Choice of parameter for DP-based FPT algorithms: four case studies

Sigve Hortemo Sæther

Dissertation for the degree of philosophiae doctor (PhD)
at the University of Bergen

2015

Dissertation date: September 7, 2015

Acknowledgements

First of all, I would like to thank my supervisor, Jan Arne Telle, for the help and guidance given to me throughout the last four years. Thank you for seeing my potential, and for urging me to do research after I finished my masters degree.

Secondly, I want to give a big thank you to my former office mate and co-author Martin Vatshelle. Your time as "ambassador" for Jan Arne while we were both in Canada, and our numerous discussions have been very valuable to my research.

I also want to give a special thanks to my two other office mates Manu Basavaraju and Mithilesh Kumar for your discussions and company in the office.

These four years have been a fantastic time in my life, and I owe a lot of it to the algorithms group. So thank you, to both current and former group members, for making the workplace so enjoyable.

Thank you also to my Korean co-authors Jisu Jeong and Sang-il Oum.

In advance I would like to thank the members of my thesis committee, Ioan Todinca, Michael Lampis, and Trond Steihaug. Thank you for taking the time to evaluate my work.

And a thank you to the Norwegian Research Council, the University of Bergen and to the Meltzer foundation for funding my research and my travels.

Lastly, I want to thank my friends and family for their support. I want to thank my girlfriend Ingunn Valde in particular, for her patience and understanding, and for always being there for me.

Abstract

This thesis studies dynamic programming algorithms and structural parameters used when solving computationally hard problems. In particular, we look at algorithms that make use of structural decompositions to overcome difficulties of solving a problem, and find alternative runtime parameterizations for some of these problems.

The algorithms we look at make use of branch decompositions to guide the algorithm when doing dynamic programming. Algorithms of this type comprise of two parts; the first part computes a decomposition of the input, and the second part solves the given problem by dynamic programming over the computed decomposition. By altering what properties of an input instance these decompositions should exploit, the runtime of the complete algorithm will change. We look at four cases where altering the structural properties of the decomposition (i.e., changing what width measure for the decomposition to minimize), is used to improve an algorithm.

The first case looks at using branch decompositions of low maximum matching-width (mm-width) instead of tree-decompositions of low treewidth when solving Dominating Set. The result of this is an algorithm that is faster than the treewidth-algorithms on instances where the treewidth is at least 1.55 times the mm-width.

In the second case, we look at using branch decompositions of low split-matching-width (sm-width) for cases when using tree-decompositions or k-expressions will not do. This study leads to new tractability results for Hamiltonian Cycle, Edge Dominating Set, Chromatic Number, and MaxCut for a class of dense graphs.

For the third case, we look at using branch decompositions of low $\mathbb{Q}$-rank-width as an alternative to using branch decompositions of low rank-width for solving a large class of problems definable as $[\sigma, \rho]$-partition problems. This class consists of many domination-type problems such as Dominating Set and Independent Set. One of the results of using such an alternative branch decompositions is that we get an improved worst case runtime for Dominating Set parameterized by the clique-width cw; namely $\mathcal{O}^*((\mathrm{cw})^{\mathcal{O}(\mathrm{cw})})$ over the previous best $\mathcal{O}^*(2^{\mathcal{O}((\mathrm{cw})^2)})$.

The fourth case looks at using branch decompositions of low projection-satisfiable-width (ps-width) for solving #SAT and MaxSAT on CNF formulas. We define the notion of having low ps-width and show that by using a dynamic programming algorithm that makes use of the ps-width of a branch decomposition, we get new tractability results for #SAT and MaxSAT, and a framework unifying many previous tractability results.

We also show that deciding boolean-width of a graph is NP-hard and deciding mim-width of a graph is W[1]-hard. In fact, under the assumption NP$\neq$ZPP, we show that we cannot approximate mim-width to within a constant factor in polynomial time.

Contents

Chapter 1

The starting point

Unless P=NP, then for any problem π in the class of NP-hard problems there must exist classes of instances for which we cannot solve π in polynomial time. However, even for NP-hard problems, there are classes of input instances for which π can be solved in polynomial time, i.e., in $\mathcal{O}(n^c)$ time for some fixed constant c. The field of *Parameterized Complexity* will account for this within the rich framework of multivariate complexity analysis. This involves measuring the algorithmic runtime not only using the input length n, but also by using a parameter k associated with the input. For algorithms with runtimes of the form $n^{f(k)}$ or $f(k)n^c$ for some constant c and function f depending only on k, it can be seen that for instances where k is constant, the resulting algorithmic runtimes will in fact be polynomial (e.g. for $k = 2$ and $f(2) = 10$, we get runtimes of n^{10} and $10n^c$). If we can solve a parameterized problem π within a runtime of the form $f(k)n^c$, then we say π is *Fixed Parameter Tractable* (FPT). Problems that are solvable in time $n^{f(k)}$ are XP. For an NP-hard problem π having a parameterized version which is FPT or XP, for any class of instances where the parameter k is fixed, solving π will become tractable[1]. However, although a problem π FPT for a parameter p_1, becomes tractable for classes of inputs where the value of p_1 is bounded, there might still be another class $\mathcal{C}$ of inputs where π is tractable even though p_1 is unbounded on $\mathcal{C}$. The reason may be that an unrelated parameter p_2, for which π is also FPT, is bounded over $\mathcal{C}$. Thus, a problem being FPT for a parameter p_1 only gives a one way relationship between parameter value and tractability. It would of course be desirable to find a parameter whose value perfectly corresponds to the tractability of a problem instance. That is, for a problem π to find a parameter so that π is tractable on a class of inputs $\mathcal{C}$ if and only if the parameter value is bounded over $\mathcal{C}$.

In this thesis, we will investigate the topic of choosing parameters in order to get the best correlation between parameter value and actual runtime complexity of solving a problem. More on this in the next chapter. Let us first state some definitions and terminology that will be used in this thesis.

[1] Allthough constant k implies a polynomial runtime for both FPT and XP, having a runtime of the form $f(k)n^{\mathcal{O}(1)}$ (FPT) is regarded as better than $n^{f(k)}$ (XP), as an increase in k only changes the coefficient of the polynomial for FPT, while it changes the degree of the polynomial for XP.

1.1 Definitions and terminology

We assume the reader is familiar with standard terminology from set theory.

1.1.1 Graph terminology

Graphs. A *graph* $G = (V, E)$ is an ordered pair consisting of a set of *vertices* V and *edges*, $E \subseteq \{uv : u, v \in V\}$. We will only look at simple and undirected graphs, meaning that we will not distinguish between uv and vu, and we will not have edges of the form $vv \in E$. For a graph $G = (V, E)$ we also denote the set of vertices V by $V(G)$, and set of edges E by $E(G)$. For an edge $uv \in E$, we say u and v are *incident* with the edge uv, and u and v are the *endpoints* of uv. We say that two vertices u, v incident with the same edge uv *share* the edge uv. Two vertices $u, v \in V(G)$ are *adjacent* in the graph G if $uv \in E(G)$.

For a set E' of edges, we denote by $V(E')$ the union of all the endpoints of E'. That is, $V(E')$ is the set of all vertices incident with edges of E'. For a graph G and vertex, edge or set of vertices/edges x, we denote by $G - x$ the subgraph by removing the edges of x and vertices of x (and adjacent edges) from G. Likewise, $G + x$ denotes the graph where we add edges and vertices of x to G. We say a set of edges E *forms* the graph $G = (V(E), E)$.

The *complement* of a graph $G = (V, E)$ is the graph G' with the same vertex set V as G, but where two vertices are adjacent in G' if and only if they are not adjacent in G.

The *open neighbourhood* of a vertex v in graph G, or simply its *neighbourhood*, is the set of vertices adjacent to v in G, denoted $N_G(v)$. The *closed neighbourhood* of v is the set $N_G(v) \cup \{v\}$, denoted $N_G[v]$. For a set $S \subseteq V(G)$, we denote by $N_G[S]$ and $N_G(S)$ the sets $\bigcup_{s \in S} N_G[s]$ and $N_G[S] \setminus S$, respectively, and call them the open (and closed resp.) neighbourhood of S. If there is no ambiguity, we may drop the subscript G. The cardinality of $N_G(v)$ is called the *degree* of v in G, denoted $d_G(v)$.

Two vertices u, v of a graph G are called *true twins* if their closed neighbourhoods are the same $N[u] = N[v]$. If u and v have the same open neighbourhoods $N(u) = N(v)$, we say they are *false twins*. A pair of vertices are *twins* if they are either true twins or false twins.

A bipartite graph. A graph is *bipartite* if its set of vertices can be partitioned into two disjoint parts A, B so that all edges are incident with exactly one vertex of A and one vertex of B. When this bipartition is known for some graph G, we might state this explicitly by describing G as $G = (A, B, E)$, where A and B are as described above, and E is the set $E(G)$.

Subgraphs and induced graphs. For a graph G', we say that it is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. A subgraph G' of G with vertex set $S \subseteq V(G)$ is said to be the *induced by* S if all vertices of S that are adjacent in G are also adjacent in G'. That is, $E(G') = \{uv \in E(G) : u, v \in S\}$. We denote this graph by $G[S]$. A bipartite subgraph G' of G with vertex set $S_1 \uplus S_2$ is said to be *induced by* S_1 *and* S_2 when its set of edges is precisely those edges from $E(G)$ that are incident with one vertex from S_1 and one vertex from S_2. That is, $E(G') = \{uv \in E(G) : u \in S_1, v \in S_2\}$. We denote this graph by $G[S_1, S_2]$.

Identifying two vertices. To *identify* two vertices u and v, means to replace them with a single vertex x for which $N(x) = N(\{v, u\})$.

Contracting and subdividing an edge. To *contract* an edge uv in a graph means to replace the two vertices u and v by a new vertex x so that $N(\{u, v\}) = N(x)$. That is, we identify u and v. To *subdivide* an edge uv in a graph means to replace the edge uv with a vertex x for which $N(x) = \{u, v\}$.

A path. A *path* in a graph is a sequence of vertices for which all pairs of consecutive vertices are adjacent. We say a graph G itself is a path if all the vertices $V(G)$ form a path, and $E(G)$ is inclusion wise minimal. Similarly, a set of edges E is called a path if it forms a graph which is a path. For a graph P which is a path, for two vertices $x, y \in V(P)$, we denote by xPy the minimal subgraph which is a path and contains the vertices x and y.

Connected. A graph is *connected* if there exists a path between any pair of vertices in the graph. A connected component of a graph is an inclusion maximal connected subgraph.

A cycle. A *cycle* is a (non-empty) connected graph of only vertices of degree two. A graph is said to have or contain a cycle if it has a subgraph which is a cycle.

Isomorphic graphs. We say two graphs G and G' are *isomorphic* to each other if there is a bijective function $\sigma : V(G) \to V(G')$ so that that u and v are adjacent in G if and only if $\sigma(u)$ and $\sigma(v)$ are adjacent in G'.

A tree. A connected graph without any cycles as induced subgraph is called a *tree.* A *rooted* tree is a tree with one distinguished vertex r called the *root.* Vertices on a shortest path from v to r are called the *ancestors* of v. The ancestor p incident with v is called the *parent* of v, and v a *child* of p. The other children of p are called the *siblings* of v. Any vertex having v as an ancestor is called a *descendant* of v. Vertices of degree more than one in a tree are called *internal* vertices, and the remaining vertices are called *leaves.*

A pendant. A *pendant* is a vertex of degree one. This can also refer to an edge incident with a degree one vertex.

Isolated and universal vertices. A vertex v of a graph is said to be *isolated* if its neighbourhood is empty. A *universal* vertex of a graph is a vertex adjacent to all other vertices in the graph.

1.1.2 More on graphs

Cliques and complete graphs. A *complete graph* is a graph where all vertices are adjacent to each other. We denote the complete graph of n vertices as K_n. A *clique* is a set of vertices S of a graph G so that $G[S]$ forms a complete graph. A *complete bipartite graph*, is an induced bipartite graph of a complete graph. We denote by $K_{a,b}$ the complete bipartite graph with part A of cardinality a and part B of cardinality b.

Distance hereditary graphs. A *distance hereditary graph* is a graph G so that for every induced subgraph $G[S]$ for $S \subseteq V(G)$, and for any pair of vertices u,v in S, either u is disconnected from v in $G[S]$, or the distance from u to v in $G[S]$ is the same as the distance from u to v in G.

Matching of a graph. A *matching* of a graph G is a set $M \subseteq E(G)$ of edges so that no vertex of $V(G)$ is incident with more than one edge in M. The number of edges in M is called the size of M. A *perfect matching* is a matching M so that every vertex of G is incident with exactly one edge of M. A vertex is said to be *saturated* by a matching M if it is incident with an edge in M. A matching M is *maximal* if no matching $M' \subseteq E(G)$ exist so that $M \subset M'$. A matching M is a *maximum* matching if for all matchings $M' \subseteq E(G)$ we have $|M| \geq |M'|$.

Induced matching. An *induced matching* of a graph G is a matching M of G so that $M = E(G[V(M)])$. In other words, for any 3 vertices a,b,c, if ab is an edge in M and bc is an edge then there does not exist an edge cd in M.

Vertex Cover. A *vertex cover* of a graph is a set $S \subseteq V(G)$ of vertices so that for every edge $uv \in E(G)$ either $u \in S$ or $v \in S$. A *minimum* vertex cover is a smallest possible vertex cover. A *minimal* vertex cover is a vertex cover S so that no proper subset $S' \subset S$ is also a vertex cover. It was shown by König [64] that in bipartite graphs, the size of a minimum vertex cover is always the same size as a maximum matching.

1.1.3 Decomposition-related terminology

Cut of a graph. A *cut* of a graph $G = (V,E)$ is a partition of the vertices V into two disjoint subsets $(S \subset V, V \setminus S)$. The edges of E with one endpoint in S and one in $V \setminus S$ is said to *cross* the cut $(S, V \subseteq S)$.

Neighbourhood equivalence. Neighbourhood equivalence is general equivalence relation based on neighbourhoods crossing the cut of a graph. For a graph G and subsets $S_1, S_2 \subseteq V(G)$ we say that S_1 and S_2 are *neighbour-equivalent*, denoted $S_1 \equiv S_2$, if they are adjacent to the same vertices in G. That is, $S_1 \equiv S_2$ if and only if $N(S_1) = N(S_2)$. Furthermore, S_1 and S_2 are *d-neighbour-equivalent*, denoted $S_1 \equiv^d S_2$, if each vertex in $V(G)$ is either adjacent to the same number of vertices

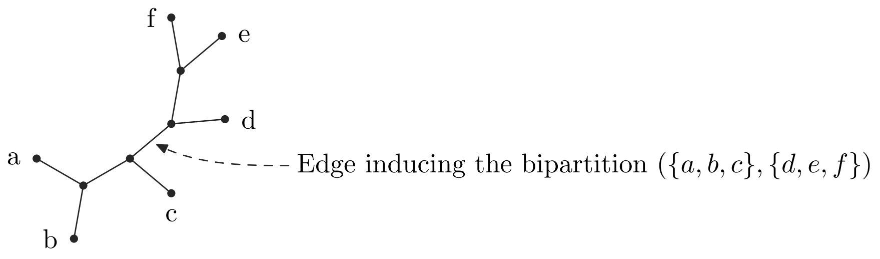


Figure 1.1: An edge inducing in a branch decomposition inducing a bipartition.

in both S_1 and S_2, or it is adjacent to at least d vertices of both S_1 and of S_2. Note that from this we have $S_1 \equiv S_2$ if and only if $S_1 \equiv^1 S_2$.

Branch decompositions A *branch decomposition* over X, for some set of elements X, is a pair (T,δ), where T is a tree over vertices of degree at most 3, and δ is a bijection from the leaves of T to the elements in X. Any edge uv disconnects T into two sub-trees T_u and T_v. Likewise, any edge uv partitions the elements of X into two parts X_u and X_v, namely the elements mapped by δ from the leaves (excluding u) of T_u, and of T_v, respectively. An edge $uv \in E(T)$ is said to *induce* the partition (X_u, X_v). We denote this partition by $\delta(uv)$.

Given a symmetric ($f(A) = f(\overline{A})$) function $f : 2^X \to \mathbb{R}$, using branch decompositions over X, we get a nice way of defining width parameters: For a branch decomposition (T,δ) and edge $e \in T$, we define the *f-value* of the edge e to be the value $f(A) = f(B)$ where A and B are the two parts of the partition induced by e in (T,δ), denoted $f(e)$. We define the *f-width* of branch decomposition (T,δ) to be the maximum f-value over all edges of T, denoted $f(T,\delta)$: $\max_{e\in T}\{f - \text{value of } e\}$. For set X of elements, we define the *f-width* of X to be the minimum f-width over all branch decompositions over X. If $|X| \leq 1$, then X admits no branch decomposition and we define its f-width to be $f(\emptyset)$.

Cut function. We say that a function $f : 2^X \to \mathbb{R}$ is a *cut function* of G if $X = V(G)$.

Adjacency matrix. The *adjacency matrix* of a bipartite graph is a matrix M where the rows are indexed by the vertices of one part, and the columns indexed by the vertices of the other part. A cell $M[u,v]$ in the matrix is one if u and v are adjacent and zero otherwise.

Rank-width. Rank-width is width parameter, introduced by Oum and Seymour [79], based on branch decompositions and the GF[2] rank of adjacency matrices, called *cut-rank*. For a graph G and set $S \subseteq V(G)$, the cut-rank of S is the GF[2] rank of the adjacency matrix of $G[S, V(G) \setminus S]$. Put differenctly, the cut-rank of S is the minimum cardinality set $S' \subseteq S$ so that for any $v \in S$ there is a set $S^* \subseteq S'$ where $x \in V(G)$ is adjacent to v if and only if the number of vertices from S^* adjacent to x is odd. The cut-rank of S is also denoted $\text{cutrk}(S)$.

Using cut-rank as our symmetric function f in the definition of f-width above, we define the *rank-width* of G to be the f-width of $V(G)$ where $f =$ cut-rank. We say that the cut-width of a branch decomposition (T,δ) of $V(G)$ is the f-width of (T,δ) with f set to be the cut-rank.

A rooted branch decomposition. A rooted branch decomposition is a branch decomposition (T,δ) where one of the vertices r of $V(T)$ are chosen as a root. In a rooted branch decomposition, for an internal vertex $v \in V(T)$, we denote by $\delta(v)$ the union of $\delta(l)$ for all leaves of l having v as its ancestor.

Branchwidth. The branchwidth of a graph G is a width parameter defined through branch decompositions over $E(G)$, introduced by Robertson and Seymour [89]. The branchwidth of G is defined to be the f-width of $E(G)$ with respect to the following symmetric function f on subsets $X \subseteq E(G)$:

$$f(X) = |\{v \in V(G) : \exists uv \in X \text{ and } \exists u'v \in E(G) \setminus X\}| \ .$$

$\mathtt{nec}_d$-width. The $\mathtt{nec}_d$-width of a graph is a general width parameter defined through branch decompositions over $V(G)$ and the d-neighbour-equivalence relation, introduced by Bui-Xuan et al. [23]. Note that d is a variable taking on any natural number, so we have $\mathtt{nec}_1$-width, $\mathtt{nec}_2$-width, $\mathtt{nec}_3$-width and so on.

Given a set $S \subseteq V(G)$, the $\mathtt{nec}_d$-value of S is the cardinality of the largest family of subsets $\mathcal{F}$ of S so that no two subsets $S_1, S_2 \in S$ are d-neighbour-equivalent in the bipartite graph $G[S, V(G) \setminus S]$. The $\mathtt{nec}_d$-value of a cut is its f-value where $f = \mathtt{nec}_d$, and the $\mathtt{nec}_d$-width of a graph is the f-width of $V(G)$ where $f = \mathtt{nec}_d$.

Boolean-width. The boolean-width of a graph G is precisely equivalent to the logarithm base two of the $\mathtt{nec}_1$-width, introduced by Bui-Xuan et al. [22]. The $\mathtt{nec}_d$-width parameter was defined as a generalization of boolean-width. We denote the boolean-width of G as boolw(G).

Maximum matching-width. The *maximum matching-width*, mm-width in short, is a width parameter defined through branch decompositions over $V(G)$ and the cardinality of matchings, introduced by Vatshelle [107]. For a cut $S \subseteq V(G)$, the maximum matching-value is defined to be the size of a maximum matching in $G[S, V(G) \setminus S]$, denoted mm(S). The mm-width of a graph G, denoted mm-w(G), is the f-width of $V(G)$ for $f =$ mm.

Maximum induced matching-width. The *maximum induced matching-width*, mim-width in short, is a width parameter defined through branch decompositions over $V(G)$ and the cardinality of induced matchings, introduced by Vatshelle [107]. For a cut $S \subseteq V(G)$, the maximum induced matching-value, denoted mim(S), is the size of a maximum induced matching in $G[S, V(G) \setminus S]$. The mim-width of a graph G, denoted mim-w(G), is the f-width of $V(G)$ for $f =$ mim.

Treewidth. The treewidth, introduced by Seymour and Thomas [88], is a graph parameter defined using a *tree-decomposition.* A *tree-decomposition* of a graph G is a pair $(T, \{X_t\}_{t \in V(T)})$ consisting of a tree T and a family $\{X_t\}_{t \in V(T)}$ of vertex sets $X_t \subseteq V(G)$, called *bags*, satisfying the following three conditions:

1. each vertex of G is in at least one bag,
2. for each edge uv of G, there exists a bag that contains both u and v, and
3. for vertices t_1, t_2, t_3 of T, if t_2 is on the path from t_1 to t_3, then $X_{t_1} \cap X_{t_3} \subseteq X_{t_2}$.

The *treewidth* of a tree-decomposition $(T, \{X_t\}_{t \in V(T)})$ is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of G, denoted by $\mathrm{tw}(G)$, is the minimum width over all possible tree-decompositions of G.

A k-expression. A k-expression is a way of describing a graph and is an algebraic expression using the following four operations:

- $i(v)$: construct a graph consisting of a single vertex with label $i \in \{1, 2, \ldots, k\}$.
- $G_1 \oplus G_2$: take the disjoint union of labelled graphs G_1 and G_2.
- $\eta_{i,j}$ for distinct $i, j \in \{1, 2, \ldots, k\}$: add an edge between each vertex of label i and each vertex of label j.
- $\rho_{i \to j}$ for $i, j \in \{1, 2, \ldots, k\}$: relabel each vertex of label i to j.

Clique-width. The *clique-width* of a graph G, introduced by Courcelle and Olariu [32], is precisely the minimum value k such that there exists a k-expression describing G. We denote this value by $\mathrm{cw}(G)$.

Submodular functions. A function $f : 2^X \to \mathbb{R}$ is said to be submodular if for any $A, B \subseteq X$ we have $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$.

Splits. A *split* of a connected graph G is a bi-partition (A, B) of the vertices $V(G)$ where $|A|, |B| \geq 2$ and for all $a \in N(B)$, $N(a) \cap B = N(A)$. That is, all vertices in A adjacent to B have the same neighbourhood in B. Notice that this property holds if and only if also for all $b \in N(A)$, $N(b) \cap A = N(B)$. A bi-partition (A, B) where either A or B consists of at most one vertex is said to be a *trivial split*.

Split-decomposing a graph. A graph G having a split (A, B) can be *decomposed* into smaller graphs G_A and G_B where G_A is the graph G with all the vertices of B replaced by a single vertex v, called a *marker*, adjacent to the same vertices in G_A as B is adjacent to in G. G_B is in the same way the graph G where we replace the vertices A by the marker vertex v so that $N_{G_B}(v) = N_G(A)$. A graph decomposed by G_A and G_B is denoted $G_A * G_B$. So if G is decomposed by G_A and G_B, we have $G = G_A * G_B$.

Prime graphs. A graph without a split is called a *prime graph.* Since all graphs of three or less vertices trivially do not contain any splits, we say that a prime graph on more than three vertices is a *non-trivial* prime graph. Note that in the context of modular decompositions a prime graph means something slightly different[2].

Split decomposition. A *split decomposition* of a graph G is a recursive decomposition of G so that all of the obtained graphs are prime.

Split decomposition tree. For a split decomposition of G into $G_1, G_2, \ldots, G_k$, a *split decomposition tree* is a tree T where each vertex corresponds to a prime graph and we have an edge between two vertices if and only if the prime graphs they correspond to share a marker. That is, the edge set of the tree is $E(T) = \{v_i v_j : v_i, v_j \in V(T) \text{ and } V(G_i) \cap V(G_j) \neq \emptyset\}$. See Figure 1.2 for an example.

1.1.4 Non-standard definitions

The following definitions are non-standard and should be read more carefully than the rest. These are definitions and terminology that will be used in Chapter 4.

The functions $\mathrm{tot}()$ **and** $\mathrm{act}()$. For a split decomposition tree T and prime graph P in $V(T)$, a marker $v \in V(P)$ 'represents' some vertices of $V(G)$, that we call $\mathrm{tot}(v : P)$. Formally, we define $\mathrm{tot}(v : P)$ to be the vertices of $V(G)$ in the prime graphs of the unique connected component of $T - P$ that contains the other occurrence of v. As an example, in Figure 1.2 the set $\mathrm{tot}(v_1 : G_2)$ is $\{f, g, h\}$; the vertices of $V(G)$ in G_4 and G_5. For a non-marker u in a prime graph G_i, we define $\mathrm{tot}(u : G_i)$ to simply be the set $\{u\}$. In this way $\mathrm{tot}(w : G_i)$ can be seen as the set of vertices from the original graph that w is representing in the prime graph G_i. We note that for a prime graph G_i, the set of $\mathrm{tot}(v : G_i)$ for all $v \in V(G_i)$ is a partition of $V(G)$. Also, for a marker v and the prime graphs G_i and G_j containing it, $\{\mathrm{tot}(v : G_i), \mathrm{tot}(v : G_j)\}$ is a bipartition of $V(G)$. For a set $V' \subseteq V(G_i)$, we define $\mathrm{tot}(V' : G_i)$ to be the union of $\mathrm{tot}(v : G_i)$ for all $v \in V'$. We define the *active set* of a vertex $v \in G_i$, denoted $\mathrm{act}(v : G_i)$, to be the vertices of $\mathrm{tot}(v : G_i)$ that are adjacent to the rest of $V(G)$. That is, $\mathrm{act}(v : G_i) = N_G(V(G) \setminus \mathrm{tot}(v : G_i))$. See Figure 1.2 for an example of $\mathrm{tot}()$ and $\mathrm{act}()$. We may also denote tot and act by the use of subscripts. That is, $\mathrm{tot}(v : G_i) = \mathrm{tot}_{G_i}(v)$ and $\mathrm{act}(v : G_i) = \mathrm{act}_{G_i}(v)$.

Weight of vertices in decomposed graphs. For a prime graph G' and vertex $v \in V(G')$, when we say the *weight* of v, we mean the cardinality of $\mathrm{act}(v)$.

Partitioning through a function. For sets X and Y, we say that a function $f : X \to 2^Y$ *partitions* Y if 1) $\forall x_1 \neq x_2 \in X : f(x_1) \cap f(x_2) = \emptyset$, and 2) $\bigcup_{x_i \in X} f(x_i) = Y$.

[2]In the context of *modular decompositions* (see [72] for the definition of modular decompositions), a prime graph is a graph for which there is no split (A, B) so that either A is disconnected from B or $N(A) = B$.

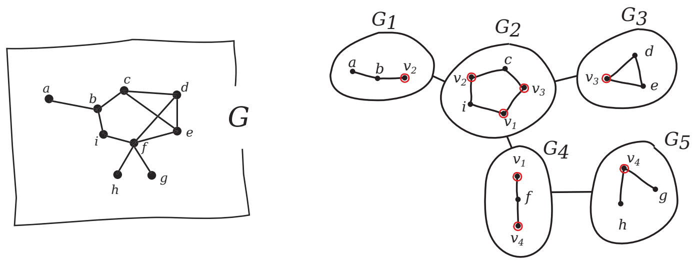


Figure 1.2: A split decomposition tree of a graph G. The markers of each prime graph are circled in red. An example of a split decomposition resulting in this tree is: $((G_1 * G_2) * G_3) * (G_4 * G_5)$. Note that $\mathrm{tot}(\{v_1, v_3\} : G_2) = \{d, e, f, g, h\}$, $\mathrm{act}(\{v_1, v_3\} : G_2) = \{d, e, f\}$.

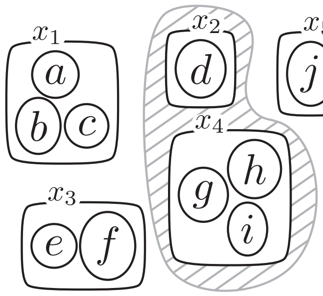


Figure 1.3: For the universes $\mathcal{U}_1 = \{a, b, c, d, e, f, g, h, i, j\}$ and $\mathcal{U}_2 = \{x_1, x_2, x_3, x_4, x_5\}$, and function $\sigma = \{x_1 \to \{a, b, c\}, x_2 \to \{d\}, x_3 \to \{e, f\}, x_4 \to \{g, h, i\}, x_5 \to \{j\}\}$ that partitions $\mathcal{U}_1$, we have that for any function $f : 2^{\mathcal{U}_1} \to \mathbb{R}$, the corresponding σ-lifted function $f^\sigma : 2^{\mathcal{U}_2} \to \mathbb{R}$ of f on the marked region is $f^\sigma(\{x_2, x_4\}) = f(\{d, g, h, i\})$.

A lifted function. Let X and Y be some set of elements and $\sigma : X \to 2^Y$ a function that partitions Y. Let $\sigma' : 2^X \to 2^Y$ be the function so that $\sigma'(S \subseteq X) = \bigcup_{s' \in S} \sigma(s')$, and $f : 2^Y \to \mathbb{R}$ any subset-function of Y. We say that the composition $(f \circ \sigma')$ of f and σ' is the σ-*lifted* function of f from X to Y. That is, the σ-lifted function of f is the function $g : 2^X \to \mathbb{R}$ so that $g(S \subseteq X) = f(\sigma'(S))$. We denote the σ-lifted function of f by f^σ. The reason why we do not simply use the well known composition notation for this is that we want to avoid explicitly defining σ'. See Figure 1.3 for an example of a lifted function. When the function σ is clear from context, we might refere to the σ-lifted function of f simply as the lifted f function.

1.1.5 Runtime and complexity

Unless otherwise specified, we denote by n and m the size cardinality of $V(G)$ and $E(G)$ for input graph G implicit from context. At times we use the notation $\mathcal{O}^*$. This means we suppress polynomials of n. That is, $\mathcal{O}^*(f(x))$ implies $\mathcal{O}(f(x)n^{\mathcal{O}(1)})$.

The complexity class P. The complexity class P is the set of problems solvable in polynomial time ($n^{\mathcal{O}(1)}$).

The complexity class NP. The complexity class NP is the set of "yes"/"no" problems solvable in polynomial time non-deterministically. Alternatively; it is the set of problems for which all "yes"-instances have a polynomial length *certificate* that can be used as a hint to verify (in polynomial time) that the instance indeed is a "yes"-instance.

A problem is NP-hard if it being in P will imply NP=P. A problem is NP-complete if it is in NP and is NP-hard.

A parameter. A parameter is a function p assigning a number to each input instance. The runtime of solving an instance I is parameterized if it is expressed in terms both of the input length n of I and the parameter value $k = p(I)$. E.g., $\mathcal{O}(n^4 2^k)$.

Classes of bounded value For a parameter p, we say that a class of instances has *bounded* p if there is some constant c so that for all instances I in the class, $p(I) \leq c$.

The complexity class FPT. The complexity class FPT is the class of problems solvable in time $n^{\mathcal{O}(1)} f(k)$ for some function f and parameter k. We abuse notation slightly by saying that a runtime (not only problems) of the form $n^{\mathcal{O}(1)} f(k)$ is FPT.

Being $\mathsf{W[1]}$-hard. A parameterized problem is $\mathsf{W[1]}$-hard if having a FPT algorithm for the problem implies k-Independent Set can be solved in FPT time parameterized by the size of the independent set (k).

The complexity class APX. The complexity class APX is the class of problems in NP that can be approximated to within a constant factor in polynomial time. That is, it is the class of problems of the form "is there a X of size k so that ..." for which there is a c so that we can in polynomial time correctly determine that an instance is not a yes instance, or determine that it is a yes instance for the same problem with k adjusted to be c times as large.

The complexity class ZPP. The complexity class ZPP is the class of problems for which there is a randomized algorithm which correctly solves the problem in polynomial expected runtime.

Exponential Time Hypothesis. The Exponential Time Hypothesis (ETH) [56] states that there is a positive real s so that 3-SAT cannot be solved in time $\mathcal{O}(2^{sn})$ where n is the number of variables in the input formula.

Strong Exponential Time Hypothesis. The Strong Exponential Time Hypothesis (Strong ETH) states that SAT can not be solved in $\mathcal{O}((2-\epsilon)^n)$ time for any constant $\epsilon > 0$. Here n denotes the number of variables.

Linearly single exponential. We say that an FPT runtime of the form $\mathcal{O}^*(2^{\mathcal{O}(k)})$, is *linearly single exponential.* Problems solvable in linearly single exponantial FPT time are also called EPT.

Chapter 2

On choosing parameters

In this chapter, we motivate the study and results addressed in this thesis, and give an outline of the remaining parts.

2.1 Dynamic programming on structure

Although the hypothesis P≠NP dictates that NP-hard problems do not have polynomial time algorithms in general, even the most notorious NP-complete problems can be solved quickly when restricting the class of input. Let us take INDEPENDENT SET as an example. The problem INDEPENDENT SET is NP-complete, yet if the input graph G is a tree we can easily solve INDEPENDENT SET in polynomial time by rooting the tree and doing dynamic programming in a bottom up traversal of the tree from the leaves to the root. This works because each vertex v in the graph disconnects the subtrees rooted at its children. Even if the input graph is not a tree, but has low treewidth, we can solve INDEPENDENT SET quickly, since each bag X_t in a rooted tree-decomposition disconnects each of the subgraphs induced by the bags of each subtree rooted at its children. In this way we solve INDEPENDENT SET using the structure inherent in a tree-decomposition, giving an FPT runtime parameterized by the treewidth, $\text{tw}(G)$, of the graph. In this thesis we will focus on that kind of FPT algorithms, namely the ones where we base our algorithm on first computing a decomposition certifying some structural property, and then using dynamic programming on said decomposition yielding solutions to the given problem with runtime depending on the decomposition computed. Here is the general scheme of many FPT algorithms solving a problem on input G[1] of size n, parameterized by some parameter x, using dynamic programming over structural decompositions:

[1] G is usually a graph. However, in Chapter 8 the structural DP-scheme will be used with CNF-formulas as input instead of a graph as input.

`[P1]`: Compute a decomposition $\mathcal{D}$ for the respective structural parameter x in $f(x(G))n^{\mathcal{O}(1)}$ time, so that the width of the structure is some value $x'(\mathcal{D}) = g(x(G))$.

`[P2]`: Solve the problem on G through dynamic programming over $\mathcal{D}$, or a decomposition computed easily from $\mathcal{D}$, with a runtime of $f'(x'(\mathcal{D}))n^{\mathcal{O}(1)}$.

From this point and onwards, let us refer to the above framework as the *structural DP-scheme.* Using this scheme, we get a total runtime $(f(x(G)) + f'(g(x(G))))n^{\mathcal{O}(1)}$ which is FPT. Many studies focus only on `[P2]`, assuming that a decomposition $\mathcal{D}$ of width $g(x(G))$ has already been computed and is given along with the input instance. A solution to `[P2]` does not imply that we can solve the problem in FPT time parameterized by $x(G)$, unless also `[P1]` is solved in FPT time. This was the case for many years regarding graph algorithms parameterized by clique-width; an FPT algorithm approximating a decomposition (a k-expression) of width $f(x(G))$ was not found until 2006 by Oum and Seymour [79], while algorithms for `[P2]` using clique-width as a parameter were in use before this. In particular, the celebrated meta-theorem by Courcelle et al. [31] that MSO_1 is FPT parameterized by clique-width (Theorem 2.3), was introduced in 2000 and needed a k-expression given along with the input. Now, because of [79], this extra limitation is not necessary.

One important aspect of the above structural DP-scheme, that we see illustrated in the story of clique-width, is that we do not need to find an optimal decomposition in `[P1]`. Any $f(x(G))$-approximation will do fine in terms of fixed parameter tractability. In fact, for clique-width, we still do not know whether or not an FPT algorithm exists for computing the optimal k-expression, so an approximation must be used in order to achieve FPT runtimes for current clique-width algorithms. This is also the case for other width-parameters studied in this thesis, like boolean-width and also mim-width. However, in Chapter 6 we will see that we actually cannot hope to compute an optimal decomposition for mim-width in FPT time unless FPT=W[1].

For a complete algorithm based on the structural DP-scheme, we must therefore weigh the runtime cost in `[P1]` of constructing an optimal decomposition against the runtime cost in `[P2]` for non-optimal decompositions. Consider treewidth as an example. The best algorithm for computing the optimal tree-decomposition is a $2^{\mathcal{O}(\text{tw}(G)^3)}n$-time algorithm by Bodlaender [12], while we can produce a constant approximation of the optimal treewidth in linearly single exponential time – $2^{\mathcal{O}(\text{tw}(G))}n^{\mathcal{O}(1)}$ – by Robertson and Seymour [90]. Thus, dynamic programming algorithms that spend asymptotically less time than $2^{\mathcal{O}(\text{tw}(\mathcal{D})^3)}n^{\mathcal{O}(1)}$ when given a decomposition $\mathcal{D}$ (i.e., part `[P2]`) will benefit from using an approximation for the first part, `[P1]`, since `[P1]` otherwise will become the bottle-neck of the algorithm.

This method of comparison also works across parameters: given an optimal decomposition (i.e. avoiding `[P1]`), Independent Set parameterized by either rank-width or clique-width both yield linearly single exponential $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ runtimes. However, for `[P1]`, in FPT time a decomposition for rank-width can be much better approximated than clique-width, and thus Independent Set using algorithms based on rank-width give better runtimes in general than those based on clique-width.

Arguably, the most natural way to hierarchically decompose a graph is through branch decompositions. Identifying the fundamental elements of the input, and recursively splitting the set of these elements into smaller and smaller parts, is a natural first step of divide and conquer algorithms, and can be well described using rooted branch decompositions. The runtime of an algorithm based on such a branch decomposition will depend on the amount of work needed to later combine partial solutions for the smaller parts induced by each of these "divide"-steps. This often correlates with how the fundamental elements within each divided part interact with the rest of the elements, and hence can be described by a cut-function. For graphs, these fundamental elements are the vertices of the graph.

In this thesis our graph algorithms[2] will use the structural DP-scheme, with a ***branch decomposition over the set of vertices***.

As we will see, this type of decomposition is incredibly versatile, by exchanging the cut function used, and can express many different structural properties of the input.

2.2 Comparing parameters

An important reason for why we can say that parameterizing by rank-width is better than parameterizing by clique-width for Dominating Set, is that the clique-width is always at least as large as the rank-width. For a problem π, to claim that a parameter p_1 is better than parameter p_2 when using the structural DP-scheme, we need to address all of the following three points:

`[A]`: What is the value of p_1 compared to p_2 for a class C of instances?

`[B]`: What is the runtime to find a decomposition for p_1? (`[P1]`)

`[C]`: What is the runtime for solving π when given a decomposition? (`[P2]`)

Leaving out any one of the above three points can lead to an incorrect conclusion regarding which of p_1 and p_2 is the best parameter for problem π. As an example, if when using the structural DP-scheme we get algorithms of runtime $2^{p_1}n^2$ and $(p_2)^{p_2}n^2$ for a problem π, looking only at `[B]` and `[C]`, it seems that p_1 is better than p_2. However, if `[A]` reveals that $p_2 \leq \log(p_1)$ for all instances, then actually p_2 is the better parameter for this problem.

[2]We again note that in Chapter 8 we do not work with graph algorithms. However, the branch decomposition of Chapter 8 can be regarded simply as branch decomposition over the vertices of the *incidence graph* of the input formula.

Not only is clique-width at least as large as the rank-width, but it is also at most a function of rank-width; $\mathrm{rw}(G) \leq \mathrm{cw}(G) \leq 2^{\mathrm{rw}(G)+1}+1$ [79]. When parameters are both upper and lower bounded like this, we say that they have the same *modelling power*. The modelling power of a parameter is the class of instances for which the parameter is bounded. Other parameters with the same modelling power as rank-width and clique-width include boolean-width (Bui-Xuan et al. [22]), NLC-width (Johansson [61]) and module-width[3] (Rao [86]). If the modelling power of one parameter is a strict superset of another, we say that its modelling power is *stronger* than the other, or simply that the parameter is stronger. Likewise, a parameter (and modelling power) is *weaker* than another parameter if its modelling power is a strict subset of the other. Note that parameters can also have incomparable modelling powers, such as the two graph parameters treewidth and maximum-degree[4]. See Figure 2.1 for an overview of the weaker/stronger relationships between the parameters mentioned in this thesis.

