More Applications of the d-Neighbor Equivalence: Connectivity and Acyclicity Constraints

Benjamin Bergougnoux[1]
Université Paris Diderot, IRIF, CNRS, Paris, France
bergougnoux@irif.fr

Mamadou Moustapha Kanté
Université Clermont Auvergne, LIMOS, CNRS, Aubière, France
mamadou.kante@uca.fr

Abstract

In this paper, we design a framework to obtain efficient algorithms for several problems with a global constraint (acyclicity or connectivity) such as Connected Dominating Set, Node Weighted Steiner Tree, Maximum Induced Tree, Longest Induced Path, and Feedback Vertex Set. For all these problems, we obtain $2^{O(k)} \cdot n^{O(1)}$, $2^{O(k \log(k))} \cdot n^{O(1)}$, $2^{O(k^2)} \cdot n^{O(1)}$ and $n^{O(k)}$ time algorithms parameterized respectively by clique-width, $\mathbb{Q}$-rank-width, rank-width and maximum induced matching width. Our approach simplifies and unifies the known algorithms for each of the parameters and match asymptotically also the running time of the best algorithms for basic NP-hard problems such as Vertex Cover and Dominating Set. Our framework is based on the d-neighbor equivalence defined in [Bui-Xuan, Telle and Vatshelle, TCS 2013]. The results we obtain highlight the importance and the generalizing power of this equivalence relation on width measures. We also prove that this equivalence relation could be useful for Max Cut: a W[1]-hard problem parameterized by clique-width. For this latter problem, we obtain $n^{O(k)}$, $n^{O(k)}$ and $n^{2^{O(k)}}$ time algorithm parameterized by clique-width, $\mathbb{Q}$-rank-width and rank-width.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases connectivity problem, feedback vertex set, d-neighbor equivalence, σ, ρ-domination, clique-width, rank-width, mim-width

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.17

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.11275`.

Acknowledgements This work is supported by French Agency for Research under the GraphEN project (ANR-15-CE40-0009).

1 Introduction

Tree-width is one of the most well-studied graph parameters in the graph algorithm community, due to its numerous structural and algorithmic properties. Nevertheless, despite the broad interest on tree-width, only sparse graphs can have bounded tree-width. But, many NP-hard problems are tractable on dense graph classes. For many graph classes, this tractability can be explained through other width measures. The most remarkable ones are certainly clique-width [8], rank-width [18], and maximum induced matching width (*a.k.a.* mim-width) [23]. We obtain most of these parameters through the notion of *rooted layout* (see Section 2).

These other width measures have a modeling power strictly stronger than the modeling power of tree-width. For example, if a graph class has bounded tree-width, then it has bounded clique-width [8], but the converse is false as cliques have clique-width at most 2

[1] Corresponding author

and unbounded tree-width. While rank-width has the same modeling power as clique-width, mim-width has the strongest one among all these width measures and is even bounded on interval graphs [1]. Despite their generality, a lot of NP-hard problems admit polynomial time algorithms when one of these width measures is fixed. But, dealing with these width measures is known to be harder than manipulating tree-width.

Concerning their computations, it is not known whether the clique-width (respectively mim-width) of a graph can be approximated within a constant factor in time $\mathsf{f}(k) \cdot n^{O(1)}$ (resp. $n^{\mathsf{f}(k)}$) for some function f. However, for rank-width and its variant $\mathbb{Q}$-rank-width [19], there are efficient FPT algorithms for computing a decomposition approximating the width of the input graph [19, 20].

Finding efficient algorithms parameterized by one of these width measures is by now standard for problems based on local constraints [7, 22]. In contrast, the task is quite complicated for problems involving a global constraint, *e.g.*, connectivity or acyclicity. For a long time, our knowledge on the parameterized complexity of this latter kind of problems, with parameters the common width measures, was quite limited even for tree-width. For a while, the FPT community used to think that for problems involving global constraints the naive $k^{O(k)} \cdot n^{O(1)}$ time algorithm, k being the tree-width of the input graph, could not be improved. But, quite surprisingly, in 2011, Cygan et al. introduced in [9] a technique called *Cut & Count* to design Monte Carlo $2^{O(k)} \cdot n^{O(1)}$ time algorithms for a wide range of problems with global constraints, including Hamiltonian Cycle, Feedback Vertex Set, and Connected Dominating Set. Later, Bodlaender et al. proposed in [5] a general toolkit, called *rank-based approach*, to design deterministic $2^{O(k)} \cdot n$ time algorithms to solve a wider range of problems.

Recently, Bergougnoux and Kanté [3] adapted the rank-based approach of [5] to obtain $2^{O(k)} \cdot n$ time algorithms, k being the clique-width of a given decomposition, for many problems with a global constraint, *e.g.* Connected Dominating Set and Feedback Vertex Set. Unlike tree-width and clique-width, algorithms parameterized by rank-width and mim-width for problems with a global constraint, were not investigated, except for some special cases such as Feedback Vertex Set [12, 15] and Longest Induced Path [14].

One successful way to design efficient algorithms with these width measures is through the notion of *d-neighbor equivalence*. This concept was introduced by Bui-Xuan, Telle and Vatshelle in [7]. Formally, given $A \subseteq V(G)$ and $d \in \mathbb{N}^+$, two sets $X, Y \subseteq A$ are d-neighbor equivalent *w.r.t.* A if, for all $v \in V(G) \setminus A$, we have $\min(d, |N(v) \cap X|) = \min(d, |N(v) \cap Y|)$, where $N(v)$ is the set of neighbors of v in G. Notice that X and Y are 1-neighbor equivalent *w.r.t.* A if and only if both have the same neighborhood in $V(G) \setminus A$. The d-neighbor equivalence gives rise to a width measure, called in this paper *d-neighbor-width* (defined in Section 2). It is worth noticing that the boolean-width of a layout introduced in [6] corresponds to the binary logarithm of the 1-neighbor-width.

These notions were used by Bui-Xuan et al. in [7] to design efficient algorithms for the family of problems called (σ, ρ)-Dominating Set problems. This family of problems was introduced by Telle and Proskurowski in [22] (we define this family in Section 4). Many NP-hard problems based on local constraints belong to this family, see [7, Table 1].

Bui-Xuan et al.[7] designed an algorithm that, given a rooted layout $\mathcal{L}$, solve any (σ, ρ)-Dominating Set problem in time $\mathsf{s\text{-}nec}_d(\mathcal{L})^{O(1)} \cdot n^{O(1)}$ where d is a constant depending on the considered problem. The known upper bounds on $\mathsf{s\text{-}nec}_d(\mathcal{L})$ and the algorithm of [7] give efficient algorithms to solve any (σ, ρ)-Dominating Set problem, with parameters tree-width, clique-width, ($\mathbb{Q}$)-rank-width, and mim-width. The running times of these algorithms are given in Table 1.

Table 1 Upper bounds on $\mathsf{s\text{-}nec}_d(\mathcal{L})^{O(1)} \cdot n^{O(1)}$ with $\mathcal{L}$ a layout and d a constant.

Tree-width	Clique-width	Rank-width	$\mathbb{Q}$-rank-width	Mim-width
$2^{O(k)} \cdot n^{O(1)}$	$2^{O(k)} \cdot n^{O(1)}$	$2^{O(k^2)} \cdot n^{O(1)}$	$2^{O(k\log(k))} \cdot n^{O(1)}$	$n^{O(k)}$

Our contributions. In this paper, we design a framework based on the 1-neighbor equivalence (presented in Section 3) and using some ideas of the rank-based approach of [5] to design efficient algorithms for many problems involving a connectivity constraint. This framework provides tools to reduce the size of the sets of partial solutions we compute at each step of a dynamic programming algorithm. We prove that many ad-hoc algorithms for these problems can be unified into a single algorithm that is almost the same as the one from [7] computing a dominating set.

