, and the boolean-width of this optimal rank decomposition is at least $q/6$. □

Proof of Theorem 1.

The bounds for both decomposition trees and graphs follow directly from Lemma 1. We define L_k as a Hsu-grid $HG_{p,q}$ such that $k \leq p/4 \leq q/6$ and $2\log p \leq q$. Then, from Lemma 4, we have that $rw(L_k) \geq p/4 \geq k$ and $boolw(L_k) \leq 2\log p$, which allows to conclude about L_k. Finally, we define U_k to be the grid of dimension $k \times k$. It is a standard exercise to check that the rank-width of U_k is at most $k + 1$. The same idea as in the proof of Claim 4.1 (inside the proof of Lemma 4) can be used to prove that $boolw(U_k) \geq k/6$. □

Proof of Theorem 2.

We first recall the definition of clique-width. Using abusive notations, we confound graphs and vertex-labelled graphs (for more concision refer to [7]). The *clique-width* of a graph G is the minimum number of labels needed to construct G using the following four operations:

- Creation of a new vertex v with label i (denoted by $i(v)$).
- Disjoint union of two labeled graphs G and H (denoted by $G \oplus H$).
- Joining by an edge each vertex with label i to each vertex with label j ($i \neq j$, denoted by $\eta_{i,j}$).
- Renaming label i to j (denoted by $\rho_{i \to j}$).

The construction tree of G from these operations is called an k*-expression* of G, where k is the number of distinct labels needed in the construction.

We continue with a useful definition. Let $A \subseteq V(G)$ be a vertex subset of G. An *external module partition* of A is a partition P of A such that, for every $z \in V(G) \setminus A$ and pair of vertices x, y belonging to the same class in P, we have x adjacent to z if and only if y adjacent to z. For any A, an maximum (coarse-wise) external module partition of A always exists [5, Lemma 3.2].

To prove the lower bound of the theorem, we consider an optimal boolean-width decomposition of G, subdivide any edge and obtain a rooted binary tree. At any internal node w of the tree with children a and b, we denote the vertex subsets induced by the leaves of the subtree rooted at w, a and b by V_w, V_a and V_b, respectively. We also denote the number of parts in the maximum external module partition of V_w, V_a and V_b by k_w, k_a and k_b, respectively. Clearly, if $G[V_a]$ and $G[V_b]$ have k_a-expression and k_b-expression, respectively, then $G[V_w]$ has a $(k_a + k_b)$-expression. Besides, it follows directly from definition of the boolean-cut function that $k_a \leq 2^{cut\text{-}bool(V_a)}$ and $k_b \leq 2^{cut\text{-}bool(V_b)}$. From those observations we can deduce by a standard induction that there exists a $2^{boolw(G)+1}$-expression of G.

To prove the upper bound, we consider an k-expression of G of optimal clique-width, and contract all internal nodes of degree 2 (corresponding to the $\rho_{i\to j}$ operations) in order to obtain a rooted binary tree T. Now, at any edge uv of the tree, with u being the parent of v, the number of parts in the maximum external module partition of V_v is no more than k, where V_v is the vertex subset induced by the leaves of the subtree rooted at v. Hence, $cut\text{-}bool(V_v) \leq k$. Whence, unrooting T results in a decomposition tree having a boolean-width at most k.

We now use the same graphs L_k and U_k as in the proof of Theorem 1 and the well-known fact that $rw(G) \leq cw(G)$ for any graph G [23] in order to conclude that $boolw(L_k) \leq 2 \log cw(L_k) + 4$. It is a standard exercise to check that the cliquewidth of U_k is at most $k + 2$. Combining this with $boolw(U_k) \geq k/6$ allows to conclude. □

Proof of Lemma 6.

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 [12], such an operation can be implemented to run in $O(|X|)$ time. If the elements of P_i are initially ordered, the refinement operations will preserve their order.

First, we subdivide an arbitrary edge of T to a new root r and obtain a rooted tree T_r. We initialize $\mathcal{P}_r = \{V(G)\}$, where the elements of $V(G)$ follow the order given at the beginning of Section 4.1. We will first perform a top-down traversal of T_r.

Let a be an internal node of T_r. Let A_a be the vertex set of G induced by the leaves of the subtrees of T_r rooted at a.

Claim 6.1. ([4, Lemma 3.2]) *It is possible to compute in global $O(n^2)$ time the twin-class partition of A_a, for every node a of T_r with A_a being the vertex subset induced by the leaves of the sub-tree of T_r rooted at a.*

For this, the main idea is as follows. Let w be the parent of a in T_r. Let b be the sibling of b in T_r, namely the (unique) other node having w as parent. Let A_w and A_b be the vertex sets of G induced by the leaves of the subtrees of T_r rooted at w and b. 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, namely when w has been visited. Then, refining $\mathcal{P}_w = \{P_1 \cap A_a, P_2 \cap A_a, \ldots, P_k \cap A_a\}$ 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 $z \in A_b$). The implementation details are described in [4, 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 want for every internal node w to compute the list $TC_{\overline{A_w}}$. We describe how to compute this in $O(n2^{2boolw(T,\delta)})$ global time by a second top-down traversal of T_r. Let w be an internal node of T_r with children a and b. Let A_w, A_a and A_b be the vertex sets of G induced by the leaves of the subtrees of T_r rooted at w, a and b. Let $\overline{A_w} = V(G) \setminus A_w$, $\overline{A_a} = V(G) \setminus A_a$ and $\overline{A_b} = V(G) \setminus A_b$. We suppose that, when visiting a, the twin-class partition $\mathcal{P}_{\overline{w}}$ of $\overline{A_w}$ has already been computed (when w was visited). Besides, the twin-class partitions $\mathcal{P}_a$ and $\mathcal{P}_b$ of A_a and A_b has already been computed as describe in Claim 6.1.

