Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems[☆]

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

[a] *CNRS – Université Paris 6, France.*
[b] *Department of Informatics, University of Bergen, Norway.*

Abstract

Given a graph G we provide dynamic programming algorithms for many locally checkable vertex subset and vertex partitioning problems. Their runtime is polynomial in the number of equivalence classes of problem-specific equivalence relations on subsets of vertices, defined on a given decomposition tree of G. Using these algorithms all these problems become solvable in polynomial time for many well-known graph classes like interval graphs and permutation graphs (Belmonte and Vatshelle [1]). Given a decomposition of boolean-width k we show that the algorithms will have runtime $O(n^4 2^{O(k^2)})$, providing the first large class of problems solvable in fixed-parameter single-exponential time in boolean-width.

1. Introduction

When solving graph problems by divide-and-conquer or by dynamic programming we need to recursively divide the input graph G. A natural way to do this is to recursively partition the vertices of the graph in two parts. The resulting decomposition of G can be stored as a full binary tree whose leaves are in bijection with the n vertices of G. In this paper we assume that we are given such a decomposition tree of G and focus on fast dynamic programming algorithms for a large class of locally checkable vertex subset and vertex partitioning problems. Depending on the problem being solved and the given decomposition tree we define equivalence relations on vertex subsets and give algorithms with runtime polynomial in n and the number of equivalence classes of these relations. In a companion paper by Belmonte and Vatshelle [1] it is shown that for many families of graphs, like permutation graphs, interval graphs, Dilworth k graphs, we can in polynomial time find a decomposition tree where the number of such equivalence classes is polynomial in n. Combined with the results in this paper we get on all those graph families polynomial-time algorithms solving the class of locally checkable vertex subset and vertex partitioning problems.

[☆]Supported by the Norwegian Research Council, project PARALGO.

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

Email addresses: `buixuan@lip6.fr` (Binh-Minh Bui-Xuan), `telle@ii.uib.no` (Jan Arne Telle), `vatshelle@ii.uib.no` (Martin Vatshelle)

σ	ρ	d	Standard name	VP	$\exists$	MAX	MIN
$\mathbb{N}$	$\{d,d+1,...\}$	d	d-Dominating set	3	P	P	NPC
$\{d\}$	$\mathbb{N}$	$d+1$	Induced d-Regular Subgraph	2	?	NPC	?
$\{d,d+1,...\}$	$\mathbb{N}$	d	Subgraph of Min Degree $\geq d$	?	P	P	NPC
$\{0,1,...,d\}$	$\mathbb{N}$	$d+1$	Induced Subg. of Max Degree $\leq d$	?	P	NPC	P
$\{0\}$	$\{0,1\}$	2	Strong Stable set or 2-Packing	4	P	NPC	P
$\{0\}$	$\{1\}$	2	Perfect Code or Efficient Dom. set	4	NPC	NPC	NPC
$\{0,1\}$	$\{0,1\}$	2	Total Nearly Perfect set	3	P	NPC	P
$\{0,1\}$	$\{1\}$	2	Weakly Perfect Dominating set	3	NPC	NPC	NPC
$\{1\}$	$\{1\}$	2	Total Perfect Dominating set	3	NPC	NPC	NPC
$\{1\}$	$\mathbb{N}$	2	Induced Matching	2	P	NPC	P
$\{1\}$	$\mathbb{N}^+$	2	Dominating Induced Matching	2	NPC	NPC	NPC
$\mathbb{N}$	$\{1\}$	2	Perfect Dominating set	2	P	P	NPC
$\{0\}$	$\mathbb{N}$	1	Independent set	3	P	NPC	P
$\mathbb{N}$	$\mathbb{N}^+$	1	Dominating set	3	P	P	NPC
$\{0\}$	$\mathbb{N}^+$	1	Independent Dominating set	3	P	NPC	NPC
$\mathbb{N}^+$	$\mathbb{N}^+$	1	Total Dominating set	2	P	P	NPC

Table 1: Some vertex subset properties expressible as (σ,ρ)-sets, with $\mathbb{N}=\{0,1,...\}$ and $\mathbb{N}^+=\{1,2,...\}$. Column d shows that we must count up to d neighbors. Column VP shows the smallest k for which the question of partition into k such sets is NP-complete. Columns $\exists$, MAX and MIN show complexity of existence, maximization and minimization over such sets, with P, NPC and ? denoting Polytime, NP-Complete and unknown (to us).

This class includes many well-known NP-hard problems related to domination, independence and homomorphism, and also their vertex weighted versions. For example, vertex subset problems like Perfect Code, Maximum Induced Matching, Minimum Perfect Dominating Set and in general existence and optimization problems over any of the vertex subset properties listed in Table 1. For fixed integer d problems like Minimum d-Dominating Set, Induced d-Regular Subgraph, Minimum Subgraph of Degrees $\geq d$, and d-Vertex Coloring. Also, the problem of deciding if the input graph has a partitioning of its vertex set into a fixed number q of sets each having a property listed in Table 1. For a fixed simple graph H also problems like H-Coloring, H-Homomorphism, H-Covering, H-Partial Covering, and H-Role Assignment, see Table 2, asking for a homomorphism, with some possible local constraints, from the input graph G to the target graph H.

These are optimization problems over locally checkable neighborhood conditions, as defined in Section 2. For example, in the Minimum d-Dominating Set problem we optimize over vertex subsets S of the input graph G such that any vertex not in S has at least d neighbors in S. To check this condition note that we must count S-neighbors up to d, but no further since it does not matter if a vertex has d neighbors in S or if it has more than d neighbors in S. In a bottom-up traversal of the decomposition tree T we solve the problem on induced subgraphs of G of increasing size. For the subgraph induced by $A \subseteq V(G)$ two subsets $X,Y \subseteq A$ will be *equivalent* w.r.t. the d-Dominating Set constraint if any vertex v

not in A either has the same number of neighbors in X and Y, or at least d in each. The number of equivalence classes nec_A^d of vertex subsets will in this way depend on the value d as given by the problem, on the various vertex subsets A as given by the decomposition tree T, and on the bipartite graphs induced by edges having exactly one endpoint in A as given by the graph G. In Section 5 we give dynamic programming algorithms solving locally checkable vertex subset problems in time polynomial in all the nec_A^d, and in Section 6 we extend this result to vertex partitioning problems.

Edges	d	Standard name	NP-Complete
$\mathbb{N}$	1	H-coloring or H-homomorphism	H bipartite
$\mathbb{N}^+$	1	H-role assignment or H-locally surj. hom.	H on 3 vertices or more
$\{1\}$	2	H-covering or H-locally bij. hom.	e.g. H k-regular for $k \geq 3$. Open
$\{0,1\}$	2	H-partial covering or H-locally inj. hom.	Harder than H-covering. Open

Table 2: Various homomorphism problems for fixed simple graph H. These are expressible as locally checkable vertex partitioning problems with the degree constraint matrix D_q being the adjacency matrix of H with 1-entries replaced by value in column Edges, 0-entries replaced by $\{0\}$, and $q = |V(H)|$. Column d shows that we must count up to d neighbors. NP-completeness known for fixed H having property listed in the last column [2], with dichotomy known for the first two rows.

As shown in the companion paper [1], for many families of intersection graphs, like convex graphs and trapezoid graphs, one can in polynomial time find a decomposition tree T such that nec_A^d is polynomial in n, for any subset A appearing as the leaves of a subtree of T, and any fixed value d. This implies polynomial-time algorithms for the locally checkable vertex subset and vertex partitioning problems on these families of intersection graphs. On the other hand, for graph families where at least one of these problems remains NP-hard we cannot expect the existence of decomposition trees with every nec_A^d polynomial in n. In such cases it is common to define a width parameter of graphs and apply the theory of fixed parameter algorithms. One can, for each value of d, define a width parameter of graphs that captures the minimum, over all decomposition trees T of a graph G, of the maximum nec_A^d for any $A \subseteq V(G)$ at the leaves of a subtree of T. For the value $d = 1$ the resulting width parameter is exactly $2^{boolw(G)}$, where $boolw(G)$ is the boolean-width of G. Moreover, we show in Section 7 that for any d and A we have $nec_A^d \leq (nec_A^1)^{d \times log_2(nec_A^1)}$. This implies that given a graph G and a decomposition tree of boolean-width $boolw$ our dynamic programming algorithms are single-exponential fixed-parameter tractable in $boolw$. For vertex subset problems the runtime becomes $O(n^4 2^{3d \times boolw^2})$ and for problems asking for a partition of the vertex set into q sets the runtime becomes $O(n^4 2^{3qd \times boolw^2})$.