In Section 4, we use our framework to design an algorithm that, given a rooted layout $\mathcal{L}$, solves any connectivity variant (a solution must induce a connected graph) of a (σ, ρ)-Dominating Set problem. This includes some well-known problems such as Connected Dominating Set, Connected Vertex Cover or Node Weighted Steiner Tree. The running time of our algorithm is polynomial in n and $\mathsf{s\text{-}nec}_d(\mathcal{L})$, with d a constant that depends on σ and ρ. Consequently, each connectivity variant of a (σ, ρ)-Dominating Set problem admits algorithms with the running times given in Table 1.

In Section 5, we introduce some new concepts to deal with acyclicity. We use these concepts to deal with the *AC* variants[2] (a solution must induce a tree) of (σ, ρ)-Dominating Set problems. Both Maximum Induced Tree and Longest Induced Path are *AC* variants of (σ, ρ)-Dominating Set problems. We prove that there exist algorithms that solve these AC variants in the running times given in Table 1. To obtain these results, we rely heavily on the d-neighbor equivalence. However, we were not able to provide an algorithm whose running time is polynomial in n and $\mathsf{s\text{-}nec}_d(\mathcal{L})$ for some constant d. Instead, we provide an algorithm whose behavior depends slightly on each width measure considered in Table 1. We moreover prove that we can modify slightly this algorithm to solve any acyclic variant (a solution must induce a forest) of a (σ, ρ)-Dominating Set problem. In particular, this shows that we can use the algorithm for Maximum Induced Tree to solve the Feedback Vertex Set problem.

Up to a constant in the exponent, the running times of our algorithms and their algorithmic consequences match those of the best known algorithms for basic problems such as Vertex Cover and Dominating Set [7, 19]. Surprisingly, our result reveal that the d-neighbor equivalence relation can be used for problems which are not based on local constraints. This highlights the importance and the generalizing power of this concept on many width measures.

We conclude in Section 6 and state our theorem for the computation of Max Cut – a problem which is W[1]-hard parameterized by clique-width – whose running time is polynomial in n and the n-neighbor width of a given rooted layout. This algorithm gives the best known algorithms parameterized by clique-width, $\mathbb{Q}$-rank-width and rank-width. It is worth mentioning that contrary to the algorithm for Max Cut given in [11], there is no need to assume that the graph is given with a clique-width expression as our algorithm can be parameterized by $\mathbb{Q}$-rank-width, which is always smaller than clique-width and for which also a fast FPT $(3k + 1)$-approximation algorithm exists [20].

[2] AC stands for "acyclic and connected".

Relation to previous works. Our framework can be used on tree-decomposition to obtain $2^{O(k)} \cdot n^{O(1)}$ time algorithms parameterized by tree-width for the variants of (σ, ρ)-Dominating Set problems. Indeed, given a vertex separator S of size k, the number of d-neighbor equivalence classes over S (resp. $V(G) \setminus S$) is upper bounded by 2^k (resp. $(d+1)^k$). For this reason, we can consider our framework as a generalization of the rank-based approach of [5]. Our framework generalizes also the clique-width adaptation of the rank-based approach used in [3] to obtain $2^{O(k)} \cdot n$ time algorithms, k being the clique-width of a given decomposition, for Connected (σ, ρ)-Dominating Set problem and Feedback Vertex Set. However, the constant in the running time of the algorithms in [3, 5] are better than those of our algorithms. Indeed, our approach is based on a more general parameter and is not optimized neither for tree-width nor clique-width. Our framework simplifies the algorithms in [3, 5] because contrary to [3, 5] we do not use weighted partitions to encode the partial solutions. Consequently, the definitions of the dynamic programming tables and the computational steps of our algorithms are simpler than those in [3, 5]. This is particularly true for Feedback Vertex Set where the use of weighted partitions to encode the partial solutions in [3] implies to take care of many technical details concerning the acyclicity.

The results we obtain simplify the $2^{O(k^2)} \cdot n^{O(1)}$ time algorithm parameterized by rank-width for Feedback Vertex Set from [12], and the $n^{O(k)}$ time algorithms parameterized by mim-width for Feedback Vertex Set and Longest Induced Path from [14, 15].

Concerning mim-width, we provide unified polynomial-time algorithms for the considered problems for all well-known graph classes having bounded mim-width and for which a layout of bounded mim-width can be computed in polynomial time [1] (*e.g.*, interval graphs, circular arc graphs, permutation graphs, Dilworth-k graphs and k-polygon graphs for all fixed k). Notice that we also generalize one of the results from [17] proving that the Connected Vertex Cover problem is solvable in polynomial time for circular arc graphs.

It is worth noticing that the approach used in [9] called Cut & Count can also be generalized to the d-neighbor-width for any Connected (σ, ρ)-dominating set problem with more or less the same arguments used in this paper (see the PhD thesis [2]).

2 Preliminaries

The size of a set V is denoted by $|V|$ and its power set is denoted by 2^V. We write $A \setminus B$ for the set difference of A from B. We denote by $\mathbb{N}$ the set of non-negative integers and by $\mathbb{N}^+$ the set $\mathbb{N} \setminus \{0\}$. We let $\min(\emptyset) := +\infty$ and $\max(\emptyset) := -\infty$. Let V be a finite set. A set function $\mathsf{f} : 2^V \to \mathbb{N}$ is *symmetric* if, for all $S \subseteq V$, we have $\mathsf{f}(S) = \mathsf{f}(V \setminus S)$.

Graphs. Our graph terminology is standard, and we refer to [10]. The vertex set of a graph G is denoted by $V(G)$ and its edge set by $E(G)$. For every vertex set $X \subseteq V(G)$, when the underlying graph is clear from context, we denote by $\overline{X}$, the set $V(G) \setminus X$. An edge between two vertices x and y is denoted by xy or yx. The set of vertices that is adjacent to x is denoted by $N_G(x)$. For a set $U \subseteq V(G)$, we define $N_G(U) := \bigcup_{x \in U} N_G(x)$. If the underlying graph is clear, then we may remove G from the subscript.

The subgraph of G induced by a subset X of its vertex set is denoted by $G[X]$. For $X, Y \subseteq V(G)$, we denote by $G[X, Y]$ the bipartite graph with vertex set $X \cup Y$ and edge set $\{xy \in E(G) : x \in X \text{ and } y \in Y\}$. Moreover, we denote by $M_{X,Y}$ the adjacency matrix between X and Y, *i.e.*, the (X, Y)-matrix such that $M_{X,Y}[x, y] = 1$ if $y \in N(x)$ and 0 otherwise. For a graph G, we denote by $\mathsf{cc}(G)$ the partition $\{V(C) : C$ is a connected component of $G\}$. For two subsets $\mathcal{A}$ and $\mathcal{B}$ of $2^{V(G)}$, we define the *merging* of $\mathcal{A}$ and $\mathcal{B}$, denoted by $\mathcal{A} \bigotimes \mathcal{B}$, as

$$\mathcal{A} \bigotimes \mathcal{B} := \begin{cases} \emptyset & \text{if } \mathcal{A} = \emptyset \text{ or } \mathcal{B} = \emptyset, \\ \{X \cup Y \,:\, X \in \mathcal{A} \text{ and } Y \in \mathcal{B}\} & \text{otherwise.} \end{cases}$$

Let $X \subseteq V(G)$. A *consistent cut* of X is an ordered bipartition (X_1, X_2) of X such that $N(X_1) \cap X_2 = \emptyset$. We denote by $\mathsf{ccut}(X)$ the set of all consistent cuts of X.