2.3 The outline of the thesis

In our investigation on choosing parameters to get the best correlation between parameter value and actual runtime complexity, we will look at the following four main cases:

1) Using maximum matching-width as an alternative parameter to treewidth for a faster FPT algorithm for Dominating Set. Chapter 3.
2) Finding a graph parameter that lies strictly between clique-width and treewidth, both in terms of modelling power, and in terms of problems being FPT. Chapter 4 and Chapter 5.
3) Finding an alternative parameter for solving $[\sigma, \rho]$-problems that gives new runtime results, which when expressed in terms of clique-width, strictly improves previous runtime analyses. Chapter 7.
4) Finding a parameter of CNF formulas better than previous structural parameterizations for solving #SAT and MaxSAT. Chapter 8.

In all four cases we make use of dynamic programming algorithms over branch decompositions that follow the structural DP-scheme described earlier (`[P1]` and `[P2]`), and in order to claim anything with respect to previous results, we need to address all the points `[A]`, `[B]`, and `[C]` from the previous section. In addition to the four case studies, we have an extra chapter where we give new (in-)tractability results on hardness of computing optimal decompositions.

5) Showing that for some parameters computing an optimal decomposition `[P1]`

[3]Module-width goes by the name of modular-width in [86], but the parameter investigated in [45] is also named modular-width and is what we refer to when the name modular-width is being used.

[4]The maximum degree of the particular graph. The class of all grids have vertices of degree at most four, while the treewidth of grids is unbounded (Robertson and Seymour [89]), and the class of all stars have unbounded degree while the treewidth is one.

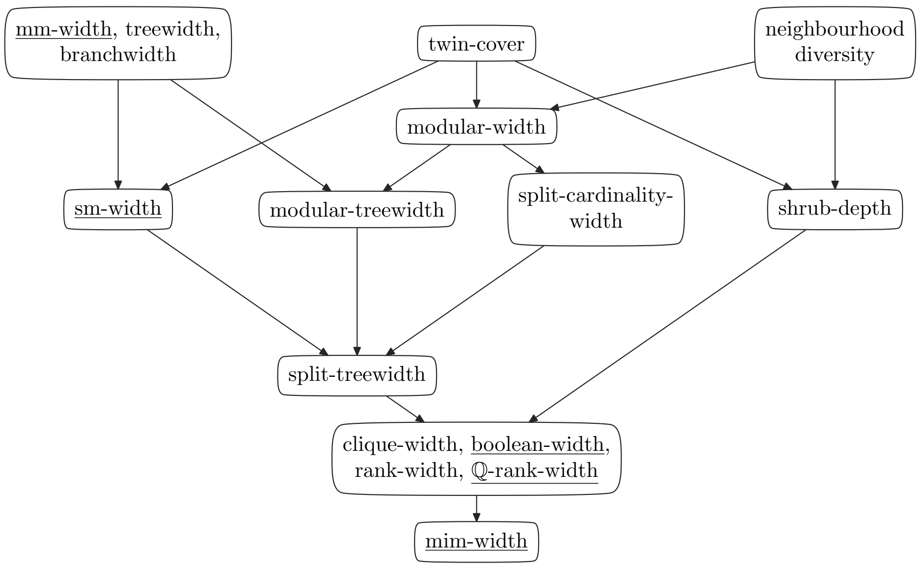


Figure 2.1: A Hasse-diagram depicting the relationship of the modelling powers of some of the parameters mentioned in this thesis. An arrow from parameter a to parameter b means that a is weaker than b. Parameters of the same modelling power are joined together. We investigate the underlined parameters in this thesis: In Chapter 3 we look at mm-width, in Chapter 4 and Chapter 5 we look at sm-width, in Chapter 6 we look at boolean-width and mim-width (mim-width is also addressed in Chapter 8), and in Chapter 7 we look at ℚ-rank-width. We also look at the parameter projection-satisfiable-width (ps-width), which is excluded from this figure, but can be found in a similar figure in Chapter 8 (Figure 8.1) relating its modelling power to that of other parameters.

is W[1]-hard or NP-hard. Chapter 6.

We will now give a brief description of each of the four main cases studies 1–4, and also 5).

2.3.1 Case 1 (Maximum matching-width)

We start our series of case studies by an investigation of the newly defined graph parameter maximum matching-width, which is of equal modelling power as treewidth. Treewidth is not defined through the general decomposition framework of branch decompositions. With the same modelling power as treewidth, we have the parameter branch-width, which is defined on branch decompositions, and is linearly bounded by treewidth, as shown by Robertson and Seymour [89]. However, branch-width is defined through branch decompositions over the edges, $E(G)$, and not the vertices, $V(G)$. Not long ago, Vatshelle [107] defined the parameter maximum matching-width which *is* defined through branch decompositions over $V(G)$, and is linearly bounded by the treewidth, as we see from the following theorem.

Theorem 2.1 ([107]). *Let G be a graph, then $\frac{1}{3}(\mathrm{tw}(G)+1) \leq \text{mm-w}(G) \leq \mathrm{tw}(G)+1$*

In this case study, we investigate properties of branch decompositions of low maximum matching-width, and deduce an alternative characterization of $\text{mm-w}(G) = k$ graphs through an intersection of subtrees representation, very similar to an alternative definition of treewidth. Through this alternative characterization, we can easily go back and forth between tree-decompositions of bounded treewidth and branch decompositions of bounded MM-width.

We will generate an algorithm for solving Dominating Set, parameterized by maximum matching-width, which turns out to be faster than the current best algorithm for treewidth whenever $\mathrm{tw}(G) > 1.55\,\text{mm-w}(G)$. Our algorithm makes use of properties that treewidth lacks in its description, namely the size of intersections between adjacent bags. As we will see in Chapter 3, maximum matching-width to some extent measures this property better than treewidth, and when this property becomes dominant in the runtime analysis, parameterizing by mm-width instead of treewidth gives a better runtime.

As mentioned earlier, for algorithms following the structural DP-scheme of parts `[P1]` and `[P2]`, both parts are necessary for a complete algorithm, and there will usually be a trade-off between the runtime of `[P1]` and how good the decomposition produced by `[P1]` is. For Dominating Set parameterized by treewidth the best runtime, in terms of the exponential part, is obtained by the following combination, with `[P1]` by Amir [3], and `[P2]` by van Roij et al. [105]:

`[P1]`: Compute in $8^{\mathrm{tw}(G)} n^{\mathcal{O}(1)}$ time a tree-decomposition $\mathcal{D}$ of treewidth at most $\mathrm{tw}(\mathcal{D}) \leq (3+2/3)\,\mathrm{tw}(G)$.

`[P2]`: Using $\mathcal{D}$ from `[P1]`, solve Dominating Set in time $3^{\mathrm{tw}(\mathcal{D})} n^{\mathcal{O}(1)}$.

This results in a total runtime of $3^{(3+3/2)\,\mathrm{tw}(G)} n^{\mathcal{O}(1)}$, which is approximately $2^{5.8\,\mathrm{tw}(G)} n^{\mathcal{O}(1)}$. There are other alternatives for `[P1]`, yielding decompositions of smaller treewidth [3, 12, 14], but then the runtime of `[P1]` takes more time than the entire algorithm of the current approach. Our algorithm for DOMINATING SET parameterized by mm-width also uses the same structural DP-scheme, and it proceeds as follows:

`[P1]`: Based on submodularity of the cut function mm, compute in $8^{\text{mm-w}(G)} n^{\mathcal{O}(1)}$ time a branch decomposition $\mathcal{D}$ of mm-width at most $\text{mm-w}(\mathcal{D}) \leq 3\,\text{mm-w}(G) + 1$.

`[P2]`: Using $\mathcal{D}$ from `[P1]`, solve DOMINATING SET in time $8^{\text{mm-w}(\mathcal{D})} n^{\mathcal{O}(1)}$.

Here `[P2]` is a completely new algorithm, and the approximation of `[P1]` is based on a new result of submodularity of mm, and a known general algorithm of Oum and Seymour approximating branch decompositions with respect to cut functions that are submodular [79]. The result of this is an algorithm for DOMINATING SET with a runtime of $2^{9\,\text{mm-w}(\mathcal{D})} n^{\mathcal{O}(1)}$. Thus, if the treewidth is at least $9/5.8 \approx 1.55$ times the mm-width, our algorithm beats the one for treewidth. We note that if we only compare `[P2]` of the two algorithms, then our mm-width approach only beats treewidth when it is at least $\log_3(8) \approx 1.9$ times the mm-width.

Vatshelle showed that the bounds of Theorem 2.1 are tight, and thus we can expect there to be many instances for which using the mm-width based algorithm will outperform the treewidth algorithm, even though treewidth needs to be larger than the mm-width. Note, that we do not hope to get an improved runtime for all instances, as the best runtime for DOMINATING SET parameterized by treewidth, $\mathcal{O}^*(3^{\mathrm{tw}(G)})$ (when given an optimal tree-decomposition), is optimal under the Strong Exponential Time Hypothesis, as shown by Lokshtanov et al. [67].

To sum up in terms of `[A]`, `[B]`, and `[C]`: For `[C]` our algorithm is worse than that of treewidth, but since both `[A]` and `[B]` compare favourably over treewidth/tree-decompositions, for classes of inputs where `[A]` says mm-width is sufficiently much smaller than treewidth, our algorithm for DOMINATING SET based on mm-width is faster than the DOMINATING SET-algorithm based on treewidth.

2.3.2 Case 2 (between treewidth and clique-width)

Among the most famous structural parameters for graphs, we have the parameters treewidth and clique-width. The modelling power of clique-width is stronger than treewidth, but as we see from the following two celebrated theorems, this power has its cost in terms of tractability.

Theorem 2.2 ([29, 90]). *Any problem expressible in* MSO_2 *is fixed parameter tractable when parameterized by the treewidth.*

Theorem 2.3 ([31, 79]). *Any problem expressible in* MSO_1 *is fixed parameter tractable when parameterized by the clique-width.*

Every problem expressible in MSO_1 can be expressed in MSO_2, but the converse is not true, and Fomin et al. [41] have shown that there are problems expressible in MSO_2 which are not FPT (unless FPT=W[1]) parameterized by clique-width, so there is indeed a trade-off between having the better modelling power of clique-width, or having the fixed parameter tractability of using treewidth. In fact Courcelle et al. [31] showed that, for any parameter p that includes the class of cliques in its modelling power, we cannot hope to have all problems expressible in MSO_2 FPT parameterized by p unless P=NP for unary languages.

Some of the most well known problems in MSO_2 but not in MSO_1 are MaxCut, Chromatic Number, Hamiltonian Cycle and Edge Dominating Set. These four problems are easily solvable for cliques, so they are not in the implication of [31], but Fomin et al. [41, 42] have showed that none of them can be FPT parameterized by clique-width, unless FPT=W[1].

In this case study, we will look for a parameter whose modelling power contains the cliques, and which is weaker than clique-width but stronger than treewidth, and for which the above four graph problems are FPT. We note that one can easily define trivial parameters having these properties (e.g. value equal to clique-width if this is at most 3, and otherwise equal to treewidth, since the four problems are in XP parameterized by clique-width), but we want a more natural parameter, which gives tractability results beyond what is already known.

This case study starts in this very subsection, where we search through possible candidate parameters, before finally arriving at our parameter choice *split-matching-width*.

There are several suggestions for parameters that are weaker than clique-width while being bounded also on some dense graphs, but few satisfy our criteria:

- ([A]) Its modelling power must include the cliques, but be weaker than clique-width, and stronger than treewidth.
- ([B]) We must be able to compute a decomposition of bounded width in FPT time.
- ([C]) Given a decomposition of bounded width, we must be able to solve the following four problems in FPT time.
  - Hamiltonian Cycle
  - MaxCut
  - Edge Dominating Set
  - Chromatic Number.

Suggested parameters that contain the class of cliques and are weaker than clique-width, but which fail our criteria on [A] of being stronger than treewidth are listed below. All of them satisfy our constraint on [B], but only partially our constraints on [C].

- neighbourhood diversity (Lampis [65])
- twin-cover (Ganian [46])
- shrub-depth (Ganian et al. [47])
- modular-width (Gajarský et al. [45])

The latter of these parameters, modular-width, arises as the maximum cardinality over all prime graphs in the modular decomposition of the input graph. A generalization of this, is to instead take the maximum treewidth over all prime graphs in the modular decomposition. We then have the following parameter:

- modular-treewidth (Paulusma et al. [81])

Modular-treewidth is stronger than treewidth, weaker than clique-width, and has the class of cliques in its modelling power, so modular-treewidth satisfies `[A]`. Furthermore, computing a modular decomposition can be done in polynomial time (James et al. [57]), and computing a tree-decomposition for each prime graph can be done in FPT time [12], so also our criteria on `[B]` is satisfied. However, we do not know any FPT algorithms for computing the four problems EDGE DOMINATING SET, CHROMATIC NUMBER, HAMILTONIAN CYCLE, and MAXCUT, so we have not been able to satisfy our criteria on `[C]` for modular-treewidth.

Modular decompositions and split decompositions are very similar, and by the same definitions as modular-width and modular-treewidth, but over split decompositions instead of modular decompositions, we arrive at the following two parameters:

- split-cardinality-width (maximum cardinality of a prime graph of the split decomposition of the input graph)
- split-treewidth (maximum treewidth over all prime graphs of the split decomposition of the input graph)

As with modular-width, split-cardinality-width is not stronger than treewidth, since it is unbounded for the class of cycles. However, similar to modular-treewidth, split-treewidth satisfies our criteria on `[A]` and `[B]`, but we do not know any FPT algorithms for the four problems in our criteria for `[C]`. In fact, CHROMATIC NUMBER parameterized by modular-treewidth, and CHROMATIC NUMBER parameterized by split-treewidth, both are polynomial time Turing reducible to and from the problem PREEMPTIVE SCHEDULING with polynomial weights parameterized by treewidth[5] (while preserving parameter value), which is an interesting open problem by itself.

Out of the parameters looked at so far, only modular-treewidth and split-treewidth can not be outright discarded for failing our criteria, as the criteria on `[A]` and `[B]` are upheld, and we have not managed to prove that `[C]` is not upheld. However, as we have not found a single FPT algorithm for either of the

[5]The PREEMPTIVE SCHEDULING problem can be regarded as a coloring problem where each vertex v has a weight $w_c(v)$ and must be assigned $w_c(v)$ colors, and no adjacent vertices may share the same color. A rough sketch of the (turing-)reduction is as follows: In the forward direction; replace each vertex v by a clique of size $w_c(v)$ having the same neighbourhood as v. In the backwards direction (for split-treewidth, which implies for modular-treewidth also); we guess a coloring number t out of the n possible, and in a bottom up manner over the split decomposition tree (root it arbitrarily), for prime graph G' with parent G^* in the split decomposition tree, sharing marker v, we assume all markers u except v have been assigned weight equal to the minimum number of colors needed by $\mathrm{act}(u : G')$ in a t-coloring. Set all non-markers to weight 1. Then find the minimum number of colors needed by $\mathrm{act}(v : G^*)$ in a t-coloring by taking t minus the maximum weight that v can have in G' while being a preemptive scheduling for G'. Continue in this way upwards the split-decomposition tree

four mentioned problems parameterized by these two parameters, and we want them all to be FPT, we continue our search for a suited parameter.

We landed on the two parameters split-treewidth and modular-treewidth by starting with a dense decomposition, and then modifying it to include treewidth bounded graphs. However, using mm-width and branch decompositions, we can do this the other way around. That is, we construct a decomposition that includes treewidth bounded graphs, and modify it to also include dense graphs. Recall from our discussion of the first case study that mm-width has the same modelling power as treewidth, and that it is defined through branch decompositions over the vertex set. Also, recall that a split is a partition (A,B) of the vertices of a graph so that $G[A,B]$ is a complete bipartite graph plus some isolated vertices (where $|A|,|B| \geq 2$). One way of extending mm-width to a width parameter which also includes some dense graph classes is to modify the cut-function as follows: $\mathrm{sm}(S) = 1$ if $(S, V \setminus S)$ is a split, and $\mathrm{sm}(S) = \mathrm{mm}(S)$ otherwise. This will be our parameter, and we call it *split-matching-width*.

In Chapter 4, we show that this parameter indeed satisfies our criteria for `[A]` and `[B]`, giving an approximation algorithm for `[P1]` and showing that its modelling power is between treewidth and clique-width. And then in Chapter 5 we show that split-matching-width also satisfies our criteria for `[C]`, by giving FPT algorithms for `[P2]` parameterized by the split-matching-width of the decompositions produced by `[P1]` in Chapter 4.

2.3.3 Case 3 ($[\sigma,\rho]$-domination)

In the third case study, we introduce an alternative parameter for $[\sigma,\rho]$-problems, called $\mathbb{Q}$-rank-width, with the same modelling power as clique-width, boolean-width and rank-width. The $[\sigma,\rho]$-problems consist of a large class of domination-type problems that generalize Independent Set and Dominating Set. Recall from earlier in this chapter that we got the best algorithms parameterized by clique-width by actually using an algorithm focusing on rank-width instead of clique-width. For a $[\sigma,\rho]$-problem π, the best clique-width algorithms are found through the following observations, yielding an algorithm of runtime $2^{\mathcal{O}(\mathrm{cw}(D)^2)} n^{\mathcal{O}(1)}$:

- `[A]` $\mathrm{rw}(G) \leq \mathrm{cw}(G)$.
- `[B]` We can compute a branch decomposition $\mathcal{D}$ of rank-width $3\,\mathrm{rw}(G)+1$ in time $8^{\mathrm{rw}(G)} n^{\mathcal{O}(1)}$.
- `[C]` We solve π in time $2^{\mathcal{O}(\mathrm{rw}(D)^2)} n^{\mathcal{O}(1)}$.

We will show in Chapter 7 that using the same algorithms, but over a decomposition of low $\mathbb{Q}$-rank-width instead of low rank-width, we get the following:

- `[A]` $\mathrm{rw}_{\mathbb{Q}}(G) \leq \mathrm{cw}(G)$.
- `[B]` We can compute a branch decomposition $\mathcal{D}$ of rank-width $3\,\mathrm{rw}(G)+1$ in time $8^{\mathrm{rw}_{\mathbb{Q}}(G)} n^{\mathcal{O}(1)}$.
- `[C]` We solve π in time $(\mathrm{rw}_{\mathbb{Q}}(D))^{\mathcal{O}(\mathrm{rw}_{\mathbb{Q}}(D))} n^{\mathcal{O}(1)}$.

This results in a faster algorithm – $2^{\mathcal{O}(\text{cw}(D)\log(\text{cw}(D)))}n^{\mathcal{O}(1)}$ – in terms of clique-width. For both the rank-width approach and the $\mathbb{Q}$-rank-width approach, the same algorithm is used in `[P2]`; namely a general algorithm of Bui-Xuan et al. [23], that solves $[\sigma, \rho]$-problems using d-neighbourhood equivalence classes, and with runtime that depends on the $\texttt{nec}_d$-width, $\texttt{nec}_d(\mathcal{D})$, of the input decomposition $\mathcal{D}$. We show in Chapter 7 that our new parameter $\mathbb{Q}$-rank-width better bounds the $\texttt{nec}_d$-width than rank-width, and thus the runtime of `[P2]` of actually solving the problem π has a smaller growing function when expressed in terms of $\mathbb{Q}$-rank-width than in terms of rank-width. We may note that rank-width is upper bounded by the $\mathbb{Q}$-rank-width, and could potentially be much smaller than $\mathbb{Q}$-rank-width for certain instances.

Note also that there are faster algorithms than ours for solving Dominating Set and Independent Set parameterized by clique-width if an optimal k-expression is given as part of the input. However, we only have exponential approximations of clique-width, but linear approximations of both rank-width and $\mathbb{Q}$-rank-width.

2.3.4 Case 4 (parameterizing #SAT and MaxSAT)

This case study differs from the rest in two ways; the first difference being that we do not look at a graph problem, and the second difference being that our result, although based on the structural DP-scheme, does not include an algorithm `[P1]` for computing a branch decomposition of bounded width. We only give heuristics for `[P1]`. In this study we look at solving #SAT and MaxSAT through dynamic programming over branch decompositions over the set of variables and clauses of the input CNF formula, and not the vertices of a graph. We give an algorithm for the last part, `[P2]`, of the structural DP-scheme, which assumes that a branch decomposition is given along with the input. When we design our algorithm in Chapter 8, we first investigate the minimal information necessary at every step of the branch decomposition to get an algorithm that does not depend on structure not actually needed for solving our problems. The result is a general algorithm with runtime depending on the structural parameter *projection satisfiable-width* (ps-width), which tries to measure this minimal information.

As mentioned, we lack the part of constructing a low ps-width branch decomposition. However, for all classes of inputs shown tractable by the use of the structural DP-scheme, Brault-Baron et al. [20] have shown that we can transform the decompositions used in `[P1]` of these previous results, into branch decompositions of low ps-width, replicating the tractability results. So our algorithm for `[P2]` can be seen as a generalization of these previous results. The algorithm also extends beyond the previous results, as we show by pointing out a new class of CNF formulas that can be solved in polynomial time by our algorithm in combination with previous results (for graphs) that help compute a branch decomposition of low ps-width for this particular class of formulas.

By the way we have constructed our algorithm, looking at the minimum information needed over a cut, we make `[P2]` rely on as little information as possible. This means that we can shift the focus of showing parameterized tractability for #SAT and MaxSAT on structural parameters x from constructing algorithms for both [P1] and [P2], to simply constructing branch decompositions of ps-width bounded by the x-width and apply our algorithm for [P2].

2.3.5 Chapter 6 - on the intractability of boolean-width and mim-width

This chapter is complementary to the four main case studies. We address situations where computing an optimal decomposition in [P1] is itself an intractable task. Since its definition, deciding the optimal boolean-width and deciding the optimal mim-width have been thought to be NP-hard to compute, but no proof has been given for this. The motivation for believing that these problems are NP-hard, is from the fact that the cut-functions they are based on are by themselves NP-hard in general.

In this part, we reduce the problems of deciding the cut-value of boolean-width and mim-width, to that of deciding the optimal boolean-width and mim-width, respectively. We thus show that indeed boolean-width and mim-width are NP-hard to compute, and by this finally prove the conjectured hardness of these two problems. In fact we show that deciding mim-width parameterized by said width is W[1]-hard.

2.4 Papers this thesis is based on

This thesis is based on work published in the following papers, which have all been peer reviewed either in its full version, or as an extended abstract for a conference. We emphasise that the case studies are only *based* on these papers.

- The first case study is based on the following paper:

  [60] Jeong, J., Sæther, S. H., and Telle, J. A. Maximum matching width: new characterizations and a fast algorithm for dominating set. To appear in *Proceedings of IPEC 2015.*

- The second case study is based on the following two papers:

  [93] Sæther, S. H., and Telle, J. A. Between treewidth and clique-width. *In proceedings of WG 2014* (2014). Invited to contribute to special section of Algorithmica.

  The full text version of this paper was recently accepted for the special section of Algorithmica, but the publication is not yet finalized.

  [92] Sæther, S. H. Solving Hamiltonian Cycle by an EPT Algorithm for a Non-sparse Parameter. In *Algorithms and Discrete Applied Mathematics.* Springer International Publishing, 2015, pp. 205–216.

  This paper won a best student paper award and was invited to contribute to a special issue of the journal *Descrete Applied Mathemathics*

- The third case study is based on the following paper:

[78] Oum, S.-i., Sæther, S. H., and Vatshelle, M. Faster algorithms for vertex partitioning problems parameterized by clique-width. *Theoretical Computer Science 535* (2014), 16–24.

- The fourth case study is based on the following paper:

[94] Sæther, S. H., Telle, J. A., and Vatshelle, M. Solving MaxSAT and #SAT by dynnamic programming. To appear in *Journal of Artificial Intelligence Research*

- Chapter 6 is based on the following paper:

[95] Sæther, S. H., and Vatshelle, M. Hardness of computing width parameters based on branch decompositions over the vertex set. *To appear in the proceedings of EuroComb 2015* (2015).

Chapter 3

Maximum matching-width

In this chapter, we study the parameter maximum matching-width (mm-width), which was introduced by Vatshelle [107], serving as a connection between treewidth and structural graph parameters defined through branch decompositions over the vertex set, by the use of the following theorem, which we saw in Chapter 2.

Theorem 2.1 ([107]). *Let G be a graph, then* $\frac{1}{3}(\mathrm{tw}(G)+1) \leq \text{mm-w}(G) \leq \mathrm{tw}(G)+1$

We show that the cut function of mm-width is submodular, leading to a $\mathcal{O}^*(8^{\text{mm-w}(G)})$ time 3-approximation algorithm for mm-width (`[P1]`). We also give an algorithm taking a branch decomposition over the vertex set as input and, parameterized by mm-width of this decomposition, solve Dominating Set in time $\mathcal{O}^*(8^{\text{mm-w}(G)})$ (`[P2]`).

Our results are based on a new characterization of graphs of mm-width at most k, as intersection graphs of subtrees of a tree. Combined with similar characterizations for treewidth and branchwidth (see Paul and Telle [80]), it can be formulated as follows, encompassing analogous formulations for all three parameters mm-width (respectively treewidth, respectively branchwidth):

For any $k \geq 2$ a graph G on vertices $v_1, v_2, ..., v_n$ has $\text{mm-w}(G) \leq k$ (resp. $\mathrm{tw}(G) \leq k-1$, resp. $\mathrm{bw}(G) \leq k$) if and only if there is a tree T of max degree at most 3 with nontrivial subtrees $T_1, T_2, ..., T_n$ such that if $v_iv_j \in E(G)$ then subtrees T_i and T_j have at least one node (resp. node, resp. edge) of T in common and for each edge (resp. node, resp. edge) of T there are at most k subtrees using it.

Thus, while treewidth has a focus on nodes and branchwidth a focus on edges, mm-width combines the aspects of both. We also arrive at the following alternative characterization: a graph G has $\text{mm-w}(G) \leq k$ if and only if it is a subgraph of a chordal graph H and for every maximal clique X of H there exists $A, B, C \subseteq X$ with $A \cup B \cup C = X$ and $|A|, |B|, |C| \leq k$ such that any subset of X that is a minimal separator of H is a subset of either A, B or C.

This chapter is organized as follows. In section 3.1 we start with a discussion of runtimes for dominating set parameterized by treewidth versus mm-width. In section 3.2 we show the cut function of mm-width is submodular and how this leads to a new approximation algorithm for mm-width. In section 3.3 we define unique minimum vertex covers for any bipartite graph, show some monotonicity properties of these, and use this, together with the fact the size of a maximum mathching in a bipartite graph equals the size of a minimum vertex cover, to give the new characterizations of mm-width. In section 3.4 we give the dynamic programming algorithm for dominating set. We end in section 3.5 with some concluding remarks of this case study.

3.1 Discussion of runtimes

We will show that given a branch decomposition over the vertex set of mm-width k we can solve Dominating Set in time $\mathcal{O}^*(8^k)$ (`[P2]`). This runtime beats the $\mathcal{O}^*(3^{\mathrm{tw}(G)})$ algorithm for treewidth [105] whenever $\mathrm{tw}(G) > \log_3 8 \times k \approx 1.893k$. We also show that mm-width has a submodular cut function, which means we can approximate mm-width to within a factor $3\,\text{mm-w}(G)+1$ in $\mathcal{O}^*(2^{3\,\text{mm-w}(G)})$ time using the generic algorithm of [79] (`[P1]` and `[B]`), giving a total runtime for solving dominating set of $\mathcal{O}^*(2^{9\,\text{mm-w}(G)})$. For treewidth we can in $\mathcal{O}^*(2^{3.7\,\mathrm{tw}(G)})$ time [3] get an approximation to within a factor $(3+2/3)\,\mathrm{tw}(G)$ giving a total runtime for solving dominating set of $\mathcal{O}^*(3^{3.666\,\mathrm{tw}(G)})$[1]. This implies that on input G, using maximum matching width gives better exponential factors whenever $\mathrm{tw}(G) > 1.549\,\text{mm-w}(G)$. This is a very nice example of a situation where the comparisons of `[A]`, `[B]`, and `[C]` are tightly coupled, and we see how these three points come together.

We may also compare with branchwidth. Let ω be the *exponent of matrix multiplication*, which practically means 2.3728639 using the current fastest matrix multiplication algorithm [66]. In 2010, Bodlaender et al. [16] gave an $\mathcal{O}^*(3^{\frac{\omega}{2}k})$ time algorithm solving Minimum Dominating Set if an input graph is given with its branch decomposition of width k. This means that given decompositions of $\mathrm{bw}(G)$ and $\text{mm-w}(G)$ our algorithm based on mm-width is faster than the algorithm in [16] whenever $\mathrm{bw}(G) > \log_3 8 \cdot \frac{2}{\omega} \cdot \text{mm-w}(G) > \frac{2\log_3 8}{2.3728639} \cdot \text{mm-w}(G) > 1.6\,\text{mm-w}(G)$.

3.2 Approximating mm-width

In [107] it was shown how any tree-decomposition $\mathcal{D}$ could be turned into a branch decomposition over the vertex set of mm-width at most $\mathrm{tw}(\mathcal{D})+1$. Thus, one way of approximating mm-width is by first computing exactly, or approximating, a tree-decomposition, and then turn it into a branch decomposition of bounded mm-width. In this section, we give an alternative way of approximating mm-width, without the use of a tree-decomposition, based on the following very general result of Oum and Seymour.

[1]Note that there is also a $\mathcal{O}^*(c^{\mathrm{tw}(G)})$ time 3-approximation of treewidth [14], but the c is so large that the approximation alone has a bigger exponential part than the entire Dominating Set algorithm when using the 3.666-approximation

Theorem 3.1 ([79])**.** *For symmetric submodular cut-function f and graph G of optimal f-width k, a branch decomposition of f-width at most $3k+1$ can be found in $\mathcal{O}^*(2^{3k})$ time.*

This theorem is central in the field of branch decompositions. There is no abundance of submodular cut functions, but the cut function mm turns out to be one of them. That mm is submodular and that we can approximate the mm-width is a very crucial part of the approximation of the parameter sm-width in Chapter 4.

Theorem 3.2. *The cut function* mm *is submodular.*

Proof. Let G be a graph and $S \subseteq V(G)$. We will say that a matching $M \subseteq E(G)$ is a matching of S if each edge of M has exactly one endpoint in S, i.e. M is a matching of the bipartite graph $G[S,\overline{S}]$. To prove that mm is submodular, we will show that for any $A, B \subseteq V(G)$ and any matching $M_{A\cup B}$ of $A \cup B$ and $M_{A\cap B}$ of $A \cap B$, there must exist two matchings M_A of A and M_B of B so that the multiset of edges $M_A \uplus M_B$ is equal to the multiset $M_{A\cup B} \uplus M_{A\cap B}$. First notice that each edge of $M_{A\cup B}$ and $M_{A\cap B}$ is a matching of either A or B (or both). As the vertices in a matching have degree one, the multiset $M_{A\cup B} \uplus M_{A\cap B}$ of edges can be regarded as a set of vertex disjoint paths and cycles (note though, we might have cycles of size two, as the same edge might be in both of the matchings). We will show that for every such path or cycle P there exist matchings M_A^P for A and M_B^P for B so that $E(P) = E(M_A^P) \cup E(M_B^P)$. Note that this suffices to prove the statement, as there will then also exist matchings M_A of A and M_B of B so that $E(M_A) \uplus E(M_B) = E(M_{A\cup B}) \uplus E(M_{A\cap B})$, by taking M_A and M_B as the disjoint union of each of the smaller matchings, for A and B respectively, that exist for each path or cycle P in $M_{A\cup B} \uplus M_{A\cap B}$. Since these paths and cycles are vertex-disjoint M_A and M_B will be matchings.

Thus, let P be a path or a cycle from $M_{A\cup B} \uplus M_{A\cap B}$. If P only contains vertices of $A \cap B$ and $\overline{A \cup B}$, each edge of P is a matching of both A and B, so we have the matchings by setting $M_A = P \cap M_{A\cap B}$ and $M_B = P \cap M_{A\cup B}$. Since the edges of $E(P)$ alternate between $M_{A\cup B}$ and $M_{A\cap B}$, and since all edges from $M_{A\cup B}$ has an endpoint in $\overline{A \cup B}$ and all edges from $M_{A\cap B}$ has an endpoint in $A \cap B$, there can be at most one vertex v in P belonging to $(B \setminus A) \cup (A \setminus B)$ (it may help to look at Figure 3.1 where it is clear that no path alternating between blue and red edges can touch $(B \setminus A) \cup (A \setminus B)$ twice). If there exists such a vertex v, assume without loss of generality that $v \in B \setminus A$. As each edge in $M_{A\cap B} \cap P$

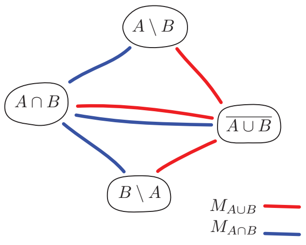


Figure 3.1: The edges of $M_{A\cap B}$ and $M_{A\cup B}$ in the proof of Theorem 3.2.

has exactly one endpoint in $A \cap B$, and P contains vertices only of $A \cap B, B \setminus A$ and $\overline{A \cup B}$, all the edges of $M_{A \cap B} \cap P$ have one endpoint in A and one endpoint in $(B \setminus A) \cup \overline{A \cup B} = \overline{A}$. So, $M_{A \cap B} \cap P$ is a matching of A. For $M_{A \cup B}$, by the same arguments, each edge in $M_{A \cup B} \cap P$ must have one endpoint in $\overline{A \cup B}$ and one endpoint in $(B \setminus A) \cup (A \cap B) = B$, making $M_{A \cup B} \cap P$ a matching of B. □

Corollary 3.3. *Given a graph G of mm-width $\text{mm-w}(G) = k$, a branch decomposition over $V(G)$ of mm-width at most $3k+1$ can be found in $\mathcal{O}^*(2^{3k})$ time.*

3.3 Subtrees of a tree representation for mm-width

Theorem 2.1 shows not only that mm-width is bounded by the treewidth, but also that the treewidth is bounded by the mm-width. However, this latter relation was in [107] not shown constructively. In this section, we give an alternative characterization of mm-width, which gives rise to a polynomial procedure to transform a branch-decomposition of mm-width k into a tree-decomposition of treewidth $3k-1$. This alternative characterization shows a nice analogy between treewidth, branchwidth and mm-width, and is crucial for our dynamic programming algorithm solving DOMINATING SET.

3.3.1 König covers.

In this subsection, we will define canonical minimum vertex covers for any bipartite graph. Our starting point is a well-known result in graph theory.

Theorem 3.4 (König's Theorem [64]). *Given a bipartite graph G, for any maximum matching M and minimum vertex cover C of G, the number of edges in M is the same as the number of vertices in C; $|M| = |C|$.*

Let (A, B) be the vertex partition of G. This statement can be proved in multiple ways. The harder direction, that a maximum matching is never smaller than a minimum vertex cover, does not hold for general graphs, and is usually proven by taking a maximum matching M and constructing a vertex cover C having size exactly $|M|$, as follows:

> For each edge $ab \in M$ (where $a \in A$, and $b \in B$), if ab is part of an alternating path starting in an unsaturated vertex of A, then put b into C, otherwise put a into C.

For a proof that C indeed is a minimum vertex cover of G, see e.g. [36]. We will call the vertex cover C constructed by the above procedure the A-*König cover* of G. A B-König cover of G is constructed similarly by changing the roles of A and B (see figure 3.2). From how a König cover is constructed, it might seem that the choice of M is important for how the resulting set C will be. Lemma 3.5 shows us that actually the König cover does not depend on the choice of M as long as M is a maximum matching. In fact this lemma shows that the A-König cover will, on the A-side consist of the A-vertices in the union over all minimum vertex covers, and on the B-side consist of the B-vertices in the intersection over all minimum vertex covers.

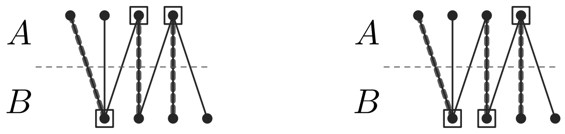


Figure 3.2: A-König cover and B-König cover.

Lemma 3.5. *For a bipartite graph $G = (A \cup B, E)$ and minimum vertex cover C of G, the set C is the A-König cover of G if and only if for any minimum vertex cover C' of G we have $A \cap C' \subseteq A \cap C$, and $B \cap C' \supseteq B \cap C$.*

Proof. Let M be a maximum matching of G, and C^* the A-König cover of G constructed from M. Since both C^* and C are minimum vertex covers, by showing that for any minimum vertex cover C' of G we have $A \cap C' \subseteq A \cap C^*$, and $B \cap C' \supseteq B \cap C^*$, as a consequence will also show that $C' = C^*$ if and only if for all minimum vertex covers C' of G we have $A \cap C' \subseteq A \cap C$ and $B \cap C' \supseteq B \cap C$. So this is precisely what we will do.

Let C' be any minimum vertex cover, and b any vertex in $C^* \cap B$. We will show that $b \in C'$, and from that conclude $B \cap C' \supseteq B \cap C^*$. As $b \in C^*$ there must be some alternating path from b to an unsaturated vertex $u \in A$. The vertices b and u are on different sides of the bipartite graph, so the alternating path P between u and b must be of some odd length $2k+1$. From Theorem 3.4, we deduce that one and only one endpoint of each edge in M must be in C'. As each vertex in $V(P)$ is incident with at most two edges of P, and all edges of P must be covered by C', we need at least $\lceil (2k+1)/2 \rceil = k+1$ of the vertices in $V(P)$ to be in C'. However, the vertices of $V(P) - b$ are incident with only k edges of M. Therefore at most k of the vertices $V(P) - b$ can be in C'. In order to have at least $k+1$ vertices from $V(P)$ in C' we thus must have $b \in C'$.

We now show that $C' \cap A \subseteq C^*$ by showing that $a \in C^*$ if $a \in A \cap C'$. Let E^* and E' be the edges of G not covered by $C^* \cap B$ and $C' \cap B$, respectively. Since $C^* \cap B \subseteq C' \cap B$, the set E^* must contain all the edges of E'. As C' is a minimum vertex cover, and all edges other than E' are covered by $C' \cap B$, a vertex a of A is in C' only if it covers an edge $e \in E'$. As $E' \subseteq E^*$, we have $e \in E^*$, and hence C^* must also cover e by a vertex in A. As G is bipartite, the only vertex from A that covers e is a, and we can conclude that $a \in C^*$. □

The following lemma establishes an important monotonicity property for A-König covers.

Lemma 3.6. *Given a graph G and tripartition (A, B, X) of the vertices $V(G)$, the following two properties hold for the A-König cover C_A of $G[A, B \cup X]$ and any minimum vertex cover C of $G[A \cup X, B]$.*

1. $A \cap C \subseteq A \cap C_A$

2. $B \cap C \supseteq B \cap C_A$.

Proof. To prove this, we will show that it holds for $X = \{x\}$, and then by transitivity of the subset relation and that a König cover is also a minimum vertex cover, it must hold also when X is any subset of $V(G)$.

Let $A' = A + x$ and $B' = B + x$, and let C' be the A-König cover of the graph $G[A,B]$ (be aware that this graph has one less vertex than G). We will break the proof into four parts, namely $A \cap C \subseteq A \cap C'$, $A \cap C' \subseteq A \cap C_A$, $B \cap C_A \subseteq B \cap C'$, and $B \cap C' \subseteq B \cap C$. Again, by transitivity of the subset relation, this will be sufficient for our proof. We now look at each part separately.

$\underline{A \cap C \subseteq A \cap C'}$: Two cases: $|C| = |C'|$ and $|C| > |C'|$. We do the latter first. This means that $C' \cup \{x\}$ must be a minimum vertex cover of $G[A',B]$. Therefore the A'-König cover C^* of $G[A',B']$ must contain $(C' \cup \{x\}) \cap A'$. This means that C^* is a minimum vertex cover of $G[A,B]$, and by C' being the A-König cover of $G[A,B]$, we have from Lemma 3.5 that $C' \cap A \supseteq C^* \cap A$. And since C^* is a A'-König cover of $G[A',B]$ we have $C' \cap A' \supseteq C \cap A'$ and can conclude that $C' \cap A \supseteq A \cap C$. Now assume that the two vertex covers are of equal size. Clearly $x \notin C$, as then $C - x$ is a smaller vertex cover of $G[A,B]$ than C', so x is not in C. This means that C is a minimum vertex cover of $G[A,B]$, so all vertices in $A \cap C$ must be in C' by Lemma 3.5.