Width parameters of graphs have many applications in the field of graph algorithms and especially in Fixed Parameter Tractable (FPT) algorithmics, see *e.g.* Downey and Fellows [3], Flum and Grohe [4], and Hliněný et al. [5]. Since the locally checkable vertex subset and vertex partitioning problems are expressible in monadic second-order logic it follows from the well-known theorem of Courcelle and Makowsky [6] that they are fixed-parameter tractable when parameterized by either the tree-width, branch-width, clique-width, rank-width or boolean-width of the input graph. Although the runtime resulting from this theorem contains a highly exponential factor (tower of powers), the problems behave very well for tree-width tw and branch-width bw: Given a decomposition tree of tree-width tw, they can be solved in $O^*(2^{O(tw)})$ and $O^*(2^{O(bw)})$ time [7]. This is not the same situation for clique-width cw, where until now the best runtime contained a $O^*(2^{2^{poly(cw)}})$ double exponential factor [8]. Having small boolean-width is witnessed by a decomposition of the graph into cuts with few different unions of neighborhoods across the cut. This makes the decomposition natural to guide dynamic programming algorithms to solve problems, like Maximum Independent Set, where vertex sets having the same neighborhoods can be treated as equivalent [9]. In this paper we extend such an observation to the much larger class of locally checkable vertex subset and vertex partitioning problems. As mentioned above, the runtime of our algorithms expressed by boolean-width $boolw$ of the given decomposition tree is $O^*(2^{O(boolw^2)})$, which can be interpreted as $O^*(2^{O(cw^2)})$ since for the clique-width cw resulting from this decomposition tree we have $cw \geq boolw$ [9]. For clique-width this improves by an exponential factor the best previous runtimes [8] and it provides for the first time a large class of problems for which dynamic programming gives runtime single exponential in boolean-width. It implies quasi-polynomial algorithms solving all these problems on random graphs, since it has been shown that a random graph on n vertices where the edges are drawn with respect to a uniform distribution almost surely has boolean-width $\Theta(\log^2 n)$, and it is easy to find a decomposition tree witnessing it [10]. On an arbitrary graph G a decomposition of optimal boolean-width can be computed in time $O(2.52^n)$ [11]. Heuristic algorithms finding decompositions for boolean-width compare well with heuristics for tree-width, in particular for dense graphs [12], opening for the possibility of a practical application of the algorithms given here.

The paper is organized as follows. In Section 2 we define the class of problems and the decomposition trees used. In Section 3 we give an intuitive description of our algorithms, using the Maximum Induced Matching problem as example. In Section 4 we give a pre-processing step computing representatives to be used as indices of the dynamic programming. In Section 5 we give algorithms for vertex subset problems and in Section 6 algorithms for vertex partitioning problems. In Section 7 we define boolean-width and show its relation to nec_A^d that allows us to express runtime of our algorithms as a function of boolean-width. We end in Section 8 with some conclusions and open problems.

2. Locally checkable problems and rooted decomposition trees

We deal with simple, undirected graphs. The complement of a vertex subset A of a graph $G = (V(G), E(G))$ is denoted by $\overline{A} = V(G) \setminus A$. The neighborhood of a vertex x is denoted by $N(x)$ and for a subset of vertices X we denote the union of their neighborhoods by $N(X) = \bigcup_{x \in X} N(x)$. We denote by $G[X]$ the graph induced by $X \subseteq V(G)$. To ensure uniqueness of certain algorithms, *e.g.* for computing representatives of the equivalence relations on vertex subsets, we assume a total ordering σ on the vertex set of G which stays the same throughout the entire paper. For easy disambiguation, we usually refer to vertices of a graph and nodes of a tree.

We want to solve graph problems using a divide-and-conquer approach. To this aim, we need to store the information on how to recursively divide the input graph instance. A standard way to do this is to use a decomposition tree, for our purposes a rooted tree.

Definition 1. A decomposition tree of a graph G is a pair (T, δ) where T is a full binary tree (i.e. T rooted with every non-leaf having two children) and δ a bijection between the leaf set of T and the vertex set of G. For a node a of T let the subset of $V(G)$ in bijection δ with the leaves of the subtree of T rooted at a be denoted by V_a.

We will be interested in the following problems as defined in [7].

Definition 2. Let σ and ρ be finite or co-finite subsets of natural numbers. A subset S of vertices of a graph G is a *sigma-rho set*, or simply (σ, ρ)*-set*, of G if

$$\forall v \in V(G) : |N(v) \cap S| \in \begin{cases} \sigma & \text{if } v \in S, \\ \rho & \text{if } v \in V(G) \setminus S. \end{cases}$$

Table 1 shows some classical vertex subset properties expressed as (σ, ρ)-sets. The class of *locally checkable vertex subset problems* consist of finding a minimum or maximum (σ, ρ)-set in an input graph G, possibly on vertex-weighted graphs. This includes many NP-hard problems as indicated in Table 1. For NP-completeness results see [13, 14, 15, 16, 17, 18].

Since σ and ρ are either finite or co-finite we can check locally if S is a (σ, ρ)-set by counting for each vertex v the number of S-neighbors only up to $d(\sigma, \rho)$, defined as follows.

Definition 3. Let $d(\mathbb{N}) = 0$. For every finite or co-finite set $\mu \subseteq \mathbb{N}$, let $d(\mu) = 1 + min(max_{x \in \mathbb{N}} x : x \in \mu, max_{x \in \mathbb{N}} x : x \notin \mu)$. Let $d(\sigma, \rho) = max(d(\sigma), d(\rho))$.

For example, $d(\{1\}) = 2$ which is one more than the largest number contained in the finite set $\{1\}$, while $d(\{1, 2, ...\}) = 1$ which is one more than the largest number not contained in the co-finite set $\{1, 2, ...\}$, and $d(\{1\}, \{1, 2, ...\}) = 2$ which is the maximum of $d(\{1\})$ and $d(\{1, 2, ...\})$.

We can also ask for a partition of $V(G)$ into q classes, with each class satisfying a certain (σ, ρ)-property, as follows.

Definition 4. A *degree constraint* matrix D_q is a q by q matrix with entries being finite or co-finite subsets of natural numbers. A D_q*-partition* in a graph G is a partition $\{V_1, V_2, ..., V_q\}$ of $V(G)$ such that for $1 \leq i, j \leq q$ we have $\forall v \in V_i : |N(v) \cap V_j| \in D_q[i, j]$.

The *locally checkable vertex partitioning problems* consist of deciding if G has a D_q partition. NP-hard problems fitting into this framework include the question of deciding if an input graph has a partition into q (σ, ρ)-sets, which is in most cases NP-complete for small values of q, see the column VP in Table 1. It also includes for any fixed graph H the homomorphism problems listed in Table 2. Let us mention that extending the algorithms we give here to handle also the case of finding an extremal value (maximum or minimum) of the cardinality of a vertex partition class over all D_q-partitions is straightforward.

3. Detailed example: Maximum Induced Matching

We first describe our algorithms intuitively, taking as our main example the vertex subset maximization problem over (σ, ρ)-sets with $\sigma = \{1\}$ and $\rho = \mathbb{N}$. We thus want to compute the cardinality of a maximum set of vertices S such that in $G[S]$ all vertices have degree one, the so-called Maximum Induced Matching problem. In a bottom-up traversal of T we will solve the problem on induced subgraphs of increasing size, at node a of T storing information on partial solutions to the problem on $G[V_a]$ in a table Tab_a.

A partial solution will have two parts (S_a, Z_a) where

- $S_a \subseteq V_a$ such that in $G[S_a]$ all vertices (of S_a) have degree *at most one*
- $Z_a \subseteq \overline{V_a}$ such that in $G[S_a \cup Z_a]$ vertices of S_a have degree *exactly one*

We call (S_a, Z_a) a partial solution to the Max Induced Matching problem on $G[V_a]$, in which Z_a is a witness of a necessary, but not sufficient, condition for S_a to be extendible into an induced matching of G.