The d-neighbor equivalence relation. Let G be a graph. The following definition is from [7]. Let $A \subseteq V(G)$ and $d \in \mathbb{N}^+$. Two subsets X and Y of A are *d-neighbor equivalent* w.r.t. A, denoted by $X \equiv_A^d Y$, if $\min(d, |X \cap N(u)|) = \min(d, |Y \cap N(u)|)$ for all $u \in \overline{A}$. It is not hard to check that $\equiv_A^d$ is an equivalence relation.

For all $d \in \mathbb{N}^+$, we let $\mathsf{nec}_d : 2^{V(G)} \to \mathbb{N}$ where, for all $A \subseteq V(G)$, $\mathsf{nec}_d(A)$ is the number of equivalence classes of $\equiv_A^d$. Notice that while nec_1 is a symmetric function [16, Theorem 1.2.3], nec_d is not necessarily symmetric for $d \geq 2$.

In order to manipulate the equivalence classes of $\equiv_A^d$, one needs to compute a representative for each equivalence class in polynomial time. This is achieved with the following notion of a representative. Let G be a graph with an arbitrary ordering of $V(G)$ and let $A \subseteq V(G)$. For each $X \subseteq A$, let us denote by $\mathsf{rep}_A^d(X)$ the lexicographically smallest set $R \subseteq A$ such that $|R|$ is minimized and $R \equiv_A^d X$. Moreover, we denote by $\mathcal{R}_A^d$ the set $\{\mathsf{rep}_A^d(X) : X \subseteq A\}$. It is worth noticing that the empty set always belongs to $\mathcal{R}_A^d$, for all $A \subseteq V(G)$ and $d \in \mathbb{N}^+$. Moreover, we have $\mathcal{R}_{V(G)}^d = \mathcal{R}_\emptyset^d = \{\emptyset\}$ for all $d \in \mathbb{N}^+$. In order to compute $\mathcal{R}_A^d$, we use the following lemma.

▶ **Lemma 1** ([7])**.** *For every $A \subseteq V(G)$ and $d \in \mathbb{N}^+$, one can compute in time $O(\mathsf{nec}_d(A) \cdot \log(\mathsf{nec}_d(A)) \cdot |V(G)|^2)$, the sets $\mathcal{R}_A^d$ and a data structure, that given a set $X \subseteq A$, computes $\mathsf{rep}_A^d(X)$ in time $O(\log(\mathsf{nec}_d(A)) \cdot |A| \cdot |V(G)|)$.*

Graph width measures. A *rooted binary tree* is a binary tree with a distinguished vertex called the *root*. Since we manipulate at the same time graphs and trees representing them, the vertices of trees will be called *nodes*. A *rooted layout* of G is a pair $\mathcal{L} = (T, \delta)$ of a rooted binary tree T and a bijective function δ between $V(G)$ and the leaves of T. For each node x of T, let L_x be the set of all the leaves l of T such that the path from the root of T to l contains x. We denote by $V_x^{\mathcal{L}}$ the set of vertices that are in bijection with L_x, *i.e.*, $V_x^{\mathcal{L}} := \{v \in V(G) : \delta(v) \in L_x\}$. When $\mathcal{L}$ is clear from the context, we may remove $\mathcal{L}$ from the superscript.

All the width measures dealt with in this paper are special cases of the following one, the difference being in each case the used set function. Given a set function $\mathsf{f} : 2^{V(G)} \to \mathbb{N}$ and a rooted layout $\mathcal{L} = (T, \delta)$, the f-width of a node x of T is $\mathsf{f}(V_x^{\mathcal{L}})$ and the f-width of (T, δ), denoted by $\mathsf{f}(T, \delta)$ (or $\mathsf{f}(\mathcal{L})$), is $\max\{\mathsf{f}(V_x^{\mathcal{L}}) : x \in V(T)\}$. Finally, the f-width of G is the minimum f-width over all rooted layouts of G.

For a graph G, we let $\mathsf{s\text{-}nec}_d, \mathsf{mw}, \mathsf{mim}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}$ be functions from $2^{V(G)}$ to $\mathbb{N}$ such that for every $A \subseteq V(G)$, $\mathsf{s\text{-}nec}_d(A) = \max\{\mathsf{nec}_d(A), \mathsf{nec}_d(\overline{A})\}$, $\mathsf{mw}(A)$ is the cardinality of $\{N(v) \cap \overline{A} : v \in A\}$, $\mathsf{mim}(A)$ is the size of a maximum induced matching of the graph $G[A, \overline{A}]$ and $\mathsf{rw}(A)$ (resp. $\mathsf{rw}_{\mathbb{Q}}(A)$) is the rank over $GF(2)$ (resp. $\mathbb{Q}$) of the matrix $M_{A,\overline{A}}$. The *d-neighbor-width*, *module-width*, *mim-width*, *rank-width* and *$\mathbb{Q}$-rank-width* of G, are respectively, its $\mathsf{s\text{-}nec}_d$-width, mw-width, mim-width, rw-width and $\mathsf{rw}_{\mathbb{Q}}$-width [23].

Observe that, for every graph G, $\mathsf{mw}(G) \leq \mathsf{cw}(G) \leq 2\mathsf{mw}(G)$ where $\mathsf{cw}(G)$ is the clique-width of G [21] , and one can moreover translate, in time at most $O(n^2)$, a given decomposition into the other one with width at most the given bounds.

In the following, we fix G an n-vertex graph, (T, δ) a rooted layout of G, and $\mathsf{w} : V(G) \to \mathbb{Q}$ a weight function over the vertices of G. We also assume that $V(G)$ is ordered.

3 Representative sets

In this section, we define a notion of representativity between sets of partial solutions *w.r.t.* the connectivity. Our notion of representativity is defined *w.r.t.* some node x of T and the 1-neighbor equivalence class of some set $R' \subseteq \overline{V_x}$. In our algorithms, R' will always belong to $\mathcal{R}^d_{\overline{V_x}}$ for some $d \in \mathbb{N}^+$. Our algorithms compute a set of partial solutions for each $R' \in \mathcal{R}^d_{\overline{V_x}}$. The partial solutions computed for R' will be completed with sets equivalent to R' *w.r.t.* $\equiv^d_{\overline{V_x}}$. Intuitively, the R''s represent some expectation about how we will complete our sets of partial solutions. For the connectivity and the domination, $d = 1$ is enough but if we need more information for some reasons (for example the (σ, ρ)-domination or the acyclicity), we may take $d > 1$. This is not a problem as the d-neighbor equivalence class of R' is included in the 1-neighbor equivalence class of R'. Hence, in this section, we fix a node x of T and a set $R' \subseteq \overline{V_x}$ to avoid to overload the statements by the sentence "let x be a node of T and $R' \subseteq \overline{V_x}$". We let $\mathsf{opt} \in \{\min, \max\}$; if we want to solve a maximization (or minimization) problem, we use $\mathsf{opt} = \max$ (or $\mathsf{opt} = \min$). We use it also, as here, in the next sections.

▶ **Definition 2** ((x, R')-representativity)**.** *For every $\mathcal{A} \subseteq 2^{V(G)}$ and $Y \subseteq V(G)$, we define* $\mathsf{best}(\mathcal{A}, Y) := \mathsf{opt}\{\mathsf{w}(X) : X \in \mathcal{A}$ *and* $G[X \cup Y]$ *is connected* $\}$.