$\underline{A \cap C' \subseteq A \cap C_A}$: Suppose C' is smaller than C_A. This means $C' + x$ is a minimum vertex cover of $G[A,B']$, and hence $(C' + x) \cap A \subseteq C_A \cap A$ by Lemma 3.5. On the other hand, if C' is of the same size as C_A. Then C_A is a minimum vertex cover of $G[A,B]$, and so $x \notin C_A$. This means $C_A \cap N(x) \cap A \subseteq C_A \cap A$. And as C_A is a minimum vertex cover of $G[A,B]$, we know from Lemma 3.5 that $C_A \cap N(x) \cap A \subseteq C'$. In particular, this means C' covers all the edges of $G[A,B']$ not in $G[A,B]$, which means that C' is also a minimum vertex cover of $G[A,B']$. This latter observation means that $C' \cap A \subseteq C_A \cap A$ from Lemma 3.5.

$\underline{B \cap C_A \subseteq B \cap C'}$: Suppose C' is smaller than C_A. This means $C' + x$ is a minimum vertex cover of $G[A,B']$, and thus $B' \cap (C' + x) \supseteq B' \cap C_A$. Which implies that $B \cap C' \supseteq B \cap C_A$. Now assume that C' is of the same size as C_A. This means C_A is a minimum vertex cover of $G[A,B]$ and $x \notin C_A$. Furthermore, this means $N(x) \cap A \subseteq C_A \cap A \subseteq C' \cap A$ by Lemma 3.5 and we conclude that C' is a minimum vertex cover of $G[A,B']$. By Lemma 3.5, this means $B' \cap C_A \subseteq B' \cap C'$ and in particular $B \cap C_A \subseteq B \cap C'$.

$\underline{B \cap C' \supseteq B \cap C}$: Suppose C' is smaller than C. This means $C' + x$ is a minimum vertex cover of $G[A,B']$, and hence by Lemma 3.5 we have $B' \cap (C' + x) \subseteq B' \cap C_2$, which implies $B \cap C' \subseteq B \cap C_2$. Now suppose C' is of the same size as C. This means that C is a minimum vertex cover of $G[A,B]$, and hence we immediately get $C \cap B \supseteq C' \cap B$ by Lemma 3.5.

This completes the proof, as we by transitivity of the subset relation have that $C_A \cap B \subseteq C \cap B$, and $C \cap A \subseteq C_A \cap A$. □

We are now ready to prove an important connectedness property of König covers that arise from cuts of a given branch decomposition.

Lemma 3.7. *Given a connected graph G and rooted branch decomposition (T,δ) over $V(G)$, for any node v in T, where $\mathcal{C}$ are the descendants of v and C_u means the $\delta(u)$-König cover of $G[\overline{\delta(u)},\delta(u)]$, we have that*

$$\left(\bigcup_{x \in V(T) \setminus \mathcal{C}} C_x\right) \cap \left(\bigcup_{x \in \mathcal{C}} C_x\right) \subseteq C_v \ .$$

Proof. First notice for all $x \in \mathcal{C}$, since C_x is a $\delta(x)$-König cover and C_v a minimum vertex cover, from Lemma 3.6 we have that $C_x \cap \overline{\delta(x)} \subseteq C_v \cap \overline{\delta(x)}$. In particular, since $\delta(x) \subseteq \delta(v)$, we have that $C_x \setminus \delta(v) \subseteq C_v \setminus \delta(v) \subseteq C_v$. Since each vertex of $V(G)$ is either in $\delta(v)$ or not in $\delta(v)$, by showing that also for all $x \in (V(T) \setminus \mathcal{C})$ we have $C_x \cap \delta(v) \subseteq C_v$ we can conclude that the lemma holds: For all $x \in V(T) \setminus \mathcal{C}$ either $\delta(v) \subseteq \delta(x)$ (when x is an ancestor of v) or $\delta(v) \subseteq \overline{\delta(x)}$ (when x is neither a descendant of v nor an ancestor of v), in either case, we can apply the $\delta(v)$-König cover C_v of $G[\delta(v), \overline{\delta(v)}]$ and the minimum vertex cover C_x of $G[\delta(x), \overline{\delta(x)}]$ to Lemma 3.6 and see that $C_x \cap \delta(v) \subseteq C_v \cap \delta(v) \subseteq C_v$. □

3.3.2 The new characterization of mm-width

We say a graph is *nontrivial* if it has an edge.

Theorem 3.8. *A nontrivial graph $G = (V, E)$ has* mm-w$(G) \leq k$ *if and only if there is a tree T of max degree at most 3 and for each vertex $u \in V$ a nontrivial subtree T_u of T such that i) if $uv \in E$ then the subtrees T_u and T_v have at least one vertex of T in common, and ii) for every edge of T there are at most k subtrees using this edge.*

Proof. Forward direction: Let (T, δ) be a rooted branch decomposition over V having mm-width at most k, and assume G has no isolated vertices. For each edge $e = uv$ of T, with u a child of v, assign the $\delta(u)$-König cover C_u of $G[\delta(u), V \setminus \delta(u)]$ to the edge uv. For each vertex x of G, define the set of edges of T whose König cover contains x and let T_x be the sub-forest of T induced by these edges. Using Lemma 3.7 we first show that T_x is a connected forest and thus a subtree of T. Consider edge $e = uv$ of T. Let p be the lowest common ancestor of u and v. For every vertex w on the path from p to u and on the path from p to v, except p, we know that exactly one of u, v is a descendant of w. By Lemma 3.7, $(C_u \cap C_v) \subseteq C_w$. It means that if a vertex x of G is in both C_u and C_v then it is also in C_w, which implies that T_x is connected.

Now, since the branch decomposition has mm-width at most k part ii) in the statement of the Theorem holds. For an arbitrary edge ab of G, consider any edge e of T on the path from $\delta^{-1}(a)$ to $\delta^{-1}(b)$ and the partition (A, B) induced by e where $a \in A$, $b \in B$. Then the König cover of e must contain one of a and b, and thus, i) holds as well. Finally, T_x is nontrivial because the edge of T incident with a leaf $\delta^{-1}(x)$ assigns the König cover $\{x\}$. If G has isolated vertices, T_x is not nontrivial for isolated vertex x. We fix this by setting T_x to consist exactly of the edge incident with $\delta^{-1}(x)$, for any isolated vertex x of G.

Backward direction: For each given subtree $\{T_u\}_{u \in V}$ of T, choose an edge in T_u (it is also in T) and append in the tree T a leaf ℓ_u, and extend T_u to contain ℓ_u and set $\delta(\ell_u) = u$. Exhaustively remove leaves (from both T and the subtrees) that are not mapped by δ. Call the resulting tree T' and subtrees $\{T'_u\}_{u \in V}$. Note that subtrees $\{T'_u\}_{u \in V}$ and T' still satisfy i) and ii). We claim that (T', δ) is a branch decomposition of mm-width at most k. It is clearly a branch decomposition over V, and for any edge e of T', if we choose $S \subseteq V$ to be those u with T_u using this edge e, then this will be a vertex cover of the bipartite graph H given by this edge e, and of size at most k because for an edge xy in H, one of T_x and T_y must contain e. $\square$

There are analogous characterizations also of treewidth and branchwidth, see Paul and Telle [80]. Combining these characterizations with ours of Theorem 3.8, we arrive at the characterization stated in the first section of this chapter, which encompass all three parameters mm-width, treewidth, and branchwidth. Using this new characteristic of mm-width in combination with the characteristic of treewidth, we get the following theorem solving an open problem of [107]:

Theorem 3.9. *Given a branch decomposition $\mathcal{D}$ over $V(G)$ of mm-width* $\text{mm-w}(\mathcal{D}) = k$*, we can in $\mathcal{O}(n^{3.5})$ time construct a tree-decomposition $\mathcal{D}'$ of treewidth* $\text{tw}(\mathcal{D}') \leq 3k-1$*. Moreover, the decomposition $\mathcal{D}'$ has max degree three and the intersection between two adjacent bags in $\mathcal{D}'$, and the bags at the leaves of $\mathcal{D}'$, are of cardinality at most k.*

Proof. The proof of Theorem 3.8 was constructive, meaning that given a branch decomposition $\mathcal{D}$ of $\text{mm-w}(\mathcal{D}) = k$, we showed how to construct a "subtrees of a tree"-representation conforming to the alternative characterization, based on König covers of the cuts induced by $\mathcal{D}$. So, since finding a maximum matching of a bipartite graph can be done in $\mathcal{O}(n^{2.5})$ time, by Hopcroft [54], and transforming this to a König cover only takes an additional linear time suppressed by the $\mathcal{O}$-notation, we can in fact construct the "subtrees of a tree"-representation T with subtrees $\{T_v : v \in V(G)\}$ in $\mathcal{O}(n^{3.5})$ time (the extra n comes from iterating over all cuts induced by $\mathcal{D}$). To construct a tree-decomposition from this, simply have the tree-decomposition $\mathcal{D}' = (T, \{X_u : u \in V(T)\})$ where $X_u = \{v \in V(G) :$ so that $u \in V(T_v)\}$. This is a tree-decomposition of treewidth at most $3k-1$, because:

(1) each vertex $v \in V(G)$ is in some bag X_u for $u \in V(T)$, in particular in all the bags $\{X_u : u \in V(T_v)\}$.
(2) for any edge $vw \in E(G)$, there must be a bag X_u containing both v and w, since we know T_v and T_w must intersect in at least one node u.
(3) The bags containing any node v of $V(G)$ must form a subtree, since the bags containing v exactly correspond to the vertices of the subtree T_v.

That T has max degree three comes directly from the definition, and that two adjacent bags X_u, X_v intersect by at most k vertices is because the intersection between X_u and X_v is precisely the set $\{v : uv \in E(T_v)\}$, and thus there can be at most k of these, by definition. $\square$

Another alternative characterization of the mm-width of a graph is the following.

Corollary 3.10. *A graph G has* $\text{mm-w}(G) \leq k$ *if and only if it is a subgraph of a chordal graph H and for every maximal clique X of H there exists $A, B, C \subseteq X$ with $A \cup B \cup C = X$ and $|A|, |B|, |C| \leq k$ such that any subset of X that is a minimal separator of H is a subset of either A, B or C.*

We only sketch the proof, which is similar to an alternative characterization of branchwidth given in [80]. We say a tree is *ternary* if it has maximum degree at most 3. Note that a graph is chordal if and only if it is an intersection graph of subtrees of a tree [50]. In the forward direction, take the chordal graph resulting from the subtrees of ternary tree representation. In the backward direction, take a clique tree of H and make a ternary tree-decomposition (which is easily made into a subtrees of ternary tree representation) by for each maximal clique X of degree larger than three making a bag X with three neighboring bags A, B, C. If minimal separators $S_1, ..., S_q$ subset of X are contained in A make a path extending from bag A of q new bags also containing A, with a single bag containing $S_i, 1 \leq i \leq q$ attached to each of them. These ternary subtrees, one for each maximal clique, is then connected together in a tree by the structure of the clique tree, adding an edge between bags of identical minimal separators.

3.4 DOMINATING SET parameterized by mm-width

For graph $G = (V, E)$ a subset of vertices $S \subseteq V$ is said to *dominate* the vertices in $N[S]$, and it is a *dominating set* if $N[S] = V$. Given a rooted branch decomposition (T, δ) over $V(G)$ of mm-width k, we will in this section give an $\mathcal{O}^*(8^k)$ algorithm for computing the size of a Minimum Dominating Set of G, i.e., the minimization version of DOMINATING SET. This is done by an algorithm doing dynamic programming along a rooted tree-decomposition $(T', \{X_t\}_{t \in V(T')})$ of G that we compute from (T, δ).

We have from Theorem 3.9 that we in polynomial time can compute a tree-decomposition $(T', \{X_t\}_{t \in V(T')})$ of treewidth at most $3k-1$ where intersections between adjacent bags are of size at most k, and bags at the leaves of T' are of size at most k. We extend this decomposition by for each pair of adjacent bags X_u, X_v adding the intersection $X_w = X_u \cap X_v$ as a bag between X_u and X_v in the decomposition, append two empty bags to each leaf, and root the decomposition in an arbitrary leaf r. The resulting decomposition will be of the same treewidth and have properties as described in Figure 3.3.

Let us now define the relevant subproblems for the dynamic programming over this tree-decomposition. For node t of the tree we denote by G_t the graph induced by the union of X_u where u is a descendant of t. A coloring of a bag X_t is a mapping $f : X_t \rightarrow \{1, 0, *\}$ with the meaning that: all vertices with color 1 are contained in the dominating set of this partial solution in G_t, all vertices with color 0 are dominated, while vertices with color $*$ might be dominated, not dominated, or in the dominating set. Thus the only restriction is that a vertex with color 1 must be a dominator, and a vertex with color 0 must be dominated. Thus, for any $S \subseteq V(G)$ there is a set $c(S)$ of $3^{|S|}2^{|N(S)|}$ colorings $f : V(G) \rightarrow \{1, 0, *\}$ compatible with taking S as set of dominators, with vertices of S colored 1, 0 or $*$, vertices of $N(S)$ colored 0 or $*$, and the remaining vertices colored $*$.

For a coloring f of bag X_t we denote by $T[t, f]$ (and view this as a 'Table' of values) the minimum $|S|$ over all $S \subseteq V(G_t)$ such that there exists $f' \in c(S)$ with $f'|_{X_t} = f$ and $f'|_{V(G_t) \setminus X_t}$ having everywhere the value 0. In other words, the minimum size of a set S of vertices of G_t that dominate all vertices in $V(G_t) \setminus X_t$, with a coloring f' compatible with taking S as set of dominators, such that f' restricted to X_t gives f. If no such set S exists, then $T[t, f] = \infty$. Note that the size of the minimum dominating set of G is the minimum value over all $T[r, f]$ where $f^{-1}(*) = \emptyset$ at the root r. We initialize the table at a leaf ℓ, with $X_\ell = \emptyset$ as $T[\ell, f] := 0$ for the only possible coloring $f : \emptyset \to \emptyset$.

For internal nodes of the tree, instead of separate 'Join, Introduce and Forget' operations we will give a single update rule with several stages. We will be using an Extend-Table subroutine which takes a partially filled table $T[t, \cdot]$ and extends it to table $T'[t, \cdot]$ so the result will adhere to the above definition, ensuring the monotonicity property that $T'[t, f] \leq T'[t, f']$ for any f we can get from f' by changing the color of a vertex from 1 to 0 or $*$, or from 0 to $*$. Intuitively, the content of $T'[t, f]$ will be the smallest value over all $T[t, f']$ where $\forall S \ \ f' \in c(S) \to f \in c(S)$. Extend-Table is implemented as follows:

(a) Initialize. For all f, if $T[t, f]$ is defined then $T'[t, f] := T[t, f]$, else $T'[t, f] := \infty$.

(b) Change from 1 to 0. For $q = |X_t|$ down to 1: for any f in $T'[t, f]$ where $|\{v : f(v) = 1\}| = q$, for any choice of a single vertex $u \in \{v : f(v) = 1\}$ set $f_u(u) = 0$ and set $f_u(x) = f(x)$ for $x \neq u$, and update $T'[t, f_u] := \min\{T'[t, f_u], T'[t, f]\}$.

(c) Change from 0 to $*$. Similarly as in step (b).

Note the transition from color 1 to $*$ will happen by transitivity. The time for Extend-Table is proportional to the number of entries in the tables times $|X_t|$.

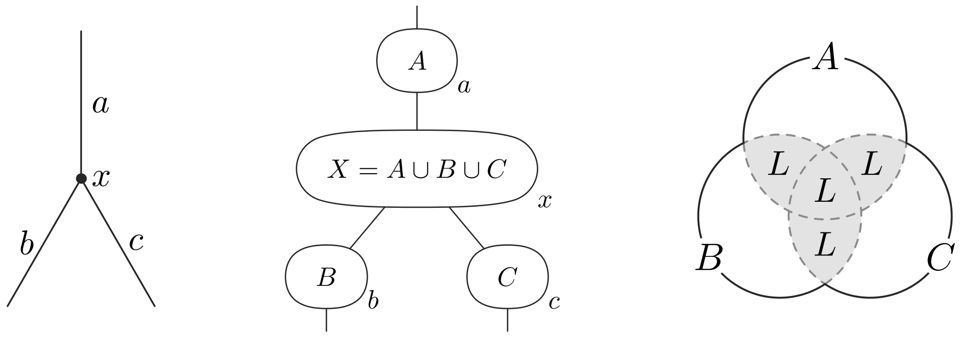


Figure 3.3: Part of ternary tree used in the subtree representation of G on the left, with node x having three incident edges a, b, c, with subtrees of vertices contained in $A, B, C \subseteq V(G)$ using these edges respectively, giving rise to the four bags in the tree-decomposition shown in the middle, with constraint $|A|, |B|, |C| \leq k$.

Assume we have the situation in Figure 3.3, corresponding to the bags surrounding any degree-three node x of the tree-decomposition. This arises from the branch decomposition (and the subtrees of tree representation) having a node incident to three edges, creating three bags a, b, c containing subsets of vertices A, B, C, respectively, each of size at most k, and giving rise to the four bags

a, b, c, x in the figure, with the latter containing subsets of vertices $X = A \cup B \cup C$. Let $L = (A \cap B) \cup (A \cap C) \cup (B \cap C)$. Assume we have already computed $T[b, f]$ and $T[c, f]$ for all $3^{|B|}$ and $3^{|C|}$ choices of f, respectively. We want to compute $T[a, f]$ for all $3^{|A|}$ choices of f, in time $\mathcal{O}^*(\max\{3^{|A|}, 3^{|B|}, 3^{|C|}, 3^{|L|}2^{|X \setminus L|}\})$. Note that we will not compute the table $T[x, \cdot]$, as it would have $3^{|X|}$ entries, which is more than the allowed time bound. Instead, we compute a series of tables:

(1) $T_b^1[x, \cdot]$ (and $T_c^1[x, \cdot]$) of size $3^{|B|}$, by for each entry $T[b, f]$ extending the coloring f of B to a unique coloring f' of X by coloring $N(f^{-1}(1))$ in $X \setminus B$ by '0'.

(2) $T_b^2[x, \cdot]$ (and $T_c^2[x, \cdot]$) of size at most $\min(3^{|B|}, 3^{|B \cap L|}2^{|X \setminus (B \cap L)|})$, by changing each coloring f of X to a coloring f' of X where vertices in $B \setminus L$ having color 1 instead are given color 0 (note these vertices have no neighbors in $V(G) \setminus V(G_x)$)

(3) $T_b^3[x, \cdot]$ (and $T_c^3[x, \cdot]$) of size exactly $3^{|B \cap L|}2^{|X \setminus (B \cap L)|}$, with $f^{-1}(1) \subseteq B \cap L$, by running Extend-Table on $T_b^2[x, \cdot]$

(4) $T_{sc}^1[x, \cdot]$ of size $3^{|L|}2^{|X \setminus L|}$ by subset convolution over parts of $T_b^3[x, \cdot]$ and $T_c^3[x, \cdot]$

(5) $T_{sc}^2[x, \cdot]$ of size $3^{|L|}2^{|X \setminus L|}$ by running Extend-Table on $T_{sc}^1[x, \cdot]$

(6) $T[a, \cdot]$ of size $3^{|A|}$ by going over all $3^{|A|}$ colorings of A and minimizing over appropriate entries of $T_{sc}^2[x, \cdot]$

Note that in step 4 we use the following:

Theorem 3.11 (Fast Subset Convolution [10]). *For two functions $g, h : 2^V \to \{-M, \ldots, M\}$, given all the $2^{|V|}$ values of g and h in the input, all $2^{|V|}$ values of the subset convolution of g and h over the integer min-sum semiring, i.e. $(g * h)(Y) = \min_{Q \cup R = Y \text{ and } Q \cap R = \emptyset} g(Q) + h(R)$, can be computed in time $2^{|V|}|V|^{O(1)} \cdot O(M \log M \log\log M)$.*

Let us now give the details of the first three steps:

(1) Compute $T_b^1[x, \cdot]$. In any order, go through all $f : B \to \{1, 0, *\}$ and compute $f' : B \cup A \cup C \to \{1, 0, *\}$ by

$$f'(v) = \begin{cases} f(x) & \text{if } v \in B \\ 0 & \text{if } v \notin B \text{ and } \exists u \in B : f(u) = 1 \wedge uv \in E(G) \\ * & \text{otherwise} \end{cases}$$

and set $T_b^1[x, f'] := T[b, f]$.

(2) Compute $T_b^2[x, \cdot]$. First, initialize $T_b^2[x, f] = \infty$ for all $f : B \cup A \cup C \to \{1, 0, *\}$ where $f^{-1}(1) \subseteq B \cap L$. In any order, go through all $f : B \cup A \cup C \to \{1, 0, *\}$

such that $T^1_b[x,f]$ was defined in the previous step, and compute $f' : B \cup A \cup C \to \{1,0,*\}$ by

$$f'(v) = \begin{cases} 0 & \text{if } v \in B \setminus L \text{ and } f(v) = 1 \\ f'(v) = f(v) & \text{otherwise} \end{cases}$$

and set $T^2_b[x,f'] := \min\{T^2_b[x,f'], T^1_b[x,f]\}$. There will be no other entries in $T^2_b[x,\cdot]$.

(3) Compute $T^3_b[x,\cdot]$ by Extend-Table on $T^2_b[x,\cdot]$.

The total time for the above three steps is bounded by $\mathcal{O}^*(\max\{3^{|B|}, 3^{|B\cap L|}2^{|X\setminus(B\cap L)|}\})$. Note that $T^3_b[x,f]$ is defined for all f where vertices in $B\cap L$ take on values $\{1,0,*\}$ and vertices in $X \setminus (B\cap L)$ take on values $\{0,*\}$. The value of $T^3_b[x,f]$ will be the minimum $|S|$ over all $S \subseteq V(G_b)$ such that there exists $f' \in c(S)$ with $f'|_X = f$ and $f'|_{V(G_b)\setminus X}$ having everywhere the value 0. Note the slight difference from the standard definition, namely that even though the coloring f is defined on X, the dominators only come from $V(G_b)$, and not from $V(G_x)$. The table $T^3_c[x,\cdot]$ is computed in a similar way, with the colorings again defined on X but with the dominators now coming from $V(G_c)$.

When computing a Join of these two tables, we want dominators to come from $V(G_b) \cup V(G_c)$. Because of the monotonicity property that holds for these two tables, we can compute their Join $T^1_{sc}[x,f]$ for any f where vertices in L take on values $\{1,0,*\}$ and vertices in $X \setminus L$ take on values $\{0,*\}$, by combining colorings as follows:

$$T^1_{sc}[x,f] = \min_{f_b,f_c}(T^3_b[x,f_b] + T^3_c[x,f_c]) - |f^{-1}(1) \cap B \cap C|$$

where f_b, f_c satisfy:

- $f(v) = 0$ if and only if $(f_b(v), f_c(v)) \in \{(0,*),(*,0)\}$
- $f(v) = *$ if and only if $f_b(v) = f_c(v) = *$
- $f(v) = 1$ if and only if $v \in B \cap C$ and $f_b(v) = f_c(v) = 1$, or $v \in B \setminus C$ and $(f_b(v), f_c(v)) = (1,*)$, or $v \in C \setminus B$ and $(f_b(v), f_c(v)) = (*,1)$.

This means that we can apply subset convolution to compute a table $T^1_{sc}[x,f]$ on $3^{|L|}2^{|X\setminus L|}$ entries based on $T^3_b[x,f]$ and $T^3_c[x,f]$. Note that $(B\cap L)\cup(C\cap L) = L$. For this step we follow the description in [34, Section 11.1.2]. Fix a set $D \subseteq L$ to be the dominating vertices. Let F_D denote the set of $2^{|X\setminus D|}$ functions $f : X \to \{1,0,*\}$ such that $f^{-1}(1) = D$, i.e. with vertices in $X \setminus D$ mapping in all possible ways to $\{0,*\}$. For each $D \subseteq L$ we will by subset convolution compute the values of $T^1_{sc}[x,f]$ for all $f \in F_D$.

We represent every $f \in F_D$ by the set $S = f^{-1}(0)$ and define $b_S : X \to \{1,0,*\}$ such that $b_S(x) = 1$ if $x \in D \cap B$, $b_S(x) = 0$ if $x \in S$, $b_S(x) = *$ otherwise. Similarly, define $c_S : X \to \{1,0,*\}$ such that $c_S(x) = 1$ if $x \in D \cap C$, $c_S(x) = 0$ if $x \in S$,

$c_S(x) = *$ otherwise. Then, as explained previously, for every $f \in F_D$ we want to compute

$$T^1_{sc}[x,f] = \min_{Q \cup R = f^{-1}(0) \text{ and } Q \cap R = \emptyset} (T^3_b[x,b_Q] + T^3_c[x,c_R]) - |f^{-1}(1) \cap B \cap C|.$$

Define functions $T_b : 2^{X \setminus D} \to \mathbb{N}$ such that for every $S \subseteq X \setminus D$ we have $T_b(S) = T^3_b[x,b_S]$. Likewise, define functions $T_c : 2^{X \setminus D} \to \mathbb{N}$ such that for every $S \subseteq X \setminus D$ we have $T_c(S) = T^3_c[x,c_S]$. Also, define $a_S : X \to \{1,0,*\}$ such that $a_S(x) = 1$ if $x \in D$, $a_S(x) = 0$ if $x \in S$, $a_S(x) = *$ otherwise. We then compute for every $S \subseteq X \setminus D$,

$$T^1_{sc}[x,a_S] := (T_b * T_c)(S) - |f^{-1}(1) \cap B \cap C|$$

where the subset convolution is over the mini-sum semiring.

(4). In step (4), by Fast Subset Convolution, Theorem 3.11, we compute $T^1_{sc}[x,a_S]$, for all a_S defined by all $f \in F_D$, in $\mathcal{O}^*(2^{|X \setminus D|})$ time each. For all such subsets $D \subseteq L$ we get the time

$$\sum_{D \subseteq L} 2^{|X \setminus D|} = \sum_{D \subseteq L} 2^{|X \setminus L|} 2^{|L \setminus D|} = 2^{|X \setminus L|} \sum_{D \subseteq L} 2^{|L \setminus D|} = 2^{|X \setminus L|} 3^{|L|}.$$

(5). In step (5) we need to run Extend-Table on $T^1_{sc}[x,\cdot]$ to get the table $T^2_{sc}[x,\cdot]$. This since the subset convolution was computed for each fixed set of dominators so the monotonicity property of the table may not hold. Note that the value of $T^2_{sc}[x,f]$ will be the minimum $|S|$ over all $S \subseteq V(G_b) \cup V(G_c)$ such that there exists $f' \in c(S)$ with $f'|_X = f$ and $f'|_{(V(G_b) \cup V(G_c)) \setminus X}$ having everywhere the value 0.

(6). In step (6) we will for each $f : A \to \{1,0,*\}$ compute $f' : B \cup A \cup C \to \{1,0,*\}$ by

$$f'(v) = \begin{cases} 1 & \text{if } v \in A \cap L \text{ and } f(v) = 1 \\ 0 & \text{if } v \in A \text{ and } f(v) = 0 \text{ and } N(v) \cap f^{-1}(1) = \emptyset \\ 0 & \text{if } v \notin A \text{ and } N(v) \cap f^{-1}(1) = \emptyset \\ * & \text{otherwise} \end{cases}$$

and set $T[a,f] := T^2_{sc}[x,f'] + |f^{-1}(1) \cap (A \setminus L)|$.

Note that when we iterate over all choices of $f : A \to \{1,0,*\}$, the vertices colored 0 (in addition to all vertices of $X \setminus A$) must either be dominated by the vertices in $f^{-1}(1)$ or by vertices in $X \setminus V_a$. As we know precisely what vertices of $f^{-1}(0)$ are dominated by $f^{-1}(1)$, we know the rest must be dominated from vertices of $X \setminus V_a$, and therefore we look in $T_{sc}[x,f']$ at an index f' which colors the rest of $f^{-1}(0)$ by 0. We can also observe that it is not important for us whether or not $f^{-1}(0)$ contains all neighbours of $f^{-1}(1)$, since we are iterating over all choices of f - also those where $f^{-1}(0)$ contains all neighbours of $f^{-1}(1)$.

The total runtime becomes $\mathcal{O}^*(\max\{3^{|A|}, 3^{|B|}, 3^{|C|}, 3^{|L|}2^{|(A \cup B \cup C) \setminus L|}\})$, with $L = (A \cap B) \cup (A \cap C) \cup (B \cap C)$ and with constraints $|A|, |B|, |C| \le k$. This runtime is maximum when $L = \emptyset$, giving a runtime of $\mathcal{O}^*(2^{3k})$. We thus have the following theorem.

Theorem 3.12. *Given a graph G and branch decomposition over its vertex set of mm-width k we can solve* Dominating Set *in time $\mathcal{O}^*(8^k)$.*

3.5 Concluding remarks

We have shown that the graph parameter mm-width can be 3-approximated in $\mathcal{O}^*(8^{\text{mm-w}(G)})$ time, from showing that the corresponding cut-function mm is submodular. Also we have shown that using mm-width will for some graphs be better than treewidth for solving DOMINATING SET. The improvement holds whenever $\text{tw}(G) > 1.549 \times \text{mm-w}(G)$, if given only the graph as input. In Figure 3.4 we list some examples of small graphs having treewidth at least twice as big as mm-width. It could be interesting to explore the relation between treewidth and mm-width for various well-known classes of graphs. The given algorithmic technique, combining 'join','introduce' and 'forget' into one operation using fast subset convolution, should extend to any graph problem expressible as a maximization or minimization over (σ, ρ)-sets, using the techniques introduced for treewidth in [105].

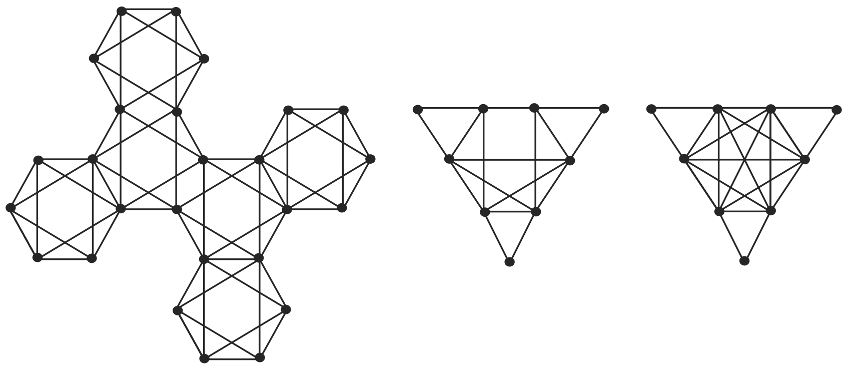


Figure 3.4: Three graphs of mm-width 2. Left and middle have treewidth 4, and right has treewidth 5.

We gave an alternative definition of mm-width using subtrees of a tree, similar to alternative definitions of treewidth and branchwidth. We saw that in the subtrees of a tree representation treewidth focuses on nodes, branchwidth focuses on edges, and mm-width combines them both. There is also a fourth way of defining a parameter through these intersections of subtrees representation; where subtrees T_u and T_v must share an edge if $uv \in E(G)$ (similar to branchwidth) and the width is defined by the maximum number of subtrees sharing a single vertex (similar to treewidth). This parameter will be an upper bound on all the other three parameters, but might it be that the structure this parameter highlights can be used to improve the runtime of DOMINATING SET beyond $\mathcal{O}^*(3^{\text{tw}(G)})$ for even more cases than those shown using mm-width and branchwidth?

Chapter 4

Split-matching-width: its modelling power and an approximation algorithm

In this chapter we will, as described in Section 2.3.2, consider the parameter split-matching-width. We first discuss its modelling power, showing that it lies between treewidth and clique-width. Next, we look at how we can find a branch decomposition of low split-matching-width in FPT time parameterized by optimal split-matching-width. There is a balance between how good of a branch decomposition we compute and the runtime needed to find it, so we do not focus on getting a decomposition of optimal sm-width. However, we show that in runtime linearly single exponential in sm-width we can find a decomposition of sm-width only a constant factor larger than the optimal. At the core of this approximation lies the following general approximation algorithm of Oum and Seymour [79] mentioned in Chapter 3.

Theorem 3.1 ([79]). *For symmetric submodular cut-function f and graph G of optimal f-width k, a branch decomposition of f-width at most $3k+1$ can be found in $\mathcal{O}^*(2^{3k})$ time.*

Split-matching-width is not submodular (see Figure 4.2) and can not be applied to this theorem directly, so the majority of this chapter is devoted to how we can massage our input graph and cut function so that we can apply this theorem. In the end we get a branch decomposition of sm-width at most $18\,\text{sm-w}(G)+1$ at a runtime of $\mathcal{O}^*(2^{18\,\text{sm-w}(G)})$.

4.1 The modelling power of split-matching-width

In this section, we discuss the modelling power of split-matching-width, and give examples of graph classes of bounded split-matching-width.

First let us recall the cut function used in definition of split-matching-width:

$$\text{sm}(A) = \begin{cases} 1 & \text{if } (A, \overline{A}) \text{ is a split} \\ \text{mm}(A) & \text{otherwise} \end{cases}$$

We see directly from this definition that cliques have split-matching-width one, as each cut $(A,\overline{A})$ of a clique either is a split or $\min\{|A|,|\overline{A}|\} \leq 1$, which means $\mathrm{sm}(A) \leq 1$. In fact, graphs of sm-width at most one are precisely the graphs of rank-width at most one (equivalently: of boolean-width at most one), and thus $\text{sm-w}(G) \leq 1$ if and only if G is a distance hereditary graph [75].

Observation 4.1. *A graph G has split-matching-width* $\text{sm-w}(G) \leq 1$ *if and only if it is a distance hereditary graph.*

To show that the modelling power of sm-width is weaker than clique-width, and stronger than treewidth, we have the following proposition, where the parameters treewidth and clique-width are substituted by the parameters mm-width and rank-width, respectively, of equal modelling power.

Proposition 4.2. *For any non-trivial graph G, we have* $\mathrm{rw}(G) \leq \text{sm-w}(G) \leq \text{mm-w}(G)$.

Proof. As G is a non-trivial graph, we have $\text{mm-w}(G), \mathrm{rw}(G), \text{sm-w}(G) \geq 1$. We notice that for cuts $(A,\overline{A})$ without crossing edges, we have $\mathrm{mm}(A) = \mathrm{cutrk}(A) = 0$ and $\mathrm{sm}(A) \leq 1$. So, if we manage to prove that for any $(A,\overline{A})$ with at least one crossing edge, we have $\mathrm{cutrk}(A) \leq \mathrm{sm}(A) \leq \mathrm{mm}(A)$, this will imply that for all branch decompositions $\mathcal{D}$ over $V(G)$ we have $\mathrm{rw}(\mathcal{D}) \leq \text{sm-w}(\mathcal{D}) \leq \text{sm-w}(\mathcal{D})$, which in turn means that $\mathrm{rw}(G) \leq \text{sm-w}(G) \leq \text{mm-w}(G)$.

When $(A,\overline{A})$ contain at least one edge, we have $1 \leq \mathrm{mm}(A)$, and thus, clearly, $\mathrm{sm}(A) \leq \mathrm{mm}(A)$, as $\mathrm{sm}(A) \in \{\mathrm{mm}(A), 1\}$. Notice that $\mathrm{sm}(A) = 1$ implies that either $(A,\overline{A})$ is a split, or there is at most one vertex in A or $\overline{A}$. In either case, we have $\mathrm{cutrk}(A) \leq 1$. So, if we can also prove that $\mathrm{sm}(A) \geq \mathrm{cutrk}(A)$ when $\mathrm{sm}(A) > 1$, we are done with the proof. We will do this by showing that $\mathrm{mm}(A) \geq \mathrm{cutrk}(A)$.

Indeed, $\mathrm{cutrk}(A)$ is defined as being the rank over $GF[2]$ for the adjacency matrix M of $G[A,\overline{A}]$. We have from Theorem 3.4 that there must be a vertex cover C of size $\mathrm{mm}(A)$ in $G[A,\overline{A}]$. This means that each cell in M that contains a non-zero value, must be in a row or a column representing a vertex of C. In particular, this means that if we remove the $\mathrm{mm}(A)$ rows and columns in M representing C, then there are only zero values left in the matrix. As the rank of a matrix does not decrease by more than one from removing a row or column, and the rank of the all-zero-matrix is zero, we have that the rank of M is at most $|C| = \mathrm{mm}(A)$. Thus, we $\mathrm{cutrk}(A) \leq \mathrm{mm}(A)$, and can conclude that $\mathrm{cutrk}(A) \leq \mathrm{sm}(A)$ also when $\mathrm{sm}(A) > 1$. □

We now compare the modelling power of split-matching-width to some of the other parameters mentioned in Section 2.3.2.

Proposition 4.3. *If twin-cover is bounded then sm-width is bounded* $(\text{sm-w}(G) \leq tc(G))$.

Proof. Twin-cover $tc(G)$ is a graph parameter introduced by Ganian [46] as a generalization of vertex cover that is bounded also for some dense classes of graphs. Using the definition of $tc(G)$ from [46] it follows that G has a set $S \subseteq V(G)$ of at most $tc(G)$ vertices such that every component C of $G \setminus S$ induces a clique and every vertex in C has the same neighborhood in S. Let $C_1, ..., C_q$ be the components of $G \setminus S$. Take any branch decomposition of $V(G)$ having for each component C_i a subtree T_i such that the leaves of T_i are mapped to the vertices of C_i and also having a subtree T_S whose leaves are mapped to S. The cuts of G induced by an edge of this branch decomposition are of three types depending on where the edge is: if it is inside the tree of S the cut has a maximum matching of size at most $|S| \leq tc(G)$; if it is inside a tree T_i the cut is a split; otherwise the cut has a maximum matching of size at most $|S| \leq tc(G)$. □

Proposition 4.4. *The parameters modular-width and modular-treewidth are incomparable to split-matching-width.*

Proof. Let G_i be the graph consisting of a clique K_i on i vertices, and a pendant $p(v)$ attached to vertex $v \in V(K_i)$ for each v in K_i. That is, G_i is a clique K_i plus a matching between the vertices of K_i and an independent set of size i. This graph is distance hereditary, and hence has split-matching-width one, while it only has one prime graph in the modular decomposition; namely itself, G_i. Thus, as the treewidth of K_i is $i-1$, we have that both modular-treewidth and modular-width of G_i is $\Theta(i)$. And so the class of graphs $\{G_i\}_{i \geq 2}$ is in the modelling power of split-matching-width, but not the two modular-* parameters.

A class of graphs that does have bounded modular-treewidth and modular-width, but not bounded split-matching-width, is the class of graphs $\{G'_i\}_{i \geq 2}$ where G'_i is constructed from taking the cycle on five vertices, and then replace each vertex v of the cycle by a clique of i vertices, with the same neighbourhood as v. The modular decomposition only has prime graphs of constant size, while the split-matching-width is unbounded, since any cut containing more than i vertices on both sides of the cut is not a split (and there must exist at least one such cut in any branch decomposition over $V(G'_i)$. □

There are several classes of graphs of bounded sm-width where no previous results implied FPT algorithms for the considered problems. We now show a class of such graphs, constructed by combining a graph of clique-width at most 3, with a graph of treewidth k and thus clique-width at most $3 * 2^{k-1}$ [27], as follows. Let G_1 be a distance-hereditary graph and let G_2 be a graph of treewidth k. Let $X \subseteq V(G_1)$ with $|X| \leq k+1$ and $(X, \overline{X})$ a split of G_1, and let $Y \subseteq V(G_2)$ be a bag of a tree decomposition of G_2 of treewidth k. Add an arbitrary set of edges on the vertex set $X \cup Y$. The resulting graph will have sm-width at most $k+1$, a result that basically follows by taking branch decompositions over $V(G_1)$ and $V(G_2)$ where X and Y each are mapped as the set of leaves of a subtree, subdividing each of the two edges above these subtrees and adding an edge on the subdivided vertices to make a single branch decomposition over the vertices of the combined graph. Adding more distance hereditary graphs to other bags of the tree-decomposition in the similar way, we can construct a large class of graphs of bounded split-matching-width containing graphs such as that of Figure 4.1, which has clique-width lower bounded by the underlying grid.