The number of partial solutions could be exponential in n but many of them are superfluous. If two subsets $Z_a, Y_a \subseteq \overline{V_a}$ have the property that for every $v \in V_a$ the vertex v has either zero neighbors in each of Z_a and Y_a, or exactly one neighbor in each, or at least two neighbors in each, then it is not hard to check that for the Maximum Induced Matching problem (S_a, Z_a) is a partial solution if and only if (S_a, Y_a) is a partial solution, and one of the two will be superfluous. This motivates the following equivalence relation on subsets of vertices, which applies to the general σ, ρ case using $d(\sigma, \rho)$-neighbor equivalence. For the Maximum Induced Matching problem we have $d(\{1\}, \mathbb{N}) = 2$.

Definition 5 (d-neighbor equivalence)**.** Let d be a non-negative integer, G be a graph and $A \subseteq V(G)$. Two vertex subsets $X \subseteq A$ and $Y \subseteq A$ are *d-neighbor equivalent* w.r.t. A, denoted by $X \equiv^d_A Y$, if:

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

Let $nec(\equiv^d_A)$ be the number of equivalence classes of $\equiv^d_A$.

In the example above we had $Z_a \equiv^2_{\overline{V_a}} Y_a$ and for fixed S_a we need to store a partial solution (S_a, Z_a) for at most one member of the equivalence class of Z_a. A similar thing applies to S_a, if $S_a \equiv^2_{V_a} S'_a$ and both (S_a, Z_a) and (S'_a, Z_a) are partial solutions and $|S_a| \geq |S'_a$ then (S'_a, Z_a) is superfluous since we are solving a maximization problem. In light of this we index the table Tab_a of partial solutions at node a of T by the Cartesian product of the two sets of equivalence classes $\equiv^2_{V_a} \times \equiv^2_{\overline{V_a}}$ (or rather by representatives of these classes) and store the following information:

$$Tab_a[X][Y] \overset{\text{def}}{=} max_{S \subseteq V_a}\{|S| : S \equiv^2_{V_a} X \text{ and in } G[S \cup Y] \text{ all vertices of } S \text{ have degree } 1\}$$

or minus ∞ if no such S exists. Note that (S, Y) is a partial solution for $G[V_a]$. In this way we contain the runtime by a function of the number of equivalence classes $nec(\equiv^2_{V_a})$ and $nec(\equiv^2_{\overline{V_a}})$.

It is instructive to consider in some detail the initialization of table entries at leaf a of T associated to vertex $\delta(a) = v \in V(G)$. In that case we have $V_a = \{v\}$ with two classes of $\equiv^2_{V_a}$ (assuming v has at least one neighbor) with representatives $\{v\}$ and $\emptyset$, we have $\overline{V_a} = V(G) \setminus \{v\}$ with three classes of $\equiv^2_{\overline{V_a}}$ (assuming v has at least two neighbors) with representatives R_0, R_1 and R_2. The class of R_i for $i = 0, 1$ consists of those subsets of $V(G) \setminus \{v\}$ containing i vertices of $N(v)$, and for $i = 2$ those containing ≥ 2 vertices of $N(v)$. According to the definition of table entries we initialize as follows.

- $Tab_a[\emptyset][R_0] = Tab_a[\emptyset][R_1] = Tab_a[\emptyset][R_2] = 0$
- $Tab_a[\{v\}][R_0] = Tab_a[\{v\}][R_2] = -\infty$
- $Tab_a[\{v\}][R_1] = 1$

At the root r of T we have $V_r = V(G)$ with a single equivalence class of $\equiv^2_{V_r}$, and we have $\overline{V_r} = \emptyset$ with a single equivalence class of $\equiv^2_{\overline{V_r}}$. The single entry of Tab_r will contain the cardinality of the largest $S \subseteq V(G)$ such that in $G[S]$ all vertices have degree 1, i.e. the solution to the Maximum Induced Matching problem.

At the combine step we have a node w of T with children a, b such that $V_w = V_a \cup V_b$ and we compute partial solutions to $G[V_a \cup V_b]$ based on already computed partial solutions to $G[V_a]$ and $G[V_b]$. For any dynamic programming on decomposition trees it is important to keep in mind the below observation, that follows directly from definitions.

Observation 1. *If in the tree T node w has children a and b then $\{V_a, V_b, \overline{V_w}\}$ forms a 3-partition of $V(G)$.*

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

Observation 2. *Let d be a non-negative integer, G be a graph with $V_a \subseteq V_w \subseteq V(G)$ and let $X, Y \subseteq V_a$. If $X \equiv^d_{V_a} Y$ then $X \equiv^d_{V_w} Y$.*

In particular we have that if $X \equiv^2_{V_a} Y$ or $X \equiv^2_{V_b} Y$ then $X \equiv^2_{V_w} Y$, so that $\equiv^2_{V_w}$ is a coarsening of the two relations $\equiv^2_{V_a}$ and $\equiv^2_{V_b}$. Likewise $\equiv^2_{\overline{V_a}}$ and $\equiv^2_{\overline{V_b}}$ are coarsenings of $\equiv^2_{\overline{V_w}}$. The algorithm will iterate over all triples of representatives $R_a, R_b, R_{\overline{w}}$ from the three most refined equivalence relations, $\equiv^2_{V_a}$, $\equiv^2_{V_b}$, $\equiv^2_{\overline{V_w}}$, compute the representatives of the resulting coarser relations: R_w of $\equiv^2_{V_w}$ (from R_a and R_b), $R_{\overline{a}}$ of $\equiv^2_{\overline{V_a}}$ (from R_b and $R_{\overline{w}}$), $R_{\overline{b}}$ of $\equiv^2_{\overline{V_b}}$ (from R_a and $R_{\overline{w}}$), and then update $Tab_w[R_w][R_{\overline{w}}]$ by $Tab[R_a][R_{\overline{a}}] + Tab[R_b][R_{\overline{b}}]$. For correctness it will be crucial that for any $S_a \equiv^2_{V_a} R_a$ and $S_b \equiv^2_{V_b} R_b$ the following holds: If in $G[S_a \cup R_b \cup R_{\overline{w}}]$ all vertices of S_a have degree 1 and in $G[S_b \cup R_a \cup R_{\overline{w}}]$ all vertices of S_b have degree 1, then in $G[S_a \cup S_b \cup R_{\overline{w}}]$ all vertices in $S_a \cup S_b$ will have degree 1. We refer to the formal description and correctness proof for details, see Figure 3.

The case of $\rho \neq \mathbb{N}$ is handled similarly. For example, consider Maximum Dominating Induced Matching for which it is NP-complete simply to decide if such a set exists. In this case we maximize over $\sigma = \{1\}, \rho = \mathbb{N}^+$ sets. Partial solutions (S_a, Z_a) must now