Let $\mathcal{A}, \mathcal{B} \subseteq 2^{V_x}$. We say that $\mathcal{B}$ (x, R')-represents $\mathcal{A}$ if, for every $Y \subseteq \overline{V_x}$ such that $Y \equiv^1_{\overline{V_x}} R'$, we have $\mathsf{best}(\mathcal{A}, Y) = \mathsf{best}(\mathcal{B}, Y)$.

Notice that the (x, R')-representativity is an equivalence relation. The set $\mathcal{A}$ is meant to represent a set of partial solutions of $G[V_x]$ associated with R' which have been computed. If a $\mathcal{B} \subseteq \mathcal{A}$ (x, R')-represents $\mathcal{A}$, then we can safely substitute $\mathcal{A}$ by $\mathcal{B}$ because the quality of the output of the dynamic programming algorithm will remain the same. Indeed, for every subset Y of $\overline{V_x}$ such that $Y \equiv^1_{\overline{V_x}} R'$, an optimum solution obtained by the union of a partial solution in $\mathcal{A}$ and Y will have the same weight as an optimum solution obtained from the union of a set in $\mathcal{B}$ and Y.

The following theorem presents the main tool of our framework: a function reduce that, given a set of partial solutions $\mathcal{A}$, outputs a subset of $\mathcal{A}$ that (x, R')-represents $\mathcal{A}$ and whose size is upper bounded by $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$. To design this function, we use ideas from the rank-based approach of [5]. That is, we define a small matrix $\mathcal{C}$ with $|\mathcal{A}|$ rows and $\mathsf{s\text{-}nec}_1(V_x)^2$ columns. Then, we show that a basis of maximum weight of the row space of $\mathcal{C}$ corresponds to an (x, R')-representative set of $\mathcal{A}$. Since $\mathcal{C}$ has $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$ columns, the size of a basis of $\mathcal{C}$ is smaller than $\mathsf{s\text{-}nec}_1(\mathcal{L})^2$. By calling reduce after each computational step, we keep the sizes of the sets of partial solutions polynomial in $\mathsf{s\text{-}nec}_1(\mathcal{L})$.

In order to compute a small (x, R')-representative set of a set $\mathcal{A} \subseteq 2^{V_x}$, the following theorem requires that the sets in $\mathcal{A}$ are pairwise equivalent *w.r.t.* $\equiv^1_{V_x}$. This is useful since in our algorithm we classify our sets of partial solutions with respect to this property. We need this to guarantee that the partial solutions computed for R' will be completed with sets equivalent to R' *w.r.t.* $\equiv^d_{\overline{V_x}}$. However, if one wants to compute a small (x, R')-representative set of a set $\mathcal{A}$ that does not respect this property, then it is enough to compute an (x, R')-representative set for each 1-neighbor equivalence class of $\mathcal{A}$. The union of these (x, R')-representative sets is an (x, R')-representative set of $\mathcal{A}$. In the following, ω is the matrix multiplication exponent.

▶ **Theorem 3.** *Let $R \in \mathcal{R}^1_{V_x}$ and $R' \subseteq \overline{V_x}$. Then, there exists an algorithm* reduce *that, given $\mathcal{A} \subseteq 2^{V_x}$ such that $X \equiv^1_{V_x} R$ for all $X \in \mathcal{A}$, outputs in time $O(|\mathcal{A}| \cdot \mathsf{nec}_1(V_x)^{2(\omega-1)} \cdot n^2)$ a subset $\mathcal{B} \subseteq \mathcal{A}$ such that $\mathcal{B}$ (x, R')-represents $\mathcal{A}$ and $|\mathcal{B}| \leq \mathsf{nec}_1(V_x)^2$.*

Sketch of proof. We assume w.l.o.g. that opt = max (the case opt = min is symmetric). If $R' \equiv^1_{\overline{V_x}} \emptyset$, then, from Definition 2, it is enough to output $\{X\}$ where X is a set in $\mathcal{A}$ of maximum weight such that $G[X]$ is connected.

Assume that R' is not equivalent to $\emptyset$ *w.r.t.* $\equiv^1_{\overline{V_x}}$. Observe that, for every $X \in \mathcal{A}$ which admits a connected component C with $N(C) \cap R' = \emptyset$, the graph $G[X \cup Y]$ is not connected for every $Y \equiv^1_{\overline{V_x}} R'$. Thus, we can safely remove from $\mathcal{A}$ all such sets, this can be done in time $|\mathcal{A}| \cdot n^2$. Let $\mathcal{D}$ be the set of all subsets Y of $\overline{V_x}$ such that $Y \equiv^1_{\overline{V_x}} R'$ and, for all $C \in \mathsf{cc}(Y)$, we have $N(C) \cap R \neq \emptyset$. It is easy to check that the sets in $2^{\overline{V_x}} \setminus \mathcal{D}$ do not matter in the (x, R')-representativity: they will not give a solution with a set in $\mathcal{A}$.

For every $Y \in \mathcal{D}$, we let v_Y be one fixed vertex of Y. In the following, we denote by $\mathfrak{R}$ the set $\{(R'_1, R'_2) \in \mathcal{R}^1_{\overline{V_x}} \times \mathcal{R}^1_{\overline{V_x}}\}$. Let $\mathcal{C}$, and $\overline{\mathcal{C}}$ be, respectively, an $(\mathcal{A}, \mathfrak{R})$-matrix and an $(\mathfrak{R}, \mathcal{D})$-matrix such that

$$\mathcal{C}[X, (R'_1, R'_2)] := \begin{cases} 1 & \text{if } \exists (X_1, X_2) \in \mathsf{ccut}(X) \text{ such that } N(X_1) \cap R'_2 = \emptyset \wedge N(X_2) \cap R'_1 = \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

$$\overline{\mathcal{C}}[(R'_1, R'_2), Y] := \begin{cases} 1 & \text{if } \exists (Y_1, Y_2) \in \mathsf{ccut}(Y) \text{ such that } v_Y \in Y_1,\ Y_1 \equiv^1_{\overline{V_x}} R'_1, \text{ and } Y_2 \equiv^1_{\overline{V_x}} R'_2, \\ 0 & \text{otherwise.} \end{cases}$$

Owing to the partial solutions we have removed from $\mathcal{A}$ and the definition of $\mathcal{D}$, we can prove that a basis of maximum weight of the row space of $\mathcal{C}$ is an (x, R')-representative set of $\mathcal{A}$. This follows from the fact that $(\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y]$ equals the number of consistent cuts (W_1, W_2) in $\mathsf{ccut}(X \cup Y)$ such that $v_Y \in W_1$. Consequently, $(\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y] = 2^{|\mathsf{cc}(G[X \cup Y])|-1}$ and thus $(\mathcal{C} \cdot \overline{\mathcal{C}})[X, Y]$ is odd if and only if $G[X \cup Y]$ is connected. The running time of reduce and the size of reduce$(\mathcal{A})$ follows from the size of $\mathcal{C}$ (*i.e.* $|\mathcal{A}| \cdot \mathsf{nec}_1(V_x)^2$) and the fact that both $\mathcal{C}$ and a basis of maximum weight of $\mathcal{C}$ are easy to compute. ◀

Now to boost up a dynamic programming algorithm P on some rooted layout (T, δ) of G, we can use the function reduce to keep the size of the sets of partial solutions bounded by $\mathsf{s\text{-}nec}_1(T, \delta)^2$. We call P' the algorithm obtained from P by calling the function reduce at every step of computation. We can assume that the set of partial solutions $\mathcal{A}_r$ computed by P and associated with the root r of (T, δ) contains an optimal solution (this will be the cases in our algorithms). To prove the correctness of P', we need to prove that $\mathcal{A}'_r$ $(r, \emptyset)$-represents $\mathcal{A}_r$ where $\mathcal{A}'_r$ is the set of partial solutions computed by P' and associated with r. For doing so, we need to prove that at each step of the algorithm the operations we use preserve the (x, R')-representativity. The following fact states that we can use the union without restriction, it follows directly from Definition 2 of (x, R')-representativity.