Note that we can also construct new tractable classes of graphs by combining several graphs in a tree structure.

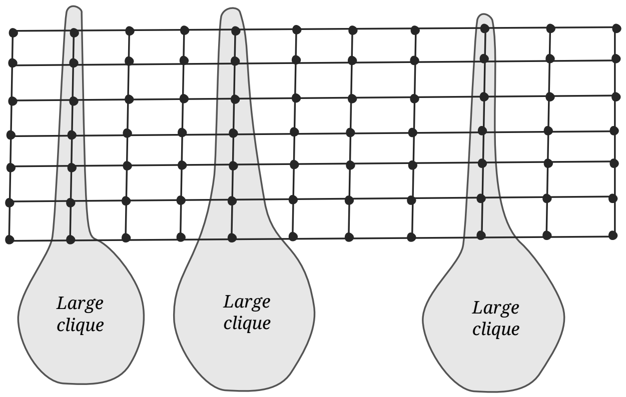


Figure 4.1: A graph of unbounded treewidth but split-matching-width comparable to the clique-width. Selected rows of a grid are connected together with distinct complete graphs to form a clique.

We now use the rest of the chapter to show how to approximate a branch decomposition over the vertex set in FPT time.

4.2 Our strategy and goal for the approximation

In [93], Sæther and Telle gave an algorithm for constructing a branch decomposition of split-matching-width $\mathcal{O}(\text{sm-w}(G)^2)$ in $2^{\mathcal{O}(\text{sm-w}(G))}n^{\mathcal{O}(1)}$ time. In the rest of this chapter, we will improve this algorithm by constructing a branch decomposition of split-matching-width $\mathcal{O}(\text{sm-w}(G))$ in the same runtime, $2^{\mathcal{O}(\text{sm-w}(G))}n^{\mathcal{O}(1)}$. We advice the reader to recall the non-standard definitions for tot(), act(), partitioning functions, and lifted functions found in Section 1.1.

The previous approximation algorithm for split-matching-width, found in [93], consists of the following four main steps;

1. construct a split decomposition $\mathcal{D}$ of G,

2. for each prime graph G_i in $\mathcal{D}$ compute a branch decomposition (T_i, δ_i) over $V(G_i)$ of split-matching-width at most $\mathcal{O}(k)$, where $k = \text{sm-w}(G)$,

3. adjust each branch decomposition (T_i, δ_i) slightly so that for the cut function $g : 2^{V(G_i)} \to \mathbb{R}$ defined as $g(S) = f(\text{tot}_{G_i}(S))$ for all $S \subseteq V(G')$, the g-width becomes $\mathcal{O}(k^2)$, and then finally

4. combine all the branch decompositions together to form a branch decomposition over $V(G)$ of sm-width $\mathcal{O}(k^2)$.

We will keep the general structure as in [93], but we will replace steps (2) and (3) by a single step where we for each prime graph G_i directly compute a branch decomposition over $V(G_i)$ of g-width bounded by $18k+1$. We manage this by the explicit use of lifted functions, which is something that was not done in [93]. If we recall the definition of a lifted function from Chapter 1.1, we see that the cut-function g for a prime graph G_i, as defined above, is precisely the tot_{G_i}-lifted cut function $\mathrm{sm}^{\mathrm{tot}_{G_i}}$ of sm, since tot_{G_i} partitions $V(G)$. So, our strategy for computing a branch decomposition of split-matching-width linear in $\mathrm{sm}(G)$ is as follows:

1. Construct a split decomposition $\mathcal{D}$ of G.
2. Compute a branch decomposition (T_i, G_i) over $V(G_i)$ of $\mathrm{sm}^{\mathrm{tot}_{G_i}}$-width at most $18k+1$ for each prime graph G_i in $\mathcal{D}$.
3. Combine all the branch decompositions together to form a branch decomposition for G of sm-width at most $18k+1$.

For step 1 there exists a well-known polynomial-time algorithm by Cunningham [33] and even linear-time ones, see e.g. [26] and see also [87] for the use of split decompositions in general. For step 2 we are dealing with a prime graph G_i, which by definition has no non-trivial splits. This will allow us to control the lifted split-matching-width of G_i. The main attention of this chapter goes towards solving step 2. The third step we do in the proof of Theorem 4.13 by joining together the branch decompositions from step 2 so that each cut of the final decomposition can be found as one of the lifted cuts induced by a branch decomposition found in step 2.

We now turn our focus to step 2. Namely finding branch decompositions for the prime graphs in a split decomposition having tot-lifted split-matching-width that approximates the split-matching-width of G.

4.3 tot-lifted cut functions in prime graphs

When looking at the tot-lifted width of a cut in a prime graph, the set of vertices $\mathrm{tot}(x)$ a marker x represents matters a lot in the computation of the lifted value. In particular, for the cut functions mm and sm, the size of $\mathrm{act}(x) \subseteq \mathrm{tot}(x)$ is very important. Recall that $\mathrm{act}(x)$ is the set of vertices from $\mathrm{tot}(x)$ that are adjacent to vertices in $V(G) \setminus \mathrm{tot}(x)$. The larger the size of $\mathrm{act}(x)$, the more impact the vertex x might have on the cut-value. For convenience, we will therefore in this chapter mean the size of $\mathrm{act}(x)$ when we say the *weight* of x. That is, if we claim the weight of a vertex x is 8 in a prime graph G', we mean that $|\mathrm{act}(x : G')| = 8$. As a side note, we may observe that using this notion of weights in a prime graph, we get the following lemma.

Lemma 4.5. *For a graph G and a prime graph G' in a split decomposition of G, the $\mathrm{tot}_{G'}$-lifted mm-value of any cut $(X, V(G') \setminus X)$ in G' is precisely the weight of a minimum weight vertex cover of $G'[X, V(G') \setminus X]$.*

Proof. If two vertices u and v of the same side of a bipartite graph are twins, then all minimal vertex covers will contain both or none of the two vertices: If one of u and v are not in the vertex cover, then $N(u) = N(v)$ must be in the vertex cover, and hence if the vertex cover is minimal, neither of u or v will be in the vertex cover. As all of $\mathrm{act}_{G'}(x)$ are twins in $G[\mathrm{tot}(X), \mathrm{tot}(V(G') \setminus X)]$, either all or none of $\mathrm{act}_{G'}(x)$ will be included in any minimum vertex cover of $G[\mathrm{tot}(X), \mathrm{tot}(V(G') \setminus X)]$. Thus, the size of a minimum vertex cover of G will be

$$\min_{X \subseteq V(G')} \left\{ |\mathrm{act}(X)| : \forall uv \in E(G[\mathrm{tot}(X), \overline{\mathrm{tot}(X)}]), u \in \mathrm{act}(X) \text{ or } v \in \mathrm{act}(X) \right\},$$

which is precisely the size of a minimum weight vertex cover of $G'[X, \overline{X}]$. □

We are aiming for a way to approximate the tot-lifted split-matching-width. Neither split-matching-width or tot-lifted split-matching-width is submodular, as can be seen by Figure 4.2. However, the closely related tot-lifted mm function is

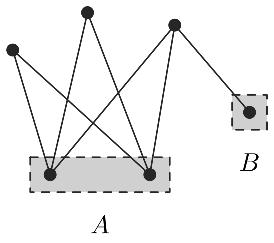


Figure 4.2: As we can see from this figure, the split-matching-value is not submodular: $f(A) = 1$, $f(B) = 1$, $f(A \cap B) = 0$, and $f(A \cup B) = 3$, breaking the submodular inequality $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$.

submodular, as can be deduced from Theorem 3.2 stating that mm is submodular, in combination with the following theorem.

Theorem 4.6. *For a set X and Y and function $\sigma : X \to 2^Y$ that partitions Y, for any function $f : 2^Y \to \mathbb{R}$, if f is submodular, then the σ-lifted function f^σ of f is also submodular.*

Proof. Let $\sigma' : 2^X \to 2^Y$ be the function defined as $\sigma'(S \subseteq X) = \bigcup_{s' \in S} \sigma(s')$. From the definition of f^σ, the value of $f^\sigma(S \subseteq X)$ is $f(\sigma'(S))$. As $\sigma'(A \cup B) = \sigma'(A) \cup \sigma'(B)$ and $\sigma'(A \cap B) = \sigma'(A) \cap \sigma'(B)$ for $A, B \subseteq X$, we see from the below following inequality that submodularity of f implies submodularity also of f^σ:

$$\begin{aligned} f^\sigma(A) + f^\sigma(B) &= f(\sigma'(A)) + f(\sigma'(B)) \\ &\geq f(\sigma'(A) \cup \sigma'(B)) + f(\sigma'(A) \cap \sigma'(B)) \\ &= f(\sigma'(A \cup B)) + f(\sigma'(A \cap B)) \\ &= f^\sigma(A \cup B) + f^\sigma(A \cap B). \end{aligned}$$

□

In combination with Theorem 3.1, the submodularity of all lifted versions of the mm function gives us the following result.

Corollary 4.7. *Given a graph G and a function $\sigma : X \to 2^{V(G)}$ that partitions $V(G)$, we can in $n^{\mathcal{O}(1)} 2^{3k}$ time compute a branch decomposition over X of σ-lifted* mm-*width at most $3k+1$, where k is the optimal σ-lifted* mm-*width of G.*

As previously mentioned, when looking at a prime graph G', the maximum matching-width and the split-matching-width are closely related. In the following series of results, ending in Corollary 4.11, we are able to find an intermediate subgraph of G' where tot-lifted mm-width and tot-lifted sm-width are equal. We use this intermediate subgraph as a way to apply the above lifted mm-approximation of prime graphs to find a good $\mathrm{sm}^{\mathrm{tot}}$-approximation of G' thereby completing step 2. Make an extra note of the first of these lemmas; Lemma 4.8, as this is a general fact for branch decompositions and will be used also in Chapter 6.

Lemma 4.8. *For any two (not necessarily disjoint) vertex subsets A and B of $V(G)$ so that $|A| \geq 2$, and in any branch decomposition (T, δ) over $V(G)$, there must exist a cut (X, Y) in (T, δ) so that $|X \cap A| \geq \left\lceil \frac{|A|}{3} \right\rceil$ and $|Y \cap B| \geq \left\lceil \frac{|B|}{2} \right\rceil$.*

Proof. For a single $S \subseteq V(G)$ it is well known that since T is a ternary tree with leaf set $V(G)$ there exists a cut (X_S, Y_S) in (T, δ) associated with an edge $(x_S, y_S) \in E(T)$ so that $|X_S \cap S| \geq \lfloor \frac{|S|}{3} \rfloor$ and $|Y_S \cap S| \geq \lfloor \frac{|S|}{3} \rfloor$. Furthermore, when $|S| \geq 2$, there exists a cut (X'_S, Y'_S) in (T, δ) associated with an edge $(x'_S, y'_S) \in E(T)$ so that $|X'_S \cap S| \geq \lceil \frac{|S|}{3} \rceil$ and $|Y'_S \cap S| \geq \lceil \frac{|S|}{3} \rceil$.

Consider such a cut (X_A, Y_A) in (T, δ) where at least one third of A is on either side of the cut. As the vertices of B also must be distributed over this cut, either X_A or Y_A must contain at least half of B. Without loss of generality, $X_A \cap B \geq |B|/2$, and thus $X_A \cap B$ contains at least $\lceil |B|/2 \rceil$ of B and Y_A contains at least $\lceil |A|/3 \rceil$ of A. □

Lemma 4.9. *Let G be a graph and $\mathcal{D}$ a split decomposition of G. For any prime graph G' in $\mathcal{D}$ there exists a branch decomposition (T, δ) over $V(G')$ of $\mathrm{sm}^{\mathrm{tot}}$-width at most $3\,\mathrm{sm\text{-}w}(G)$.*

Proof. We will give a proof by construction. Let (T, δ) be a branch decomposition over $V(G)$ of optimal split-matching-width. We will transform (T, δ) into a branch decomposition (T', δ') over $V(G')$. For each vertex $v \in V(G')$ of weight one, we set $\sigma'(\mathrm{tot}_v) = v$, where tot_v is the leaf of T so that $\sigma(\mathrm{tot}_v) = \mathrm{act}(v)$. For the vertices $v \in V(G')$ of weight two or more, by Lemma 4.8 applied to $A = B = \mathrm{act}(v)$, there will always be an edge e_v in T where at least one third of $\mathrm{act}(v)$ is on each side of the cut induced by e_v. For each vertex $v \in V(G')$ of weight at least two, we append a vertex mapping to v to such an edge e_v as depicted in Figure 4.3. We then repeatedly remove all leaves not mapping to a vertex in $V(G')$, so that we are left with a branch decomposition (T', δ') of $V(G')$ (see Figure 4.3). We now claim that this branch decomposition has tot-lifted sm-width no more than $3\,\mathrm{sm\text{-}w}(G)$.

For each trivial cut $(\{v\}, V(G') \setminus \{v\})$ of (T', δ') the lifted sm-value is 1, as it is either a split or trivial-split. For each non-trivial cut (A', B') of (T', δ'), there is an associated non-trivial cut (A, B) in (T, δ) where for each $v \in A'/B'$ at least one third of $\mathrm{act}(v)$ is in A/B. Let C be a minimum vertex cover of $G[A, B]$. For a minimum vertex cover, two twins are either both in the cover, or both not in the cover, so for each vertex $v \in A/B$ either one third of $\mathrm{act}(v)$ is in C or none of $\mathrm{act}(v) \cap A/B$ is in C. Let C' be the vertices $v \in V(G')$ for which at least one third of $\mathrm{act}(v)$ is in C'. The size of $\mathrm{act}(C')$ is at most three times C and we will now show that $\mathrm{act}(C')$ in fact is a vertex cover of $G[\mathrm{tot}(A), \mathrm{tot}(B)]$, proving that the lifted sm-width of G' is at most $3\,\mathrm{sm}(G)$.

There is an edge between a and b in $G[A', B']$ if and only if there is an edge between $\mathrm{tot}(a)$ and $\mathrm{tot}(b)$ in $G[\mathrm{tot}(A), \mathrm{tot}(B)]$. So, if for all edges uv in $G[A', B']$, either u or v is in C', we are done. Assume $uv \in E(G[A', B']$ where $u \in A'$ and $v \in B'$. This means one third of $\mathrm{act}(u)$ and one third of $\mathrm{act}(v)$ is on the opposite side of each other in (A, B). However, as $\mathrm{act}(u)$ are twins and adjacent to all of $\mathrm{act}(v)$, this means one third of either $\mathrm{act}(u)$ or $\mathrm{act}(v)$ must be in C, and hence u or v must be in C'. □

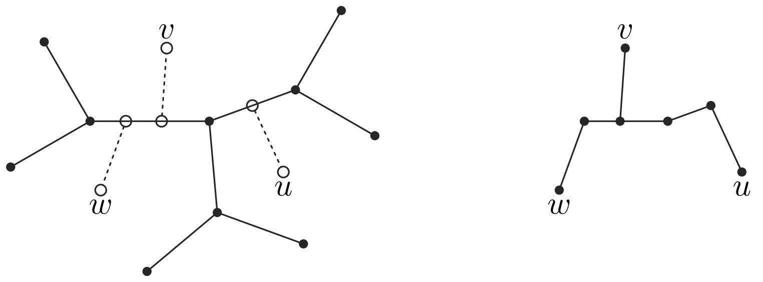


Figure 4.3: As described in the proof of Lemma 4.9. u,v, and w are vertices of the prime graph and the dotted lines are the pendants added to T (as described in the proof). To the right is the resulting decomposition (T', δ').

Lemma 4.10. *Let G be a graph of split-matching-width k and G' a non-trivial prime graph in a split decomposition of G. For any vertex v in G' of weight more than $3k$, v is either adjacent to exactly one other vertex of weight more than $3k$ or the* mm*-value of* $\mathrm{tot}(v)$ *is at most* $6k$.

Proof. Assume for contradiction that v has two neighbours a and b of weight more than $3k$ in G', and let (T', δ') be a branch decomposition of G' of lifted sm-width at most $3k$, as by Lemma 4.9. Since G' is a non-trivial prime graph, there must be a fourth vertex z in $V(G')$. By Lemma 4.8 applied on $A = B = \{v, a, b, z\}$, there must be a cut (X, Y) in (T', δ') so that at $\lceil |A|/3 \rceil = 2$ of A is in X and $\lceil |B|/2 \rceil = 2$ is in Y. Assume without loss of generality that $v \in X$. Since Y contains two of the three remaining vertices a, b and z, at least one of the neighbours a and b must be in Y, without loss of generality $a \in Y$. Since X and Y both contain at least two vertices, the cut (X, Y) is a non-trivial cut of G' and hence it cannot be a split, since G' is a prime graph. That means that the lifted sm-value of this cut is the same as its lifted mm-value. However, $\mathrm{act}(v) \cap X$ and $\mathrm{act}(y) \cap Y$ alone induce a complete bipartite graph with more than $3k$ vertices on each side of the bipartion, so the lifted mm-value of (X, Y) is larger than $3k$, a contradiction of (T', δ') having lifted sm-value at most $3k$.

We know from Lemma 4.9 that there must exist a branch decomposition of G' of $\mathrm{sm}^{\mathrm{tot}}$-width at most $3k$, and all non trivial cuts have $\mathrm{mm}^{\mathrm{tot}}$-value at most $3k$. This means there must be a tri-partition $(\{v\}, A, B)$ of $V(G')$ so that $\mathrm{sm}^{\mathrm{tot}}(\{v\})$, $\mathrm{sm}^{\mathrm{tot}}(A)$, and $\mathrm{sm}^{\mathrm{tot}}(B)$ are all at most $3k$.

Without loss of generality, we show that the number of vertices in $\text{tot}(A)$ adjacent to $\text{tot}(v)$ is no more than $3k$. Since G' is a prime graph, $(A, \{v\} \cup B)$ cannot be a split and A must either consist of a single vertex, or have $\text{sm}^{\text{tot}}(A) = \text{mm}^{\text{tot}}(A)$. If A consist of a single vertex, by our assumptions that v was not adjacent to any vertex of weight more than $3k$, we have $|N(\text{tot}(v)) \cap \text{tot}(A)| \leq 3k$. If A consists of more than a single vertex, we know $\text{mm}^{\text{tot}}(A) = \text{sm}^{\text{tot}} \leq 3k$. Since any minimum vertex cover of $G[\text{tot}(A), \text{tot}(\{v\} \cup B)]$ must either consist of all of $\text{act}(v)$ or none of it (since $\text{act}(v)$ are twins in this bipartite graph), and $|\text{act}(v)| > 3k \geq \text{mm}^{\text{tot}}(A)$, the minimum vertex cover of $G[\text{tot}(A), \text{tot}(\{v\} \cup B)]$ must consist of all the vertices of A adjacent to $\text{tot}(v)$. Hence, $|N(\text{tot}(v)) \cap \text{tot}(A)| \leq \text{mm}^{\text{tot}}(A) \leq 3k$. The same arguments applies to B, so we can conclude that

$$|N(\text{tot}(v))| \leq |N(\text{tot}(v)) \cap A| + |N(\text{tot}(v)) \cap B| \leq 3k + 3k \ .$$

And so the mm-value of $\text{tot}(v)$ is no more than $6k$ as $N(\text{tot}(v))$ is a vertex cover of $(\text{tot}(v), \overline{\text{tot}(v)})$. □

As we notice from Lemma 4.10, vertices of weight $3k$ are more restricted than the rest of the vertices. We say that a vertex of weight more than $3k$ is *heavy*, and an edge incident with two heavy vertices is called a *heavy* edge.

For a heavy edge uv, if a branch decomposition of lifted-sm-width at most $3k$ induce a non-trivial cut (A, B), it must be the case that u and v are either both in A or both in B. Otherwise, the lifted-sm-width will be too large. This means that in any branch decomposition of lifted-sm-width at most $3k$, $(\{uv\}, \overline{\{uv\}})$ must be a cut induced by the decomposition. Combining this observation with Lemma 4.10, we get the following corollary.

Corollary 4.11. *By contracting each heavy edge uv in a non-trivial prime graph G' of G to single vertices v_{uv}, and defining $g(v_{uv}) = \text{tot}(\{u, v\})$ and $g(x) = \text{tot}(x)$ for all other vertices x, we get a graph of mm^g-width at most $6\,\text{sm-w}(G)$.*

Using Corollary 4.11 connecting the lifted mm-width to the sm-width of G, and Corollary 4.7 giving a 3-approximation of the lifted mm-width, we are able to solve step 2, as is shown by the following theorem.

Theorem 4.12. *Given a graph G of split-matching-width at most k and a prime graph G' in a split decomposition of G, in $\mathcal{O}^*(2^{18k})$ time we are able to construct a branch decomposition (T', δ') over $V(G')$ of $\text{tot}_{G'}$-lifted split-matching width at most $18k + 1$.*

Proof. If G' is a trivial prime graph, any branch decomposition will have lifted split-matching width at most 1, and the theorem trivially holds. If G' is a non-trivial prime graph, then by first contracting each heavy edge as described in Corollary 4.11, we have a graph G^* of g-lifted-mm-width less than $6k$, where $g(v) = \text{tot}_{G'}(v)$ for all vertices $v \in V(G')$, and where $g(v_{uv}) = \text{tot}_{G'}(u) \cup \text{tot}_{G'}(v)$ for all new vertices v_{uv} resulting in contracting a heavy edge uv. As g partitions $V(G)$, we have by Corollary 4.7 that we can construct a branch decomposition (T^*, δ^*) over $V(G^*)$ of mm^g-width no more than $18k + 1$. Let X^* be any subset of vertices from $V(G^*)$, and X' the corresponding set of vertices in $V(G')$ where every contracted vertex $v_{uv} \in X^*$ is replaced by the two endpoints u and v of the heavy contracted edge uv. From the definition of g we have $\text{tot}_{G'}(X') = g(X^*)$. Now, let (T',δ') be the branch decomposition over $V(G')$ we get by for each leaf ℓ in T' mapping to a contracted edge uv of G', appending two vertices to ℓ that map to u and v, respectively. Now every non trivial cut $(X',\overline{X'})$ in $V(G')$ induced by (T',δ') has a corresponding cut $(X^*,\overline{X^*})$ in $V(G^*)$ so that $(g(X^*),g(\overline{X^*}))$ equals $(\text{tot}_{G'}(X'),\text{tot}_{G'}(\overline{X'}))$ in G. The tot-lifted mm-value and the g-lifted mm-value, respectively, of these cuts must thus be the same. The trivial cuts induced by (T',δ') are all splits, so we can conclude that the $\text{tot}_{G'}$-lifted sm-width of (T',δ') is the same as the g-lifted mm-width of (T^*,δ^*), which is at most $18k+1$. □

Lemma 4.12 completes the part of finding branch decompositions of the prime graphs with tot-lifted sm-width only a linear factor larger than the sm-width of the original graph. Putting Lemma 4.12 together with the fact that we can find a split decomposition in polynomial time by [33] and a procedure for how to combine these branch decompositions together to form a branch decomposition for G, we get the following theorem, as promised.

Theorem 4.13. *Given a graph G of split-matching-width k, we can in $\mathcal{O}^*(2^{18k})$ time construct a branch decomposition (T,δ) of G of split-matching-width at most $18k+1$.*

Proof. First we construct a split decomposition $\mathcal{D}$ of G in polynomial time, as shown by Cunningham [33]. Then, for each prime prime graph G_i in $\mathcal{D}$, we construct a branch decomposition (T_i,δ_i) over $V(G_i)$ of tot_{G_i}-lifted split matching-width no more than $18k+1$, as shown by Lemma 4.12. We then combine the decompositions to a branch decomposition (T,δ) of G, as follows:

For each pair of prime graphs G_i and G_j that share a marker m, we identify the leaves of T_i and T_j that map to m. That is, for the leaf ℓ_i of T_i so that $\delta_i(\ell_i) = m$ and the leaf ℓ_j of T_j so that $\delta_j(\ell_j) = m$, we connect the trees T_i and T_j by combining ℓ_i and ℓ_j into a single vertex ℓ (see Figure 4.4). We let this be our tree T, and let δ be the union of all the mappings δ_i restricted to the remaining leaves. This gives us a branch decomposition (T,δ) over $V(G)$.

This branch decomposition (T,δ) is very similar to how the split decomposition tree $T_{\mathcal{D}}$ of $\mathcal{D}$ is constructed. In T each T_i occurs as a subtree, and these subtrees are placed relative to each other in T just as their respective prime graphs are placed relative to each other in the split decomposition tree $T_{\mathcal{D}}$. As a consequence, if we were to remove the edges of T_i from E, then the edges of T fall into as many components as there are markers in $V(G_i)$, just like when removing G_i from $T_{\mathcal{D}}$. Furthermore, the set of vertices that δ maps to from the leaves of each such component is precisely the set of vertices residing in the prime graphs of the corresponding component of $T_{\mathcal{D}} - G_i$. Thus, for each leaf v and incident pendant e_v in T_i, the corresponding edge e_v in T induces the cut $(\text{tot}_{G_i}(v), V(G) \setminus \text{tot}_{G_i}(v))$. So, for each edge e in T, the same edge e occurs in one of the branch decompositions (T_j,δ_j) of some prime graph G_j, and this edge induces cuts $(X, V(G) \setminus X)$ and $(X', V(G_j) \setminus X')$ in $V(G)$ and $V(G')$, respectively, so that $X = \text{tot}_{G_j}(X')$. And hence $\text{sm}_G(X) = \text{sm}^{\text{tot}_{G_j}}(X) \leq 18k+1$. From this we can conclude that the split-matching-width of the entire branch decomposition (T,δ) over $V(G)$ is no larger than $18k+1$. □

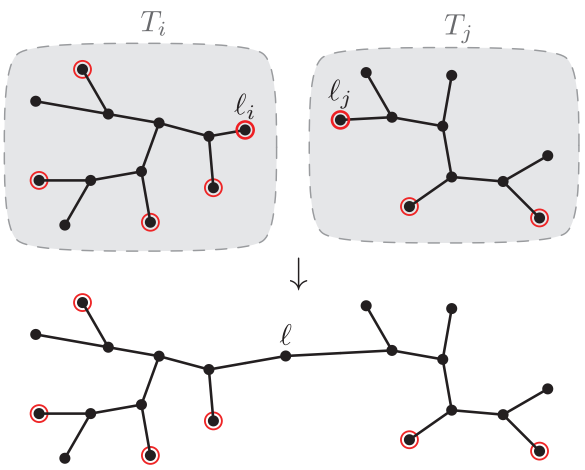


Figure 4.4: For two prime graphs G_i and G_j sharing a marker m, we combine the their respective branch decomposition trees at the leaf mapping to m, as described in the proof of Theorem 4.13. Markers are marked with a red circle.

4.4 Concluding remarks

Using split decompositions and submodularity of the mm cut-function, we managed to construct a branch decomposition $\mathcal{D}$ over the vertex set which has $\text{sm-w}(\mathcal{D}) = \mathcal{O}(\text{sm-w}(G))$. With that, we can conclude that split-matching-width satisfies our criteria on [B] from Section 2.3.2.

We also showed that the modelling power of split-matching-width contains the class of cliques, and is stronger than the modelling power of treewidth, and weaker than the modelling power of clique-width. As could be seen from the examples of graphs of bounded split-matching-width, split-matching-width combines bounded treewidth with distance hereditary properties. As distance hereditary means rank width at most one, this leads us to wonder what about rank width more than one? That is, if we slightly modify the definition of split-matching-width to instead use the following cut function:

$$\text{sm}(A) = \begin{cases} \text{cutrk}(A) & \text{if } \text{cutrk}(A) \leq 1 \\ \text{mm}(A) & \text{otherwise} \end{cases}$$

What happens for the variant of split-matching-width where we change $\text{cutrk}(A) \leq 1$ with some other constant, for instance 4?

$$\text{sm}_4(A) = \begin{cases} \text{cutrk}(A) & \text{if } \text{cutrk}(A) \leq 4 \\ \text{mm}(A) & \text{otherwise} \end{cases}$$

Clearly the sm_4-width has more modelling power than sm-width, but is still weaker than clique-width. However, our approximation algorithm for split-matching works because of how easily we can find split decompositions, and relies heavily on them to do the approximation. We are not aware of any similar decompositions that can be used for cuts of $\text{cutrk}(A) > 1$, and thus fail the criteria for `[B]`.

Chapter 5

Dynamic programming parameterized by split-matching-width

In this chapter, we will, as described in Section 2.3.2, construct FPT algorithms for MaxCut, Chromatic Number, Hamiltonian Cycle and Edge Dominating Set parameterized by the split-matching-width, showing that indeed more problems are FPT parameterized by split-matching-width than clique-width (at the cost of its modelling power). Here we only focus on the second part of the structural DP-scheme; solving a problem when assuming a branch decomposition is given together with the input (`[P2]`). We know from Chapter 4 that we can find a constant approximation of split-matching-width in linearly single exponential time (`[P1]`), which will result in FPT algorithms for these problems.

5.1 The framework

We solve MaxCut, Chromatic Number, Hamiltonian Cycle and Edge Dominating Set on a graph G by a bottom-up traversal of a rooted branch decomposition (T,δ) over $V(G)$, in time FPT parameterized by the sm-width of (T,δ). We will assume T is rooted, as this will help guide the algorithm by introducing parent-child relationships. To root T, we first pick any edge of T and subdivide it. We then root the tree in the newly introduced vertex, resulting in a rooted binary tree consisting of the exact same cuts as the original decomposition.

In the bottom-up traversal of the rooted tree we encounter two disjoint subsets of vertices $A,B \subseteq V(G)$, as leaves of two already processed subtrees, and need to process the subtree on leaves $A \cup B$. There are three cuts of G involved: $\left(A,\overline{A}\right),\left(B,\overline{B}\right),\left(A\cup B,\overline{A\cup B}\right)$, and each of them can be of type split, or of type non-split (also called type mm for maximum-matching). This gives six cases that need to be considered, at least if we use the standard framework of table-based dynamic programming. We instead use an algorithmic framework for decision problems where we JOIN sets of certificates while ensuring that the result preserves witnesses for a "yes" instance. Under this framework, the algorithm for MaxCut becomes particularly simple, and only two cases need to be handled in the JOIN, depending on whether the "parent cut" $\left(A\cup B,\overline{A\cup B}\right)$ is a split or not. For the other three problems we must distinguish between the two types of "children cuts" in order to achieve FPT runtime, and the algorithms are more complicated.

Let us describe the algorithmic framework. As usual, e.g. for problems in NP, a verifier is an algorithm that given a problem instance G and a certificate c, will verify if the instance is a "yes" instance, and if so we call c a witness. As an analogy to other dynamic programming routines, the certificates can be looked at as partial solutions, and a witness as a solution to the problem. For our algorithms we will use a commutative and associative function $\oplus(x,y)$, that takes two certificates x,y and creates a set of certificates. This is extended to sets of certificates S_A, S_B by $\oplus(S_A, S_B)$ which creates the set of certificates $\bigcup_{x \in S_A, y \in S_B} \oplus(x,y)$. For a graph decision problem, an input graph G, and any $X \subseteq V(G)$ we define $\text{cert}(X)$ to be a set of certificates on only a restricted part of G, which must be subject to the following constraints:

- If G is a "yes" instance, then $\text{cert}(V(G))$ contains a witness.
- For disjoint $X, Y \subseteq V(G)$ we have $\oplus(\text{cert}(X), \text{cert}(Y)) = \text{cert}(X \cup Y)$.

For FPT runtime we need to restrict the size of a set of certificates, and the following will be useful. For $X \subseteq V(G)$ and certificates $x, y \in \text{cert}(X)$, we say that x *preserves* y if for all $z \in \text{cert}(\overline{X})$ so that $\oplus(y,z)$ contains a witness, the set $\oplus(x,z)$ also contains a witness. We denote this as $x \preceq_X y$. A set S preserves $S' \subseteq \text{cert}(X)$, denoted $S \preceq_X S'$, if for every $x' \in S'$ there exists a $x \in S$ so that $x \preceq_X x'$. A certificate $x \in \text{cert}(X)$ so that there exists a $y \in \text{cert}(\overline{X})$ where $\oplus(x,y)$ contains a witness, is called an *important* certificate.

For a rooted branch decomposition (T, δ) over the vertices of a graph G and vertex $v \in V(T)$, we denote by V_v the set of vertices of $V(G)$ mapped by δ from the leaves of the subtree in T rooted at v. With these definitions we give a generic recursive (or bottom-up) algorithm called Recursive that takes (T, δ) and a vertex w of T as input and returns a set $S \preceq_{V_w} \text{cert}(V_w)$, as follows:

- at a leaf w of T initialize and return the set $\text{cert}(\{\delta(w)\})$
- at an inner node w first call Recursive on each of the children nodes a and b and then run procedure Join on the returned input sets S_1, S_2 of certificates, with $S_1 \preceq_{V_A} \text{cert}(V_a)$ and $S_2 \preceq_{V_b} \text{cert}(V_b)$, and return a set $S \preceq_{V_a \cup V_b} \oplus(S_1, S_2)$
- at the root we will have a set of certificates $S \preceq_{V(G)} \text{cert}(V(G))$

Calling Recursive on the root r of T and running a verifier on the output solves any graph decision problem in NP. Correctness of this procedure follows from the definitions. The extra time spent by the verifier is going to be $\mathcal{O}^*(|S|)$, and for an FPT algorithm we will require that all $|S|$ be $\mathcal{O}^*(f(k))$, i.e. FPT in the sm-width k of (T, δ).

In the following sections we show how to solve each of the respective four problems in FPT time. A rough sketch of the idea of how this can be achieved for each of the problems is shown below. A formal definition of each of the problems is given in each of their respective sections.

Maximum Cut. MaxCut is the one out of the four problems which has the most simple algorithm. To compute a maximum cut of a graph (a cut of maximum number of crossing edges), we will give an algorithm to solve t-MaxCut, which asks for a cut of with at least t crossing edges. Running t-MaxCut for increasing values of t, will determine MaxCut. The certificates for this problem are subsets of vertices and a witness is a subset S so that the number of edges with one endpoint in S and one in $V(G) \setminus S$ is at least t (i.e., witnesses are cuts of size at least t). We show that for a cut $(A,\overline{A})$ and subsets S_1 and S_2 in cert(A), if the neighbourhood of $S_1 \subseteq A$ and $S_2 \subseteq A$ in $\overline{A}$ are the same, then one of the sets preserves the other in A ($S_1 \preceq_A S_2$ or $S_2 \preceq_A S_1$). As the number of distinct neighbourhoods over the cut $(A,\overline{A})$ is bounded by 2 and $2^{\text{mm}(A)}$, for split and non-split cuts, respectively, we will be able to give an FPT algorithm for solving MaxCut.

Hamiltonian Cycle. For Hamiltonian Cycle, certificates are disjoint paths or cycles, and a witness is a Hamiltonian cycle. For a cut $(A,\overline{A})$ two certificates C_1, C_2 will preserve each other in cert(A) if for any certificate in cert$(\overline{A})$ the paths of C_1 can connect to the same paths (and in the same way) as the paths of C_2 can, and vice versa. Whether this is the case depends on what vertices in $\overline{A}$ the two endpoints of each of the paths in C_2 and C_1 are adjacent to. Therefore we classify the paths into *path classes*, where two paths are in the same path class if the two neighbourhoods of the one path is equal to the two neighbourhoods of the other path. Two certificates will preserve each other as long as they consist of paths of the same path classes and the cardinality of each path class is the same for both of the certificates.

For splits the number of paths might be linear in n, but the number of path classes is constant, so we have a preserving set of size $\mathcal{O}(n)$. For non-splits we use techniques from algorithms for tree-decompositions ([43]) to produce a preserving set of size $2^{\mathcal{O}(k)}$.

Chromatic Number. For Chromatic Number, we will actually solve t-Coloring, which asks whether the input graph can be colored by at most t colors, and from this conclude that Chromatic Number can be solved in the same time when excluding polynomial factors. We note that a graph of sm-width k, unlike graphs of treewidth k, may need more than $k+1$ colors. We let all partitions into t parts where the parts induce independent sets be our certificates. What matters for a certificate is what kind of certificates it can be combined with to yield a new certificate, i.e. inducing an independent set also across the cut. For non-split cuts, this means the number of important certificates is bounded by the number of ways to t-partition the vertices in the k-vertex cover of the cut, which is a function of k. For a split cut, what is important is the number of parts of a partition/certificate that have neighbors across the cut. The certificate minimizing this number will preserve all other certificates. Based on this the Join operation will be able to find a preserving set of certificates of FPT-size.

Edge Dominating Set. For Edge Dominating Set (or t-Edge Dominating Set which is what we actually solve) the certificates are subgraphs of G and a witness is a graph $G' = (V', E')$ so that each vertex in V' is incident with an edge in E', and E' is an edge dominating set of G of size at most t. The idea of how to make an FPT Join-procedure is that for a vertex cover C of a cut, the number of ways a certificate restricted to C can look is limited by a function of the size of C. Based on this we find a preserving set of FPT cardinality when $|C|$ is at most k. When $|C|$ is not bounded by k, we have a split. For splits we limit the max number of certificates needed for a preserving set by a polynomial of n. This is because almost all edges on one side of the cut affect the rest of the edges uniformly, and the other way around.

5.2 Maximum Cut

5.2.1 The Problem.

The problem t-MaxCut asks, for a graph G, whether there exists a set $W \subseteq V(G)$ so that the number of edges in $G[W, \overline{W}]$ is at least t. For a set X, we denote by $\partial_G(X)$ the number of edges in $G[V(G) \cap X, V(G) \setminus X]$ (note that X does not need to be a subset of $V(G)$).

5.2.2 The certificates and $\oplus$.

For t-MaxCut, we define cert(X) for $X \subseteq V(G)$ to be all the subsets of X, and we define $\oplus(x,y)$ to be the union function; $\oplus(x,y) = \{x \cup y\}$. We solve t-MaxCut by use of Recursive and the below procedure Join$_{maxcut}$ with input specification as described above.

5.2.3 The Join$_{maxcut}$ function.

```
Procedure Join$_{maxcut}$
  Input:  $S_1 \subseteq \text{cert}(A_1)$ and $S_2 \subseteq \text{cert}(A_2)$
          for disjoint sets $A_1$, $A_2$ and $A = A_1 \cup A_2$
  Output: $S \preceq_A \oplus(S_1, S_2)$

 $S' \leftarrow \{s_1 \cup s_2 : s_1 \in S_1, s_2 \in S_2\}$ /* note $S' = \oplus(S_1, S_2)$ */
 $S \leftarrow \emptyset$

 if $\left(A, \overline{A}\right)$ is a split then for $z = 0, \ldots, n$ do
       $s' \leftarrow \text{argmax}_{s \in S'} \{\partial_{G[A]}(s) : \left|N(\overline{A}) \cap s\right| = z\}$
       $S \leftarrow S \cup \{s'\}$
```

```
else for all subsets $S_C \subseteq C$ do
    $s' \leftarrow \mathrm{argmax}_{s \in S'} \left\{\partial_{G[A \cup C]}(s \cup S_C) : S_C \cap A = s \cap A\right\}$
    $S \leftarrow S \cup \{s'\}$
return $S$
```

Lemma 5.1. *Procedure* $\textsc{Join}_{maxcut}$ *is correct and runs in time* $\mathcal{O}^*((|S_1||S_2|)2^k)$ *for* $k = \max\{\mathrm{sm}(A_1), \mathrm{sm}(A_2), \mathrm{sm}(A)\}$*, producing a set* S *of cardinality* $\mathcal{O}(n+2^k)$.

Proof. We $S' = \oplus(S_1, S_2)$, and S' can be calculated in time $\mathcal{O}^*(|S_1||S_2|)$. Finding a minimum vertex cover of a $G[A, \overline{A}]$ can be done in polynomial time, since $G[A, \overline{A}]$ is a bipartite graph. Also, when $\left(A, \overline{A}\right)$ is not a split, then $\mathrm{mm}(A) \leq k$ and $|C| \leq k$. Combined with a polynomial amount of work for each iteration of the for loops, and loops iterating at most $\max\{n, 2^k\}$ times (making the size of S also bounded by $n + 2^k$), the total runtime is $\mathcal{O}^*(|S|2^k)$.

To show that $S \preceq_A S'$ we have to make sure that if there exists a witness x of t-MaxCut so that for $x_A \in S'$ and $x_{\overline{A}} \in \mathrm{cert}(\overline{A})$ we have $\{x\} \subseteq \oplus(x_A, x_{\overline{A}})$, then there must exist a certificate $x' \in S$ so that $x' \preceq_A x_A$. We assume there exists such a witness x with x_A and $x_{\overline{A}}$ defined as above. We have two cases to consider; when $(A, \overline{A})$ is a split, and when it is not.