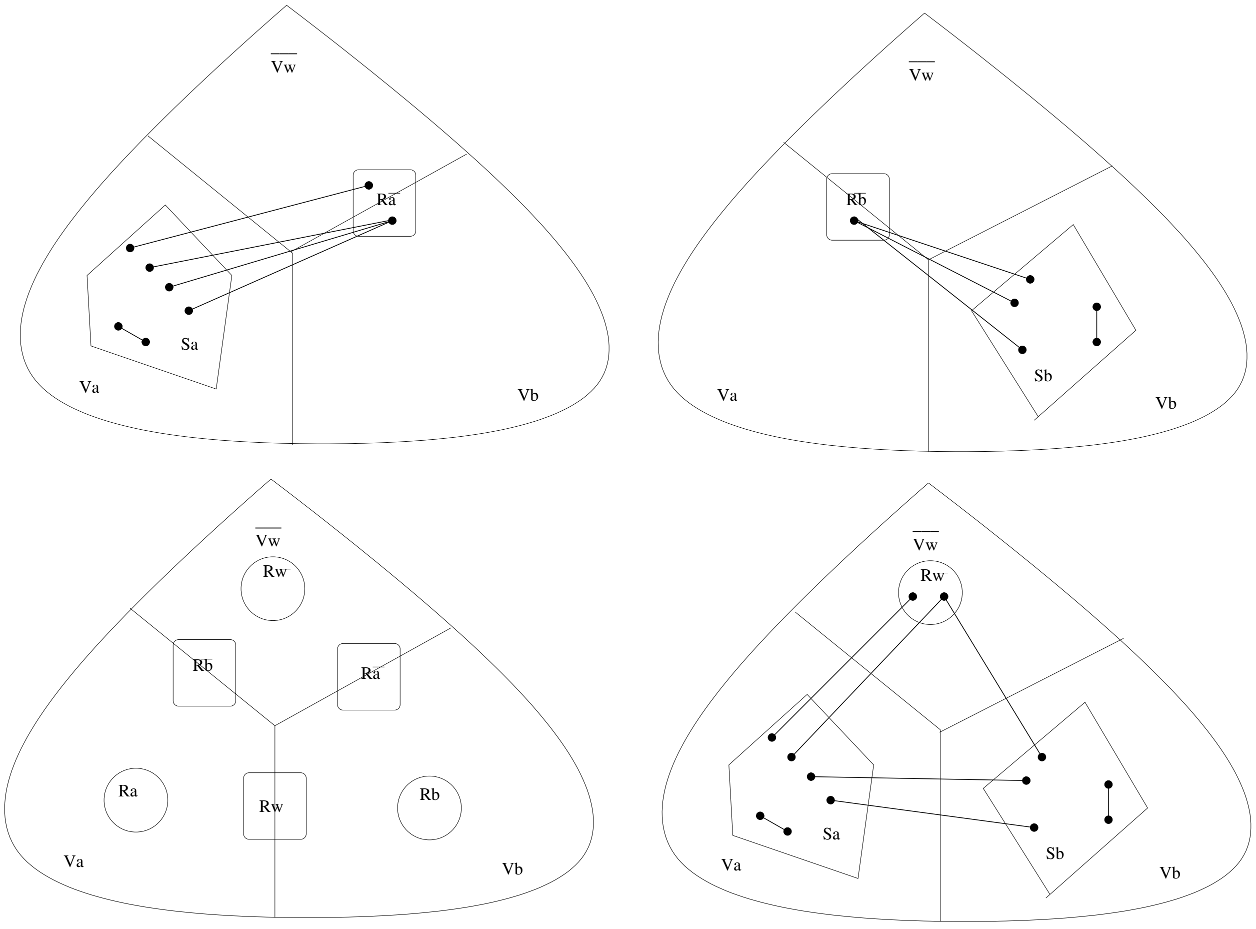

Figure 1: Solving Maximum Induced Matching by dynamic programming using the 2-neighbor equivalence relation on vertex subsets. At inner node w of T with children a, b we have partition $V_a, V_b, \overline{V_w}$ of $V(G)$. Bottom left shows three arbitrary representatives $R_a, R_b, R_{\overline{w}}$ of refined relations as circles, and as boxes the computed representatives $R_{\overline{a}}, R_{\overline{b}}, R_w$ of the resulting coarser relations. Top line shows partial solutions $(S_a, R_{\overline{a}})$ from $Tab_a[R_a][R_{\overline{a}}]$ and $(S_b, R_{\overline{b}})$ from $Tab_b[R_b][R_{\overline{b}}]$. Bottom right shows resulting partial solution $(S_a \cup S_b, R_{\overline{w}})$ from $Tab_w[R_w][R_{\overline{w}}]$.

also ensure that in $G[S_a \cup Z_a]$ vertices in $V_a \setminus S_a$ have degree at least one. The fact that $d(\mathbb{N}^+) = 1 < d(\{1\}) = 2$ does not matter, as the 1-neighbor equivalence relation is a coarsening of the 2-neighbor equivalence relation.

4. Computing representatives of d-neighbor equivalence classes

Before explaining the dynamic programming we show how to compute representatives of the $\equiv^d_{V_w}$ and $\equiv^d_{\overline{V_w}}$ relations used for indexing the table Tab_w at node w of the tree T. We denote by $rep^d_{V_w}(X)$ the representative of $X \subseteq V_w$ in the relation $\equiv^d_{V_w}$, and by $rep^d_{\overline{V_w}}(Y)$ the representative of $Y \subseteq \overline{V_w}$ in the relation $\equiv^d_{\overline{V_w}}$. For simplicity we define this using V_w instead of a generic subset A, but note that everything we say about $X \subseteq V_w$, $rep^d_{V_w}(X)$ and $\equiv^d_{V_w}$ will hold also for $rep^d_{\overline{V_w}}(Y)$, $Y \subseteq \overline{V_w}$ and $\equiv^d_{\overline{V_w}}$.

Definition 6 (representative)**.** We assume that a total ordering of the vertices of $V(G)$ is given. For every $X \subseteq V_w$, the representative $rep^d_{V_w}(X)$ is defined as the lexicographically smallest set $R \subseteq V_w$ such that $|R|$ is minimized and $R \equiv^d_{V_w} X$.

To check algorithmically if two subsets of V_w are equivalent w.r.t. V_w we use the d-neighborhood vector w.r.t. V_w defined as follows.

Definition 7 (d-neighborhood)**.** For $X \subseteq A \subseteq V(G)$, the d-neighborhood of X w.r.t. A, denoted $N^d_A(X)$, is a multiset of nodes from $\overline{A}$, such that, $\forall v \in \overline{A}$ the number of occurrences of v in $N^d_A(X)$ is equal to $\min\{|N(v) \cap X|, d\}$. Since we have assumed a fixed ordering of the vertices we will store $N^d_A(X)$ as an $|\overline{A}|$-vector over $\{0, 1, ..., d\}$.

Note that $X \equiv^d_A X'$ if and only if $N^d_A(X) = N^d_A(X')$.

Lemma 1. *For any node w of T we can compute a list LR_{V_w} containing all representatives w.r.t. $\equiv^d_{V_w}$ in time $O(nec(\equiv^d_{V_w}) \cdot \log(nec(\equiv^d_{V_w})) \cdot n^2)$.*

We also compute a data structure that given $X \subseteq V_w$, in time $O(\log(nec(\equiv^d_{V_w})) \cdot |X| \cdot n)$ will allow us to find a pointer to $rep^d_{V_w}(X)$ in LR_{V_w}.

Proof. Algorithm 1 computes the list LR_{V_w} and the list LNR_{V_w} of d-neighborhoods w.r.t. V_w of the members of LR_{V_w}. Before adding a representative R to the list LR_{V_w} we check if the list LNR_{V_w} contains the d-neighborhood $N^d_{V_w}(R)$. Therefore all elements of the list LR_{V_w} have different d-neighbourhoods. All the representatives added to the list LR_{V_w} are also expanded by any of the vertices of V_w. Assume for contradiction that X is a minimal representative such that $N^d_{V_w}(X)$ is not in the list LNR_{V_w}. Then $\forall u \in X$ we have: $\forall Y \in LNR_{V_w} : X \setminus u \not\equiv Y$ since then $N^d_{V_w}(X)$ would have been added to LNR_{V_w}. Meaning that $N^d_{V_w}(X \setminus u)$ is not in LNR_{V_w} contradicting that X is minimal.

The total number of representatives to be added to LR_{V_w} and d-neighborhoods added to LNR_{V_w} is $nec(\equiv^d_{V_w})$. The total number of possible representatives R' to be considered is $nec(\equiv^d_{V_w}) \cdot n$. Computing the union $R \cup \{v\}$ and the d-neighbouthood $N^d_{V_w}(R')$ can be done in $O(n)$ time by copying the d-neigborhood vector of R and updating the entries

Algorithm 1 List of representatives and their d-neighborhood

```
INPUT: Graph $G$, subset $A \subseteq V(G)$ and integer $d$
Initialize $LR_A$, $LNR_A$, $NextLevel$ to be empty
Initialize $LastLevel = \{\emptyset\}$
while LastLevel != $\emptyset$ do
  for $R$ in LastLevel do
    for every vertex $v$ of $A$ do
      $R' = R \cup \{v\}$
      compute $N' = N^d_A(R')$
      if $R' \not\equiv^d_A R$ and $N'$ is not contained in $LNR_A$ then
        add $R'$ to both $LR_A$ and $NextLevel$
        add $N'$ to $LNR_A$ at the proper position
        add pointers between $R'$ and $N'$
      end if
    end for
  end for
  set $LastLevel = NextLevel$, and $NextLevel = \emptyset$
end while
OUTPUT: $LR_A$ and $LNR_A$
```

for vertices in $N(v) \cap \overline{V_w}$. If we realize the list LNR_{V_w} as a balanced binary search tree checking for containment can be done in $O(\log(nec(\equiv^d_{V_w})) \cdot n)$. Inserting into the list LR_{V_w} can be done in constant time. So in total the construction of LR_{V_w} and LNR_{V_w} takes time $O(nec(\equiv^d_{V_w}) \cdot \log(nec(\equiv^d_{V_w})) \cdot n^2)$.