▶ **Fact 4.** *If $\mathcal{B}$ and $\mathcal{D}$ (x, R')-represents, respectively, $\mathcal{A}$ and $\mathcal{C}$, then $\mathcal{B} \cup \mathcal{D}$ (x, R')-represents $\mathcal{A} \cup \mathcal{C}$.*

The second operation we use in our dynamic programming algorithms is the merging operator $\bigotimes$. In order to safely use it, we need the following notion of compatibility that just tells which partial solutions from V_a and V_b can be joined to possibly form a partial solution in V_x. (It was already used in [7] without naming it.)

▶ **Definition 5** (d-(R, R')-compatibility). *Suppose that x is an internal node of T with a and b as children. Let $d \in \mathbb{N}^+$ and $R \in \mathcal{R}^d_{V_x}$. We say that $(A, A') \in \mathcal{R}^d_{V_a} \times \mathcal{R}^d_{\overline{V_a}}$ and $(B, B') \in \mathcal{R}^d_{V_b} \times \mathcal{R}^d_{\overline{V_b}}$ are d-(R, R')-compatible if we have (1) $A \cup B \equiv^d_{V_x} R$, (2) $A' \equiv^d_{\overline{V_a}} B \cup R'$, and (3) $B' \equiv^d_{\overline{V_b}} A \cup R'$.*

▶ **Lemma 6.** *Suppose that x is an internal node of T with a and b as children. Let $d \in \mathbb{N}^+$ and $R \in \mathcal{R}^d_{V_x}$. Let $(A, A') \in \mathcal{R}^d_{V_a} \times \mathcal{R}^d_{\overline{V_a}}$ and $(B, B') \in \mathcal{R}^d_{V_b} \times \mathcal{R}^d_{\overline{V_b}}$ that are d-(R, R')-compatible. Let $\mathcal{A} \subseteq 2^{V_a}$ such that, for all $X \in \mathcal{A}$, we have $X \equiv^d_{V_a} A$, and let $\mathcal{B} \subseteq 2^{V_b}$ such that, for all $W \in \mathcal{B}$, we have $W \equiv^d_{V_b} B$. If $\mathcal{A}' \subseteq \mathcal{A}$ (a, A')-represents $\mathcal{A}$ and $\mathcal{B}' \subseteq \mathcal{B}$ (b, B')-represents $\mathcal{B}$, then $\mathcal{A}' \bigotimes \mathcal{B}'$ (x, R')-represents $\mathcal{A} \bigotimes \mathcal{B}$.*

4 Connected (Co)-(σ, ρ)-Dominating Sets

Let σ and ρ be two (non-empty) finite or co-finite subsets of $\mathbb{N}$. We say that a subset D of $V(G)$ (σ, ρ)-*dominates* a subset $U \subseteq V(G)$ if, for every vertex $u \in U \cap D$, we have $|N(u) \cap D| \in \sigma$, and, for every vertex $u \in U \setminus D$, we have $|N(u) \cap D| \in \rho$. A subset D of $V(G)$ is a (σ, ρ)-*dominating set* (resp. co-(σ, ρ)-*dominating set*) if D (resp. $V(G) \setminus D$) (σ, ρ)-*dominates* $V(G)$. A problem is a Connected (σ, ρ)-Dominating Set if it consists in finding a connected (σ, ρ)-dominating set of maximum (or minimum) weight. Similarly, one can define the Connected Co-(σ, ρ)-Dominating Set problems. Examples are Connected Perfect Dominating Set and Connected Vertex-Cover[3].

Let $d(\mathbb{N}) := 0$, and for a finite or co-finite subset μ of $\mathbb{N}$, let $d(\mu) := \min(\max(\mu), \max(\mathbb{N} \setminus \mu)) + 1$. Let $d := \max\{1, d(\sigma), d(\rho)\}$. The definition of d is motivated by the following lemma.

▶ **Lemma 7** ([7]). *Let $A \subseteq V(G)$. Let $X \subseteq A$ and $Y, Y' \subseteq \overline{A}$ such that $Y \equiv^d_A Y'$. Then $(X \cup Y)$ (σ, ρ)-dominates A if and only if $(X \cup Y')$ (σ, ρ)-dominates A.*

In this section, we present an algorithm that computes a maximum (or minimum) connected (σ, ρ)-dominating set with a graph G and a layout (T, δ) as inputs. Its running time is $O(\mathsf{s\text{-}nec}_d(T, \delta)^{O(1)} \cdot n^3)$. The same algorithm, with some little modifications, will be able to find a minimum Steiner tree or a maximum (or minimum) connected co-(σ, ρ)-dominating set as well.

To deal with the local constraint, *i.e.*, the (σ, ρ)-domination, we use the ideas of Bui-Xuan et al. [7]. For every $x \in V(T)$ and all pairs $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, we let $\mathcal{A}_x[R, R'] := \{X \subseteq V_x : X \equiv^d_{V_x} R$ and $X \cup R'$ (σ, ρ)-dominates $V_x\}$. To compute a maximum (or minimum) (σ, ρ)-dominating set, Bui-Xuan et al. [7] proved that it is enough to compute, for each node $x \in V(T)$ and each pair (R, R'), a partial solution X in $\mathcal{A}_x[R, R']$ of maximum (or minimum) weight. Indeed, by Lemma 7, if a partial solution $X \in \mathcal{A}_x[R, R']$ could be completed into a (σ, ρ)-dominating set of G with a set $Y \equiv^d_{\overline{V_x}} R'$, then it is the case for every partial solution in $\mathcal{A}_x[R, R']$. To deal with the connectivity constraint, for each node x of $V(T)$ and each pair (R, R'), our algorithm will compute a set $\mathcal{D}_x[R, R']$ that satisfies the following invariant.

Invariant. For every $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$, the set $\mathcal{D}_x[R, R']$ is a subset of $\mathcal{A}_x[R, R']$ of size at most $\mathsf{s\text{-}nec}_1(T, \delta)^2$ that (x, R')-represents $\mathcal{A}_x[R, R']$.

Notice that, by the definition of $\mathcal{A}_r[\emptyset, \emptyset]$ (r being the root of T) and the definition of (x, R')-representativity, if G admits a connected (σ, ρ)-dominating set, then $\mathcal{D}_r[\emptyset, \emptyset]$ must contain a maximum (or minimum) connected (σ, ρ)-dominating set.

[3] More can be found in [7, Table 1] by adding a connectivity constraint to the sets or their complements.

We compute the tables $\mathcal{D}_x$'s by a usual bottom-up dynamic programming algorithm, starting at the leaves of T. The computational steps are trivial for the leaves, and for each internal node x with children a and b, and for each $(R, R') \in \mathcal{R}^d_{V_x} \times \mathcal{R}^d_{\overline{V_x}}$ we let $\mathcal{D}_x[R, R']$ be an (x, R')-representative set of $\left(\bigcup_{(A,A'),\ (B,B')\ d\text{-}(R,R')\text{-compatible}} \mathcal{A}_a[A, A'] \bigotimes \mathcal{A}_b[B, B']\right)$ computed with the function reduce defined in Section 3. This guarantees that each set $\mathcal{D}_x[R, R']$ contains at most $\mathsf{s\text{-}nec}_1(T, \delta)^2$ partial solutions. Thanks to Lemma 6, we can state.