We first consider the case when $\left(A, \overline{A}\right)$ is a split. Since x is a witness, $\partial_G(x) \geq t$. Let $z = \left|N(\overline{A}) \cap x\right|$. We have $\partial_G(x) = \partial_{G[A]}(x_A) + \partial_{G[\overline{A}]}(x_{\overline{A}}) + \partial_{G[A,\overline{A}]}(x)$, and $\partial_{G[A,\overline{A}]}(x) = \left|N(\overline{A}) \cap x_A\right| \times \left|N(A) \setminus x_{\overline{A}}\right| = z\left|N(A) \setminus x_{\overline{A}}\right|$. Since S contains $s \in S'$ maximizing $\max_{s \in S'}\{\partial_{G[A]}(s) : \left|N(\overline{A} \cap s)\right| = z\}$, we have $\partial_G(\oplus(s, x_{\overline{A}})) = \partial_G(x) + \partial_{G[A]}(s) - \partial_{G[A]}(x_A) \geq \partial_G(x)$ meaning $\oplus(s, x_{\overline{A}})$ is a witness, and so $S \preceq_A S'$.

Now, consider the case when $\left(A, \overline{A}\right)$ is not a split (this means $\mathrm{mm}(A) \leq k$). Let C be the vertex cover used in the procedure and x_C be $x \cap C$. As C disconnects A and $\overline{A}$, for any certificate $x' \in \mathrm{cert}(A)$ where $C \cap x' = x_C \cap A$, including x_A, we have the following:

$$\partial_G(x' \cup x_{\overline{A}}) = \partial_{G[C \cup A]}(x_C \cup x_A) + \partial_{G[C \cup \overline{A}]}(x_C \cup x_{\overline{A}}) - \partial_{G[C]}(x_C) \ .$$

This means that as $\oplus(x_A, x_{\overline{A}})$ is a witness, as long as there is a certificate $x' \in S$ so that $x' \cap C = x_C \cap A$ and $\partial_{G[C \cup A]}(x_C \cup x') \geq \partial_{G[C \cup A]}(x_C \cup x)$, we can be sure that S preserves $\{x\}$. And since we for all choices of $x_C \subseteq C$ add a certificate from S to S that maximizes this value the set S must preserve $S' = \oplus(S_1, S_2)$. $\square$

Theorem 5.2. *Given a graph* G *and branch decomposition* (T, δ) *of sm-width* k*, we can solve* MaxCut *in time* $\mathcal{O}^*(8^k)$.

Proof. In Lemma 5.1 we show $\textsc{Join}_{maxcut}$ is correct and produces a preserving set S of size at most $\mathcal{O}^*(2^k)$ in time $\mathcal{O}^*(|S_1||S_2|2^k)$. So, using Recursive with $\textsc{Join}_{maxcut}$, we know the size of both of the inputs of $\textsc{Join}_{maxcut}$ is at most the size of its output, i.e., $|S_1|, |S_2| \leq \mathcal{O}^*(2^k)$. So, each call to Recursive has runtime at most $\mathcal{O}^*(8^k)$. As there are linearly many calls to Recursive and there is a polynomial time verifier for the certificates Recursive produces, by the definition of $\preceq$, the total runtime for solving t-MaxCut is also bounded by $\mathcal{O}^*(8^k)$. To solve MaxCut, we run the t-MaxCut algorithm $\mathcal{O}(\log(n))$ times while binary searching for the maximum cut value $t \in \{0, n^2\}$, giving the same runtime when excluding polynomials of n. □

A rough analysis of the runtimes of this section, in terms also of n, gives a total running time of $\mathcal{O}(n^4 \log(n) + 8^k n \log(n))$ for solving MaxCut.

5.3 Hamiltonian Cycle

Recently, it was shown that Hamiltonian Cycle parameterized by treewidth can be solved in linearly single exponential time [13, 43], meaning it can be solved in $n^{\mathcal{O}(1)}2^{\mathcal{O}(k)}$-time. In this section, using tools from [43], we show that also parameterized by split-matching-width we can solve Hamiltonian Cycle in linearly single exponential time. To the best of our knowledge, this is the first linearly single exponential algorithm for any "globally constrained" graph problem parameterized by a non-trivial and non-sparse structural parameter. Note that there is another algorithm for deciding Hamiltonian Cycle which does not use these new treewidth-tools, but this only achieves a runtime of $n^{\mathcal{O}(1)}2^{\mathcal{O}(k^2)}$ [93].

5.3.1 The problem.

For a graph G, a subgraph G' of G where G' is a cycle, we say that G' is a *Hamiltonian cycle* of G if $V(G') = V(G)$. The decision problem Hamiltonian Cycle asks, for an input graph G, whether there exists a Hamiltonian cycle of G.

5.3.2 The certificates and $\oplus$

We notice for a set $A \subseteq V(G)$ and Hamiltonian cycle G' of G that $G'[A]$ is either the Hamiltonian cycle itself (if $A = V(G)$) or a set of vertex disjoint paths and isolated vertices. For ease of notation, we will throughout this section regard isolated vertices as paths (of length zero). Based on this observation, for $X \subseteq V(G)$, we let $\text{cert}(X)$ on the problem Hamiltonian Cycle be all subgraphs G' of G so that $V(G') = X$ and G' consists only of disjoint paths or of a cycle of length $|V(G)|$. The witnesses of $\text{cert}(V(G))$ are exactly the certificates that are Hamiltonian cycles of G. Clearly a polynomial time verifier exists, as we can easily confirm, in polynomial time, that a Hamiltonian cycle in fact is a Hamiltonian cycle. Also, as $\text{cert}(V(G))$ contains all Hamiltonian cycles of $V(G)$, it must contain a witness if G is a "yes" instance.

For disjoint sets $A, B \subset V(G)$, certificate $G_x \in \text{cert}(A)$, and certificate $G_y \in \text{cert}(B)$, we define $\oplus(G_x, G_y)$ to be the set of all certificates $G_z = (A \cup B, E(G_x) \cup E(G_y) \cup E')$ where E' is a subset of the edges crossing (A, B). That is, $\oplus(G_x, G_y)$ is the set of all graphs generated by the disjoint union of G_x and G_y and adding edges from G with one endpoint in A and one endpoint in B that are also valid certificates.

5.3.3 The Join$_{HC}$ function.

In Join$_{maxcut}(S_1,S_2)$ we first calculated $S = \oplus(S_1,S_2)$, and later reduced the size of S. However, by our definition of $\oplus$ for Hamiltonian Cycle, even for certificate sets S_1 and S_2 of restricted cardinality, the set $\oplus(S_1,S_2)$ might be huge. Therefore, in Join$_{HC}$ we cannot allow to always run $\oplus$ inside our algorithm. Instead we will for each pair of certificates in S_1 and S_2 construct a set $S' \preceq \oplus(S_1,S_2)$ where $|S'|$ is bounded by a linearly single exponential FPT function of n and k while possibly $S' \subset \oplus(S_1,S_2)$.

We will handle cuts in two ways; one way when we are dealing with only splits, and one way when we are dealing with cuts that are not splits. For the latter part, when we do not have splits, we will use results from [43] to reduce the cardinality of preserving sets.

5.3.4 Hamiltonian Cycle on splits

For a certificate $G' \in \text{cert}(A)$ where $A \subset V(G)$, each path P of G' can be categorized by an unordered pair (N_1,N_2) so that for its two endpoints v_1 and v_2 (or single endpoint $v_1 = v_2$ if P is an isolated vertex) we have $N_1 = N(v_1)\setminus A$ and $N_2 = N(v_2)\setminus A$. We say that two paths are from the same *class* of paths if they get categorized by the same unordered pair. Two certificates $G', G'' \in \text{cert}(A)$ are *path equivalent* if there exists a bijection σ between the paths of G' and the paths in G'' so that for each pair of paths $P \in G'$ and $\sigma(P) \in G''$ the path P is in the same path class as $\sigma(P)$ (see Figure 5.1.

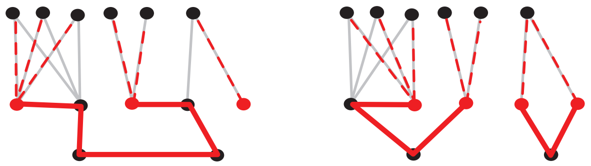


Figure 5.1: Two path equivalent certificates.

Claim 5.3. *For any subset $A \subset V(G)$ and certificates $G_1, G_2 \in \text{cert}(A)$, we have $G_1 \preceq_A G_2$ if G_1 is path equivalent to G_2.*

Proof. Suppose there is a certificate $G_3 \in \text{cert}(\overline{A})$ and witness $W \in \oplus(G_3,G_2)$. That means that for a set of edges $E_W \subseteq E(G[A,\overline{A}])$ we have $W = G_2 \cup G_3 + E_w$. From the definition of path classes and path equivalence, there exists a bijection σ from paths of G_2 to paths in G_1 so that for each path P in G_2 and edges $a_1p_1, a_2, p_2 \in E_W$ so that p_1, p_2 are the endpoints of P, there must exist two edges $a_1p'_1, a_1p'_2 \in E(G[A,\overline{A}])$ where p'_1, p'_2 are the endpoints of $\sigma(P)$ in G_1. Thus, if replacing the edges E_W with these edges, and replacing each path P in G_2 with the path $\sigma(P)$ of G_3, we have an Hamiltonian cycle. So, if $\oplus(G_2,G_3)$ contains a witness, so must $\oplus(G_1,G_3)$. □

For a certificate $G' \in \text{cert}(A)$ and path P of G', we say that P is an *isolated* path if one of its endpoints is not incident with an edge in $E(G[A,\overline{A}])$. That is, P is an isolated path if it is categorized by a pair containing an empty set.

As each vertex of a Hamiltonian cycle has degree exactly two, and for two certificates $G_1 \in \mathrm{cert}(A)$, $G_2 \in \mathrm{cert}(\overline{A})$, each of the edges in certificate $G' \in \oplus(G_1, G_2)$ is either in $E(G_1)$, $E(G_2)$, or $E(G[A, \overline{A}])$, we get the following observation.

Observation 5.4. *If G' is an important certificate, G' can not contain any isolated paths.*

Lemma 5.5. *Given two splits $(A, \overline{A})$ and $(B, \overline{B})$ so that $A \cap B = \emptyset$, and two sets $S_a \subseteq \mathrm{cert}(A)$ and $S_b \subseteq \mathrm{cert}(B)$, we can in $\mathcal{O}^*(|S_a| + |S_b|)$ time compute a new set $S' \preceq_{A \cup B} \oplus(S_a, S_b)$ where $|S'| \leq n^3$ and $S' \subseteq \oplus(S_a, S_b)$.*

Proof. First we trim the number of certificates in S_a: Since by Observation 5.4 all path classes are of the same type for important certificates when $(A, \overline{A})$ is a split, and by Claim 5.3 when two certificates are path equivalent one preserves the other, we can in $\mathcal{O}^*(|S_a|)$ time find a set S'_a of at most n certificates that preserves S_a, where all the certificates are pairwise not path equivalent. We do the same for S_b, producing a set S'_b.

Let us now notice that for any important certificate G_x in $\oplus(S'_a, S'_b)$ each of the paths of G_x are of one of the following three path classes.

(1) $(N(A) \setminus (A \cup B),\ N(A) \setminus (A \cup B))$

(2) $(N(B) \setminus (A \cup B),\ N(B) \setminus (A \cup B))$

(3) $(N(A) \setminus (A \cup B),\ N(B) \setminus (A \cup B))$

The number of paths of each of these path classes can be any number $0 \leq z < n$. So there are at most n choices as to how many paths G_x might have of each single path class, and thus in total no more than n^3 choices. This is a good indication that we can create a set S' of size no larger than n^3, but we also need to say how such a set can be created. For $s_a \in S'_a$ and $s_b \in S'_b$, let $c_{z_1, z_2}(s_a, s_b)$ be the certificate resulting in combining z_1 paths of s_a and of s_b together (by using edges of $E(G[A, B])$) to form z_2 paths of type (3). Since all paths in s_a are of the same path class, and all paths in s_b are of the same path class, whenever it is possible to construct $c_{z_1, z_2}(s_a, s_b)$, we can do so straight forward in polynomial time (e.g., by pairwise connecting z_2 paths of s_b to z_2 paths of s_a and extending one of the resulting paths with the $z_1 - z_2$ remaining paths of s_a and s_b in a way alternating between paths of s_a and of s_b). Clearly when $G_x \in \oplus(s_a, s_b)$, there must be two corresponding numbers z_1 and z_2 so that G_x is path equivalent to $c_{z_1, z_2}(s_a, s_b)$. Thus, if we for each pair of choices of z_1 and z_2 (with $0 \leq z_2 \leq z_1 \leq n$) construct $c_{z_1, z_2}(s_a, s_b)$ whenever possible, we have a set preserving $\oplus(s_a, s_b)$ of size $\mathcal{O}(n^2)$. By doing this for all pairs in $S'_a \times S'_b$ we thus end up with a set S of size $\mathcal{O}(n^4)$. We can again trim this set to a size of at most n^3 by removing path equivalent certificates. The result is a set S' of size at most n^3 which preserves $\oplus(S_a, S_b)$ and we can construct this set in $\mathcal{O}^*(|S_a| + |S_b|)$ time. □

5.3.5 Hamiltonian Cycle on general cuts

From Theorem 3 of [43] applied to a *graphic matroid* we have the following corollary, which will help us not only get an FPT algorithm, but a linearly single exponential FPT algorithm.

Corollary 5.6 ([43])**.** *Let G be a connected graph on n vertices and S a family of p-sized subsets of $E(G)$. We can for any integer q find a subset $\hat{S}$ of S with $|\hat{S}| \leq 2^n$ so that for any q-sized subset Y of $E(G)$, if there exists a set $X \in S$ disjoint from Y so that $X \cup Y$ is a forest, then there exists a set $\hat{X} \in \hat{S}$ disjoint from Y so that $\hat{X} \cup Y$ is a forest. Furthermore, the set $\hat{S}$ can be computed in $\mathcal{O}(|S|3^n)$ time.*

This might not seem like something related to finding Hamiltonian cycles, since finding a largest forest is polynomial time solvable and finding a largest cycle is NP-complete. However, as shown in [43], many NP-complete problems that contain a global connectivity constraint, for instance Steiner Tree and Hamiltonian Cycle, can be solved faster by the use of Corollary 5.6. In [43], the authors focus more on the Steiner Tree problem and less on Hamiltonian Cycle, so the precise usage of Corollary 5.6 applied to Hamiltonian Cycle is not very explicit. The following two lemmas can, however, be deduced from their paper, but as it is such a crucial part of this algorithm, we need all the details formally stated.

Lemma 5.7. *Given a graph G of k vertices, and family S of edge-sets, we can in $n^{\mathcal{O}(1)}(|S|9^k)$ time find a subset $\hat{S}$ of S of size at most 6^k so that for any set Y of edges in $E(G)$, if there exists a set $s \in S$ disjoint from Y so that $s \cup Y$ form a Hamiltonian cycle, then there is a set $\hat{s} \in \hat{S}$ disjoint from Y so that $\hat{s} \cup Y$ form a Hamiltonian cycle.*

Proof. For a set $x \in S$ let $D_i(x)$ denote the set of vertices in $V(G)$ incident to exactly i of the edges in x, i.e., the vertices of degree i in the graph $(V(G), x)$. We notice that if $Y \cup s$ is a Hamiltonian cycle, then s consists only of vertex disjoint paths. Furthermore, when $Y \cup s$ is a Hamiltonian cycle, then $Y \cup s'$ is a Hamiltonian cycle only if $D_i(s) = D_i(s')$ for $i = 0, 1, 2$. So, to produce a smaller subset $\hat{S}$ of S as stated in the lemma, we start by categorizing each of the $s \in S$ that consist of only vertex disjoint paths into one of at most $3^{|V(G)|}$ classes $S[D_0][D_1][D_2]$, depending on the content of $D_i(s)$ (we put s into the class where $D_0 = D_0(s), D_1 = D_1(s)$ and $D_2 = D_2(s)$). One result of this is that all sets in the same class will consist of exactly the same number of edges.

Now we apply Corollary 5.6 on each of the classes $S[D_0][D_1][D_2]$ with p being the size of the edge set in the particular class, and $q = n - p - 1$. We put the result into $\hat{S}[D_0][D_1][D_2]$ for each tri-partition D_0, D_1, D_2. We will now show that when $\hat{S}$ is the union of each $\hat{S}[D_0][D_1][D_2]$, then $\hat{S}$ satisfies the statement.

First of all, the size of $\hat{S}$ is at most $3^k 2^k$, since there are at most 3^k equivalence classes, and each class contributes by at most 2^k sets.

Second, we will have to show for every $Y \subseteq E(G)$ so that there exists a set $s \in S$ disjoint from Y where $Y \cup s$ is a Hamiltonian cycle of G, there also exists a set $\hat{s} \in \hat{S}$ so that also $Y \cup \hat{s}$ is a Hamiltonian cycle:

Suppose for disjoint s and Y, $Y \cup s$ is a Hamiltonian cycle of G. This means $D_0(s) = D_2(Y)$, $D_1(s) = D_1(Y)$ and $D_2(s) = D_0(Y)$, and s is an element of $S[D_0(s)][D_1(s)][D_2(s)]$. As all vertices of a Hamiltonian cycle have degree exactly two, for any element $s' \in S[D_0(s)][D_1(s)][D_2(s)]$ each vertex in

$V(G)$ must also have degree exactly two in $Y \cup s'$. So $Y \cup s'$ must be a disjoint set of cycles containing all vertices of $V(G)$. We will show that there exists a $s' \in \hat{S}[D_0(s)][D_1(s)][D_2(s)]$ so that the cycles of $Y \cup s'$ is connected, and hence must be one large Hamiltonian cycle. Let e be any edge in Y, and let Y_e be $Y \setminus \{e\}$. We notice that $s \cup Y_e$ is a (Hamiltonian) path and hence a subtree in G. That means by Corollary 5.6, there must be a $\hat{s} \in \hat{S}[D_0(s)][D_1(s)][D_2(s)]$ so that $\hat{s} \cup Y_e$ is a forest. As we know all the elements of $S[D_0(s)][D_1(s)][D_2(s)]$ have the same number of edges, $H' = \hat{s} \cup Y_e$ must contain $|V(G)| - 1$ edges. This means H' is one single component, as we know H' is acyclic. Furthermore, we know $H' \cup \{e\} = \hat{s} \cup (Y_e \cup \{e\})$ is a cycle cover, so $H' \cup \{e\} = \hat{s} \cup Y$ is indeed a Hamiltonian cycle of G. □

Lemma 5.8. *Let G be a graph and $A \subseteq V(G)$ some set of vertices separated from $V(G) \backslash A$ by $C \subseteq V(G)$ of size at least three. Given a family S of subsets of edges of $E(G[A \cup C])$, we can in $n^{\mathcal{O}(1)}(|S|9^{|C|})$ time construct a family $\hat{S} \subseteq S$ of size at most $6^{|C|}$ so that for any set Y of edges in $E(G[C \cup \overline{A}])$, if there exists a set $X \in S$ disjoint from Y so that $Y \cup X$ is a Hamiltonian cycle, then there exists a set $\hat{X} \in \hat{S}$ disjoint from Y so that $\hat{X} \cup Y$ is a Hamiltonian cycle.*

To prove Lemma 5.8, we use what is called a *torso*. For a graph G and subset $S \subseteq V(G)$, we say that the *torso* of G in S is the graph we get by taking $G[S]$ and then add the edges $\{uv : \exists uPv \in G \text{ s.t. } P \in G \setminus S\}$. In general this means we take $G[S]$ and for each component $C \in G \setminus S$ make $N(C)$ into a clique. We will only use torsos in the restricted case when the components $C \in G \setminus S$ have exactly two neighbours in C, and hence each component will only contribute to the torso by a single edge.

Proof of Lemma 5.8. Let G' be the complete graph on $V(G)$ (i.e., $V(G') = V(G)$ and $E(G') = \{uv : u \in V(G), v \in V(G)\}$). We notice that for disjoint sets $X \subseteq E(G[A])$ and $Y \subseteq E(G[\overline{A} \cup C])$ $X \cup Y$ form a Hamiltonian cycle in G only if $X \cup Y$ also form a Hamiltonian cycle in G'. Also, $X \cup Y$ can be a Hamiltonian cycle only if for some $Y' \subseteq E(G'[C])$ $X \cup Y'$ is a Hamiltonian cycle (namely the torso Y' of Y in C). So, we will from here on assume $V(G') = V(G) = A \cup C$.

Our goal now will be to reduce the problem from looking at $G'[A \cup C]$ to only looking at $G'[C]$. First notice that unless X induces a disjoint set of paths ending in C which covers all the vertices in $A \setminus C$, then $X \cup Y$ cannot be a Hamiltonian cycle for any $Y \subseteq E(G'[C])$. Let $T \subseteq S$ be the set consisting of all $X \in S$ where X is consisting of such edge disjoint set of paths. For any $X \in T$ and $Y \subseteq E(G'[C])$ $X \cup Y$ is a Hamiltonian cycle of $G'[A \cup C]$ if and only if for the torso X' of X in X, $X' \cup Y$ is a Hamiltonian cycle of $G'[C]$.Therefore, to construct the set $\hat{S}$ in the statement of the lemma, we do the following:

We first let T' be the set of these torsos. By first removing all the duplicates in T' and then applying Lemma 5.7 to it, we get a subset $\hat{T}'$ of size at most $6^{|C|}$. The set $\hat{S} = \{X \in T : X' \in \hat{T}'\}$ will thus be a set of size at most $6^{|C|}$ and such that for any $X \in S$ and $Y \subseteq E(G[\overline{A} \cup C])$, if X and Y are disjoint and $X \cup Y$ is a Hamiltonian cycle of G, then $\exists \hat{X} \in \hat{S}$ disjoint from Y so that $\hat{X} \cup Y$ is a Hamiltonian cycle in G. □

Lemma 5.8 gives an insight to how it is possible to make linearly single exponential algorithms for HAMILTONIAN CYCLE parameterized by treewidth, as done in [43]. The idea is to build a set of partial solutions using dynamic programming in a bottom up manner in a tree decomposition, and at each step use Lemma 5.8 to reduce the number of partial solutions needed to ensure you will find a Hamiltonian cycle in the end of your algorithm. This works because at each step of the algorithm all partial solutions will be disjoint paths that have all their endpoints inside a small separator (a bag in the tree decomposition). However, when parameterizing by split-matching width, even for cuts of small mm-value and a vertex cover C of small size, the partial solutions (certificates) will not necessarily consist of paths that have endpoints inside C, but possibly in $N(C)$, which could be large. To overcome this problem, we define what we call an *extension*.

An extension of a certificate is a certificate plus some extra edges. The idea is that an extension will encompass how a certificate $G' \in \operatorname{cert}(X)$ looks after adding more edges than those in $E(G[X])$. Formally, for a certificate $G' \in \operatorname{cert}(X)$ and set of edges E^* disjoint from $E(G[X])$, we say that a set $S \subseteq \{G' \cup E' : E' \subseteq E^*\}$ is an extension of G' by the set E^*. For a set of certificates P, the set S is an extension of P by E' if it is a union of extensions by E' of the certificates in P. For a set of certificates P we say that an extension S of P by E^* is *preserving* if for any edge set Y not intersecting $E(G[A]) \cup E^*$, if there is a certificate $C \in P$ and $E' \subseteq E^*$ so that $Y \cup G' \cup E'$ is a Hamiltonian cycle, then there is an element $G'' \in S$ so that $G'' \cup Y$ is a Hamiltonian cycle. A preserving extension of a single certificate G' is simply a preserving extension of $\{G'\}$.

Observation 5.9. *For $P \subseteq \operatorname{cert}(A)$ and edges $E' \subseteq E(G[A,\overline{A}])$, if $\mathcal{S}$ is a preserving extension of P by E', then $\{S \setminus E' : S \in \mathcal{S}\}$ preserves P.*

Motivated by Observation 5.9, we give the following lemma, which will be used to reduce the number of certificates needed to preserve certificate sets over sets of small mm-value. The result of this is captured in Corollary 5.11.

Lemma 5.10. *Given a set of certificates $S \subseteq \operatorname{cert}(A)$ and a vertex cover C of $G[A,\overline{A}]$ of size at least three, we can in $\mathcal{O}^*(|S|9^{3|C|})$ time create a preserving extension of S by $E^* = E(G[A, C \setminus A])$ of size no larger than $6^{|C|}$.*

Proof of Lemma 5.10. We prove this first for a single certificate $S = \{G'\}$ and then generalize to a set S containing multiple certificates in the end of the proof.

Suppose Y is a set of edges outside of $E(G[A])$ and E^* so that $Y \cup G' \cup E'$ is a Hamiltonian cycle for some $E' \subseteq E^*$. Neither E' nor Y consist of edges in $E(G[A])$, so by temporarily removing the edges of $E(G[A])$ not in $E(G')$ we get a new graph G^* for which $Y \cup G' \cup E'$ is also a Hamiltonian cycle. Clearly, all Hamiltonian cycles of G^* are also Hamiltonian cycles of G. Let $Q = E(G[A,\overline{A}]) \cap (Y, E^*)$. All paths in G' must be incident to exactly two edges of Q and no two paths in G' can be adjacent to the same edge in Q (since all edges in Q only have one endpoint in A). Furthermore, all edges of Q must be incident with a vertex of C, but no vertex of C can be incident with more than two vertices of Q. So $|Q| \leq 2|C|$ and we can conclude that the number of paths in G' is at most $|C|$. Now let X be the endpoints of the paths in G'. Each path has at most 2 endpoints, so $|X|$ is at most $2|C|$. The set $X \cup C$ is of size at most $3k$ and disconnects G' from the rest of the graph. We then do the following subroutine to construct a preserving extension.

```
$S' = \{G'\}$
for each $e \in E'$ incident with $X$:
    for each $s \in S'$ add $s \cup \{e\}$ to $S'$
    reduce the size of $S'$ using Lemma 5.8 with $X \cup C$ as separator
end for
```

The set S' resulted from this subroutine will be a preserving extension of G' by E', and by Lemma 5.8, its size will not exceed $2^{|C|+|X|} \leq 2^{3|C|}$. The runtime to create this set will be $n^{\mathcal{O}(1)}(9^{3|C|})$.

For a set S of multiple certificates, we find for each $G' \in S$ a preserving extension $S_{G'}$ by E' and then in a similar manner as above, construct a preserving set S' by adding the preserving extensions $S_{G'}$ to S' one by one for each $G' \in S$, and for each time we add a new extension reduce the size of S' using Lemma 5.8 with C as a separator. The size of the ending set of certificates will be bounded by $6^{|}C|$ and the total runtime as follows: To produce all the preserving extensions we use $\mathcal{O}^*(|S|9^{3|C|})$ time, and to put all of these at most $|S|8^{|C|}$ preserving extensions into S', we use $\mathcal{O}^*(|S|8^{|C|}9^{|C|})$ time. The total runtime is thus dominated by finding preserving extensions for single certificates, which is bounded by $\mathcal{O}^*(|S|9^{3|C|})$. □

Corollary 5.11. *Given a set $S \subseteq \mathrm{cert}(A)$, and a vertex cover C of $G[A,\overline{A}]$ where $3 \leq |C| = k$, we can in $\mathcal{O}^*(|S|9^{3k})$ time find a set $\hat{S} \subseteq S$ so that $\hat{S} \preceq_A S$ and $|\hat{S}| \leq 6^k$.*

Now that we have defined preserving extensions, and already shown how we can use this to reduce the size of a preserving set of certificates, we will show how we can also use extensions to produce small sets S preserving $\oplus(S_1,S_2)$ for certificates S_1 and S_2. This is the last step needed to create our dynamic programming algorithm for Hamiltonian Cycle.

Lemma 5.12. *For a tri-partition (A,B,W) of $V(G)$, and $S_a \in \mathrm{cert}(A)$ and $S_b \in \mathrm{cert}(B)$, for $k = \max\{\mathrm{sm}(A),\mathrm{sm}(B)\}$ we can in $n^{\mathcal{O}(1)}2^{\mathcal{O}(k)}$ time compute a set $S \subseteq \oplus(S_a,S_b)$ so that $S \preceq_{A\cup B} \oplus(S_a,S_b)$ of size at most $2^{\mathcal{O}(\mathrm{sm}(W))}n^{\mathcal{O}(1)}$.*

Proof. The case when both $(A,\overline{A})$ and $(B,\overline{B})$ are splits, the statement holds by Lemma 5.5. So, we only need to prove that it holds for the case when when at least one of $(A,\overline{A})$ and $(B,\overline{B})$ are not splits.

We first assume that there exists a certificate $S_w \in \mathrm{cert}(W)$ so that the set $\oplus(\oplus(S_a,S_b),\{S_w\})$ contains a witness $H = S_a \cup S_b \cup S_w \cup E'$ where E' is disjoint from the three certificates. Let $X_a \subseteq A$ and $X_b \subseteq B$ be the set of vertices in A and B incident with less than two edges of S_a and S_b, respectively. That is, X_a and X_b are exactly the vertices of A and B, respectively, that are incident with E'.

We now show that if a witness H as described above exists, then $|X_a| \leq 2\,\mathrm{mm}(A)$ and $|X_b| \leq 2\,\mathrm{mm}(B)$. Without loss of generality, we prove that this holds for X_a. As each vertex in X_a must be incident with an edge of E', and each vertex in $\overline{A}$ can be incident to at most two edges of E' since H is a simple cycle, there is a matching in E' of at least half the size of X_a, implying that $|X_a| \leq 2\,\mathrm{mm}(A)$. The same also holds for X_b. Let C be a vertex cover of $G[A,\overline{A}]$. If both $\mathrm{mm}(A), \mathrm{mm}(B) \leq k$, then $C \cup X_a \cup X_b$ is a vertex cover of size $\leq 5k$. This means that by Lemma 5.10 we can construct a preserving extension $\hat{S_a}$ of S_a by $E(G[A,(C \cup X_b) \setminus A])$ of size 6^{5k}, which combined with S_b must preserve $\oplus(S_a, S_b)$. That is, $S' = \{S'_a \cup S_b : S'_a \in \hat{S_a}\}$.

For the case when either $\mathrm{mm}(A) > \mathrm{sm}(A)$ or $\mathrm{mm}(B) > \mathrm{sm}(B)$, we need a slightly different argument. Assume without loss of generality that $\mathrm{mm}(B) > \mathrm{sm}(B)$, and thus $(B, \overline{B})$ is a split. As before $|X_a| \leq 2k$, but now $|X_b|$ is possibly very large. What we notice though, is that as each vertex of X_a can be incident to at most two edges of E', the number of edges in E' incident with X_a is at most $2|X_a| \leq 4k$. This means that no more than $4k$ of the paths in S_b will connect directly to S_b by the edges in E'. As all endpoints of all paths in S_b (including isolated vertices, which can be thought of as paths of length zero) have the same neighbourhood in $\overline{B}$ and are interchangeable, we can simply disregard all but $4k$ paths of S_b, and do the same for the remaining $4k$ paths as we did for S_b for the case when there were no splits. This means the set X_b is of size at most $8k$ instead of $2k$, giving a runtime of $\mathcal{O}^*(|S_a||S_b|9^{3(11k)})$ to compute a preserving set of $\oplus(S_a, S_b)$. After computing this set, we can trim it further to a size of either n as we did in the proof of Lemma 5.5 when $(A \cup B, W)$ is a split, or trim it to a size of $6^{\mathrm{mm}(W)}$ by Corollary 5.11, depending on whether $(A \cup B, W)$ is a split or not. As $\mathrm{sm}(W) = \mathrm{mm}(W)$ when $(A \cup B, W)$ is not a split, we end up with a preserving set of $\oplus(S_a, S_b)$ of size bounded by $n + 6^{\mathrm{sm}(W)}$. □

Theorem 5.13. *Given a graph G and branch decomposition (T, δ) of sm-width k, we can solve* Hamiltonian Cycle *in time $\mathcal{O}^*(2^{\mathcal{O}(k)})$.*

Proof. Let Join$_{hc}$ be the algorithm of Lemma 5.12. Using Recursive with Join$_{hc}$, we know the size of both of the inputs of Join$_{hc}$ is at most the size of its output, i.e., $|S_a|, |S_b| \leq \mathcal{O}^*(2^k)$. So, each call to Recursive has runtime at most $\mathcal{O}^*(2^{\mathcal{O}(k)})$. As there are linearly many calls to Recursive and there is a polynomial time verifier for the certificates Recursive produces, by the definition of $\preceq$, the total runtime is also bounded by $\mathcal{O}^*(2^{\mathcal{O}(k)})$. □

5.4 t-Coloring

5.4.1 The problem.

The decision problem t-Coloring asks for an input graph G, whether there exists a labelling function c of the vertices of $V(G)$ using only t colors in such a way that no edge has its endpoints labelled with the same color. Equivalently, it asks whether there exists a t-partitioning of the vertices so that each part induces an independent set. For simplicity, we will allow empty parts in a partition (e.g., $\{\{x_1, x_2, x_3\}, \{x_4\}, \emptyset, \emptyset, \emptyset\}$ is a 5-partition of $\{x_1, x_2, x_3, x_4\}$).

For a set A and partition $p = \{p_1, p_2, \ldots, p_j\}$, we denote by $A \cap p$ the partition $\{p_1 \cap A, p_2 \cap A, \ldots, p_j \cap A\}$.

5.4.2 The certificates and $\oplus$

For t-Coloring, we define $\mathrm{cert}(X)$ for $X \subseteq V(G)$ to be all t-partitions of X and $\oplus(p,q)$ for $p = p_1, \ldots, p_t$ and $q = q_1, \ldots, q_t$ to be the following set of partitions

$$\left\{ \bigcup_{i \le t} \left\{ \left\{ p_i \cup q_{\sigma(i)} \right\} \right\} : \sigma \text{ is a permutation of } \{1, \ldots, t\} \right\} .$$

We can easily construct a polynomial time algorithm that given a t-partition p of independent sets (which there must exist at least one of if G is a "yes" instance) is able to confirm that G is a "yes" instance. So t-partitions of $V(G)$ forming independent sets will be our witnesses.

5.4.3 The Join$_{col}$ function.

The main observation for the design of this procedure is that whenever $\left(A, \overline{A}\right)$ is a split, there exists a single element $x \in \mathrm{cert}(A)$ so that $\{x\} \preceq_A \mathrm{cert}(A)$. Also, when $\mathrm{mm}(A) < k$ there is a separator of A and $\overline{A}$ so that for any t-partition witness, the separator is intersecting at most k of the parts of the t-partition. In the procedure Trim$_{\text{COL}}$ below we use this to trim the number of certificates to store at each step of the algorithm; if two certificates restricted to the vertices in the separator of A and $\overline{A}$ are equal, we only store one of them. This results in less than k^k certificates to store.

For two t-partitions P, Q, we say that we *merge* P and Q when we generate a new partition R by pairwise combining the parts (by union) of P with the parts of Q in such a way that R is a t-partition where each part is an independent set. If P and Q can be merged, we say that P and Q are *mergeable.*

Lemma 5.14. *Given two t-partitions P and Q, deciding whether P and Q are mergeable, and merging P and Q if they are, can be done in polynomial time.*

Proof. We can check whether P and Q are mergeable by reducing it to deciding whether a bipartite graph has a perfect matching: We generate a bipartite graph $B = (P, Q)$ where each vertex/part $p \in P$ is adjacent to $q \in Q$ if and only if $p \cup q$ is an independent set. When P and Q partition sets that do not intersect, then we can merge P and Q by for each edge in the matching, combine the respective parts the edge is incident with. If there is no perfect matching in B, then that must mean there is no way of pairwise combining the parts of P with the parts Q so that all combined parts are independent.

If P contain parts that share elements with parts of Q, then we know these parts must be combined in all merged partitions, so we combine all these sets and then run the above reduction to perfect matching (on the parts that do not have intersecting elements). If a part intersect with more than one other part, then P and Q cannot be merged. $\square$

Lemma 5.15. *Let A be a subset of $V(G)$, $C \subseteq V(G)$ a separator of A and $\overline{A}$, and P_A and P_C be t-partitions of A and C, respectively. If P_C and P_A merge to a t-partition P'_A, then for any set $B \subseteq (C \cup \overline{A})$ and t-partition P_B of B, P_B is mergeable with P'_A if and only if P_C is mergeable with P_B.*

Proof. For the forward direction, we notice that for any t-partition P' resulting from merging P'_A with P_C, the t-partition $P^* = P' \cap (C \cup B)$ is a result of merging $P'_A \cap (C \cup B) = P_C$ with $P_B \cap (C \cup B) = P_B$, and hence P_C and P_B are mergeable.

We will prove the backwards direction by constructing a partition P' which is the result of merging P'_A with P_B. As P_C partitions exactly the set C, and both P_A merge with P_C to a t-partition P'_A and P_B merge with P_C to a t-partition P'_B, there is an ordering of the parts of P'_A and P'_B so that the i-th part of P'_A intersected with C equals the i-th part of P'_B intersected with C. We let P' be the multiset resulting from pairwise combining the i-th part of P'_A with the i-th part of P'_B. As the only vertices that occur in parts of both P'_A and of P'_B is the set of vertices C, by combining the parts based on the ordering we described, we have made sure that each vertex appear in exactly on part of P', and hence P' is a t-partition.

To show that each part in P' is an independent set, we assume by contradiction that two adjacent vertices x and y are in the same part of P'. By how we constructed P', no two vertices in different parts in either P'_A or P'_B will be in the same part in P'. Therefore, as P'_A partitions $A \cup C$ into parts that are independent sets, and P'_B does the same for $B \cup C$, if x and y are adjacent and in the same part, one of them must be in $A \setminus C$ and the other in $B \setminus C$. However, C disconnects A and B, and hence no edge exists between vertices in $A \setminus C$ and $B \setminus C$, contradicting that x and y are adjacent. □

Procedure Trim$_{\text{COL}}$
Input: $S \subseteq \text{cert}(A)$
Output: $S' \preceq_A S$ of size $|S'| \leq \text{sm}(A)^{\text{sm}(A)}$

```
remove from $S$ the partitions containing parts that are not independent sets
if $\left(A, \overline{A}\right)$ is a split then
    mark one certificate $P' \in S$ where
            the number of non-empty parts in $N(\overline{A}) \cap P'$ is minimized
else
    $C \leftarrow$ minimum vertex cover of $G[A, \overline{A}]$
    for each $t$-partition $P_C$ of $C$ do
        mark one certificate $P'_C \in S$ which is mergeable with $P_C$
return the marked certificates of $S$
```

Lemma 5.16. *The procedure* Trim$_{\text{COL}}$ *with input $S \subseteq A$ for $A \subseteq V(G)$ returns a set $S' \preceq_A S$ of size at most* $\text{sm}(A)^{\text{sm}(A)}$, *and has a runtime of* $\mathcal{O}^*(|S|\,\text{sm}(A)^{\text{sm}(A)})$.

Proof. The runtime is correct as a result of the fact that merging (Lemma 5.14) takes polynomial time to execute, and it produces a set $S' \subseteq S$ of cardinality at most $1 = \mathrm{sm}(A)^{\mathrm{sm}(A)}$ if $(A, \overline{A})$ is a split, and cardinality at most $\mathrm{mm}(A)^{\mathrm{mm}(A)} = \mathrm{sm}(A)^{\mathrm{sm}(A)}$ otherwise. We now show that $S' \preceq_A S$:

For contradiction, let us assume there exists a certificate $P_w \in \mathrm{cert}(\overline{A})$ so that for a certificate $P \in S$ the set $\oplus(P_w, P)$ contains a witness, while $\oplus(\{P_w\}, S')$ does not contain a witness. Let us first assume $\left(A, \overline{A}\right)$ is a split:

As $\left(A, \overline{A}\right)$ is a split, each part of P is either adjacent to all of $N(A)$, or not adjacent to $\overline{A}$ at all. Let z be the number of parts in P that are adjacent to all of $N(A)$. This also means z is a lower bound to the number of parts in P_w that do not have neighbours in A. In Trim$_{\mathrm{COL}}$ we mark (and thus output) one certificate P' where at most z parts have neighbours $\overline{A}$. Thus for each of the at most z parts in P' that are adjacent to A, there is a part in P_w not adjacent to any vertex in A. So, we can conclude that P' and P_w can be merged together to form a witness. This contradicts that $\oplus(\{P_w\}, S')$ does not contain a witness when $\left(A, \overline{A}\right)$ is a split.