Given a subset $X \subseteq V_w$ we can generate the d-neighborhood $N^d_{V_w}$ in $O(|X| \cdot n)$ time. Then we can binary search in the list LNR_{V_w} to find a pointer to the representative in time $O(\log(nec(\equiv^d_{V_w})) \cdot |X| \cdot n)$. □

5. Dynamic programming for vertex subset problems

We show in this section how to apply dynamic programming on a decomposition tree (T, δ) of a graph G to solve a (σ, ρ) locally checkable vertex subset optimization problem. Note that we do not assume any further information from the input of (T, δ) other than T being a tree with internal nodes of degree three and δ a bijection between its leaves and $V(G)$. As is customary, we let the algorithm follow a bottom-up traversal of T.

With each node w of T we associate a table data structure Tab_w that will store partial solutions to the problem we are solving. Note that we must satisfy the constraint imposed both by σ and by ρ and that we must account for the domination both 'from the inside', i.e. from V_w, and also the expectation 'from the outside', i.e. from $\overline{V_w}$. This motivates the following definitions.

Definition 8. Let G be a graph, $A \subseteq V(G)$, and $\mu \subseteq \mathbb{N}$. For $X \subseteq V(G)$, we say that X μ-dominates A if $\forall v \in A : |N(v) \cap X| \in \mu$. For $X \subseteq A$, $Y \subseteq \overline{A}$, the pair (X, Y) *σ, ρ-dominates* A if $(X \cup Y)$ σ-dominates X and $(X \cup Y)$ ρ-dominates $A \setminus X$.

Letting $d = d(\sigma, \rho)$ we note that for $Y \equiv^d_{\overline{A}} Y'$ we have $N^d_{\overline{A}}(Y) = N^d_{\overline{A}}(Y')$, i.e. the d-neighborhood of Y and Y' w.r.t. $\overline{A}$ are equal, see Definition 7. This proves the following lemma showing that σ, ρ-domination behaves well w.r.t. the $d(\sigma, \rho)$-neighbor equivalence relation.

Lemma 2. *Let G be a graph and $A \subseteq V(G)$ and σ, ρ finite or co-finite subsets of non-negative integers with $d(\sigma, \rho) = d$. Let $X \subseteq A$, $Y, Y' \subseteq \overline{A}$, and $Y \equiv^d_{\overline{A}} Y'$. Then (X, Y) σ, ρ-dominates A if and only if (X, Y') σ, ρ-dominates A.*

The index set of the table Tab_w at w will be $LR_{V_w} \times LR_{\overline{V_w}}$ and its contents is defined as follows.

Definition 9. Let *opt* stand for either function *max* or function *min*, depending on whether we are looking for a maximum or minimum (σ, ρ)-set, respectively. For every node w of T, for $X \subseteq V_w$ and $Y \subseteq \overline{V_w}$, let $R_X = rep^d_{V_w}(X)$ and $R_Y = rep^d_{\overline{V_w}}(Y)$. Let $d = d(\sigma, \rho)$. We define the contents of $Tab_w[R_X][R_Y]$ as:

$$Tab_w[R_X][R_Y] \overset{\text{def}}{=} \begin{cases} opt_{S \subseteq V_w}\{|S| : S \equiv^d_{V_w} R_X \text{ and } (S, R_Y)\ \sigma, \rho\text{-dominates } V_w\}, \\ -\infty \text{ if no such set } S \text{ exists and } opt = max, \\ +\infty \text{ if no such set } S \text{ exists and } opt = min. \end{cases}$$

Note that Tab_w has $nec(\equiv^d_{V_w}) \times nec(\equiv^d_{\overline{V_w}})$ entries, cf. Definition 5. At the root r of T the value of $Tab_r[rep^d_{V_r}(X)][\emptyset]$ (for all $X \subseteq V(G)$) will be the size of a maximum, resp. minimum, (σ, ρ)-set of G (cf. $V_r = V(G)$ and $\equiv^d_{V_r}$ has only one equivalence class).

For initialization, firstly, for every node w of T the value of every entry of Tab_w will be set to $+\infty$ or $-\infty$ depending on whether we are solving a minimization or maximization problem, respectively.

Updating the leaves: For a leaf l of T, we then perform the following update: letting $\delta(l) = v \in V(G)$, for every representative R w.r.t. $\equiv^d_{V(G)\setminus\{v\}}$, we set:

- If $|N(v) \cap R| \in \sigma$ then $Tab_l[\{v\}][R] = 1$.
- If $|N(v) \cap R| \in \rho$ then $Tab_l[\emptyset][R] = 0$.

Updating the internal nodes: In a bottom-up traversal of the tree T, for an inner node w of T with children a and b, the algorithm proceeds as follows.

- Loop over all triples $R_{\overline{w}} \in LR^d_{\overline{V_w}}$, $R_a \in LR^d_{V_a}$, $R_b \in LR^d_{V_b}$ and do:

  Compute $R_w = rep^d_{V_w}(R_a \cup R_b)$, $R_{\overline{a}} = rep^d_{\overline{V_a}}(R_b \cup R_{\overline{w}})$ and $R_{\overline{b}} = rep^d_{\overline{V_b}}(R_a \cup R_{\overline{w}})$

  Update $Tab_w[R_w][R_{\overline{w}}] = opt(Tab_w[R_w][R_{\overline{w}}], Tab_a[R_a][R_{\overline{a}}] + Tab_b[R_b][R_{\overline{b}}])$.

The following lemma will be useful in the correctness proof of this update.

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

Proof. Let $S = S_a \cup S_b \cup S_w$. Consider $x \in S_a, x' \in A \setminus S_a$ and $y \in S_B, y' \in B \setminus S_b$. By Definition 8 $(S_a, S_b \cup S_w)$ σ, ρ-dominates A iff for every such x, x' we have $|N(x) \cap S| \in \sigma$ and $|N(x') \cap S| \in \rho$. Also, $(S_b, S_a \cup S_w)$ σ, ρ-dominates B iff for every such y, y' we have $|N(y) \cap S| \in \sigma$ and $|N(y') \cap S| \in \rho$.

Again, by Definition 8 $(S_a \cup S_B, S_w)$ σ, ρ-dominates $A \cup B$ iff for all $z \in S_a \cup S_b$ and $z' \in (A \cup B) \setminus (S_a \cup S_b)$ we have $|N(z) \cap S| \in \sigma$ and $|N(z') \cap S| \in \rho$, to finish the proof. □

Lemma 4. *The table of an inner node is updated correctly.*

Proof. Let node w have children a, b and assume Tab_a, Tab_b have been filled correctly. We show that after executing the update at node w the table Tab_w is filled according to Definition 9. We first show show that if $Tab_w[R_w][R_{\overline{w}}] = k$ then there exists $S_w \subseteq V_w$ with $|S_w| = k$ and $S_w \equiv^d_{V_w} R_w$ such that $(S_w, R_{\overline{w}})$ σ, ρ-dominates V_w in G. For this note that, based on the update step and assumed correctness of children tables, there must exist indices $R_a \in LR_{V_a}$ and $R_b \in LR_{V_b}$, with $S_a \equiv^d_{V_a} R_a$ and $S_b \equiv^d_{V_b} R_b$ such that $(S_a, R_{\overline{a}})$ σ, ρ-dominates V_a, and $(S_b, R_{\overline{b}})$ σ, ρ-dominates V_b, and $|S_a \cup S_b| = s$, and with $R_{\overline{a}} = rep^d_{\overline{V_a}}(R_b \cup R_{\overline{w}})$ and $R_{\overline{b}} = rep^d_{\overline{V_b}}(R_a \cup R_{\overline{w}})$. We claim that $S_a \cup S_b$ is the desired S_w. Since $(S_b \cup R_{\overline{w}}) \equiv^d_{\overline{V_a}} R_{\overline{a}}$ and $(S_a, R_{\overline{a}})$ σ, ρ-dominates V_a it follows from Lemma 2 that $(S_a, S_b \cup R_{\overline{w}})$ σ, ρ-dominates V_a. Likewise, $(S_b, S_a \cup R_{\overline{w}})$ σ, ρ-dominates V_b. We deduce from Lemma 3 that $(S_a \cup S_b, R_{\overline{w}})$ σ, ρ-dominates $V_a \cup V_b = V_w$. It remains to show that $S_a \cup S_b \equiv^d_{V_w} R_w$. By Observation 2 we have $R_a \equiv^d_{V_w} S_a$ and $R_b \equiv^d_{V_w} S_b$ so that $R_a \cup R_b \equiv^d_{V_w} S_a \cup S_b$. Since we assumed $R_a \cup R_b \equiv^d_{V_w} R_w$ we therefore have $S_a \cup S_b \equiv^d_{V_w} R_w$ as desired.