▶ **Theorem 8.** *There exists an algorithm that, given an n-vertex graph G and a rooted layout (T, δ) of G, computes a maximum (or minimum) connected (σ, ρ)-dominating set in time $O(\mathsf{s\text{-}nec}_d(T, \delta)^3 \cdot \mathsf{s\text{-}nec}_1(T, \delta)^{2(\omega+1)} \cdot \log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot n^3)$ with $d := \max\{1, d(\sigma), d(\rho)\}$.*

As a corollary, we can solve in time $\mathsf{s\text{-}nec}_1(T, \delta)^{O(1)} \cdot n^3$ the Node-Weighted Steiner Tree problem that asks, given a subset of vertices $K \subseteq V(G)$ called *terminals*, a subset T of minimum weight such that $K \subseteq T \subseteq V(G)$ and $G[T]$ is connected.

▶ **Corollary 9.** *There exists an algorithm that, given an n-vertex graph G, a subset $K \subseteq V(G)$, and a rooted layout (T, δ) of G, computes a minimum node-weighted Steiner tree for (G, K) in time $O(\mathsf{s\text{-}nec}_1(T, \delta)^{(2\omega+5)} \cdot \log(\mathsf{s\text{-}nec}_1(T, \delta)) \cdot n^3)$.*

Observe that Corollary 9 simplify and generalize the algorithm from [5] for (Edge-Weighted) Steiner Tree. Indeed the incidence graph of a graph of tree-width k has tree-width at most $k + 1$, and one can reduce the computation of a edge-weighted Steiner tree on a graph to the computation of a node-weighted Steiner tree on its incidence graph.

With few modifications of the algorithm from Theorem 8, we can state the following.

▶ **Corollary 10.** *There exists an algorithm that, given an n-vertex graph G and a rooted layout (T, δ) of G, computes a maximum (or minimum) connected co-(σ, ρ)-dominating set in time $O(\mathsf{s\text{-}nec}_d(T, \delta)^3 \cdot \mathsf{s\text{-}nec}_1(T, \delta)^{(2\omega+5)} \cdot \log(\mathsf{s\text{-}nec}_d(T, \delta)) \cdot n^3)$ with $d := \max\{1, d(\sigma), d(\rho)\}$.*

5 Acyclic variants of (Connected) (σ, ρ)-Dominating Set

We call AC-(σ, ρ)-Dominating Set (resp. Acyclic (σ, ρ)-Dominating Set) the family of problems which consists in finding a subset $X \subseteq V(G)$ of maximum (or minimum) weight such that X is a (σ, ρ)-dominating set of G and $G[X]$ is a tree (resp. a forest). Examples are Longest Induced Path and Feedback Vertex Set.

In this section, we present an algorithm that solves any AC-(σ, ρ)-Dominating Set problem. Unfortunately, we were not able to obtain an algorithm whose running time is polynomial in n and the d-neighbor-width of the given layout (for some constant d). But, for the other parameters, by using their respective properties, we get the running time presented in Table 1. Moreover, we show, via a polynomial reduction, that we can use our algorithm for AC-(σ, ρ)-Dominating Set problems (with some modifications) to solve any Acyclic (σ, ρ)-Dominating Set problem.

Let us first explain why we cannot use the same trick as in [5] on the algorithms of Section 4 to ensure the acyclicity, that is classifying the partial solutions X – associated with a node $x \in V(T)$ – with respect to $|X|$ and $|E(G[X])|$. Indeed, for two sets $X, W \subseteq V_x$ with $|X| = |W|$ and $|E(G[X])| = |E(G[W])|$, we have $|E(G[X \cup Y])| = |E(G[W \cup Y])|$, for all $Y \subseteq \overline{V_x}$, if and only if $X \equiv^n_{V_x} W$. Hence, the trick used in [5] would imply to classify the partial solutions with respect to their n-neighbor equivalence class. But, the upper bounds we have on $\mathsf{nec}_n(V_x)$ with respect to module-width, ($\mathbb{Q}$-)rank-width would lead to an XP algorithm.

In the following, we introduce some new concepts that extends the framework designed in Section 3 in order to manage acyclicity. All along, we give intuitions on these concepts through a concrete example: Maximum Induced Tree. Finally, we present the algorithms for the AC-(σ,ρ)-Dominating Set problems and the algorithms for Acyclic (σ,ρ)-Dominating Set problems. We start by defining a new notion of representativity to deal with the acyclicity constraint. This new notion of representativity is defined *w.r.t.* to the 2-neighbor equivalence class of a set $R' \subseteq \overline{V_x}$. We consider 2-neighbor equivalence classes instead of 1-neighbor equivalence classes in order to manage the acyclicity (see the following explanations). Similarly to Section 3, every concept introduced in this section is defined with respect to a node x of T and a set $R' \subseteq \overline{V_x}$. To simplify this section, we fix a node x of T and $R' \subseteq \overline{V_x}$. In our algorithm, R' will always belong to $\mathcal{R}^d_{\overline{V_x}}$ for some $d \in \mathbb{N}^+$ with $d \geq 2$. For Maximum Induced Tree $d = 2$ is enough and in general, we use $d := \max\{2, d(\sigma), d(\rho)\}$. The following definition extends Definition 2 of Section 3 to deal with the acyclicity.

▶ **Definition 11** ($(x,R')^{\mathsf{acy}}$-representativity). *For every $\mathcal{A} \subseteq 2^{V(G)}$ and $Y \subseteq V(G)$, we define* $\mathsf{best}(\mathcal{A},Y)^{\mathsf{acy}} := \mathsf{opt}\{\mathsf{w}(X) : X \in \mathcal{A}$ *and $G[X \cup Y]$ is a tree*$\}$.

Let $\mathcal{A},\mathcal{B} \subseteq 2^{V_x}$. We say that $\mathcal{B}$ $(x,R')^{\mathsf{acy}}$-represents $\mathcal{A}$ if, for every $Y \subseteq \overline{V_x}$ such that $Y \equiv^2_{\overline{V_x}} R'$, we have $\mathsf{best}^{\mathsf{acy}}(\mathcal{A},Y) = \mathsf{best}^{\mathsf{acy}}(\mathcal{B},Y)$.

In order to compute a maximum induced tree, we design an algorithm similar to those of Section 4. That is, for each $(R,R') \in \mathcal{R}^2_{V_x} \times \mathcal{R}^2_{\overline{V_x}}$, our algorithm will compute a set $\mathcal{D}_x[R,R'] \subseteq 2^{V_x}$ that is an $(x,R')^{\mathsf{acy}}$-representative set of small size of the set $\mathcal{A}_x[R] := \{X \subseteq V_x$ such that $X \equiv^2_{V_x} R\}$. This is sufficient to compute a maximum induced tree of G since we have $\mathcal{A}_r[\emptyset] = 2^{V(G)}$ with r the root of T. Thus, by Definition 11, any $(r,\emptyset)^{\mathsf{acy}}$-representative set of $\mathcal{A}_r[\emptyset]$ contains a maximum induced tree.

The key to compute the tables of our algorithm is a function that, given $\mathcal{A} \subseteq 2^{V_x}$, computes a small subset of $\mathcal{A}$ that $(x,R')^{\mathsf{acy}}$-represents $\mathcal{A}$. This function starts by removing from $\mathcal{A}$ some sets that will never give a tree with a set $Y \equiv^2_{\overline{V_x}} R'$. For doing so, we characterize the sets $X \in \mathcal{A}$ such that $G[X \cup Y]$ is a tree for some $Y \equiv^2_{\overline{V_x}} R'$. The following gives a formal definition of these important and unimportant partial solutions.