Now, let us assume $\left(A, \overline{A}\right)$ is not a split. Let W be a witness in $\oplus(\{P_w\}, S)$. For the vertex cover C, we have the following smaller t-partitions of W; $P_C = W \cap C$ and $P_W = W \cap C$. By Lemma 5.15, as C disconnects A from the rest of the graph, any t-partition of A mergeable with P_C is mergeable with P_W. The algorithm assures that for all t-partitions P_C of C, whenever a t-partition in S is mergeable with P_C, at least one t-partition mergeable with P_C exists in S'. From this we can conclude that $\oplus(\{P_w\}, S')$ preserves $\oplus(\{P_w\}, S)$. □

Procedure Join$_{col}$
Input: $S_1 \subseteq \mathrm{cert}(A_1)$ and $S_2 \subseteq \mathrm{cert}(A_2)$ for disjoint sets A_1, A_2 and $A = A_1 \cup A_2$
Output: $S \preceq_A \oplus(S_1, S_2)$

```
$S \leftarrow \emptyset$
$C_1, C_2 \leftarrow$ minimum vertex cover of $G[A_1, \overline{A_1}]$ and $G[A_2, \overline{A_2}]$, respectively
for $P_1 \in S_1$ and $P_2 \in S_2$ do
    $N_1 \leftarrow$ parts of $P_1$ not adjacent to $N(A_1)$
    $N_2 \leftarrow$ parts of $P_2$ not adjacent to $N(A_2)$
    if both $(A_1, \overline{A_1})$ and $(A_2, \overline{A_2})$ are splits then
        for $0 \le z \le \min\{|N_1|, |N_2|\}$ do
            add to $S$ a $t$-partition generated (if possible) by
                    merging the two partitions $P_1$ and $P_2$ together
                    in such a way that exactly $z$ of the parts of $N_1$
                    are merged with parts of $N_2$
```

```
  else if neither $(A_1,\overline{A_1})$ nor $(A_2,\overline{A_2})$ is a split then
    for each $t$-partition $P_c$ of $C_1 \cup C_2$ do
      if both $P_1$ and $P_2$ are mergeable with $P_c$ then
        $P_1^c \leftarrow$ merge $P_c$ and $P_1$
        $P' \leftarrow$ merge $P_1^c$ and $P_2$
        add $P' \cap (A \cup B)$ to $S$

  else // up to symmetry let $(A_1,\overline{A_1})$ be split and $(A_2,\overline{A_2})$ not
    for each $t$-partition $P_c$ of $C_2$ mergeable with $P_2$ do
      for each subset $Q$ of the non-empty parts of $P_c$ do
        $P_q \leftarrow$ merge (if possible) $P_c$ with $P_1$ so that $Q$ is exactly the
                non-empty parts of $P_c$ that get combined with $N_1$
        $P' \leftarrow$ the $t$-partition generated by merging $P_q$ with $P_1$
        add to $S$ the $t$-partition $P' \cap (A \cup B)$

return TRIM$_{col}(S)$
```

Lemma 5.17. *Procedure* JOIN$_{col}$ *is correct and runs in time* $\mathcal{O}^*(|S_1||S_2|k^{3k})$ *for* $k = \max\{\mathrm{sm}(A_1), \mathrm{sm}(A_2), \mathrm{sm}(A)\}$*, producing a set* S *of cardinality* $\mathcal{O}(k^k)$*.*

Proof. We will go through the three cases in the algorithm (when zero, one or two out of the two cuts $(A_1,\overline{A_1})$ and $(A_2,\overline{A_2})$ are splits), and show that for each pair (P_1,P_2) of certificates in $S_1 \times S_2$, the output of the algorithm preserves $\oplus(P_1,P_2)$. As the last line of the algorithm assures that the output preserves S (by Lemma 5.16) and the cardinality of the output is of the correct size, we only need to show that S preserves $\oplus(P_1,P_2)$ and that the runtime is correct.

Suppose neither of the two cuts are splits. In this case, whenever there exists a certificate of $\oplus(P_1,P_2)$ mergeable with a t-partition P_C of the separator $C = C_1 \cup C_2$ of A and $\overline{A}$, the algorithm assures that there is going to be at least one t-partition of A in S mergeable with P_C. By Lemma 5.15, this means S is going to preserve $\oplus(P_1,P_2)$.

If both of the two cuts are splits, then for each part p_i of every t-partition of A the neighbourhood $N(p_i) \setminus A$, must either be empty, $N(A_1)$, $N(A_2)$, or $N(A)$. In this case the algorithm generates, for each possible z so that there exists an important certificate $P^* \in \oplus(P_1,P_2)$ where the number of parts with empty neighbourhoods is exactly z (and thus, the number of parts with neighbourhood equal $N(A_1)$, $N(A_2)$ and $N(A)$, is $|N_2|-z$, $|N_1|-z$, and $t-|N_1|+|N_2|+z$, respectively), generates a t-partition of independent sets where there the same number of parts with each of the particular four neighbourhoods is equal to the number of parts of each neighbourhood for P^*. We can observe that when two t-partitions over the same set both consist of only independent sets, and there is a correspondence between the parts of both partitions so that the neighbourhood in $\overline{A}$ of each pair of corresponding parts are the same, the two partitions are mergeable to exactly the same t-partitions of $\overline{A}$. Therefore, the set of t-partitions the algorithm generates in the case when both cuts are splits, preserves $\oplus(P_1,P_2)$.

If exactly one of the cuts are splits, let us assume without loss of generality that $(A_1,\overline{A_1})$ is the split. As above, each part of P_1 is one of two types; adjacent to $N(A_1)$, or not adjacent to $N(A_1)$. When C is a vertex cover of $G[A_2, \overline{A_2}]$, for each important certificate $P^* \in \oplus(P_1, P_2)$, we have a t-partition $P_C = P^* \cap C$ of C. Of the parts in P_c, some subset Z of the non-empty parts get combined with the parts of P_1 that do not have any neighbours in $N(A_1)$ and the rest of the non-empty parts get combined with the other type of parts in P_1. Since C disconnects A_2 from the rest of the graph, there must be a bijection from each part in P^* to parts with the exact same neighbourhood in $N(A)$ for the t-partition generated by the algorithm in the last if/else case, for $Q = Z$. This, in turn, means that the two partitions preserve each other, and so we can conclude that the set of certificates the algorithm produces preserves $\oplus(P_1, P_2)$.

The runtime of the algorithm we get as follows: Excluding polynomials of n, we get the worst case runtime when neither A_1 nor A_2 are splits. Then S grows to be as large as $\mathcal{O}(k^{2k})$ (the number of t-partitions of $C_1 \cup C_2$) for each pair of certificates in $S_1 \times S_2$. This implies a runtime of $\mathcal{O}^*(|S_1||S_2|k^{2k})$ for the entire part before the execution of TRIM$_{\text{COL}}$, which by Lemma 5.16 takes $\mathcal{O}^*(|S_1||S_2|k^{3k})$-time given the size of S. □

Theorem 5.18. *Given a graph G and branch decomposition (T, δ) of sm-width k, we can solve* CHROMATIC NUMBER *in time $\mathcal{O}^*(k^{5k})$.*

Proof. In Lemma 5.17 we show JOIN$_{col}$ is correct and runs in time $\mathcal{O}^*(|S_1||S_2|k^{3k})$, producing a set S of cardinality $\mathcal{O}(k^k)$. So, using RECURSIVE with JOIN$_{col}$, we know the size of both of the inputs of JOIN$_{col}$ is at most the size of its output, i.e., $|S_1|, |S_2| \leq \mathcal{O}^*(k^k)$. So, each call to RECURSIVE has runtime at most $\mathcal{O}^*(k^{5k})$. As there are linearly many calls to RECURSIVE and there is a polynomial time verifier for the certificates RECURSIVE produces, by the definition of $\preceq$, the total runtime is also bounded by $\mathcal{O}^*(k^{5k})$. To solve CHROMATIC NUMBER when we have an algorithm for t-COLORING, we simply run the t-COLORING algorithm for each value of $t \leq n$, giving the same runtime for CHROMATIC NUMBER when excluding polynomial factors of n. □

With a somewhat relaxed analysis of the runtimes of this section in terms also of n, we get a total runtime of $\mathcal{O}(k^{5k}n^5)$ for solving CHROMATIC NUMBER.

5.5 Edge Dominating Set

5.5.1 The problem.

The decision problem t-EDGE DOMINATING SET asks for an input graph G, whether there exists a set $E' \subseteq E(G)$ of cardinality at most t, so that for each edge $e \in E(G)$ either $e \in E'$ or e shares an endpoint with an edge $e' \in E'$. We say that an edge $e' \in E'$ *dominates* $e \in E(G)$ if e and e' share an endpoint.

5.5.2 The certificates and $\oplus$

For t-EDGE DOMINATING SET, we define cert(X) for $X \subseteq V(G)$ to be all subgraphs of $G[X]$. This might seem odd, as we are looking for a set of edges. However, for a certificate $G' \in \text{cert}(X)$, the set $V(G')$ is going to make the algorithm simpler. A witness will be a subgraph G_w such that $E(G_w)$ is an edge dominating set of size at most t and each vertex of $V(G_w)$ is incident with an edge of $E(G_w)$. Checking the latter can obviously be done in polynomial time and checking that $E(G_w)$ is an edge dominating set of size at most t can also be done in polynomial time since t-Edge Dominating Set is in NP.

For disjoint sets $A, B \subseteq V(G)$ and certificates $G_A \in \text{cert}(A)$ and $G_B \in \text{cert}(B)$, we define $\oplus(G_A, G_B)$ to be the set

$$\left\{G_A + G_B + E' : E' \subseteq E(G[V(G_A), V(G_B)])\right\}.$$

5.5.3 The Join$_{\text{eds}}$ function.

Given a graph G' and vertex $v \in V(G')$, we say that v is an *isolated* vertex of G' if it is not incident with any edge of $E(G')$. If a vertex is not isolated, it is *non-isolated*. We say a set of edges E' *span* a set of vertices X if $X \subseteq V(E')$. We denote by $I(G')$ the isolated vertices of G'.

We say that a certificate $G' \in \text{cert}(A)$ is *locally correct* if all edges in $G[A]$ have an endpoint in $V(G')$ and all the isolated vertices of $V(G')$ are in $N(\overline{A})$. A certificate is *locally incorrect* if it is not locally correct. We see that a certificate in $\text{cert}(A)$ which is locally incorrect cannot also be an important certificate as there exists an edge in $G[A]$ which is not dominated by edges in $E(G')$ and cannot be dominated by edges in $G[A, \overline{A}]$.

We observe that for two locally correct certificates $G_1, G_2 \in \text{cert}(A)$, if $N(\overline{A}) \cap G_1 = N(\overline{A}) \cap G_2$ and the isolated vertices of G_1 equal those of G_2, then $G_1 \preceq_A G_2$ if $|E(G_1)| \le |E(G_2)|$.

Lemma 5.19. *Given a graph G without isolated vertices and a subset $A \subseteq V(G)$, a minimum cardinality set $X \subseteq E(G)$ of edges spanning the vertices A can be found in polynomial time.*

Proof. Let M be a maximum matching of $G[A]$. Any set spanning A must be of size at least $|M| + (|A| - 2|M|) = |A| - |M|$, as otherwise there must exist a matching in $G[A]$ larger than M. Let R be the set of vertices in A not incident with any edge in M. As no vertex of G is isolated, each vertex in R is incident with at least one edge in G. Thus, we can easily find a set E_R of $|R|$ edges spanning R. We claim that the set $X = M \cup E_R$ is a minimum cardinality set of edges spanning A; Clearly X spans A, as each vertex not spanned by M is by definition spanned by E_R. Furthermore, the size of R is exactly $|A| - 2|M|$, so the size of X is $|M| + (|A| - 2|M|) = |A| - |M|$. Hence, the set X is a minimum cardinality set spanning A. □

As usual, our join-procedure will consist of a part where we join together pairs of certificates, imitating $\oplus$, and a part where the size of the output is reduced using a trim-procedure that ensures the output is preserving. For the Edge Dominating Set-problem, the trimming part consists of two procedures; one for when the cut in question is a split, and one for when it is not a split. We first present the simplest of the two, namely the one used when the cut is a split (Trim$_{\text{EDS}}$-split).

Procedure TRIM$_{\text{EDS}}$-SPLIT
Input: $S \subseteq \text{cert}(A)$ where $(A, \overline{A})$ is a split
Output: $S' \preceq_A S$ of size $|S'| \leq \mathcal{O}(n^2)$.

```
remove from $S$ all locally incorrect certificates
for each $0 \leq x_1, x_2 \leq n$ do
    mark one certificate $G' \in S$ of minimized $|E(G')|$ where
        $x_1$ equals $|I(G')|$, and
        $x_2$ equals $\left|V(G') \cap N(\overline{A})\right|$
return all the marked certificates in $S$
```

Lemma 5.20. *The procedure* TRIM$_{\text{EDS}}$-SPLIT *on $S \subseteq \text{cert}(A)$ for $A \subseteq V(G)$ where $(A, \overline{A})$ is a split, returns a set $S' \subseteq S$ so that $S' \preceq_A S$ and $|S'| \leq \mathcal{O}(n^2)$ in $\mathcal{O}^*(|S|)$-time.*

Proof. The algorithm clearly does $(n+1)^2$ number of iterations in the for-loop, and for each of these iterations it iterates through the list S. For each element in S, it does a polynomial amount of work, so the total runtime is $\mathcal{O}^*(|S|)$. Furthermore, at most one element is marked to be put in the output at each iteration, so the output of the algorithm is of size $\mathcal{O}(n^2)$.

For the correctness of the algorithm, notice that for any two locally correct certificates $G_1, G_2 \in \text{cert}(A)$, if $|I(G_1)| = |I(G_2)|$ and $|V(G_1) \cap N(\overline{A})| = |V(G_2) \cap N(\overline{A})|$, then G_1 preserves G_2. This is because each vertex in $N(\overline{A})$ is adjacent to exactly the same vertices of $\overline{A}$, and so for any set $X_2 \subseteq N(\overline{A})$, if there is a set of edges spanning $X \subseteq \overline{A}$ and $X_1 \subseteq N(\overline{A})$, there is also a set of edges of the same cardinality spanning X_2 and X as long as $|X_2| = |X_1|$. □

Now we describe the trim-procedure for when the respective cut is not a split (TRIM$_{\text{EDS}}$-NON-SPLIT). This procedure is more complicated than TRIM$_{\text{EDS}}$-SPLIT. The core idea of the procedure is that when $G_a + G_{\overline{a}} + E_w \in \oplus(G_a, G_{\overline{a}})$ is a witness, then also for any locally correct certificate G_i, the certificate $G_i + G_{\overline{a}} + E'_w \in \oplus(G_i, G_{\overline{a}})$ is also a witness, as long as $V(G_i) \cup V(G_{\overline{a}})$ is a vertex cover of $G[A, \overline{A}]$, E'_w spans $I(G_i)$ and the vertices in $V(E_w) \cap \overline{A}$, and $|E(G_i)| + |E'_w| \leq |E(G_a)| + |E_w|$.

Procedure TRIM$_{\text{EDS}}$-NON-SPLIT
Input: $S \subseteq \text{cert}(A)$
Output: $S' \preceq_A S$ of size $|S'| \leq 3^{\text{mm}(A)}$.

```
remove from $S$ all locally incorrect certificates
$C \leftarrow$ minimum vertex cover of $G[A, \overline{A}]$
for all $Q \subseteq R \subseteq C$ do
    $R_a \leftarrow R \cap A$
    $R_{\overline{a}} \leftarrow R \cap \overline{A}$
    mark one certificate $G_i \in S$ minimizing $|E_i| + |E(G_i)|$ where:
        $V(G_i) \cap C = R_a$,
        $C \setminus (R_{\overline{a}} \cup A) \subseteq N(V(G_i))$, and
        $E_i \leftarrow$ a minimum subset of $E(G[V(G_i), R_{\overline{a}}])$ spanning $(I(G_i) \cup R_{\overline{a}}) \setminus Q$
    (if no such $G_i$ exists, then don't mark any certificate)
return the set of all the marked certificates in $S$
```

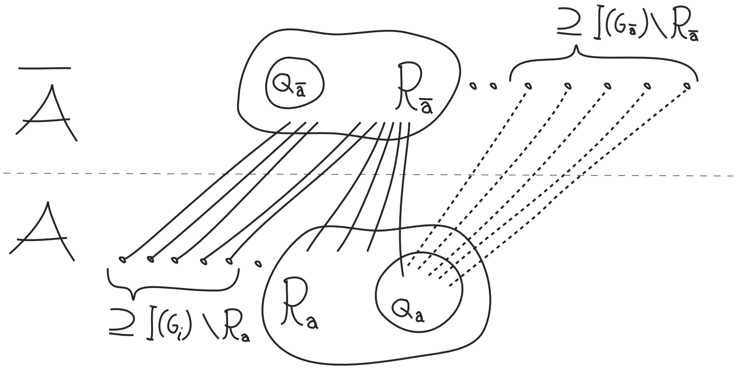

$\supseteq I(G_{\overline{a}}) \setminus R_{\overline{a}}$
$\overline{A}$
$Q_{\overline{a}}$
$R_{\overline{a}}$
A
$\supseteq I(G_i) \setminus R_a$
R_a
Q_a

Figure 5.2: As described in the proof of Lemma 5.21. The dotted lines constitute the set E_I.

Lemma 5.21. *Procedure* TRIM$_{\text{EDS}}$-NON-SPLIT *produces a set $S' \preceq_A S$ of size at most $3^{\text{mm}(A)}$ in time $\mathcal{O}^*(3^{\text{mm}(A)}|S|)$.*

Proof. The size of S' is apparent from the fact that we iterate through all sets $Q \subseteq R \subseteq C$ (at most $3^{\text{mm}(A)}$ triples), and mark one certificate in S to later be put in the output set. When deciding which certificate to mark, we possibly search through the entire set S, and do a polynomial amount of work on each certificate in S to check if it is the certificate that should be marked in this iteration. This gives a total runtime of $\mathcal{O}^*(3^{\text{mm}(A)}|S|)$.

By the definition of the $\preceq_A$-relation, we have $S' \preceq_A S$ if for every important certificate $G_a \in S$ and $G_{\overline{a}} \in \text{cert}(\overline{A})$ so that $\oplus(G_a, G_{\overline{a}})$ contains a witness, the set $\oplus(S', \{G_{\overline{a}}\})$ also contains a witness.

Let $G_w = G_a + G_{\overline{a}} + E_w$ be a witness of $\oplus(G_a, G_{\overline{a}})$. We show that the algorithm will mark a certificate G_i so that the set $\oplus(G_i, G_{\overline{a}})$ also contains a witness, thereby establishing the lemma. By our definition of $\oplus$, the set E_w is a subset of edges from $E(G[V(G_a),V(G_{\overline{a}})])$. Let R' be the vertices of $V(G_w)\cap C$ and let E_I be the edges of E_w where one endpoint is in $C\cap A$ and the other endpoint is in $\overline{A}\setminus C$. Note that E_I must span the set $I(G_{\overline{a}})\setminus R'$. We let Q_a be the set of vertices in $R'\cap A$ that are incident with edges of E_I, and let $Q_{\overline{a}}$ be the vertices of $R'\cap\overline{A}$ not incident with any edge in E_w (see Figure 5.2).

At the iteration of the algorithm where $Q=Q_a\cup Q_{\overline{a}}$ and $R=R'$, the algorithm marks a certificate G_i to be put into S'. This means for the certificate G_i there is a set $E_i\subseteq E(G[V(G_i),R_{\overline{a}}])$ spanning $I(G_i)\setminus Q_a$ and $R_{\overline{a}}\setminus Q_{\overline{a}}$. Furthermore, we have $|E_i|+|E(G_i)|\leq|E_w\setminus E_I|+|E(G_a)|$, as otherwise the algorithm would prefer marking G_a over marking G_i.

We will now show that the certificate $G'_w=G_i+G_{\overline{a}}+E_I+E_i\in\oplus(G_i,G_{\overline{a}})$ is a witness. To do this, it will be enough to show the following three things: (1) $V(G'_w)$ is a vertex cover of the original graph G, (2) $|E(G'_w)|\leq|E(G_w)|$, and (3) $I(G'_w)$ is empty.

The first point, that $V(G'_w)$ is a vertex cover, we get from the fact that G'_w is a vertex cover of $G[A,\overline{A}]$ and that as both G_i and $G_{\overline{a}}$ are locally correct $V(G_i)$ and $V(G_{\overline{a}})$ are vertex covers of $G[A]$ and $G[\overline{A}]$, respectively. So, G'_w is a vertex cover of $G[A]+G[\overline{A}]+G[A,\overline{A}]=G$.

The second point, $|E(G'_w)|\leq|E(G_w)|$, holds by the following inequality.

$$\begin{aligned}|E(G'_w)|&=|E(G_i)|+|E_i|+|E(G_{\overline{a}})|+|E_I|\\&\leq|E(G_a)|+|E_w\setminus E_I|+|E(G_{\overline{a}})|+|E_I|=|E(G_w)|\ .\end{aligned}$$

What remains to prove is the third point, that $I(G'_w)$ is empty. To do this, we will show that each vertex in $I(G_i)$ and $I(G_{\overline{a}})$ is spanned by the edges E_I+E_i. We defined the vertices of $Q_{\overline{a}}$ to not be incident with any edges of E_w, and we know $I(G_w)$ is empty, so we can conclude that $Q\cap I(G_{\overline{a}})$ is empty also. This means that as E_i spans the set $R\cap\overline{A}\setminus Q$, in fact E_i spans all the vertices of $I(G_{\overline{a}})\cap R$. We defined E_I to be all the edges of E_w with endpoints in $\overline{A}\setminus R$, and E_w spans $I(G_{\overline{a}})$, so E_I must span $I(G_{\overline{a}})\setminus R$. From this we conclude that $I(G'_w)\setminus A$ is empty. Now we must show that also $I(G'_w)\cap A$ is empty. We know E_i spans all the vertices of $I(G'_w)\cap A\setminus Q_a$ by definition, and the set Q_a is exactly the set of vertices in A that are incident with edges of E_I. So, $E_I\cup E_i$ also spans $I(G'_w)\cap A$, and hence $I(G'_w)=\emptyset$. □

Next, we give the following lemma, which will be used directly in the join operation of our algorithm solving t-Edge Dominating Set. It will be helpful in reducing the number of certificates needed to ensure we have a set preserving $\oplus(G_1,G_2)$ for given certificates G_1 and G_2. The idea behind this lemma is that for any witness $G_1+G_2+G_3+E_w$, we can substitute the set E_w with another set E'_w spanning all the isolated vertices of G_1,G_2 and G_3 to produce a new witness, as long as $|E'_w|\leq|E_w|$. This means that a lot of different certificates $G_1+G_2+E_t\in\oplus(G_1,G_2)$ will generate a witness when combined with the same certificate G_3 as long as there is a set E'_t so that E'_t+E_t spans the isolated vertices of G_1,G_2 and G_3. Having this in mind, we are able to limit the number of certificates needed to preserve $\oplus(G_1,G_2)$ greatly.

Lemma 5.22. *For any disjoint sets $A_1, A_2 \subseteq V(G)$ and certificates $G_1 \in \text{cert}(A_1)$ and $G_2 \in \text{cert}(A_2)$, we can in $\mathcal{O}^*(2^{\text{sm}(A_1)} + 2^{\text{sm}(A_2)})$-time compute two set families $\mathcal{F}(G_1, G_2) \subseteq 2^{V(G_1)}$ and $\mathcal{F}(G_2, G_1) \subseteq 2^{V(G_2)}$ where the following holds:*

1. *$|\mathcal{F}(G_1, G_2)| \leq \max\left\{2^{\text{sm}(A_1)}, n\right\}$ and $|\mathcal{F}(G_2, G_1)| \leq \max\left\{2^{\text{sm}(A_2)}, n\right\}$, and*
2. *there is a set $S \subseteq \oplus(G_1, G_2)$ preserving $\oplus(G_1, G_2)$, where for each certificate $G_1 + G_2 + E_s \in S$ the set E_s has $V(E_s) \cap A_1 \in \mathcal{F}(G_1, G_2)$ and $V(E_s) \cap A_2 \in \mathcal{F}(G_2, G_1)$.*

Proof. Let $A_3 = \overline{A_1 \cup A_2}$. We will give a construction of the set families $\mathcal{F}(G_1, G_2)$ and show that for any certificate $G_z = G_1 + G_2 + E_z$ in $\oplus(G_1, G_2)$, there is a certificate $G'_z = G_1 + G_2 + E'_z$ in $\oplus(G_1, G_2)$ so that $G'_z \preceq_{\overline{A_3}} G_z$ and that $V(E'_z) \cap A_2 = V(E_z) \cap A_2$ and $V(E'_z) \cap A_1 \in \mathcal{F}(G_1, G_2)$. By similar construction and argument for $\mathcal{F}(G_2, G_2)$, we can conclude that constraint (*2.*) holds for the two constructed set families. That the two set families can be constructed within the proposed time bound and that the size of the sets are as stated in constraint (*1.*) is evident from how we are going to construct them.

Suppose for certificate $G_z = G_1 + G_2 + E_z \in \oplus(G_1, G_2)$ there is some certificate $G_3 \in \text{cert}(A_3)$ so that we have a witness $G_w = G_1 + G_2 + G_3 + E_w$ in $\oplus(G_3, G_z)$. Let C be a vertex cover of $G[A_1, \overline{A_1}]$. We are particularly interested in the following three parts of E_w incident with A_1 (see Figure 5.3).

- E_1: the subset of edges that go from $A_1 \cap C$ to A_2,
- E_2: the subset of edges between $A_1 \setminus C$ and $A_2 \cap C$, and
- E_3: the edges that go from $A_1 \setminus C$ to $A_3 \cap C$.

We denote by R_1, R_2 and R_3 the endpoints $A_1 \cap V(E_1)$, $A_2 \cap V(E_2)$, and $A_3 \cap V(E_3)$, respectively.

As G_w is a witness, the edges in E_w must span all the isolated vertices of G_1, G_2, and G_3. We notice that the edges $E(G_w) \setminus (E_2 \cup E_3)$ span all the vertices of $V(G_w)$ except possibly some vertices in $I(G_1) \cup R_2 \cup R_3$. Therefore, for any subset $E' \subseteq E(I(G_1) \setminus R_1,\ R_2 \cup R_3)$ spanning $(I(G_1) \setminus R_1) \cup R_2 \cup R_3$, it will be the case that $E(G_w) + E' - E_2 - E_3$ spans $V(G_w)$. Furthermore, if $|E'| \leq |E_2 \cup E_3|$, then the certificate $G'_z = G_w - E_2 - E_3 + E'$ will be a witness. To ensure that constraint (*2.*) is satisfied, it thus suffice to ensure that the vertices in A_1 adjacent to vertices in A_2 in G'_z constitute a set in $\mathcal{F}(G_1, G_2)$. For G'_z, these vertices are exactly $R_1 \cup (V(E') \cap A_1)$. What we notice is that E' only depended on G_1, R_1, R_2 and R_3. So, if for each choice of R_1, R_2, and R_3, we compute the E' as we did above, we will have made a set family satisfying constraint (*2.*) and which can be computed within a polynomial factor in n of its size.

What we may now observe, is that since $R_1 \subseteq C \cap A_1$, $R_2 \subseteq C \cap A_2$ and $R_3 \subseteq C \cap A_3$, the different possibilities for R_1, R_2 and R_3 combined is at most $2^{|C|}$. This means the size of the set family is at most $2^{\text{mm}(A_1)}$ and the time to compute it is $\mathcal{O}^*(2^{\text{mm}(A_1)})$.

If $(A_1, \overline{A_1})$ is a split, however, it might be the case that $\text{sm}(A_1) < \text{mm}(A_1)$. But when it is a split, as the neighbourhood of each vertex in $I(G_1)$ is the same, as long as there is a set $S_i \in \mathcal{F}(G_1, G_2)$ of i vertices maximizing $\max\{|S_i \cap I(G_1)|\}$ for each $i \in \{0, \ldots, |V(G_1)|\}$, the set family $\mathcal{F}(G_1, G_2)$ satisfies constraint (*2.*). The size constraint follows from the fact that we only need at most n sets S_i, and clearly we can compute the set family in the runtime stated, as it only takes polynomial amount of time to generate S_i greedily for each i. □

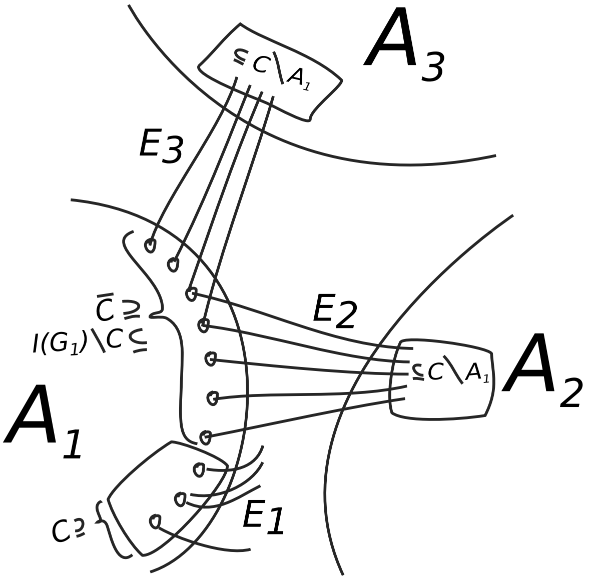

$\subsetneq C \setminus A_1$
A_3
E_3
$\bar{C} \supsetneq$
$I(G_1) \setminus C \subsetneq$
E_2
$\subsetneq C \setminus A_1$
A_2
A_1
$C \supsetneq$
E_1

Figure 5.3: As described in proof of Lemma 5.22. The vertices in each of the three rectangles are subsets of the vertex cover C of A_1, so these sets can be of at most $2^{|C|}$ possibilities.

Procedure JOIN$_{\text{EDS}}$
Input: $S_1 \subseteq \text{cert}(A_1)$ and $S_2 \subseteq \text{cert}(A_2)$
for disjoint sets A_1, A_2 and $A = A_1 \cup A_2$
Output: $S \preceq_A \oplus(S_1, S_2)$

```
$S \leftarrow \emptyset$
for each $G_1 \in S_1$, $G_2 \in S_2$ do
    $\mathcal{V}_1 \leftarrow \mathcal{F}(G_1, G_2)$ from Lemma 5.22
    $\mathcal{V}_2 \leftarrow \mathcal{F}(G_2, G_1)$ from Lemma 5.22
    for each $X_1 \in \mathcal{V}_1$, $X_2 \in \mathcal{V}_2$ do
        $E' \leftarrow$ minimum sized subset of $E(G[X_1, X_2])$ spanning $X_1 \cup X_2$
        add to $S$ the certificate $G_1 + G_2 + E'$
if $(A, \overline{A})$ is a split then return TRIM$_{\text{EDS}}$-SPLIT($S \subseteq \text{cert}(A)$)
else return TRIM$_{\text{EDS}}$-NON-SPLIT($S \subseteq \text{cert}(A)$)
```

Lemma 5.23. *The algorithm* $\textsc{Join}_{\text{EDS}}$ *is correct and runs in time* $\mathcal{O}^*(|S_1||S_2|12^k)$*, producing a set* S *of cardinality* $\mathcal{O}(n^2+3^k)$ *for* $k = \max\{\text{sm}(A_1), \text{sm}(A_2), \text{sm}(A)\}$*.*

Proof. For $G_1 \in \text{cert}(A_1)$ and $G_2 \in \text{cert}(A_2)$, if $G^* = G_1 + G_2 + E^* \in \oplus(G_1, G_2)$ and $G' = G_1 + G_2 + E'$ where $V(E') = V(E^*)$, then $G' \preceq_A G^*$ as long as $|V(E')| \leq |V(E^*)|$. Therefore, the set of certificates S generated in the first part of the algorithm must preserve the set "S" from point 2. in the statement of Lemma 5.22, which in turn implies that $S \preceq_A \oplus(S_1, S_2)$. By Lemma 5.20 and Lemma 5.21 this means the output of the algorithm is a set of size at most $n^2 + 3^{\text{sm}(A)}$ that preserves $\oplus(S_1, S_2)$.

From Lemma 5.22, for each pair of certificates, it takes $\mathcal{O}^*(2^k)$ time to compute $\mathcal{F}(G_1, G_2)$ and $\mathcal{F}(G_2, G_1)$, and for each pair we generate at most $|\mathcal{F}(G_1, G_2)||\mathcal{F}(G_2, G_1)|$ certificates. So, before the call to one of the $\textsc{Trim}_{\text{EDS}}$-procedures, the algorithm uses $\mathcal{O}^*((2^{2k})|S_1||S_2|)$-time and S contains at most $(2^{2k})|S_1||S_2|$ certificates. The total runtime, including the call to the respective $\textsc{Trim}_{\text{EDS}}$-procedure, must then by Lemma 5.20 and Lemma 5.21 be $\mathcal{O}^*(3^k 2^{2k}|S_1||S_2|)$. □

Theorem 5.24. *Given a graph* G *and branch decomposition* (T, δ) *of sm-width* k*, we can solve* Edge Dominating Set *in time* $\mathcal{O}^*(3^{5k})$*.*

Proof. In Lemma 5.23 we showed that the procedure $\textsc{Join}_{\text{EDS}}$ is correct and runs in time $\mathcal{O}^*(|S_1||S_2|12^k)$, producing a set S of cardinality $\mathcal{O}(n^2+3^k)$. So, using Recursive with $\textsc{Join}_{\text{EDS}}$, we know the size of both of the inputs of $\textsc{Join}_{\text{EDS}}$ is at most the size of its output, i.e., $|S_1|, |S_2| \leq \mathcal{O}^*(3^{5k})$. So, each call to Recursive has runtime at most $\mathcal{O}^*(3^k)$. As there are linearly many calls to Recursive and there is a polynomial time verifier for the certificates Recursive produces, by the definition of $\preceq$, the total runtime is also bounded by $\mathcal{O}^*(3^{5k})$. To solve Edge Dominating Set, we run the t-Edge Dominating Set algorithm for all values of $t \leq n$ and hence this is also solvable in $\mathcal{O}^*(3^{5k})$ time as we exclude polynomials of n. □

With a rough analysis of the runtimes of this section in terms also of n, we get a total runtime of $\mathcal{O}(3^{5k}n^4 + 3^k n^{10})$ for solving Edge Dominating Set.

5.6 Conclusions

We have given FPT algorithms, parameterized by split-matching-width, for part `[P2]` of the structural DP-scheme for four basic problems that cannot be FPT parameterized by clique-width unless FPT=W[1]. In combination with the results on `[B]` of Chapter 4, saying that we in FPT time can construct a branch decomposition of sm-width only a constant factor from the optimal, we have `[P1]` and `[P2]` of the structural DP-scheme, yielding complete FPT algorithms for all of the four problems on any input graph G.

Corollary 5.25. *Given a graph* G *of* $\text{sm-w}(G) = k$*, we can solve* Hamiltonian Cycle*,* MaxCut*, and* Edge Dominating Set *in* $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ *time, and we can solve* Chromatic Number *in* $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ *time.*

In fact, the runtimes of Corollary 5.25 for all the four problems are optimal under the Exponential Time Hypothesis [67, 68][1], in the sense that we cannot change any $\mathcal{O}$ into o.

This concludes the second case study, as we have now looked at `[C]` and in Chapter 4 looked at `[A]` and `[B]`.

[1]The optimality of EDGE DOMINATING SET under ETH follows follows from looking at the NP-hardness reduction from 3-SAT to a restricted version of 3-SAT where each variable occurs at most 3 times [104], and the NP-hardness reduction from the restricted version of 3-SAT where each variable occurs at most 3 times to EDGE DOMINATING SET in bipartite graphs of maximum degree 3 [108]. The resulting EDGE DOMINATING SET instance G resulting of applying these two reductions on 3-SAT formula ϕ has its number of vertices bounded by a constant factor c times the number of variables in ϕ. So, any $2^{\delta n} n^{\mathcal{O}(1)}$ algorithm for EDGE DOMINATING SET implies a $2^{\delta c n} n^{\mathcal{O}(1)}$ algorithm for 3-SAT.

Chapter 6

On the hardness of computing optimal decompositions

We have seen how branch decompositions serve as a general decomposition that is well suited for dynamic programming. This is under the assumption that the f-width of the decomposition, for the corresponding function f is low. In this chapter, we will look at situations where finding a branch decomposition of optimal f-width is intractable. In these cases we should rather find good approximations to the best decomposition, such as what we did for split-matching-width in Chapter 4.

6.1 Hardness of branch decompositions

Some of the most studied, and well known width parameters; treewidth, clique-width, branch-width and rank-width, are all NP-hard to compute [4, 39, 76, 97]. Treewidth, branch-width and rank-width can all be computed in FPT time, whereas it is a long-standing open problem if computing clique-width is FPT or W[1]-hard. In contrast, for the parameters maximum matching-width, split-matching-width, boolean-width and mim-width, no hardness results have been shown. In this chapter, we show that both mim-width and boolean-width are NP-hard to compute and also that mim-width is W[1]-hard. To our knowledge, this is the first width parameter of graphs based on non-linear[1] decompositions that have been shown to be W[1]-hard to compute. As opposed to maximum matching-width and split-matching-width, the cut-functions of both boolean-width and mim-width are known to be NP-hard to compute. This is what allows us to show that both boolean-width and mim-width are NP-hard to compute. We give a reduction from deciding the value of these cut-functions on a given vertex subset to the problem of deciding boolean-width and mim-width, respectively. This reduction is not only applicable to boolean-width and mim-width, but to all width parameters based on branch decompositions over the vertex set of a graph where the cut-function in question satisfies certain constraints. One of these constraints being that the

[1]We say that a decomposition is a linear decomposition if the underlying structure of the decomposition is a linear ordering or a path, as opposed to a tree. An example of a parameter based on a linear decomposition which is W-hard to compute is the *bandwidth* of a graph [15, 37].

value of the cut function should not increase when adding a twin vertex to the graph.

Our reduction preserves the parameter to within a constant factor of the original decision problem. Many parameterized hardness results will therefore also translate to parameterized hardness of the width parameter in question. For instance we get W[1]-hardness of computing mim-width, and that no polynomial time constant factor approximation for mim-width can exist (unless NP=ZPP), because of similar hardness results for computing the mim-width of any particular cut.

The main result of this chapter is Theorem 6.1, which follows from Lemma 6.5 and 6.8 described later. Using this theorem, we are able to show hardness results of computing f-width for any cut-function f from a large class of functions we name $\mathcal{C}$-*satisfying cut-functions*, as long as computing f on a single cut is hard. The graph G_k^A is a specific graph constructible in polynomial time and is described in detail in Section 6.3.

Theorem 6.1. *Given a graph G, a subset $A \subseteq V(G)$, a $\mathcal{C}$-satisfying cut function f and a non-negative number $k \in \mathbb{R}$, the graph $G^A_{\lfloor k \rfloor +1}$ has f-width at most $k + \lfloor k \rfloor + 1$ if and only if $f_G(A)$ is at most k.*

So given a graph G and subset $A \subseteq V(G)$, answering the question "is $f_G(A) \leq k$?" can be done by instead answering the question "for $t = \lfloor k \rfloor + 1$, does G_t^A have f-width at most $k+t$?"

Using Theorem 6.1 in combination with known NP-hardness results for counting the number of Maximal Independent Sets in a bipartite graph $G[A, \overline{A}]$ (equivalent to counting $\mathtt{nec}_1(A)$, by Rabinovich et al. [83]) by Provan and Ball [82], and deciding the size of a maximum induced matching in a bipartite graph $G[A, \overline{A}]$ (equivalent to computing $\operatorname{mim}(A)$) by Cameron [24], we get the following corollary.

Corollary 6.2. *Both deciding the mim-width of a graph, and deciding the boolean-width of a graph is* NP*-hard.*

By Moser and Sikdar [73] finding a maximum induced matching in a bipartite graph is W[1]-hard, which gives the following corollary.

Corollary 6.3. *Deciding the maximum induced matching-width of a graph is* W[1]*-hard.*

By Elbassioni et al. [38], deciding the size of a maximum induced matching in a bipartite graph is not in APX unless NP=ZPP, which gives us the following corollary.

Corollary 6.4. *There is no polynomial time algorithm for approximating the mim-width of a graph to within a constant factor of the optimal, unless* NP=ZPP.

6.2 Terminology

In this section, we give some terminology local to this chapter. The definitions used here will not be used elsewhere in the thesis.