For the other direction, we need to show for every $R_w \in LR_{V_w}$ and $R_{\overline{w}} \in LR_{\overline{V_w}}$ that if there is a set $S_w \equiv^d_{V_w} R_w$ such that $(S_w, R_{\overline{w}})$ σ, ρ-dominates V_w, then after the update the value of $Tab_w[R_w][R_{\overline{w}}]$ is $\leq |S_w|$ if $opt = min$ and $\geq |S_w|$ if $opt = max$. Let $S_a = S_w \cap V_a$ and $S_b = S_w \cap V_b$. The algorithm loops over all triples of representatives: at some point it will check $(R_a, R_b, R_{\overline{w}})$, where $R_a = rep^d_{V_a}(S_a)$ and $R_b = rep^d_{V_b}(S_b)$. We know that $(S_a \cup S_b, R_{\overline{w}})$ σ, ρ-dominates V_w so it follows from Lemma 3 that $(S_a, S_b \cup R_{\overline{w}})$ σ, ρ-dominates V_a. Note that $R_{\overline{a}}$ as computed in the update satisfies $R_{\overline{a}} \equiv^d_{\overline{V_a}} (S_b \cup R_{\overline{w}})$ so that it follows from Lemma 2 that $(S_a, R_{\overline{a}})$ σ, ρ-dominates V_a. Hence, after the update the value of $Tab_a[R_a][R_{\overline{a}}]$ will be $\leq |S_a|$ if $opt = min$ and $\geq |S_a|$ if $opt = max$. Arguing analogously we have that the value of $Tab_b[R_b][R_{\overline{b}}]$ will be $\leq |S_b|$ if $opt = min$ and $\geq |S_b|$ if $opt = max$. Thus, to conclude that the value of $Tab_w[R_w][R_{\overline{w}}]$ will be $\leq |S_a| + |S_b| = |S_w|$ if $opt = min$ and $\geq |S_a| + |S_b| = |S_w|$ if $opt = max$ all we need to show is that $R_w \equiv^d_{V_w} R_a \cup R_b$. By Observation 2 we have $R_a \equiv^d_{V_w} S_a$ and $R_b \equiv^d_{V_w} S_b$ so that $R_a \cup R_b \equiv^d_{V_w} S_a \cup S_b$. Since $S_w = S_a \cup S_b$ and we assumed $R_w \equiv^d_{V_w} S_w$ we therefore have $R_a \cup R_b \equiv^d_{V_w} R_w$ as desired. □

Theorem 1. *For every n-vertex graph G given along with a decomposition tree (T,δ), with $nec_d(T,\delta)$ the maximum $nec(\equiv^d_{V_w})$ and $nec(\equiv^d_{\overline{V_w}})$ over all nodes w of this tree, any (σ,ρ)-vertex subset problem on G with $d = d(\sigma,\rho)$ can be solved in time $O(n^4 \cdot nec_d(T,\delta)^3)$.*

Proof. Correctness follows by structural induction on the tree T using Lemma 4 with the base case being the leaf initialization, so that at the root r of T the single index of the table Tab_r will contain the size of the optimal (σ,ρ) set in G. For complexity analysis, for every node w of T, we basically call the first computation of Lemma 1 once, then loop through every triplet $R_{\overline{w}}$, R_a, R_b of representatives, and there are at most $nec_d(T,\delta)^3$ such triplets. For each triplet we call the second computation of Lemma 1 three times, and since $|R_{\overline{w}}|, |R_a|, |R_b|$ and $\log(nec_d(T,\delta))$ all are at most n, we can perform the table update in $O(n^3)$ time. □

If the input graph G comes with a weight function on the vertices $w : V(G) \rightarrow \mathbb{R}$ we may wish to find the (σ,ρ) set with largest sum of weights, or with smallest sum of weights. This can be accomplished in the same runtime and requires only a very small change to the algorithm. For $S \subseteq V(G)$ let $w(S) = \Sigma_{v \in S} w(v)$. The tables must be defined to store the optimum value over $w(S)$ rather than over $|S|$ and the algorithms remain the same apart from the leaf initialization.

6. Dynamic programming for vertex partitioning problems

We show in this section how to apply dynamic programming on a decomposition tree (T,δ) of a graph G to solve a locally checkable vertex partitioning problem defined by a degree constraint matrix D_q of finite and co-finite entries, see Definition 4. Let $d = d(D_q) = \max_{i,j} d(D_q[i,j])$.

Definition 10. Let G be a graph and let $A \subseteq V(G)$ be a vertex subset of G. Two q-tuples $(X_1, X_2, ..., X_q)$ and $(Y_1, Y_2, ..., Y_q)$ of subsets of A are d-neighbor equivalent w.r.t. A, denoted by $(X_1, X_2, ..., X_q) \equiv^{q,d}_A (Y_1, Y_2, ..., Y_q)$, if:

$$\forall i \forall v \in \overline{A} : \min(d, |N(v) \cap X_i|) = min(d, |N(v) \cap Y_i|)$$

Observation 3. *$(X_1, X_2, ..., X_q) \equiv^{q,d}_A (Y_1, Y_2, ..., Y_q)$ if and only if $\forall i X_i \equiv^d_A Y_i$. A consequence is that the number of equivalence classes of $\equiv^{q,d}_A$ is at most $nec(\equiv^d_A)$ to the power q.*

This Observation follows directly from Definitions 5 and 10. The dynamic programming algorithm will again follow a bottom-up traversal of T and maintain a table at each node of T with partial solutions. In the sequel we will define the values of Tab_w directly indexed by the equivalence classes. For this we need to first define representatives. For a node w of T, and $\mathcal{X} = (X_1, X_2, ..., X_q) : X_i \subseteq V_w$, we define $rep^{q,d}_{V_w}(\mathcal{X}) = (rep^d_{V_w}(X_1), rep^d_{V_w}(X_2), ..., rep^d_{V_w}(X_q))$.

Definition 11. Let G be a graph and $A \subseteq V(G)$. Let $\mathcal{X} = (X_1, X_2, ..., X_q) \in A^q$ and $\mathcal{Y} = (Y_1, Y_2, ..., Y_q) \in \overline{A}^q$. We say that $(\mathcal{X}, \mathcal{Y})$ D_q-dominates A if for all i,j we have that $(X_j \cup Y_j)$ $D_q[i,j]$-dominates X_i (cf. Definition 8).

Definition 12. For every node w of T, for every $\mathcal{X} = (X_1, X_2, ..., X_q) \in A^q$ and every $\mathcal{Y} = (Y_1, Y_2, ..., Y_q) \in \overline{A}^q$, let $\mathcal{R}_\mathcal{X} = rep^{q,d}_{V_w}(\mathcal{X})$ and $\mathcal{R}_\mathcal{Y} = rep^{q,d}_{\overline{V_w}}(\mathcal{Y})$. We define the contents of $Tab_w[\mathcal{R}_\mathcal{X}][\mathcal{R}_\mathcal{Y}]$ as

$$Tab_w[\mathcal{R}_\mathcal{X}][\mathcal{R}_\mathcal{Y}] \overset{\text{def}}{=} \begin{cases} TRUE & \text{if } \exists \text{ partition } \mathcal{S} = (S_1, S_2, ..., S_q) \text{ of } V_w \text{ such that:} \\ & \mathcal{S} \equiv^{q,d}_{V_w} \mathcal{R}_\mathcal{X} \text{ and } (\mathcal{S}, \mathcal{R}_\mathcal{Y})\ D_q\text{-dominates } V_w \\ FALSE & \text{otherwise.} \end{cases}$$

It follows by definition that G has a D_q-partition if and only if some entry in the table at the root of T has value $TRUE$. The computation of the list of all representatives w.r.t. $\equiv^{q,d}_{V_w}$ is basically q times the one given in the previous section. The same situation holds for the computation of a representative from the input of a q-tuplet. Firstly, initialize all values in all tables to $FALSE$.