Now 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 part of $\mathcal{P}_b \cup \mathcal{P}_{\overline{w}}$ containing x. We then compute H, the graph having $R_a \cup R_{\overline{a}}$ as vertex set and defined such that $xy \in E(H)$ if and only if $x \in R_a$, $y \in R_{\overline{a}}$ and $xy \in E(G)$. We now initialize $\mathcal{P}_{\overline{a}} = \{R_{\overline{a}}\}$ and, for every $z \in R_a$, refine $\mathcal{P}_{\overline{a}}$ using the neighbourhood of z in H as pivot. Finally, for every part P of $\mathcal{P}_{\overline{a}}$, we replace every element x of P by all the elements belonging to the part 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 this computation. By definition of boolean-width, we have $|R_a| \leq 2^{boolw(T,\delta)}$ and $|R_{\overline{a}}| \leq 2 \times 2^{boolw(T,\delta)}$. Accordingly, we can compute R_a, $R_{\overline{a}}$ and H in $O(2^{2boolw(T,\delta)})$ time (we just compute H brute-force by adjacency check from the list of vertices of 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 $O(2^{2boolw(T,\delta)})$ (cf. [12] for further implementations details). 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 $O(2^{2boolw(T,\delta)})$, leading to an $O(n2^{2boolw(T,\delta)})$ computation for every node a of T_r. □

Proof of Lemma 11.

Let a, b be the children of w in T_r, assume Tab_a and Tab_b are correct. We first show that if $Tab_w[R_w][R_{\overline{w}}] = s$ and hence not ∞, then there exist a set S_w satisfying all the conditions. For a value of Tab_w to be set, it means that an update happened in the algorithm, hence there exists R_a and R_b such that: $R_{\overline{a}}$ is the representative of $[R_b \cup R_{\overline{w}}]_{\equiv_{\overline{A_a}}}$ and $R_{\overline{b}}$ is the representative of $[R_a \cup R_{\overline{w}}]_{\equiv_{\overline{A_b}}}$ and $Tab_a[R_a][R_{\overline{a}}] + Tab_b[R_b][R_{\overline{b}}] = s$. Then we know that there exists S_a and S_b such that $(S_a, R_{\overline{a}})$ dominates A_a and $(S_b, R_{\overline{b}})$ dominates A_b and that $|S_a \cup S_b| = s$. Let $S_w = S_a \cup S_b$, then S_w fulfil the two conditions $|S| = s$ and $R_w \equiv_{A_w} S_w$, now we need to show that $(S_w, R_{\overline{w}})$ dominates A_w. Since $(S_b \cup R_{\overline{w}}) \equiv_{V_{\overline{a}}} R_{\overline{a}}$ and $(S_a, R_{\overline{a}})$ dominates A_a it follows from Lemma 9 that $(S_a, S_b \cup R_{\overline{w}})$ dominates A_a. Similarly we conclude that $(S_b, S_a \cup R_{\overline{w}})$ dominates A_b. Then from Lemma 10 we get $(S_w, R_{\overline{w}})$ dominates $A_a \cup A_b$.

Next we will for all $R_w, R_{\overline{w}}$ show that if there exists a set $S_w \equiv_{A_w} R_w$ such that $(S_w, R_{\overline{w}})$ dominates A_w then for R the representative of $[S_w]_{\equiv_{A_w}}$ we have $Tab_w[R][R_{\overline{w}}] \leq |S_w|$. Let $S_a = S_w \cap A_a$ and $S_b = S_w \cap A_b$. Since the algorithm goes through all triples of representatives, it will at some point go through $(R_a, R_b, R_{\overline{w}})$, where R_a is the representative of $[S_a]_{\equiv_{A_a}}$ and R_b is the representative of $[S_b]_{\equiv_{A_b}}$. Since $(S_w, R_{\overline{w}})$ dominates A_w, $(S_a \cup S_b, R_{\overline{w}})$ dominates A_w. Using Lemma 10 we get $(S_a, S_b \cup R_{\overline{w}})$ dominates A_a and $(S_b, S_a \cup R_{\overline{w}})$ dominates A_b. Since in the algorithm $R_{\overline{a}}$ is the representative of $[S_b \cup R_{\overline{w}}]_{\equiv_{V_{\overline{a}}}}$, using Lemma 9 we get that $(S_a, R_{\overline{a}})$ dominates A_a. Similarly we get that $(S_b, R_{\overline{b}})$ dominates A_b. This means that $Tab_a[R_a][R_{\overline{a}}] + Tab_b[R_b][R_{\overline{b}}] \leq |S_a \cup S_b| = |S_w|$, hence $Tab_w[R_w][R_{\overline{w}}] \leq |S_w|$.

By induction all tables will be correct. □