▶ **Definition 12** (R'-important). *We say that $X \subseteq V_x$ is R'-important if there exists $Y \subseteq \overline{V_x}$ such that $Y \equiv^2_{\overline{V_x}} R'$ and $G[X \cup Y]$ is a tree, otherwise, we say that X is R'-unimportant.*

By definition, any set obtained from a set $\mathcal{A}$ by removing R'-unimportant sets is an $(x,R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$. The following lemma gives some necessary conditions on R'-important sets. It follows that any set which does not respect one of these conditions can safely be removed from $\mathcal{A}$. These conditions are the key to obtain the running times of Table 1. At this point, we need to introduce the following notations. For every $X \subseteq V_x$, we define $X^0 := \{v \in X : N(v) \cap R' = \emptyset\}$, $X^1 := \{v \in X : |N(v) \cap R'| = 1\}$, and $X^{2+} := \{v \in X : |N(v) \cap R'| \geq 2\}$. Notice that, for every $Y \equiv^2_{\overline{V_x}} R'$ and $X \subseteq V_x$, the vertices in X^0 have no neighbor in Y, those in X^1 have exactly one neighbor in Y and those in X^{2+} have at least 2 neighbors in Y.

▶ **Lemma 13.** *If $X \subseteq V_x$ is R'-important, then $G[X]$ is a forest and the following properties are satisfied:*

1. *for every pair of distinct vertices a and b in X^{2+}, we have $N(a) \cap \overline{V_x} \neq N(b) \cap \overline{V_x}$,*
2. *$|X^{2+}|$ is upper bounded by* $2\mathsf{mim}(V_x)$, $2\mathsf{rw}(V_x)$ *and* $2\mathsf{rw}_{\mathbb{Q}}(V_x)$.

In order to prove these two necessary conditions, we need the properties of the 2-neighbor equivalence relation. More precisely, we use the fact that, for all $X \subseteq V_x$ and $Y \equiv^2_{\overline{V_x}} R'$, the vertices in X having at least two neighbors in Y corresponds exactly to those having at least two neighbors in R'. These vertices play a major role in the acyclicity and the computation of representatives in the following sense. By removing from $\mathcal{A}$ the sets that do not respect the two above properties, we are able to decompose $\mathcal{A}$ into a small number of sets $\mathcal{A}_1, \dots, \mathcal{A}_t$ such that an (x, R')-representative set of $\mathcal{A}_i$ is an $(x, R')^{\mathsf{acy}}$-representative set of $\mathcal{A}_i$ for each $i \in \{1, \dots, t\}$. We find an $(x, R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$, by computing an (x, R')-representative set $\mathcal{B}_i$ for each $\mathcal{A}_i$ with the function reduce. This is sufficient because $\mathcal{B}_1 \cup \dots \cup \mathcal{B}_t$ is an $(x, R')^{\mathsf{acy}}$-representative set of $\mathcal{A}$. The following definition gives a characterization of the subsets of 2^{V_x} for which an (x, R')-representative set is also an $(x, R')^{\mathsf{acy}}$-representative set.

▶ **Definition 14.** *We say that $\mathcal{A} \subseteq 2^{V_x}$ is R'-consistent if, for each $Y \subseteq \overline{V_x}$ such that $Y \equiv^2_{\overline{V_x}} R'$, if there exists $W \in \mathcal{A}$ such that $G[W \cup Y]$ is a tree, then, for each $X \in \mathcal{A}$, either $G[X \cup Y]$ is a tree or $G[X \cup Y]$ is not connected.*

The following lemma proves that an (x, R')-representative set of an R'-consistent set is also an $(x, R')^{\mathsf{acy}}$-representative set of this latter.

▶ **Lemma 15.** *Let $\mathcal{A} \subseteq 2^{V_x}$. For all $\mathcal{D} \subseteq \mathcal{A}$, if $\mathcal{A}$ is R'-consistent and $\mathcal{D}$ (x, R')-represents $\mathcal{A}$, then $\mathcal{D}$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$.*

The next lemma proves that, for each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, we can decompose a set $\mathcal{A} \subseteq 2^{V_x}$ into a small collection $\{\mathcal{A}_1, \dots, \mathcal{A}_t\}$ of pairwise disjoint subsets of $\mathcal{A}$ such that each $\mathcal{A}_i$ is R'-consistent. Even though some parts of the proof are specific to each parameter, the ideas are roughly the same. First, we remove the sets X in $\mathcal{A}$ that do not induce a forest. If $\mathsf{f} = \mathsf{mw}$, we remove the sets in $\mathcal{A}$ that do not respect Condition (1) of Lemma 13, otherwise, we remove the sets that do not respect the upper bound associated with f from Condition (2) of Lemma 13. These sets can be safely removed as, by Lemma 13, they are R'-unimportant. After removing these sets, we obtain the decomposition of $\mathcal{A}$ by taking the equivalence classes of some equivalence relation that is roughly the n-neighbor equivalence relation. Owing to the set of R'-unimportant sets we have removed from $\mathcal{A}$, we prove that the number of equivalence classes of this latter equivalence relation respects the upper bound associated with f that is described in Table 2.

▶ **Lemma 16.** *Let $\mathcal{A} \subseteq 2^{V_x}$. For each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, there exists a collection $\{\mathcal{A}_1, \dots, \mathcal{A}_t\}$ of pairwise disjoint subsets of $\mathcal{A}$ computable in time $O(|\mathcal{A}| \cdot \mathcal{N}_{\mathsf{f}}(T, \delta) \cdot n^2)$ such that (1) $\mathcal{A}_1 \cup \dots \cup \mathcal{A}_t$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$, (2) $\mathcal{A}_i$ is R'-consistent for each $i \in \{1, \dots, t\}$, and (3) $t \leq \mathcal{N}_{\mathsf{f}}(T, \delta)$; where $\mathcal{N}_{\mathsf{f}}(T, \delta)$ is the term defined in Table 2.*

Table 2 Upper bounds $\mathcal{N}_{\mathsf{f}}(T, \delta)$ on the size of the decomposition of Lemma 16 for each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$.

f	mw	$\mathsf{rw}_{\mathbb{Q}}$	rw	mim
$\mathcal{N}_{\mathsf{f}}(T, \delta)$	$2^{\mathsf{mw}(T,\delta)} \cdot 2n$	$2^{\mathsf{rw}_{\mathbb{Q}}(T,\delta) \log_2(2\mathsf{rw}_{\mathbb{Q}}(T,\delta)+1)} \cdot 2n$	$2^{2\mathsf{rw}(T,\delta)^2} \cdot 2n$	$2n^{2\mathsf{mim}(T,\delta)+1}$

We are now ready to give an adaptation of Theorem 3 to the notion of $(x, R')^{\mathsf{acy}}$-representativity.

▶ **Theorem 17.** *Let $R \in \mathcal{R}^2_{V_x}$. For each $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$, there exists an algorithm $\mathsf{reduce}^{\mathsf{acy}}_{\mathsf{f}}$ that, given a set $\mathcal{A}$ such that $X \equiv^2_{V_x} R$ for every $X \in \mathcal{A}$, outputs in time $O((\mathsf{nec}_1(V_x)^{2(\omega-1)} + \mathcal{N}_{\mathsf{f}}(T, \delta)) \cdot |\mathcal{A}| \cdot n^2)$, a subset $\mathcal{B} \subseteq \mathcal{A}$ such that $\mathcal{B}$ $(x, R')^{\mathsf{acy}}$-represents $\mathcal{A}$ and $|\mathcal{B}| \leq \mathcal{N}_{\mathsf{f}}(T, \delta) \cdot \mathsf{nec}_1(V_x)^2$.*

Now, The algorithm for any AC-(σ,ρ)-Dominating Set problem is essentially the same as the algorithm from Theorem 8, except that we use $\mathsf{reduce}_{\mathsf{f}}^{\mathsf{acy}}$ instead of reduce.