Updating the leaves: for a leaf l of T, like before, let $\delta(l) = v \in V(G)$ and let $A = \{v\}$. Firstly, there are q possible classes v could belong to in a q-partition of A (recall that empty sets are allowed). We call their representatives respectively $\mathcal{R}_{\mathcal{X}_1}$, $\mathcal{R}_{\mathcal{X}_2}$, $\dots$, $\mathcal{R}_{\mathcal{X}_q}$. Secondly, for vertices in $B = V(G) \setminus \{v\}$ note that they are either neighbors of v or not. Hence we have at most $d+1$ choices (namely $0, 1, ..., d-1, \geq d$) for each of the q partition classes. (A consequence is that Tab_l has at most $q(d+1)^q$ entries.) For every representative $\mathcal{R}_\mathcal{Y} = (Y_1, Y_2, \dots, Y_q)$ w.r.t. $\equiv^{q,d}_B$, we have that $(\mathcal{R}_{\mathcal{X}_i}, \mathcal{R}_\mathcal{Y})$ D_q-dominates $\{v\}$ if and only if $\forall j |N(l) \cap Y_j| \in D_q[i,j]$. Accordingly, we perform the following leaf update for every i and for every $\mathcal{R}_\mathcal{Y}$:

- $Tab_l[\mathcal{R}_{\mathcal{X}_i}][\mathcal{R}_\mathcal{Y}]$ is set to be $TRUE$ if and only if $\forall j\ |N(v) \cap Y_j| \in D_q[i,j]$.

Updating the internal nodes: in the following, $\bigcup_q$ denotes the componentwise union of two q-tuples. For a node w with children a and b, the algorithm performs the following steps.

- Loop over all triples of representatives $\mathcal{R}_{\overline{w}}$ of $\equiv^{q,d}_{\overline{V_w}}$, $\mathcal{R}_a$ of $\equiv^{q,d}_{V_a}$, $\mathcal{R}_b$ of $\equiv^{q,d}_{V_b}$ and do:

  Compute $\mathcal{R}_w = rep^{q,d}_{V_w}(\mathcal{R}_a \bigcup_q \mathcal{R}_b)$, $\mathcal{R}_{\overline{a}} = rep^{q,d}_{\overline{V_a}}(\mathcal{R}_b \bigcup_q \mathcal{R}_{\overline{w}})$, $\mathcal{R}_{\overline{b}} = rep^{q,d}_{\overline{V_b}}(\mathcal{R}_a \bigcup_q \mathcal{R}_{\overline{w}})$

  If $Tab_w[\mathcal{R}_w][\mathcal{R}_{\overline{w}}] = FALSE$ then $Tab_w[\mathcal{R}_w][\mathcal{R}_{\overline{w}}] = Tab_a[\mathcal{R}_a][\mathcal{R}_{\overline{a}}] \wedge Tab_b[\mathcal{R}_b][\mathcal{R}_{\overline{b}}]$

Theorem 2. *For every n-vertex, m-edge graph G given along with a decomposition tree (T, δ) and an integer d. Deciding if G has a D_q-partition, with $d = \max_{i,j} d(D_q[i,j])$, can be solved in time $O(n^4 \cdot q \cdot nec_d(T, \delta)^{3q})$.*

Proof. The complexity analysis is very similar to the one given in Theorem 1, except we need to compute one reprsentative for each of the q parts, and uses the bound in Lemma 3. The correctness proof follows the same style as the proof of Lemma 4, Some steps are not explained here because they were explained in Lemma 4.

For the correctness, let a, b be the children of w in T, assume Tab_a and Tab_b are correct. ($\Rightarrow$) For this direction of the proof we have that $Tab_w[\mathcal{R}_w][\mathcal{R}_{\overline{w}}] = TRUE$. Then there must exist some $\mathcal{R}_a, \mathcal{R}_b$ such that $Tab_a[\mathcal{R}_a][\mathcal{R}_{\overline{a}}] = TRUE$ and $Tab_b[\mathcal{R}_b][\mathcal{R}_{\overline{b}}] = TRUE$, where $\mathcal{R}_{\overline{a}} = rep^d_{V_a}(\mathcal{R}_b \bigcup_q \mathcal{R}_{\overline{w}})$ and $\mathcal{R}_{\overline{b}} = rep^d_{V_b}(\mathcal{R}_a \bigcup_q \mathcal{R}_{\overline{w}})$. Hence there exists $\mathcal{S}_a$ partition of V_a and $\mathcal{S}_b$ partition of V_b such that $(\mathcal{S}_a, \mathcal{R}_{\overline{a}})$ D_q-dominates V_a $(\mathcal{S}_b, \mathcal{R}_{\overline{b}})$ D_q-dominates V_b. This means that $\forall i,j : (S_{a_j} \cup R_{\overline{a}_j})$ $D_q[i,j]$-dominates S_{a_i} and $\forall i,j : (S_{b_j} \cup R_{\overline{b}_j})$ $D_q[i,j]$-dominates S_{b_i}. It then follows that: $\forall i,j : (S_{a_j} \cup S_{b_j} \cup R_{\overline{w}_j})$ $D_q[i,j]$-dominates S_{a_i} and $\forall i,j : (S_{a_j} \cup S_{b_j} \cup R_{\overline{w}_j})$ $D_q[i,j]$-dominates S_{b_i}. It then follows that: $\forall i,j : (S_{w_j} \cup R_{\overline{w}_j})$ $D_q[i,j]$-dominates S_{w_i}. Which means $(\mathcal{S}, \mathcal{R}_{\overline{w}})$ D_q-dominates V_w.

($\Leftarrow$) For this direction of the proof we have that there exists a partition $\mathcal{S} = (S_1, ...S_q)$ of V_w such that: $(\mathcal{S}, \mathcal{R}_{\overline{w}})$ D_q-dominates V_w. This means that $\forall i,j : (S_{w_j} \cup R_{\overline{w}_j})$ $D_q[i,j]$-dominates S_{w_i}. Let $\mathcal{S}_a, \mathcal{S}_b$ be the componentwise intersection of $\mathcal{S}_w$ with V_a and V_b respectively. We then have: $\forall i,j : (S_{w_j} \cup R_{\overline{w}_j})$ $D_q[i,j]$-dominates S_{a_i} and $\forall i,j : (S_{w_j} \cup R_{\overline{w}_j})$ $D_q[i,j]$-dominates S_{b_i}. Hence $\forall i,j : (S_{a_j} \cup S_{b_j} \cup R_{\overline{w}_j})$ $D_q[i,j]$-dominates S_{a_i} and $\forall i,j : (S_{a_j} \cup S_{b_j} \cup R_{\overline{w}_j})$ $D_q[i,j]$-dominates S_{a_i}. Let $\mathcal{R}\overline{a} = rep^d_{\overline{V_a}}(\mathcal{S}_b \bigcup_q \mathcal{R}_{\overline{w}})$ and $\mathcal{R}_{\overline{b}} = rep^d_{\overline{V_b}}(\mathcal{S}_a \bigcup_q \mathcal{R}_{\overline{w}})$ then $\forall i,j : (S_{a_j} \cup R_{\overline{a}_j})$ $D_q[i,j]$-dominates S_{a_i} and $\forall i,j : (S_{b_j} \cup R_{\overline{b}_j})$ $D_q[i,j]$-dominates S_{b_i}. Let $\mathcal{R}_a = can^d_{V_a}(\mathcal{S}_a)$ and $\mathcal{R}_b = can^d_{V_b}(\mathcal{S}_b)$ then $Tab_a[\mathcal{R}_a][\mathcal{R}_{\overline{a}}] = TRUE$ and $Tab_b[\mathcal{R}_b][\mathcal{R}_{\overline{b}}] = TRUE$. Since the algorithm goes through all triples, it will at some point go through $(\mathcal{R}_a, \mathcal{R}_b, \mathcal{R}_{\overline{w}})$. And it will set $Tab_w[\mathcal{R}_w][\mathcal{R}_{\overline{w}}]$ to true, once it is true it will never change.

By induction all tables will be correct. $\square$

7. Runtime expressed by boolean-width

We give an alternative definition of boolean-width, equivalent to the standard one [9].