For a grid graph G, we denote by C_i and R_i its i-th column and row, respectively. A subdivided grid graph is a graph resulting from replacing each edge uv in a grid by a vertex v_{uv} with neighbourhood $N_G(v_{uv}) = \{u,v\}$. In this chapter, we refer to the vertices added by this operation as *sub-vertices*. The non sub-vertices we refer to as *cell-vertices*. For a subdivided grid, we denote by C_i and R_i the same set of vertices as C_i and R_i denote in the original grid graph (i.e., cell-vertices). For a set of cell-vertices X, we denote by $\texttt{sub}(X)$ the set of sub-vertices adjacent to exactly two vertices of X. For two sets X_1, X_2 of cell-vertices, we denote by $\texttt{sub}(X_1, X_2)$ the set of sub-vertices with one neighbour in X_1 and one neighbour in X_2. For a column C_i, we call $\texttt{sub}(C_i) \cup C_i$ the *sub-column* of C_i.

6.3 Deciding cut value through graph width

In this section we will show that we can reduce the problem of deciding the value of a cut-function f on a cut to the problem of deciding the f-width of a graph (Theorem 6.1).

The idea of how to achieve such a reduction is that we construct, based on the input graph G and cut (A,B), a new graph consisting of a subdivided grid of known f-width, and attach copies of A to the left-hand side of the grid, and copies of B to the right-hand side of the grid. The grid will enforce the existence of a cut separating A from B in any optimal decomposition, making us able to deduce the value of $f_G(A,B)$.

In order to enforce a cut such as mentioned above, we cannot allow all kinds of cut-functions. In fact, we need our cut-functions to satisfy the following five constraints in order to work. However, these constraints are upheld by cut functions of many known through branch decompositions. If a cut function satisfies the below constraints $\mathcal{C}$, we say that it is a $\mathcal{C}$-*satisfying* cut-function. The constraints $\mathcal{C}$ are as follows, and must hold for any graph G and any set $S \subseteq V(G)$:

1. $f_G(S) = f_G(\overline{S})$ and $f_G(S)$ depends only on the unlabeled graph $G[S,\overline{S}]$.
2. $f_G(S)$ is zero if $G[S,\overline{S}]$ has no edges and at least one otherwise.
3. Removing a vertex $x \in \overline{S}$ from G does not increase $f_G(S)$, and reduces $f_G(S)$ by at most one.
4. If $G[S,\overline{S}]$ is the disjoint union of $G_1 = G[A_1,B_1]$ and $G_2 = G[A_2,B_2]$, then $f_G(S) = f_{G_1}(A_1) + f_{G_2}(A_2)$.
5. If $v \in \overline{S}$ has a twin vertex in $G[S,\overline{S}]$, then $f_G(S) = f_{G-v}(S)$.

On most known width parameters defined using branch decompositions over $V(G)$, all but the last constraint is upheld, as they are natural properties that come as a result of wanting to measure how many objects of a certain kind lie between the two parts of a cut. The last constraint is the only real limitation of the cut-parameters we investigate.

As a result of the four first constraints, any $\mathcal{C}$-satisfying cut function f on $A \subseteq V(G)$ will always have value at least as large as a maximum induced matching

M in $G[A,\overline{A}]$ (i.e. $f(A) \geq \text{mim}(A)$), since removing all vertices other than those in M does not increase the f-value, and we then have $|M|$ disjoint graphs of at least one edge, implying an f-value of at least $|M|$.

Some examples of cut functions that are $\mathcal{C}$-satisfying are the cut functions of the width-parameters mim-width, boolean-width, and rank-width.

To prove Theorem 6.1 we show that given a graph G, a cut A and non-negative integer k, we can in polynomial time construct the graph G_k^A having f-width no more than $k + f_G(A)$ and no less than $\min\{2k, k + f_G(A)\}$. This upper and lower bound is proved by Lemma 6.5 and Lemma 6.8, respectively. Let us now look at how the graph G_k^A is defined.

The graph G_k^A. Given a graph G, a cut $(A, B = \overline{A})$ of G and an integer k, we construct G_k^A as follows. We first start with a subdivided grid G' of height k and width $6k$. Then, for each vertex $a \in A$, we add to G_k^A a set S_a of k vertices, and let $S_A = \bigcup_{a \in A} S_a$. Similarly, for each vertex $b \in B$, we add to G_k^A a k-vertex set S_b and let S_B denote the union $\bigcup_{b \in B} S_b$. Then, for each $a \in A$ we add edges making up a matching between the vertices of S_a and the set C_1 of G' and for each $b \in B$ a matching between S_b and C_{6k}. Now we add edges between the vertices of S_A and S_B in such a manner that the induced subgraph on $S_A \cup S_B$ will be the graph $G[A,B]$ with the addition that each vertex has $k-1$ extra twins. That is, we add the edges $E' = \{uv : u \in S_a, v \in S_b, a \in A, b \in B\}$. So, the vertices of G_k^A are $V(G_k^A) = V(G') \cup S_A \cup S_B$ and the edges are $E(G') \cup E'$ plus a matching from S_a to C_1 for each $a \in A$, and a matching from S_b to C_{6k} for each $b \in B$.

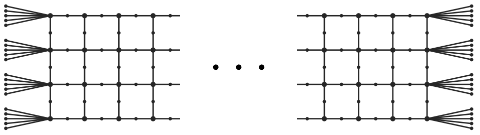


Figure 6.1: The graph G_4^A for some set $A \subset V(G)$ so that $|A| = |\overline{A}| = 5$. Edges between S_A and $S_{\overline{A}}$ are omitted in this figure.

We now show the first part of proving Theorem 6.1, namely upper bounding the f-width of G_k^A.

Lemma 6.5. *Given a graph G and subset $A \subseteq V(G)$, the f-width of G_k^A for a $\mathcal{C}$-satisfying cut function f is at most $f_G(A) + k$, for any non-negative integer k.*

Proof. Let $B = V(G) \setminus A$. The way we construct our decomposition will be to (in an arbitrary way) decompose S_A by itself, S_B by itself, and each C_i, $\texttt{sub}(C_i)$, and $\texttt{sub}(C_i, C_{i+1})$ by themselves, and then in a specific way combine them together by attaching each of the decomposition trees to a distinct node in a path of length $18k-2$ (one node for all but two of the smaller decomposition trees). We attach the decomposition trees to the path by, from left to right, attaching the decompositions of the sets of the subdivided grid in the order $C_1, \texttt{sub}(C_1), \texttt{sub}(C_1, C_2), C_2, \texttt{sub}(C_2), \ldots, C_{6k}$. Then attach the decomposition tree of S_A to the leftmost node of the path (which is also attached to the decomposition tree of C_1), and attach the decomposition tree of S_B to the rightmost

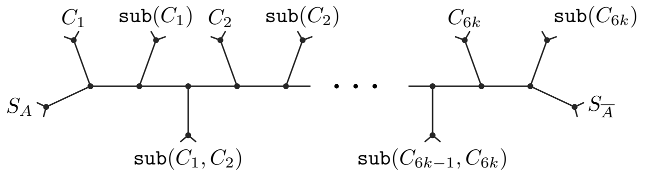


Figure 6.2: A high-level view of the decomposition of width at most $f_G(A)+k$ described in Lemma 6.5.

node of the path (which is also attached to the decomposition tree of C_{6k}). See Figure 6.3.

We now argue that the width of this decomposition is at most $f_G(A)+k$. First notice that since $N(S_A)\setminus C_1 \subseteq S_B$, we have by the constraints in $\mathcal{C}$ that $f(S_A) \leq f(S_A,S_B)+|C_1| = f(S_A,S_B)+k$. Furthermore, as $G[S_A,S_B]$ is isomorphic to the graph $G[A,B]$ with a lot of twins added, by $\mathcal{C}$ we have $f(S_A,S_B) = f_G(A)$. So, $f(S_A) \leq f_G(A)+k$. The same holds for S_B. As $S_A \cap N(S_A) = S_B \cap N(S_B) = \emptyset$, we also have $\forall S \subseteq S_A : f(S) \leq f(S_A)$ and $\forall S \subseteq S_B : f(S) \leq f(S_B)$. So all the cuts induced by the decompositions made for S_A and S_B themselves have low enough f-width. For each of the remaining sets we made decomposed by themselves all have size at most k, so the width is also at most k from the constraints in $\mathcal{C}$.

Now we prove that the cuts induced by the edges of the path we attached all the smaller decompositions to also have low enough f-width. Each such cut (other than $(S_A,\overline{S}_A)$ and $(\overline{S}_B,S_B)$) will be of the form $(S_A\cup Y_1, Y_2\cup S_B)$ where Y_1 and Y_2 are vertices of the subdivided grid. If we look at the set Y_1, the set of vertices $Z \subseteq Y_1$ that have neighbours in Y_2 is either only a set C_i or a set $\texttt{sub}(C_i,C_{i+1})$. Therefore, as each such set has size at most k, by $\mathcal{C}$ we get the inequalities $f(S_A\cup Y_1, Y_2\cup S_B) \leq f(S_A,S_B)+|Z| \leq f_G(A)+k$. This completes the proof, as each cut will have f-width at most $f_G(A)+k$. □

What remain now in order for proving Theorem 6.1, is to lower bound the f-width of G_k^A. To do that, we show that for any branch decompositions, there must be a cut of at least a specific width. However, to prove such a thing we need to know a little bit more about the structure of a branch decomposition. For this purpose we have Proposition 6.6 and Lemma 6.7, which enforces a certain structure to any branch decomposition that could possibly contradict the lower bound we aim for in Lemma 6.8.

Proposition 6.6. *For a subdivided grid G, an integer t and cut (X_1,X_2), if for t distinct sub-columns $\mathcal{C}$ we have that for all C in $\mathcal{C}$ it holds that $C\cap X_1 \neq \emptyset$ and $C\cap X_2 \neq \emptyset$, then* $\text{mim}(X_1,X_2) \geq t$.

Proof. Since (X_1,X_2) is a bipartition of $V(G)$, each vertex in the graph is either in X_1 or X_2. Therefore, as each of the sub-columns in $\mathcal{C}$ contain a vertex in X_1 and a vertex in X_2, there must be a pair of adjacent vertices of each of the t sub-columns so that one is in X_1 and one is in X_2. As no two sub-columns are adjacent to each other, this means we have an induced matching of size t in the bipartite graph $G[X_1,X_2]$ and hence $\text{mim}(X_1,X_2) \geq t$. □

Lemma 6.7. *Let f be a $\mathcal{C}$-satisfying cut function and $\mathcal{D}$ a branch decomposition of a subdivided $h \times w$-grid G. If the f-width of $\mathcal{D}$ is less than $2h$ and $w \geq 6h$, then there exists a cut (X_1, X_2) in $\mathcal{D}$ for which there are four sub-columns $C'_{i_1}, C'_{i_2}, C'_{i_3}, C'_{i_4}$ where $C'_{i_1} \cup C'_{i_2} \subseteq X_1$ and $C'_{i_3} \cup C'_{i_4} \subseteq X_2$.*

Proof. Let C'_i be the i-th sub-column. That is, $C'_i = \mathtt{sub}(C_i) \cup C_i$, and let C' be the union of all C'_i for $1 \leq i \leq w$. We recall from Lemma 4.8 of Chapter 4 that there must be a cut (X_1, X_2) so that $|C' \cap X_1|, |C' \cap X_2| \geq \left\lfloor \frac{1}{3}|C'| \right\rfloor$. In particular, at least $2h$ of the sub-columns contain a vertex of X_1 and at least $2h$ contain a vertex of X_2. In fact, if one sub-column contain a vertex of both X_1 and of X_2, the number of sub-columns containing a vertex of X_1 must be strictly more than $2h$ and the number of sub-columns containing vertices of X_2 strictly more than $2h$. By Poroposition 6.6, the number of sub-grids that contain vertices of both X_1 and X_2 (simultaniously), is less than $2h$, so there will always be at least two sub-columns C'_{i_1} and C'_{i_2} subset of X_1 and sub-columns C'_{i_3} and C'_{i_4} subset of X_2. □

Lemma 6.8. *Given a graph G, a non-negative integer k and subset $A \subseteq V(G)$, for any $\mathcal{C}$-satisfying cut function f we have $f(G^A_k) \geq \min\{2k,\ f_G(A) + k\}$.*

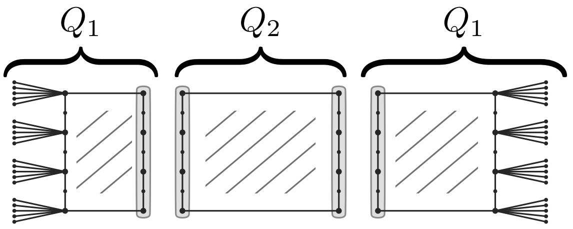


Figure 6.3: The four columns (marked in grey) mentioned in the proof of Lemma 6.8 act as a separator between the parts Q_1 and Q_2 depicted. (Again, edges from S_A to $S_{\overline{A}}$ are omitted from the drawing.)

Proof. Let $t = f_G(A)$ and $B = \overline{A}$, and G' be the subdivided grid of G^A_k. Let C' be the set of all sub-columns $C' = C'_1 \cup C'_2 \cup \ldots C'_{6k}$ of G'. Suppose for contradiction there is a branch decomposition $\mathcal{D}$ of f-width less $2k$ and $f_G(A)$. Then $f(\mathcal{D}) < 2k$, and by Lemma 6.7, there must be a cut (X_1, X_2) in $\mathcal{D}$ so that two sub-columns C'_{l_1}, C'_{l_2} and two sub-columns C'_{r_1}, C'_{r_2} are subsets of X_1 and X_2, respectively. Without loss of generality let $l_1 < l_2, r_1, r_2$ and let $r_1 < r_2$. Since $f(X_1, X_2) < 2k$, we must have $l_2 < r_1$, as otherwise we have two disjoint induced subgrids from column l_1 to r_1 and one between l_2 and r_2, both of width at least k by Proposition 6.6, which by the constraints $\mathcal{C}$ implies a total width of at least $2k$. So we have $l_1 < l_2 < r_1 < r_2$.

Let Z be the vertices between (but not including) C'_{l_1} and C'_{l_2} and between C'_{r_1} and C'_{r_2} in G'. In the graph $G^A_k - Z$ we have at least two connected components (possibly three if $E(G[A, \overline{A}]) = \emptyset$). Let Q_2 be the vertices of the same component as C'_{l_2} in $G^A_k - Z$ and let Q_1 be the vertices of the remaining component(s) ($Q_1 =$

$V(G_k^A) \setminus (Z \cup Q_2))$, as depicted in Figure 6.3. We will now show that the width of G_k^A is at least $k + f(A)$ by first showing that the cut $(X_1 \cap Q_2, X_2 \cap Q_2)$ has f-value at least k and then that $(X_1 \cap Q_1, X_2 \cap Q_1)$ has f-value at least $f_G(A)$.

We notice that the induced subgraph $G_k^A[Q_2]$ is a subdivided grid of height k with one column only containing vertices of X_1 (namely C_{l_2}) and one column only containing vertices only of X_2 (namely C_{r_1}). By regarding columns as rows and rows as columns, we can see that $G_k^A[Q_2]$ is also isomorphic to a subdivided grid of width k where all sub-columns contain at least one vertex from X_1 and one vertex from X_2. By Proposition 6.6, this means the mim-value of $(X_1 \cap Q_2, X_2 \cap Q_2)$ is at least k.

Now we show that $f(X_1 \cap Q_1, X_2 \cap Q_1) \geq f_G(A)$. Notice that if for any $a \in A$ we have $S_a \subseteq X_2$ then there will be k induced paths from $S_a \subseteq X_2$ to $C_{l_1} \subseteq X_1$, implying the mim-value, and hence f-value, of $(X_1 \cap Q_1, X_2 \cap Q_1)$ to be at least k. Likewise, if $f(X_1 \cap Q_1, X_2 \cap Q_1) < k$, then for all $b \in B$ be must have $S_b \not\subseteq X_1$, as otherwise we have k induced paths from $C_{r_2} \subseteq X_2$ to $S_b \subseteq X_1$. As a result, for each $a \in A$ and $b \in B$ there must be a vertex $a' \in S_a \cap X_1$ and $b' \in S_b \cap X_2$. Let A' and B' be the set of one such a' and b' for each $a \in A$ and $b \in B$, respectively. The way we constructed G_k^A, a vertex $a' \in A'$ is adjacent to $b' \in B'$ if and only if a is adjacent to b in G, so $G[A,B]$ is isomporhic to $G_k^A[A' \subseteq X_1, B' \subseteq X_2]$. This, in turn, implies that $f(X_1 \cap Q_1, X_2 \cap Q_1) \geq f_G(A)$ just as we wantet to show.

From this we can conclude that either the width of $\mathcal{D}$ is at least $2k$, or there is a cut in $\mathcal{D}$ of width at least $k + f_G(A)$. □

This completes the part of proving Theorem 6.1, as we now have a bound on the f-width of G_k^A by $k + f_G(A)$ from above and $\min\{2k, k + f(G)\}$ from below. When we know $f_G(A) < k$ we thus have a strict bound of $f_G(A) + k$ for the f-width of G_k^A.

Chapter 7

Faster algorithm for $[\sigma, \rho]$-problems

In this chapter, we investigate a set of parameterized problems already known to be FPT. Namely, the class of $[\sigma, \rho]$-partition problems parameterized by clique-width and parameters of the same modelling power as clique-width. As mentioned in Section 2.3.3, the best known runtimes for these problems is achieved by the use of a general algorithm of Bui-Xuan et al. [23] that requires a branch decomposition given as part of the input. This branch decomposition approximates the optimal rank-width. However, as we see in this chapter, by instead finding a branch decomposition approximating a parameter which is a modification of rank-width, called $\mathbb{Q}$-rank-width we end up with a faster total runtime in terms of clique-width.

Note, however, that if we compare rank-width with $\mathbb{Q}$-rank-width directly, for the points `[A]`, `[B]`, and `[C]`, we see that for the parameterized runtimes of this new algorithm parameterized by $\mathbb{Q}$-rank-width and the former alorithm parameterized by rank-width, neither of them are strictly better than the other. This is because rank-width can be better than $\mathbb{Q}$-rank-width in terms of `[A]`, while $\mathbb{Q}$-rank-width is better than rank-width in terms of `[C]`.

7.1 Introduction

Clique-width is a well-studied parameter in parameterized complexity theory. It is therefore interesting to be able to expand our knowledge on the parameter and to improve on the preciseness of problem complexity when parameterizing by clique-width. For the definition of clique-width, see Section 1.1.

Courcelle, Makowsky, and Rotics [31] showed that, for an input graph of clique-width at most k, every problem expressible in MSO_1 (monadic second-order logic of the first kind) can be solved in FPT time parameterized by k if a k-expression for the graph is given together with the input graph. Later, Oum and Seymour [79] gave an algorithm to find a $(2^{3k+2}-1)$-expression of a graph having clique-width at most k in time $2^{3k}n^{\mathcal{O}(1)}$.[1] By combining these results, we deduce that for an input graph of clique-width at most k, every MSO_1 problem is in FPT, even if a k-expression is not given as an input. However the dependency in k is huge and

[1]Later, Oum [76] obtained an improved algorithm to find a $(2^{3k}-1)$-expression of a graph having clique-width at most k in time $2^{3k}n^{\mathcal{O}(1)}$.

can not be considered of practical interest. In order to increase the practicality of FPT algorithms, it is very important to control the runtime as a function of k.

If we rely on finding an approximate k-expression first and then doing dynamic programming on the obtained k-expression, we have two ways to make improvements; either we improve the algorithm that uses the k-expression, or we find a better approximation for clique-width. Given a k-expression, Independent Set and Dominating Set can be solved in time $2^k n^{\mathcal{O}(1)}$ [52] and $4^k n^{\mathcal{O}(1)}$ [11], respectively. Lokshtanov et al. [67] show that unless the Strong ETH fails, Dominating Set can not be solved in $(3-\epsilon)^k n^{\mathcal{O}(1)}$ time even if a k-expression is given[2]. Hence, there is not much room for improvement in the existing algorithms when a k-expression is given.

There are no known FPT algorithms for computing optimal k-expressions, and the best known FPT algorithm for approximating an optimal k-expression via rank-width has an approximation ratio which is exponential in the optimal clique-width [76]. Therefore, even for the simple NP-hard problems such as Independent Set and Dominating Set, all known algorithms following this procedure have a runtime where the dependency is double exponential in the clique-width. The question of finding a better approximation algorithm for clique-width is an important and challenging open problem.

However, there is a way around this by avoiding a k-expression: Bui-Xuan, Telle and Vatshelle [21] showed that by doing dynamic programming directly on a rank decomposition, Dominating Set can be solved in $2^{k^2} n^{\mathcal{O}(1)}$ for graphs of clique-width k. Their algorithm with a runtime of $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)}$ is not only for Independent Set and Dominating Set but also for a wide range of problems, called the $[\sigma,\rho]$ partition problems. Tables 7.1 and 7.2 list some well known $[\sigma,\rho]$ partition problems.

In this chapter we improve on these results by using a slightly modified definition of rank-width, called *$\mathbb{Q}$-rank-width*, based on the rank function over the rational field instead of the binary field. The idea of using fields other than the binary field for rank-width was investigated earlier in [62], but our work is the first to use $\mathbb{Q}$-rank-width to speed up an algorithm.

We will show the following:

- For any graph, its $\mathbb{Q}$-rank-width is no more than its clique-width.
- There is an algorithm to find a decomposition confirming that $\mathbb{Q}$-rank-width is at most $3k+1$ for graphs of $\mathbb{Q}$-rank-width at most k in time $2^{3k} n^{\mathcal{O}(1)}$.
- If a graph has $\mathbb{Q}$-rank-width at most k, then every fixed $[\sigma,\rho]$ partition problem can be solved in $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$-time.

This allows us to construct an algorithm that runs in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ for graphs of clique-width at most k and solve every fixed $[\sigma,\rho]$ partition problem, improving the previous runtime $2^{\mathcal{O}(k^2)} n^{\mathcal{O}(1)}$ of the algorithm by Bui-Xuan et al. [23].

We also relate the parameter $\mathbb{Q}$-rank-width to other existing parameters. There are several factors affecting the quality of a parameter, such as: Can we compute

[2]Their proof uses pathwidth, but the statement holds since clique-width is at most 1 higher than pathwidth.

$d(\pi)$	Standard name
d	d-Dominating set
$d+1$	Induced d-Regular Subgraph
d	Subgraph of Min Degree $\geq d$
$d+1$	Induced Subgraph of Max Degree $\leq d$
2	Strong Stable set or 2-Packing
2	Perfect Code or Efficient Dominating set
2	Total Nearly Perfect set
2	Weakly Perfect Dominating set
2	Total Perfect Dominating set
2	Induced Matching
2	Dominating Induced Matching
2	Perfect Dominating set
1	Independent set
1	Dominating set
1	Independent Dominating set
1	Total Dominating set

Table 7.1: A table of some vertex subset properties whose optimization problems are $[\sigma,\rho]$ partition problems. The meaning of the problem specific constant $d(\pi)$ is discussed in subsection 7.2.2.

$d(\pi)$	Standard name
1	H-coloring or H-homomorphism
1	H-role assignment or H-locally surjective homomorphism
2	H-covering or H-locally bijective homomorphism
2	H-partial covering or H-locally injective homomorphism

Table 7.2: A table of some homomorphism problems that are also $[\sigma,\rho]$ partition problems for fixed simple graph H. These are expressible with a degree constraint matrix D_q where $q(\pi) = |V(H)|$. The meaning of D_q, $d(\pi)$ and $q(\pi)$ is explained in subsection 7.2.2.

or approximate the parameter? Which problems can we solve in FPT time? Can we reduce the exponential dependency in the parameter for specific problems? And, how large and natural is the class of graphs having a bounded parameter value?

This chapter is organized as follows: In Section 7.2 we introduce the main parts of the framework used by Bui-Xuan et al. [23], including the general algorithm they give for $[\sigma,\rho]$ partition problems. Section 7.3 revolves around $\mathbb{Q}$-rank-width and is where the results of this chapter reside. We show how $\mathbb{Q}$-rank-width relates to clique-width, and reveal why we have a good FPT algorithm for approximating a decomposition. In Section 7.4, we give our main result, which is an improved upper bound on solving $[\sigma,\rho]$ partition problems parameterized by clique-width when we are not given a decomposition. We end the chapter with Section 7.5 containing some concluding remarks and open problems.

7.2 Framework

The algorithm of Bui-Xuan et al. [23] requires a branch decomposition as input, and has a running time depending on the $\mathtt{nec}_d$-width of the decomposition. Here the value of d depends on which of the $[\sigma,\rho]$ partition problems are being solved. The definition of the class of $[\sigma,\rho]$ partition problems is not used directly in this chapter, but for completeness we give the definitions in Subsection 7.2.2. Before this let us recall the definition of $\mathtt{nec}_d$-width.

7.2.1 Neighbourhood Equivalence and $\mathtt{nec}_d$

Two sets of vertices S_1, S_2 are *neighbourhood equivalent* if they have the same set of neighbours, in other words, $N(S_1) = N(S_2)$. We are particularly interested in neighbourhood equivalence in bipartite graphs, or more specifically, cuts defined by a branch decomposition. This concept was generalized with respect to cuts in [23]. We define the *d-neighbour equivalence* relation $\equiv_A^d$, and use this to define the parameter $\mathtt{nec}_d$.

For a cut $\left(A,\overline{A}\right)$ of a graph G, and a positive integer d, two subsets $X, Y \subseteq A$ are *d-neighbour equivalent*, $X \equiv_A^d Y$, over $\left(A,\overline{A}\right)$ if:

$$\text{for each vertex } v \in \overline{A}, \quad \min\{d,\ |N(v) \cap X|\} = \min\{d,\ |N(v) \cap Y|\}.$$

The *number of d-neighbour equivalence classes*, $\mathtt{nec}_d(A)$, is the number of equivalence classes of $\equiv_A^d$ over $\left(A,\overline{A}\right)$.

In other words, $X \equiv_A^d Y$ over the cut $\left(A,\overline{A}\right)$ if each vertex in $\overline{A}$ is either adjacent to at least d vertices in both X and Y, or is adjacent to exactly the same number of vertices in X as in Y. The algorithm in [23] uses this relation to limit the number of partial solutions to try. Therefore, the runtime is dependent on the number of d-neighbour equivalence classes.

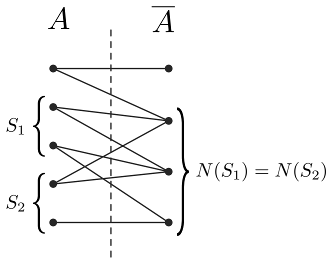


Figure 7.1: The sets S_1 and S_2 are neighbourhood equivalent over $\left(A, \overline{A}\right)$. That is, $S_1 \equiv^1_A S_2$. However, in this example it is not the case that $S_1 \equiv^2_A S_2$.

7.2.2 Locally Checkable Vertex Subset and Vertex Partitioning Problems

Telle and Proskurowski [101] introduced the $[\sigma,\rho]$-problems, also called *Locally Checkable Vertex Subset and Vertex Partitioning problems* (LC-VSP), and D_q-partition problems (see below for the meaning of D_q). This is a framework to describe many well-known graph problems, see [23, 101]. Tables 7.1 and 7.2 list some of them.

For finite or co-finite sets σ and ρ of non-negative integers, a set S of vertices of a graph G is a *$[\sigma,\rho]$-set* of G if for each vertex v of G,

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

The *$[\sigma,\rho]$-problems* are those problems that consist of finding a minimum or maximum $[\sigma,\rho]$-set of the input graph.

The $[\sigma,\rho]$ *partition problems*, or *LC-VSP problems* or *D_q-partition problems*, is a generalization of the $[\sigma,\rho]$-problems. A *degree constraint matrix D_q* is a $q \times q$ matrix such that each cell is a finite or co-finite set of non-negative integers. We say that a partition $V_1, V_2, \dots, V_q$ of $V(G)$ satisfies D_q if for $1 \le i,j \le q$, the number of neighbours in V_j of a vertex of V_i is in the set $D_q[i,j]$. In other words,

$$|N(v) \cap V_j| \in D_q[i,j] \text{ for all } 1 \le i,j \le q \text{ and } v \in V_i.$$

For a given degree constraint matrix D_q, the $[\sigma,\rho]$-partition problem is to decide whether the vertex set of a graph admits a partition satisfying D_q.

For each $[\sigma,\rho]$-partition problem π, there are two problem-specific constants $d(\pi)$ and $q(\pi)$. The number $q(\pi)$ equals the number of parts in a partition that the problem requests, or equivalently, the row/column size of the constraint matrix (i.e., for problem π with degree constraint matrix D_q we have $q = q(\pi)$. The number $d(\pi)$ is defined to be one more than the largest number in all the finite sets and in all the complements of the co-finite sets of the degree constraint matrix used for expressing π. If all the finite sets and complements of co-finite sets are empty, $d(\pi)$ is zero.

For example, Dominating Set can be described by a degree constraint matrix D_2 where $D_2[1,1] = D_2[1,2] = D_2[2,2] = \mathbb{N}$ and $D_2[2,1] = \mathbb{N} \setminus \{0\}$, and we ask to minimize $|V_1|$. If we alter $D_2[1,1]$ to $\{0\}$, the problem is changed to Independent Dominating Set, as no vertex in V_1 can be adjacent to another vertex in V_1. For both problems we have a 2×2-matrix, and so $q(\pi)=2$ and $d(\pi)=0+1=1$.

The algorithm of Bui-Xuan et al. [23] solves each of the $[\sigma,\rho]$ partition problems with a runtime dependent on $\mathtt{nec}_{d(\pi)}$-width and $q(\pi)$ by using the d-neighbour equivalence relation $\equiv_A^{d(\pi)}$.

Theorem 7.1 (Bui-Xuan et al. [23, Theorem 2])**.** *Let π be a $[\sigma,\rho]$ partition problem. For a graph G given with its branch decomposition of $\mathtt{nec}_{d(\pi)}$-width k, the problem π can be solved in time $\mathcal{O}\left(|V(G)|^4 \cdot q(\pi) \cdot k^{3q(\pi)}\right)$.*

7.3 $\mathbb{Q}$-rank-width of a Graph

For a graph G, the $\mathbb{Q}$-*cut-rank* function is a cut function on $V(G)$ that maps $X \subseteq V(G)$ to the rank of an $|X| \times \left|\overline{X}\right|$-matrix $A=(a_{ij})_{i\in X, j\in\overline{X}}$ over the rational field such that $a_{ij}=1$ if i and j are adjacent in G and $a_{ij}=0$ otherwise. We let $\mathrm{cutrk}^{\mathbb{Q}}(X)$ denote the $\mathbb{Q}$-cut-rank of X. For a subset $X\subseteq V(G)$, the matrix A associated with $\mathrm{cutrk}^{\mathbb{Q}}(X)$ is the *adjacency matrix* of the cut $\left(X,\overline{X}\right)$. Note that if the underlying field of the matrix A is the binary field $GF(2)$, then we obtain the definition of the usual cut-rank function [79]. By $\mathbb{Q}$-*rank-width* of a graph G, we mean the $\mathbb{Q}$-cut-rank-width over $V(G)$. We may denote the $\mathbb{Q}$-rank-width simply as $\mathrm{rw}_{\mathbb{Q}}$.

$$\begin{array}{cc} & \overline{A} \\ A & \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Figure 7.2: The adjacency matrix of the cut depicted in Figure 7.1. The rank over the rational field is 4, so the $\mathbb{Q}$-cut-rank of this cut is 4.

Since the $\mathbb{Q}$-cut-rank function is symmetric submodular and is computable in polynomial time, by applying Theorem 3.1 of Oum and Seymour, we get the following theorem.

Theorem 7.2 (Oum and Seymour [79])**.** *There is a $2^{3k}n^{\mathcal{O}(1)}$-time algorithm for which, given a graph G as input and a parameter k, either outputs a branch decomposition for G of $\mathbb{Q}$-rank-width at most $3k+1$ or confirms that $\mathbb{Q}$-rank-width of G is more than k.*

7.3.1 $\mathbb{Q}$-rank-width versus clique-width/rank-width

The question of how useful the $\mathbb{Q}$-rank-width is as a width parameter is hard to answer. To better understand this question, it would be interesting to know the relation to other well-known width parameters such as treewidth, rank-width and clique-width.

The following relates $\mathbb{Q}$-rank-width to the closely related parameter rank-width, yet we see that rank-width can be substantially lower than $\mathbb{Q}$-rank-width.

Lemma 7.3. *For any graph G we have* $\mathrm{rw}(G) \leq \mathrm{rw}_{\mathbb{Q}}(G) \leq \mathrm{cw}(G) \leq 2^{\mathrm{rw}(G)+1} - 1$.

Proof. The first inequality is from the fact that a set of 0-1 vectors linearly dependent over $\mathbb{Q}$ must also be linearly dependent over $GF(2)$.

The second and third inequalities follow from [79, Proposition 6.3] since their proof is not dependent on the type of field rank-width uses. They show that a k-expression can be translated to a branch decomposition where for every cut $\left(A, \overline{A}\right)$ in the decomposition, either the number of distinct rows or the number of distinct columns in the adjacency matrix M of its induced bipartite graph, is bounded by k. Since this means the rank of M over $\mathbb{Q}$ is at most k, we have $\mathrm{rw}_{\mathbb{Q}}(G) \leq \mathrm{cw}(G)$. The idea of showing $\mathrm{cw}(G) \leq 2^{\mathrm{rw}(G)+1} - 1$, is that a branch decomposition where the adjacency matrix of each cut has its number of distinct columns/rows (approximately) bounded by some k, can be translated to a k-expression. As the number of distinct columns/rows for any 0-1 matrix of rank rw is at most 2^{rw}, we get our inequality. The last two inequalities are also proved in [62]. □

We believe Lemma 7.3 is tight. There are existing results showing that it is almost tight. A $n \times n$ grid has rank-width $n-1$ [59] and clique-width $n+1$ [51], hence the first two inequalities are almost tight. There exist graphs with treewidth k and hence $\mathbb{Q}$-rank-width at most k and clique-width at least $2^{\lfloor k/2 \rfloor - 1}$ [28].

7.3.2 $\mathbb{Q}$-rank-width versus treewidth/mm-width

Oum [77] proved that the rank-width of a graph is less than or equal to its treewidth plus 1. We prove a similar result for $\mathbb{Q}$-rank-width, using maximum matching-width as an intermediate parameter between treewidth and $\mathbb{Q}$-rank-width that lets us focus only on the relation between two cut functions instead of entire decompositions. Recall Theorem 2.1 from Chapter 3 by Vatshelle [107] stating that for any graph G, its optimal mm-width is at most $\mathrm{tw}(G)+1$. Based on this, we will prove that also the optimal $\mathbb{Q}$-rank-width is at most $\mathrm{tw}(G)+1$, by showing that for any $X \subseteq V(G)$ the $\mathrm{cutrk}^{\mathbb{Q}}(X)$ is upper bounded by $\mathrm{mm}(X)$.

Theorem 7.4. *Let G be a graph, then* $\mathrm{rw}_{\mathbb{Q}}(G) \leq \text{mm-w}(G) \leq \mathrm{tw}(G) + 1$.

Proof. The last inequality we know from Theorem 2.1, and the first inequality follows by the exact same arguments as the proof of Proposition 4.2, since that proof did not rely on the underlying field. □

Figure 7.3 shows a comparison diagram of graph parameters. The idea of such a diagram is that parameterized complexity results will propagate up and down in this diagram. Positive results propagate upward; for instance, since Dominating Set is solvable in $2^{\mathcal{O}(tw)} n^{\mathcal{O}(1)}$ for a graph of treewidth tw [100], we see that Dominating Set is solvable in $2^{\mathcal{O}(pw)} n^{\mathcal{O}(1)}$ for a graph of pathwidth pw.

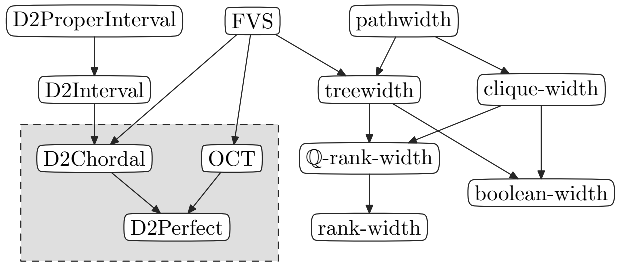


Figure 7.3: A comparison diagram of some graph parameters. A parameter κ_1 is drawn below a parameter κ_2 if there is a constant c such that $\kappa_1(G) \leq c \cdot \kappa_2(G)$ for all graphs G (this does not mean the modelling power of κ_1 and κ_2 are different from each other). The abbreviations are: FVS = Feedback Vertex Set number, OCT = Odd Cycle Transversal number, D2Π = Vertex Deletion distance to a member of Π. For the parameters outside of the shaded region, all the $[\sigma,\rho]$ partition problems are in FPT, and unless $\mathsf{P} = \mathsf{NP}$ for each of the parameters inside the shaded region, at least one of the $[\sigma,\rho]$ partition problems is not in FPT.

Negative results propagate downward; for example, since unless ETH fails, Dominating Set can not be solved in $2^{o(pw)}n^{\mathcal{O}(1)}$ where pw is the pathwidth of the input graph [68], so is the case for treewidth, clique-width, ℚ-rank-width, rank-width and boolean-width. From this table, we can deduce that the entire class of $[\sigma,\rho]$ partition problems cannot be in FPT parameterized by OCT, D2Chordal or D2Perfect unless $\mathsf{P} = \mathsf{NP}$, since Dominating Set is NP-hard for both bipartite [7] and chordal graphs [17]. Furthermore, the class of $[\sigma,\rho]$ partition problems parameterized by either of the remaining parameters is in fact in FPT, since we have FPT algorithms solving each $[\sigma,\rho]$ partition problem parameterized by rank-width, boolean-width, and D2Interval[3].

7.4 Bounding $\mathtt{nec}_d$-width by ℚ-rank-width and its Algorithmic Consequences

Now we know how to find a branch decomposition with a low ℚ-rank-width. We are going to discuss its $\mathtt{nec}_d$-width to apply Theorem 7.1. Theorem 7.1 provides the runtime of the algorithm in terms of the $\mathtt{nec}_d$-width of the given decomposition. So, if we manage to give a bound on the $\mathtt{nec}_d$-width of a decomposition in terms of the ℚ-rank-width, we will also get a bound on the runtime of the algorithm in terms of ℚ-rank-width. We will prove such a bound shortly, but in order to do

[3]Given a graph of D2Interval$(G) = k$, we have a fixed-parameter tractable algorithm to construct a branch deceomposition of $\mathtt{nec}_d$-width at most $2^k n^d$ and thus by the algorithm of [23], we have a fixed-parameter tractable algorithm to solve the problem. To do this, we first find a vertex set S of size k so that $G - S$ is an interval graph, and then find a branch decomposition of $\mathtt{nec}_d$-width at most n^d. Arbitrarily adding the vertices of S anywhere in the branch decomposition cannot increase the $\mathtt{nec}_d$-width by more than $2^{|S|}$, and thus the resulting branch decomposition has $\mathtt{nec}_d$-width at most $2^k n^d$. We have a fixed-parameter tractable algorithm to find S shown in [25] and constructing a branch decomposition for $G - S$ of $\mathtt{nec}_d$-width at most n^d can be done in polynomial time by [6].

this we first need the following lemma, based on a proof of Belmonte and Vatshelle [6, Lemma 1].

Lemma 7.5. *Given a positive integer d and a cut $\left(A,\overline{A}\right)$ of $\mathbb{Q}$-cut-rank k, for every subset $S \subseteq A$, there exists a subset $R \subseteq S$ so that $|R| \leq dk$ and $R \equiv_A^d S$ over the cut.*

Proof. We proceed by induction on d. If $d = 1$, then let S' be a minimal subset of S so that $S' \equiv_A^1 S$. Since S' is minimal, removing any vertex of S' will decrease $|N(S')|$. Therefore, every vertex of S' is adjacent to at least one vertex that none of the other vertices in S' are adjacent to. In the adjacency matrix M of $\left(A,\overline{A}\right)$, this means that each of the corresponding rows of S' has a 1 in a column where all the other rows of S' has a 0. Hence, the rows of S' are linearly independent and so $|S'| \leq \operatorname{cutrk}^{\mathbb{Q}}(A) = k$.

So we may assume that $d > 1$. By the above, there exists a subset $S_1 \subseteq S$ such that $|S_1| \leq k$ and $S_1 \equiv_A^1 S$. By the induction hypothesis, there exists a set $S_2 \subseteq (S \setminus S_1)$ so that $S_2 \equiv_A^{d-1} (S \setminus S_1)$ and $|S_2| \leq (d-1)k$.