▶ **Theorem 18.** *For each* $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$*, there exists an algorithm that, given an* n*-vertex graph* G *and a rooted layout* (T,δ) *of* G*, solves any* AC-(σ,ρ)-Dominating Set *problem, in time* $O(\mathsf{s\text{-}nec}_d(T,\delta)^3 \cdot \mathsf{s\text{-}nec}_1(T,\delta)^{2(\omega+1)} \cdot \mathcal{N}_{\mathsf{f}}(T,\delta)^2 \cdot \log(\mathsf{s\text{-}nec}_d(T,\delta)) \cdot n^3)$*, with* $d := \max\{2, d(\sigma), d(\rho)\}$*.*

By constructing for any graph G a graph G' such that the width measure of G' is linear in the width measure of G, and any optimum acyclic (σ,ρ)-dominating set of G corresponds to an optimum AC-(σ,ρ)-dominating set of G' and vice-versa, we obtain the following which allows for instance to compute a feedback vertex set in time $n^{O(c)}$, c the mim-width.

▶ **Theorem 19.** *For each* $\mathsf{f} \in \{\mathsf{mw}, \mathsf{rw}, \mathsf{rw}_{\mathbb{Q}}, \mathsf{mim}\}$*, there exists an algorithm that, given an* n*-vertex graph* G *and a rooted layout* (T,δ) *of* G*, solves any* Acyclic (σ,ρ)-Dominating Set *problem in time* $O(\mathsf{s\text{-}nec}_d(T,\delta)^{O(1)} \cdot \mathcal{N}_{\mathsf{f}}(T,\delta)^{O(1)} \cdot n^3)$ *with* $d := \max\{2, d(\sigma), d(\rho)\}$*.*

6 Conclusion

Prior to this work, the d-neighbor-equivalence relation was used only for problems with a locally checkable property like Dominating Set [7, 13, 19]. We prove that the d-neighbor-equivalence relation can also be useful for problems with a connectivity constraint and an acyclicity constraint. Is this notion also useful for other kinds of problems? Can we use it for the problems which are unlikely to admit FPT algorithms parameterized by clique-width, $\mathbb{Q}$-rank-width or rank-width? This is the case for well-known problems such as Hamiltonian Cycle, Edge Dominating Set, and Max Cut. The complexity of these problems parameterized by clique-width is well-known. Indeed, for each of these problems, we have an ad-hoc $n^{O(k)}$ time algorithm with k the clique-width of a given k-expression [4, 11]. On the other hand, little is known concerning rank-width and $\mathbb{Q}$-rank-width. For mim-width, we know that Hamiltonian Cycle is para-NP-hard parameterized by the mim-width of a given rooted layout [15].

As these problems are W[1]-hard parameterized by clique-width, we cannot expect to rely only on the d-neighbor equivalence relation for d a constant. Maybe, we can avoid this dead-end by using the n-neighbor equivalence relation. In fact, we prove the following theorem for Max Cut.

▶ **Theorem 20.** *There exists an algorithm that, given an* n*-vertex graph* G *and a rooted layout* (T,δ)*, solves* Max Cut *in time* $O(\mathsf{s\text{-}nec}_n(T,\delta)^2 \cdot \log(\mathsf{s\text{-}nec}_n(T,\delta)) \cdot n^3)$*.*

Consequently, this theorem implies the existence of $n^{O(\mathsf{mw}(\mathcal{L}))}$, $n^{O(\mathsf{rw}_{\mathbb{Q}}(G))}$ and $n^{2^{O(\mathsf{rw}(G))}}$ time algorithms for Max Cut. Notice that, unless ETH fails, there are no $n^{o(\mathsf{mw}(\mathcal{L}))}$ and $n^{o(\mathsf{rw}_{\mathbb{Q}}(G))}$ time algorithms for Max Cut [11].

References

1 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoret. Comput. Sci.*, 511:54–65, 2013. doi:10.1016/j.tcs.2013.01.011.

2 Benjamin Bergougnoux. *Matrix Decomposition and Algorithmic Application to (Hyper)Graphs.* PhD thesis, Université Clermont Auvergne, 2019. lien vers chapitre.

3 Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parametrized by clique-width. *To appear at Theoretical Computer Science*, 2017. URL: `https://hal.archives-ouvertes.fr/hal-01560555`.

4 Benjamin Bergougnoux, Mamadou Moustapha Kanté, and O-joung Kwon. An Optimal XP Algorithm for Hamiltonian Cycle on Graphs of Bounded Clique-Width. In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, pages 121–132, 2017. `doi:10.1007/978-3-319-62127-2_11`.

5 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inform. and Comput.*, 243:86–111, 2015. `doi:10.1016/j.ic.2014.12.008`.

6 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-Width of Graphs. In Jianer Chen and Fedor V. Fomin, editors, *IWPEC*, volume 5917 of *Lecture Notes in Computer Science*, pages 61–74. Springer, 2009. `doi:10.1007/978-3-642-11269-0_5`.

7 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoret. Comput. Sci.*, 511:66–76, 2013. `doi:10.1016/j.tcs.2013.01.009`.

8 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

9 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time (extended abstract). In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science—FOCS 2011*, pages 150–159. IEEE Computer Soc., Los Alamitos, CA, 2011. `doi:10.1109/FOCS.2011.23`.

10 Reinhard Diestel. *Graph Theory*. Number 173 in Graduate Texts in Mathematics. Springer, third edition, 2005.

11 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. `doi:10.1137/130910932`.

12 Robert Ganian and Petr Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Appl. Math.*, 158(7):851–867, 2010. `doi:10.1016/j.dam.2009.10.018`.

13 Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve Hortemo Sæther, and Yngve Villanger. Output-Polynomial Enumeration on Graphs of Bounded (Local) Linear MIM-Width. *Algorithmica*, 80(2):714–741, 2018. `doi:10.1007/s00453-017-0289-1`.

14 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Polynomial-Time Algorithms for the Longest Induced Path and Induced Disjoint Paths Problems on Graphs of Bounded Mim-Width. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, pages 21:1–21:13, 2017. `doi:10.4230/LIPIcs.IPEC.2017.21`.

15 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A Unified Polynomial-Time Algorithm for Feedback Vertex Set on Graphs of Bounded Mim-Width. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 42:1–42:14, 2018. `doi:10.4230/LIPIcs.STACS.2018.42`.

16 Ki Hang Kim. *Boolean matrix theory and applications*, volume 70. Dekker, 1982.

17 Pedro Montealegre and Ioan Todinca. On Distance-d Independent Set and Other Problems in Graphs with "few" Minimal Separators. In *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, pages 183–194, 2016. `doi:10.1007/978-3-662-53536-3_16`.

18 Sang-Il Oum. *Graphs of Bounded Rank Width*. PhD thesis, Princeton University, 2005.

19 Sang-il Oum, Sigve Hortemo Sæther, and Martin Vatshelle. Faster algorithms for vertex partitioning problems parameterized by clique-width. *Theoret. Comput. Sci.*, 535:16–24, 2014. `doi:10.1016/j.tcs.2014.03.024`.

20 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

21 Michaël Rao. *Décompositions de Graphes et Algorithmes Efficaces*. PhD thesis, Université Paul Verlaine, Metz, 2006.

22 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math.*, 10(4):529–550, 1997. doi:10.1137/S0895480194275825.

23 Martin Vatshelle. *New width parameters of graphs*. PhD thesis, University of Bergen, Bergen, Norway, 2012.