Definition 13 (Boolean-width). Let G be a graph and $A \subseteq V(G)$. The *bool-dim* $: 2^{V(G)} \rightarrow \mathbb{R}$ function of a graph G is defined as

$$bool\text{-}dim(A) = \log_2(nec(\equiv^1_A))$$

Let (T, δ) be a rooted decomposition tree of G. The boolean-width of (T, δ) is

$$boolw(T, \delta) = \max_{a \in V(T)} \{bool\text{-}dim(V_a)\}$$

The boolean-width of a graph G is the minimum boolean-width over all its rooted decomposition trees

$$boolw(G) = \min_{(T,\delta) \text{ of } G} \{boolw(T, \delta)\}$$

The classes of $\equiv^1_A$ are in a bijection with what is called the Boolean row space of the bipartite adjacency matrix of the graph on edges with exactly one endpoint in A, i.e. the set of vectors that are spanned via Boolean sum (1+1=1) by the rows of this matrix, see the monograph [19] on Boolean matrix theory. ¿From this bijection we get that the *bool-dim* function is symmetric, see [19, Theorem 1.2.3]. In particular, for any node w of T we have $nec(\equiv^1_{V_w}) = nec(\equiv^1_{\overline{V_w}})$.

Lemma 5. *Let G be a graph and $A \subseteq V(G)$. Then, for every $X \subseteq A$, there is $R \subseteq X$ such that $R \equiv_A^d X$ and $|R| \leq d \cdot bool\text{-}dim(A)$. Moreover, $nec(\equiv_A^d) \leq 2^{d \cdot bool\text{-}dim(A)^2}$.*

Proof. We prove the first statement, namely bounding $|R|$ by induction on d. For $d \leq 1$ the lemma follows from Lemma 6 in [9]. Let $S \subseteq X$ be an inclusion minimal set such that $N(S) \cap \overline{A} = N(X) \cap \overline{A}$ e.g. $S \equiv_A^1 X$. Hence from this Lemma with $d = 1$ we have that $|S| \leq bool\text{-}dim(A)$. Assume the induction hypothesis true up to $d-1$, then we show it true for d. By induction hypothesis there exists $R' \subseteq (X \setminus S)$ such that $R' \equiv_A^{d-1} (X \setminus S)$ and $|R'| \leq bool\text{-}dim(A) \cdot (d-1)$. Thus it is enough to show $R = R' \cup S \equiv_A^d X$.

We partition the nodes of $\overline{A}$ into (P, Q) such that $\forall v \in P$, we have $|N(v) \cap (X \setminus S)| = |N(v) \cap R'|$ and $\forall v \in Q$, we have $|N(v) \cap (X \setminus S)| \geq d-1$ and $|N(v) \cap R'| \geq d-1$. For every vertex $v \in P$, since $S \cap R' = \emptyset$ and $S \subseteq X$, we know $|N(v) \cap X| = |N(v) \cap (X \setminus S)| + |N(v) \cap S| = |N(v) \cap R'| + |N(v) \cap S| = |N(v) \cap R|$. We have $N(X) = N(S)$ and since $d > 1$ we have $Q \subseteq N(S)$. For every vertex $v \in Q$, since $|N(v) \cap (X \setminus S)| \geq d-1$ we get $|N(v) \cap X| \geq d$ and since $|N(v) \cap R'| \geq d-1$ we get $|N(v) \cap R| \geq d$. Since (P, Q) is a partition we get $R \equiv_A^d X$ and $|R| \leq bool\text{-}dim(A) \cdot d$, thus by induction the lemma holds for all d.

To bound the number of equivalence classes $nec(\equiv_A^d)$ we know from the previous arguments that we only need to find the equivalence classes among the subsets of A of size at most $d \cdot bool\text{-}dim(A)$. Two vertices $x, x' \in A$ are twins across $\{A, \overline{A}\}$ if $N(x) \cap \overline{A} = N(x') \cap \overline{A}$. Let H be obtained from the bipartite subgraph of G with color classes $A, \overline{A}$ after doing twin contraction of all twins. We know that every node of $V(H) \cap A$ has a unique neighborhood, hence $|V(H) \cap A| \leq 2^{bool\text{-}dim(A)}$. For any subset of A there is a multiset of $V(H) \cap A$ with the same d-neighbourhood, and a trivial bound on number of multisets of size $d \cdot bool\text{-}dim(A)$ of $V(H) \cap A$ gives us: $nec(\equiv_A^d) \leq 2^{d \cdot bool\text{-}dim(A)^2}$. □

Together with Theorem 1 we get the following.

Corollary 1. *For every graph G given along with a decomposition tree (T, δ) any (σ, ρ)-vertex subset problem on G with $d = d(\sigma, \rho)$ can be solved in time $O(n^4 \cdot q \cdot 2^{3d \cdot boolw(T,\delta)^2})$.*

Together with Theorem 2 and Observation 3 we get the following.

Corollary 2. *For every graph G given along with a decomposition tree (T, δ), deciding if G has a D_q-partition, with $d = \max_{i,j} d(D_q[i,j])$, can be done in time $O(n^4 \cdot 2^{3qd \cdot boolw(T,\delta)^2})$.*

8. Conclusions and Open Problems

The runtime of the algorithms given here for (σ, ρ)-problems and D_q-problems have the square of the boolean-width $boolw$ as a factor in the exponent, i.e. $O(boolw^2)$ in the exponent. For problems where $d = 1$ we can in fact improve this to a factor linear in the exponent [9], but that requires a special focus on these cases. We hope that also for the other problems (with any constant value of d) we could get runtimes with a better exponential factor, say $O(boolw \log boolw)$ in the exponent or maybe even linear. We must then improve the bound in Lemma 5. For the linear bound we must show that the number of

d-neighborhood equivalence classes is no more than the number of 1-neighborhood equivalence classes raised to some function of d. This runtime question can also be formulated as a purely algebraic one. First generalize the concept of Boolean sums $(1+1=1)$ to d-Boolean sums $(i+j=\min(i+j,d))$. For a Boolean matrix A let $R_d(A)$ be the set of vectors over $\{0,1,...,d\}$ that arise from all possible d-Boolean sums of rows of A. To get $O(\mathit{boolw}\log\mathit{boolw})$ in the exponent it would suffice to show that there is a function f such that $|R_d(A)| \leq |R_1(A)|^{f(d)\log\log|R_1(A)|}$.

References

[1] R. Belmonte, M. Vatshelle, Graph classes with structured neighborhoods and algorithmic applications, Theoretical Computer Science Submitted to this issue (?) (2012) ?

[2] J. Fiala, J. Kratochvíl, Locally constrained graph homomorphisms - structure, complexity, and applications, Computer Science Review 2 (2008) 97–111.

[3] R. G. Downey, M. R. Fellows, Parameterized Complexity, Springer Verlag, 1999.

[4] J. Flum, M. Grohe, Parameterized Complexity Theory, Springer Verlag, 2006.

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

[6] B. Courcelle, J. A. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique width, Theory of Computing Systems 33 (1999) 125–150.

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

[8] M. U. Gerber, D. Kobler, Algorithms for vertex-partitioning problems on graphs with fixed clique-width, Theoretical Computer Science 299 (1-3) (2003) 719–734.

[9] B.-M. Bui-Xuan, J. A. Telle, M. Vatshelle, Boolean-width of graphs, Theoretical Computer Science 412 (39) (2011) 5187–5204.

[10] I. Adler, B.-M. Bui-Xuan, Y. Rabinovich, G. Renault, J. A. Telle, M. Vatshelle, On the boolean-width of a graph: Structure and applications, in: Proceedings of WG, 2010, pp. 159–170.

[11] M. Vatshelle, New width parameters of graphs, Ph.D. thesis, University of Bergen (2012).

[12] E. M. Hvidevold, S. Sharmin, J. A. Telle, M. Vatshelle, Finding good decompositions for dynamic programming on dense graphs, in: Proceedings of IPEC, 2011, pp. 219–231.

[13] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Co., 1978.

[14] J. Kratochvíl, Perfect Codes in General Graphs, Academia Praha, 1991.

[15] T. J. Schaefer, The complexity of satisfiability problems, Proceedings of STOC 10 (1978) 216–226.

[16] P. Heggernes, J. A. Telle, Partitioning graphs into generalized dominating sets, Nordic Journal of Computing 5 (2) (1998) 128–142.

[17] J. A. Telle, Vertex partitioning problems: Characterization, complexity and algorithms on partial k-trees, Ph.D. thesis, University of Oregon (1994).

[18] O. Amini, I. Sau, S. Saurabh, Parameterized complexity of the smallest degree-constrained subgraph problem, in: Proceedings of IWPEC, 2008, pp. 13–29.

[19] K. H. Kim, Boolean matrix theory and applications, Marcel Dekker, 1982.