We claim that $S_1 \cup S_2 \equiv_A^d S$. Let $v \in \overline{A}$. We may assume that v has at most $d-1$ neighbours in $S_1 \cup S_2$.

If v has a neighbour in $S \setminus (S_1 \cup S_2)$, then $|N(v) \cap (S \setminus S_1)| > |N(v) \cap S_2|$ and therefore v has at least $d-1$ neighbours in S_2 and so v has no neighbors in S_1. This contradicts our assumption that $S_1 \equiv_A^1 S$.

Thus v has no neighbour in $S \setminus (S_1 \cup S_2)$. This proves the claim. Since $|S_1 \cup S_2| \leq dk$, this completes the proof of the lemma. □

Lemma 7.5 implies that to count distinct d-neighbour equivalence classes for a cut of a branch decomposition of $\mathbb{Q}$-rank-width k, it is enough to search subsets of size at most dk. The same result is true, even if we replace $\mathbb{Q}$-rank-width with rank-width or boolean-width ([107], [23, Lemma 5]).

Then what is the contribution of $\mathbb{Q}$-rank-width instead of rank-width or boolean-width? Here comes the crucial difference. For both rank-width k or boolean-width k, the number of vertices with distinct neighbourhoods over the cut is no more than 2^k [23, 107]. Putting this together gives a trivial bound of $\mathtt{nec}_d \leq 2^{dk^2}$. We can improve this bound if k is $\mathbb{Q}$-rank-width, thanks to the fact that the row space of a matrix over $\mathbb{Q}$ not only contains all the rows of the matrix, but also all the different sums of the rows in the matrix. So, we can bound $\mathtt{nec}_d(A)$ by using a more direct connection between $\mathbb{Q}$-rank-width and the number of distinct d-neighbourhoods than that of the trivial bound.

Theorem 7.6. *If the $\mathbb{Q}$-rank-width of a branch decomposition is k, then the $\mathtt{nec}_d$-width of the same decomposition is no more than $(dk+1)^k = 2^{k\log_2(dk+1)}$.*

Proof. It is enough to prove that if a cut $\left(A,\overline{A}\right)$ has $\mathbb{Q}$-cut-rank k, then $\mathtt{nec}_d(A) \leq (dk+1)^k$. Let M be the $A \times \overline{A}$ adjacency matrix of the cut $\left(A,\overline{A}\right)$ over $\mathbb{Q}$.

For a subset S of A, let $\sigma(S)$ be the sum of the row vectors of M corresponding to S. If $\sigma(S) = \sigma(S')$ then $S \equiv_A^d S'$ for all d, because the entries of $\sigma(S)$ represent the number of neighbours in S for each vertex in $\overline{A}$.

By Lemma 7.5, each equivalence class of $\equiv_A^d$ can be represented by a subset S of A having at most dk vertices. Notice that for such S, each entry of $\sigma(S)$ is in $\{0,1,2,\dots,dk\}$.

Let B be a set of k linearly independent columns of M. Since M has rank k, every linear combination of row vectors of M is completely determined by its entries in B. Thus the number of possible values of $\sigma(S)$ is at most $(dk+1)^k$ (see Figure 7.4). This proves the theorem. $\square$

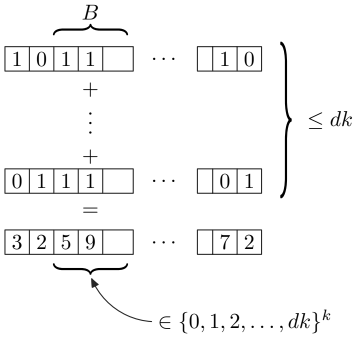


Figure 7.4: As described by Theorem 7.6, we can determine the sum of the vectors by looking at the values in the columns B. As we sum over at most dk rows, and each row either increases the value of a column by exactly one or exactly zero, the number of unique sums possible is at most $|\{0,1,\dots\}|^{|B|} = (1+dk)^k$.

This result, combined with Theorems 7.1 and 7.2, shows that all the $[\sigma,\rho]$ partition problems can be solved in time $2^{\mathcal{O}(k\log k)}n^{\mathcal{O}(1)}$. Expressing the runtime in terms of clique-width, we get the following corollary.

Corollary 7.7. *Every $[\sigma,\rho]$ partition problem π on n-vertex graphs of clique-width* cw *can be solved in $2^{\mathcal{O}(\mathrm{cw}\log(\mathrm{cw}\cdot d(\pi))q(\pi))}n^{\mathcal{O}(1)}$-time.*

Proof. Let k be the $\mathbb{Q}$-rank-width of G. By Theorem 7.2 we can find a branch decomposition of $\mathbb{Q}$-rank-width at most $3k+1$ in time $2^{3k}n^{\mathcal{O}(1)}$. By Theorems 7.1 and 7.6, the $[\sigma,\rho]$ partition problem π can be solved in time $2^{9k\log(3k\cdot d(\pi)+1)q(\pi)}n^{\mathcal{O}(1)}$. This completes the proof because $k \leq \mathrm{cw}$ by Lemma 7.3. $\square$

7.5 Conclusion

If we are given a k-expression as input, the best known FPT algorithm parameterized by k solving the Dominating Set is by Bodlaender et al. [11] and runs in time $4^k n^{\mathcal{O}(1)}$. However, it is currently open whether we can construct a $\mathcal{O}(k)$-expression of an input graph of clique-width at most k in polynomial time. We have shown the existence of algorithms with runtime $2^{\mathcal{O}(\mathrm{cw}\log\mathrm{cw})}n^{\mathcal{O}(1)}$ for all $[\sigma,\rho]$ partition problems, without assuming that a k-expression is given as an input. This still leaves the natural open question:

Can Independent Set or Dominating Set be solved in $2^{\mathcal{O}(\mathrm{cw})}n^{\mathcal{O}(1)}$ time, where cw is the clique-width of the graph?

We know that for a graph of treewidth tw, Independent Set can be solved in time $2^{\mathcal{O}(tw)}n^{\mathcal{O}(1)}$ time. This leads us to an interesting question of what parameters give a linearly single exponential runtime for Independent Set. Two such parameters are the Vertex Deletion Distance to Proper Interval graphs (D2ProperInterval) and the Odd Cycle Transversal number (OCT number):

1. For a graph G, the D2ProperInterval of a graph is the minimum number of vertices needed to be removed in order to make G into a proper interval graph. For a graph G with D2ProperInterval equal k, Villanger and van 't Hof [106] gave a $6^k n^{\mathcal{O}(1)}$-time algorithm for finding such a set S to be removed. To solve Independent Set on a graph $G = (V,E)$, we guess the intersection S' of S and an optimal solution, and then combining it with the optimal solution of Independent Set on the proper interval graph $G-(S \cup N(S'))$. As Independent Set is solvable in $n^{\mathcal{O}(1)}$ time on proper interval graphs, this yields a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm.

2. The OCT number of a graph G is the minimum number of vertices needed to remove from G in order to make it bipartite. For a graph G with OCT number equal k, Lokshtanov, Saurabh and Sikdar [69] gave a $3^k n^{\mathcal{O}(1)}$-time algorithm for finding the minimum set S of vertices to remove from G to make it bipartite. As with the algorithm above, we can solve Independent Set by guessing the intersection S' of S and the optimal solution and then combine it with the optimal solution of the bipartite graph $G-(S \cup N(S'))$. As Independent Set is trivially solvable in $n^{\mathcal{O}(1)}$ time on bipartite graphs, this yields a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm.

Note, however, that these parameters are not bounded by treewidth (and thus also not bounded by clique-width), see Figure 7.3.

Chapter 8

Solving #SAT and MaxSAT by dynamic programming

In this chapter we look at dynamic programming algorithms for propositional model counting, also called #SAT, and MaxSAT. Tools from graph structure theory, in particular treewidth, have been used to successfully identify tractable cases in many subfields of AI, including SAT, Constraint Satisfaction Problems (CSP), Bayesian reasoning, and planning. We attack #SAT and MaxSAT using branch decompositions.

The decompositions used can be seen as being over the vertex set of the incidence graph $I(F)$ of the CNF formula F. However, our corresponding width measure will depend on more properties of the CNF formula than is captured by $I(F)$, so we rather see it as a branch decomposition over the variables and clauses of F.

As a partition function, we introduce the *projection-satisfiable-value*, which gives rise to a parameter which correlates well, in terms of value, with known tractability results of solving #SAT and MaxSAT through dynamic programming along structural decompositions. In fact, as mentioned in Section 2.3.4, our proposed algorithm extends all previous tractability results achieved using structural decompositions of the incidence graph of the input formula. This is, however, by upper bounding the value of the projection-staisfiable-width through other structural measures, while we do not know a way of computing an optimal decomposition, or even an approximation of an optimal one, in FPT time. Our result is thus an algorithm for `[P2]` in terms of the structural DP-scheme of Chapter 2.

8.1 A lengthier introduction

The propositional satisfiability problem (SAT) is a fundamental problem in computer science and in AI. Many real-world applications such as planning, scheduling, and formal verification can be encoded into SAT and a SAT solver can be used to decide if there exists a solution. To decide how many solutions there are, the propositional model counting problem (#SAT), which finds the number of satisfying assignments, could be useful. If there are no solutions, it may be interesting to know how close we can get to a solution. When the propositional formula is encoded in Conjunctive Normal Form (CNF) this may be solved by the maximum satisfiability problem (MaxSAT), which finds the maximum number of clauses that can be satisfied by some assignment. In this chapter we investigate classes of CNF formulas where these two problems, #SAT and MaxSAT, can be solved in polynomial time. Tools from graph structure theory, in particular treewidth, have been used to successfully identify tractable cases in many subfields of AI, including SAT, Constraint Satisfaction Problems (CSP), Bayesian reasoning, and planning, see for example [5, 35, 40, 96]. In this chapter we attack #SAT and MaxSAT through the use of branch decompositions of the formula. The tractable cases will include formulas whose class of incidence graphs have not only unbounded treewidth but also unbounded clique-width.

Both #SAT and MaxSAT are significantly harder than simply deciding if a satisfying assignment exists. #SAT is #P-hard [49] even when restricted to Horn 2-CNF formulas, and to monotone 2-CNF formulas [91]. MaxSAT is NP-hard even when restricted to Horn 2-CNF formulas [58], and to 2-CNF formulas where each variable appears at most 3 times [85]. Both problems become tractable under certain structural restrictions obtained by bounding width parameters of graphs associated with formulas, see for example [40, 48, 96, 99]. The work we present here is inspired by recent results of Paulusma et al. [81] and Slivovsky and Szeider [98] showing that #SAT is solvable in polynomial time when the incidence graph[1] $I(F)$ of the input formula F has bounded modular treewidth, and more strongly, bounded symmetric clique-width.

These tractability results work by dynamic programming along a decomposition of $I(F)$. Using the structural DP-scheme of Chapter 2, there are thus two steps involved: `[P1]` find a good decomposition, and `[P2]` perform dynamic programming along the decomposition. The goal is to have a fast runtime, and this is usually expressed as a function of some known graph width parameter of the incidence graph $I(F)$ of the formula F, like its treewidth. Step `[P1]` is solved by a known graph algorithm for computing a decomposition of low (tree-)width, while step `[P2]` solves #SAT or MaxSAT by dynamic programming with runtime expressed in terms of the (tree-)width k of the decomposition.

The algorithms we give in this chapter also work by dynamic programming along a decomposition, but in a slightly different framework. Since we are not solving a graph theoretic problem, expressing runtime by a graph theoretic parameter may be a limitation. Therefore, our strategy will be to develop a framework based on the following strategy

(I) consider, for #SAT or MaxSAT, the amount of information needed to combine solutions to subproblems into global solutions, then

(II) define the notion of good decompositions based on a parameter that minimizes this information, and then

(III) design a dynamic programming algorithm along such a decomposition with runtime expressed by this parameter (`[P2]`).

[1] $I(F)$ is the bipartite incidence graph between the clauses of F on the one hand and the variables of F on the other hand. Information about positive or negative occurrences of variables is not encoded in $I(F)$ so sometimes a signed or directed version is used that includes also this information.

Both Paulusma et al. [81] and Slivovsky and Szeider [98] consider two assignments to be equivalent if they satisfy the same set of clauses. When carrying out (I) for #SAT and MaxSAT this led us to the concept of ps-value of a CNF formula. Let us define it and give an intuitive explanation. A subset $\mathcal{C}$ of the clauses of a CNF formula F is called *projection satisfiable* if there is some complete assignment satisfying every clause in $\mathcal{C}$ but not satisfying any clause not in $\mathcal{C}$. The ps-value of F is the number of projection satisfiable subsets of clauses. Let us consider its connection to dynamic programming, which in general applies when an optimal solution can be found by combining optimal solutions to certain subproblems. For #SAT and MaxSAT these subproblems, at least in the cases we consider, take the form of a subformula of F induced by a subset S of clauses and variables, i.e. first remove from F all variables not in S and then remove all clauses not in S. Consider for simplicity the two subproblems F_S and $F_{\overline{S}}$ defined by S and its complement $\overline{S}$. When combining the 'solutions' to F_S and $F_{\overline{S}}$, in order to find solutions to F, it seems clear that we must consider a number of cases at least as big as the ps-values of the two disjoint subformulas 'crossing' between S and $\overline{S}$, i.e. the subformulas obtained by removing from clauses in S the variables of S, and by removing from clauses in $\overline{S}$ the variables of $\overline{S}$. See Figure 8.2 for an example.

We did not find in the literature a study of the ps-value of CNF formulas, so we start by asking for a characterization of formulas having low ps-value. We were led to the concept of the mim-value of $I(F)$, which is the size of a maximum induced matching of $I(F)$, where an induced matching is a subset M of edges with the property that any edge of the graph is incident to at most one edge in M. Note that this value can be much lower than the size of a maximum matching, e.g. any complete bipartite graph has mim-value 1. We show that the ps-value of F is upper bounded by the number of clauses of F raised to the power of the mim-value of $I(F)$, plus 1. For a CNF formula F where $I(F)$ has mim-value 1 the interpretation of this result is straightforward: its clauses can be totally ordered such that for any two clauses $C < C'$ the variables occurring in C are a subset of the variables occurring in C', and this has the implication that the number of subsets of clauses for which some complete assignment satisfies exactly this subset is at most the number of clauses plus 1.

Families of CNF formulas having small ps-value are themselves of algorithmic interest, but in this chapter we continue with part (II) of the above strategy, and focus on how to decompose a CNF formula F based on the concept of ps-value. We will decompose the formula by the use of rooted branch decompositions, but this time not over sets of vertices, but rather over the variables and clauses of F. For a rooted branch decompsition $(T.\delta)$, a node v in T then represents the subset $X = \delta(v)$ of variables and clauses at the leaves of its subtree. Which branch decomposition are good for efficiently solving #SAT and MaxSAT? In accordance with the above discussion under part (I) the answer is that the good branch decomposition are those where all subformulas 'crossing' between X and $\overline{X}$, for some X defined by a node of the tree, have low ps-value. See Figure 8.2 for an example. Using ps-value as our partition-function, we arrive at the definition of the ps-width of a CNF formula F. It is important to note that a formula can have ps-value exponential in formula size while ps-width is polynomial, and that in general the class of formulas of low ps-width is much larger than the class of formulas of low ps-value.

To finish the above strategy, we must carry out part (III) and show how to solve #SAT and MaxSAT by dynamic programming along the branch decomposition of the formula, and express its runtime as a function of the ps-width. This is not complicated, as dynamic programming when everything has been defined properly simply becomes an exercise in brute-force computation of the sufficient and necessary information, but it is technical and quite tedious. It leads to the following theorem.

Theorem 8.4. *Given a formula F over n variables and m clauses, and a branch decomposition (T,δ) of F of* ps*-width k, we solve* MaxSAT*,* #SAT*, and weighted* MaxSAT *in time $\mathcal{O}(k^3 m(m+n))$.*

Thus, given a decomposition having a ps-width k that is *polynomially-bounded* in the number of variables n and clauses m of the formula, we get polynomial-time algorithms. Let us compare our result to the strongest previous result in this direction, namely that of Slivovsky and Szeider [98] for #SAT. Their algorithm takes as input a branch decomposition over the vertex set of $I(F)$, which is the same as the ground set of F, and evaluates its runtime by the cut function they call 'index'. They show that this cut function is closely related to the symmetric clique-width *scw* of the given decomposition, giving runtime $(n+m)^{\mathcal{O}(scw)}$. Considering the clique-width *cw* of the given decomposition the runtime of [98] becomes $(n+m)^{\mathcal{O}(2^{cw})}$ since symmetric clique-width and clique-width is related by the essentially tight inequalities $0.5cw \leq scw \leq 2^{cw}$ [30]. Their algorithm is thus a polynomial-time algorithm if given a decomposition with *constantly bounded scw*. The result of Theorem 8.4 encompasses this, since our Corollary 8.6 ties ps-width to mim-width and Vatshelle [107] shows that mim-width is upper bounded by clique-width, see also [86] for symmetric clique-width, so that a decomposition of $I(F)$ having constantly bounded (symmetric) clique-width also has polynomially bounded ps-width. In this way, given the decomposition assumed as input in [98], the algorithm of Theorem 8.4 will have runtime $\mathcal{O}(m^{3cw}s)$, for *cw* the clique-width of the given decomposition.

In a paper by Brault-Baron et al. [20], appearing after a preliminary presentation of our results [94], it is argued that the framework behind Theorem 8.4 gives a uniform explanation of all tractability results for #SAT in the literature, in particular those using dynamic programming based on structural decompositions of the incidence graph. Brault-Baron et al. [20] also goes beyond this, giving a polynomial-time algorithm, *not* by dynamic programming, to solve #SAT on β-acyclic CNF formulas, being exactly those formulas whose incidence graphs are chordal bipartite. They show that these formulas do *not* have bounded ps-width and that their incidence graphs do *not* have bounded mim-width. See Figure 8.1 which gives an overview of the results in this chapter and in other papers. Using the concept of mim-width of graphs, and the connection between ps-value and mim-value alluded to earlier, we show that a rich class of formulas, including classes of unbounded clique-width, have polynomially bounded ps-width and are thus covered by Theorem 8.4. Firstly, this holds for classes of formulas having incidence graphs that can be represented as intersection graphs of certain objects,

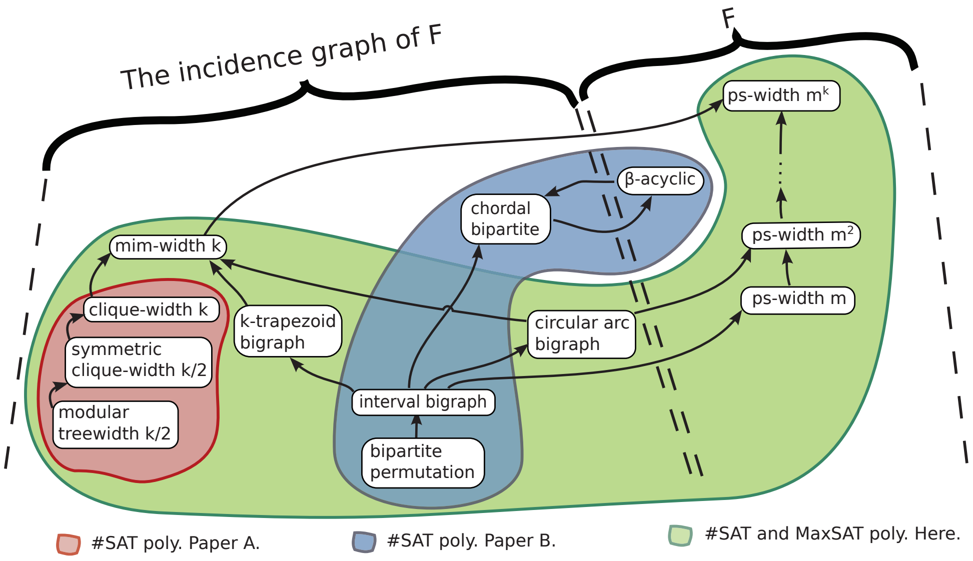


Figure 8.1: We believe, as argued in [20], that any dynamic programming approach working along a structural decomposition to solve #SAT (or MaxSAT) in polynomial time cannot go beyond the green box. Paper A is [98] and Paper B is [20]. On the left of the two dashed lines are 4 classes of graphs with bound $k/2$ or k on some structural graph width parameter, and 5 classes of bipartite graphs. On the right are β-acyclic CNF formulas and 3 classes of CNF formulas with ps-width varying from linear in the number of clauses m, to m^2 and m^k. There is an arc from P to Q if any formula F or incidence graph $I(F)$ having property P also has property Q. This is a Hasse diagram, so lack of an arc in the transitive closure means this relation provably does not hold.

like interval graphs [6]. Secondly, it holds also for the much larger class of bipartite graphs achieved by taking bigraph bipartizations of these intersection graphs, obtained by imposing a bipartition on the vertex set and keeping only edges between the partition classes. Some such bigraph bipartizations have been studied previously, in particular the interval bigraphs. The interval bigraphs contain all bipartite permutation graphs, and these latter graphs have been shown to have unbounded clique-width [19]. See Figure 8.1.

Let us discuss step `[P1]`, finding a good decomposition. Note that Theorem 8.4 assumes that the input formula is given along with a decomposition of some ps-width k. The value k need not be optimal, so any heuristic finding a reasonable branch decomposition could be used in practice. Computing decompositions of optimal ps-width is probably not doable in polynomial-time, but the complexity of this question is not adressed in this chapter. However, we are able to efficiently decide if a CNF formula has a certain linear structure guaranteeing low ps-width. By combining an alternative definition of interval bigraphs [53] with a fast recognition algorithm [74, 84] we arrive at the following. Say that a CNF formula F has an interval ordering if there exists a total ordering of variables and clauses such that for any variable x occurring in clause C, if x appears before C then any variable between them also occurs in C, and if C appears before x then x occurs also in any clause between them.

Theorem 8.11. *Given a CNF formula F over n variables and m clauses each of at most t literals. In time $\mathcal{O}((m+n)mn)$ we can decide if F has an interval ordering (yes iff $I(F)$ is an interval bigraph), and if yes we solve* #SAT *and weighted* MaxSAT *with an additional runtime of* $\mathcal{O}(\min\{m^2, 4^t\}(m+n)m)$.

Formulas with an interval ordering are precisely those whose incidence graphs are interval bigraphs, so Theorem 8.11 encompasses classes of formulas whose incidence graphs have unbounded clique-width.

Could parts of our algorithms be of interest for practical applications? Answering this question is beyond the scope of this chapter and our case study. However, we have performed some limited testing, in particular for formulas with a linear structure, as a simple proof of concept. All our code can be found online [1]. We have designed and implemented a heuristic for step `[P1]` finding a good decomposition (T, δ), in this case a linear one where T is a path with attached leaves. We have also implemented step `[P2]` dynamic programming solving #SAT and MaxSAT along such decompositions. We then run `[P1]` followed by `[P2]` and compare against one of the best MaxSAT solvers from the Max-SAT-2014 event of the SAT-2014 conference and the latest version of the #SAT solver called sharpSAT developed by Marc Thurley [103]. These solvers beat our implementation on most inputs, which is not suprising since our code does not include any techniques beyond our algorithm. Nevertheless, we were able to generate some classes of CNF formulas having interval orderings where our implementation is by far the better. This lends support to our belief that methods related to ps-value warrants further research to investigate if they could be useful in practice.

This chapter is organized as follows. In Section 8.2 we give formal definitions of ps-value and ps-width of a CNF formula and show the central combinatorial lemma linking ps-value of a formula to the size of the maximum induced matching in the incidence graph of the formula. In Section 8.3 we present the dynamic programming algorithm for `[P2]` that, when given a formula and a decomposition, solves #SAT and weighted MaxSAT, proving Theorem 8.4. In Section 8.4 we investigate classes of formulas having decompositions of low ps-width, basically proving the correctness of the hierarchy presented in Figure 8.1. In Section 8.5 we consider formulas having an interval ordering and prove Theorem 8.11. In Section 8.6 we present the results of the implementations and testing. We end in Section 8.7 with some open problems.

8.2 Framework

We consider propositional formulas in Conjunctive Normal Form (CNF). A *literal* is a propositional *variable* or a negated variable, x or $\neg x$, a *clause* is a set of literals, and a *formula* is a multiset of clauses. For a formula F, $\texttt{cla}(F)$ denotes the clauses in F. The incidence graph of a formula F is the bipartite graph $I(F)$ having a vertex for each clause and variable, with variable x adjacent to any clause C in which it occurs. We consider only input formulas where $I(F)$ is connected, as otherwise we would solve our problems on the separate components of $I(F)$.

For a clause C, $\texttt{lit}(C)$ denotes the set of literals in C and $\texttt{var}(C)$ denotes the variables of the literals in $\texttt{lit}(C)$. For a formula F, $\texttt{var}(F)$ denotes the union $\bigcup_{C\in\texttt{cla}(F)}\texttt{var}(C)$. For a set X of variables, an *assignment* of X is a function $\tau : X \to \{0,1\}$. For a literal ℓ, we define $\tau(\ell)$ to be $1-\tau(\texttt{var}(\ell))$ if ℓ is a negated variable ($\ell = \neg x$ for some variable x) and to be $\tau(\texttt{var})$ otherwise ($\ell = x$ for some variable x). A clause C is said to be *satisfied* by an assignment τ if there exists at least one literal $\ell \in \texttt{lit}(C)$ so that $\tau(\ell) = 1$. We notice that this means an empty clause will never be satisfied. A formula is satisfied by an assignment τ if τ satisfies all clauses in $\texttt{cla}(F)$.

The problem #SAT, given a formula F, asks how many distinct assignments of $\texttt{var}(F)$ satisfy F. The optimization problem weighted MaxSAT, given a formula F and weight function $w : \texttt{cla}(F) \to \mathbb{N}$, asks what assignment τ of $\texttt{var}(F)$ maximizes $\sum_C w(C)$ for all $C \in \texttt{cla}(F)$ satisfied by τ. The problem MaxSAT is weighted MaxSAT where all clauses have weight one. For weighted MaxSAT, we assume the sum of all the weights are at most $2^{O(\texttt{cla}(F))}$, and thus we can do summation on the weights in time linear in $\texttt{cla}(F)$.

For a set A, with elements from a universe U we denote by $\overline{A}$ the elements in $U \setminus A$, as the universe is usually given by the context.

8.2.1 Cut of a formula

In this chapter, we will solve MaxSAT and #SAT by the use of dynamic programming. We will be using a divide and conquer technique where we solve the problem on smaller subformulas of the original formula F and then combine the solutions to each of these smaller formulas to form a solution to the entire formula F. Note however, that the solutions found for a subformula will depend on the interaction between the subformula and the remainder of the formula. We use the following notation for subformulas.

For a clause C and set X of variables, by $C|_X$ we denote the clause $\{\ell \in C : \texttt{var}(\ell) \in X\}$. We say $C|_X$ is the clause C *induced* by X. Unless otherwise specified, all clauses mentioned in this chapter are from the set $\texttt{cla}(F)$ (e.g., if we write $C|_X \in \texttt{cla}(F')$, we still assume $C \in \texttt{cla}(F)$). For a formula F and subsets $\mathcal{C} \subseteq \texttt{cla}(F)$ and $X \subseteq \texttt{var}(F)$, we say the subformula $F_{\mathcal{C},X}$ of F *induced* by $\mathcal{C}$ and X is the formula consisting of the clauses $\{C_i|_X : C_i \in \mathcal{C}\}$. That is, $F_{\mathcal{C},X}$ is the formula we get by removing all clauses not in $\mathcal{C}$ followed by removing each literal of a variable not in X. For a set $\mathcal{C}$ of clauses, we denote by $\mathcal{C}|_X$ the set $\{C|_X : C \in \mathcal{C}\}$. As with a clause, for an assignment τ over a set X of variables, we say the assignment τ *induced* by $X' \subseteq X$ is the assignment $\tau|_{X'}$ where the domain is restricted to X'.

For a formula F and sets $\mathcal{C} \subseteq \texttt{cla}(F)$, $X \subseteq \texttt{var}(F)$, and $S = \mathcal{C} \cup X$, we call S a *cut* of F and note that it breaks F into four subformulas $F_{\mathcal{C},X}$, $F_{\overline{\mathcal{C}},X}$, $F_{\mathcal{C},\overline{X}}$, and $F_{\overline{\mathcal{C}},\overline{X}}$. See Figure 8.2. One important fact we may observe from this definition is that a clause C in F is satisfied by an assignment τ of $\texttt{var}(F)$, if and only if C (induced by X or $\overline{X}$) is satisfied by τ in at least one of the formulas of any cut of F.

8.2.2 Projection satisfiable sets and ps-value of a formula

For a formula F and assignment τ of some of the variables in $\texttt{var}(F)$, we denote by $\texttt{sat}(F,\tau)$ the inclusion maximal set $\mathcal{C} \subseteq \texttt{cla}(F)$ so that each clause in $\mathcal{C}$ is satisfied by τ. If for a set $\mathcal{C} \subseteq \texttt{cla}(F)$ we have $\texttt{sat}(F,\tau) = \mathcal{C}$ for some τ over all the variables in $\texttt{var}(F)$, then $\mathcal{C}$ is known as a *projection* (see e.g. [63, 98]) and we say $\mathcal{C}$ is *projection satisfiable* in F. We denote by $\texttt{PS}(F)$ the family of all projection satisfiable sets in F. That is,

$$\texttt{PS}(F) = \{\texttt{sat}(F,\tau) : \tau \text{ is an assignment of the entire set } \texttt{var}(F)\}.$$

The cardinality of this set, $|\texttt{PS}(F)|$, is referred to as the ps-value of F.

To get a grasp of the structure of formulas having low ps-value we consider induced matchings in the incidence graph of a formula. The incidence graph of a formula F is the bipartite graph $I(F)$ having a vertex for each clause and variable, with variable x adjacent to any clause C in which it occurs. The following result provides an upper bound on the ps-value of a formula in terms of the maximum size of an induced matching of its incidence graph.

Lemma 8.1. *Let F be a CNF formula with no clause containing more than t literals, and let k be the maximum size of an induced matching in $I(F)$. We then have $|\texttt{PS}(F)| \leq \min\{|\texttt{cla}(F)|^k + 1, 2^{tk}\}$.*

Proof. We first argue that $|\texttt{PS}(F)| \leq |\texttt{cla}(F)|^k + 1$. Let $\mathcal{C} \in \texttt{PS}(F)$ and $\mathcal{C}_f = \texttt{cla}(F) \setminus \mathcal{C}$. Thus, there exists a complete assignment τ such that the clauses not satisfied by τ are $\mathcal{C}_f = \texttt{cla}(F) \setminus \texttt{sat}(F,\tau)$. Since every variable in $\texttt{var}(F)$ appears in some clause of F this means that $\tau|_{\texttt{var}(\mathcal{C}_f)}$ is the unique assignment of the variables in $\texttt{var}(\mathcal{C}_f)$ which do not satisfy any clause of $\mathcal{C}_f$. Let $\mathcal{C}_f' \subseteq \mathcal{C}_f$ be an inclusion minimal set such that $\texttt{var}(\mathcal{C}_f) = \texttt{var}(\mathcal{C}_f')$, hence $\tau|_{\texttt{var}(\mathcal{C}_f)}$ is also the unique assignment of the variables in $\texttt{var}(\mathcal{C}_f)$ which do not satisfy any clause of $\mathcal{C}_f'$. An upper bound on the number of different such minimal $\mathcal{C}_f'$, over all $\mathcal{C} \in \texttt{PS}(F)$, will give an upper bound on $|\texttt{PS}(F)|$. For every $C \in \mathcal{C}_f'$ there is a variable v_C appearing in C and no other clause of $\mathcal{C}_f'$, otherwise $\mathcal{C}_f'$ would not be minimal. Note that we have an induced matching M of $I(F)$ containing all such edges v_C, C. By assumption, the induced matching M can have at most k edges and hence $|\mathcal{C}_f'| \leq k$. It is easy to show by induction on k that there are at most $|\texttt{cla}(F)|^k + 1$ sets of at most k clauses and the lemma follows.

We now argue that $|\texttt{PS}(F)| \leq 2^{tk}$. As the maximum induced matching has size k there is some set $\mathcal{C}$ of k clauses so that $var(\mathcal{C}) = var(F)$. As each clause $C \in \mathcal{C}$ has $|var(C)| \leq t$, we have $|var(F)| = |var(\mathcal{C})| \leq tk$. As there are no more than $2^{|var(F)|}$ assignments for F, the ps-value of F is upper bounded by 2^{tk}. □

8.2.3 The ps-width of a formula

We use branch decompositions over the set of variables and clauses, and for a formula F, when we say (T,δ) is a branch decomposition of F, we mean (T,δ) is a branch decomposition over $\texttt{var}(F) \cup \texttt{cla}(F)$. Similar as for branch decompositions over set of vertices, we say that the bipartitions induced by the decomposition is a *cut* of F. Furthermore, as we are working with rooted branch decompositions, also the nodes of T induce cuts of F; namely the cut with parts $\delta(v)$ and $\overline{\delta(v)}$ for node v in T.

For each node v in T, by F_v we denote the formula induced by the clauses in $\texttt{cla}(F) \setminus \delta(v)$ and the variables in $\delta(v)$, and by $F_{\overline{v}}$ we denote the formula on the complement sets; i.e. the clauses in $\delta(v)$ and the variables in $\texttt{var}(F) \setminus \delta(v)$. In other words, if $\delta(v) = \mathcal{C} \cup X$ with $\mathcal{C} \subseteq \texttt{cla}(F)$ and $X \subseteq \texttt{var}(F)$ then $F_v = F_{\overline{\mathcal{C}},X}$ and $F_{\overline{v}} = F_{\mathcal{C},\overline{X}}$. To simplify the notation, we will for a node v in a branch decomposition and a set $\mathcal{C}$ of clauses denote by $\mathcal{C}|_v$ the set $\mathcal{C}|_{\texttt{var}(F_v)}$. We define the ps*-value* of the cut $\delta(v)$ to be

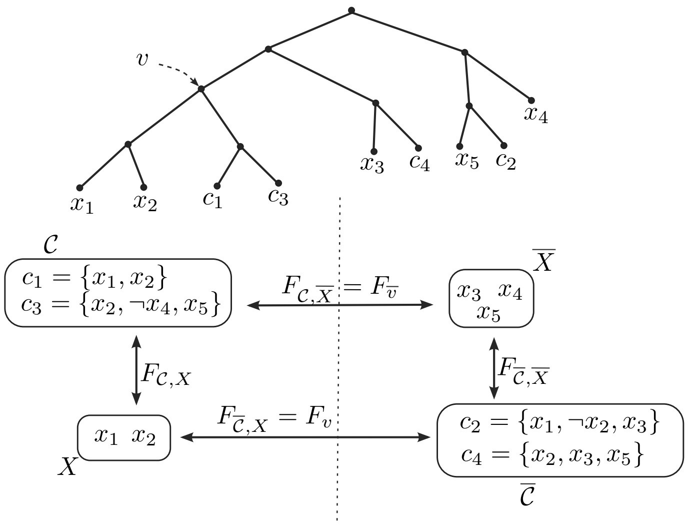
$$\text{ps}\,(\delta(v)) = \max\{|PS(F_v)|, |PS(F_{\overline{v}})|\}$$

We define the ps*-width* of a branch decomposition to be

$$\texttt{psw}(T,\delta) = \max\{\text{ps}\,(\delta(v)) : v \text{ is a node of } T\}$$

We define the ps*-width* of a formula F to be

$$\texttt{psw}(F) = \min\{\texttt{psw}(T,\delta) : (T,\delta) \text{ is a branch decomposition of } F\}$$



v
x_1
x_2
c_1
c_3
x_3
c_4
x_5
c_2
x_4
$\mathcal{C}$
$c_1 = \{x_1, x_2\}$
$c_3 = \{x_2, \neg x_4, x_5\}$
$F_{\mathcal{C},\overline{X}} = F_{\overline{v}}$
$\overline{X}$
x_3 x_4 x_5
$F_{\mathcal{C},X}$
$F_{\overline{\mathcal{C}},\overline{X}}$
x_1 x_2
X
$F_{\overline{\mathcal{C}},X} = F_v$
$c_2 = \{x_1, \neg x_2, x_3\}$
$c_4 = \{x_2, x_3, x_5\}$
$\overline{\mathcal{C}}$

Figure 8.2: On top is a branch decomposition of a formula F with $\texttt{var}(F) = \{x_1, x_2, x_3, x_4, x_5\}$ and the 4 clauses $\texttt{cla}(F) = \{c_1, c_2, c_3, c_4\}$ as given in the boxes. The node v of the tree defines the cut $\delta(v) = \mathcal{C} \cup X$ where $\mathcal{C} = \{c_1, c_3\}$ and $X = \{x_1, x_2\}$. There are 4 subformulas defined by this cut: $F_{\mathcal{C},X}, F_{\overline{\mathcal{C}},\overline{X}}, F_{\overline{\mathcal{C}},X}, F_{\mathcal{C},\overline{X}}$. For example, $F_{\overline{\mathcal{C}},X} = \{\{x_1, \neg x_2\}, \{x_2\}\}$ and $F_{\mathcal{C},\overline{X}} = \{\emptyset, \{\neg x_4, x_5\}\}$. We have $F_v = F_{\overline{\mathcal{C}},X}$ and $F_{\overline{v}} = F_{\mathcal{C},\overline{X}}$ with projection satisfiable sets of clauses $\texttt{PS}(F_v) = \{\{c_2|_v\}, \{c_4|_v\}, \{c_2|_v, c_4|_v\}\}$ and $\texttt{PS}(F_{\overline{v}}) = \{\emptyset, \{c_3|_{\overline{v}}\}\}$ and the ps-value of this cut is $\text{ps}\,(\delta(v)) = \max\{|PS(F_v)|, |PS(F_{\overline{v}})|\} = 3$.

Note that the ps-value of a cut is a symmetric function. That is, the ps-value of cut S equals the ps-value of the cut $\overline{S}$. See Figure 8.2 for an example.

8.3 Dynamic programming for MaxSAT and #SAT

Given a branch decomposition (T,δ) of a CNF formula F over n variables and m clauses and of total size s, we will give algorithms that solve MaxSAT and #SAT on F in time $\mathcal{O}(\mathtt{psw}(T,\delta)^3 m(m+n))$. Our algorithms are strongly inspired by the algorithm of [98], but in order to achieve a runtime polynomial in ps-width, and also to solve MaxSAT, we must make some crucial changes. In particular, we must index the dynamic programming tables by PS-sets rather than the 'shapes' used in [98].

Special terminology. In this dynamic programming section, we will combine partial solutions to subformulas into solutions for the input formula F. To improve readability we introduce notation PS' and $\mathtt{sat}'$ that allows us to refer directly to the clauses of F, also when working on the subformulas. Thus, for a formula F and branch decomposition (T,δ), for each node v in T, and induced subformula F_v of F, by $\mathtt{PS}'(F_v)$ we denote the subsets of clauses $\mathcal{C}$ from $\mathtt{cla}(F) \setminus \delta(v)$ so that $\mathtt{PS}(F_v) = \mathcal{C}|_{\mathtt{var}(F_v)}$. Similarly, for an assignment τ over $\mathtt{var}(F_v)$, by $\mathtt{sat}'(F_v,\tau)$ we denote the set of clauses $\mathcal{C}$ from $\mathtt{cla}(F) \setminus \delta(v)$ so that $\mathtt{sat}(F_v,\tau) = \mathcal{C}|_{\mathtt{var}(F_v)}$. Note that $|\mathtt{PS}'(F_v)| = |\mathtt{PS}(F_v)|$ and $|\mathtt{sat}'(F_v,\tau)| = |\mathtt{sat}(F_v,\tau)|$. We take the liberty to call also these sets projection satisfiable and refer to them as 'PS-sets' in the text, but it will be clear from context that we mean clauses of $\mathtt{cla}(F)$ and not $\mathtt{cla}(F_v)$.

Implementation details. We regard PS-sets as boolean vectors of length $|\mathtt{cla}(F)|$, and assume we can identify clauses and variables by integer numbers. So, checking if a clause is in a PS-set can be done in constant time, and checking if two PS-sets are equal can be done in $\mathcal{O}(|\mathtt{cla}(F)|)$ time. To manage our PS-sets, we use a binary `trie` datastructure (see [44]). We can add and retrieve a PS-set to and from a `trie` in $\mathcal{O}(|\mathtt{cla}(F)|)$ time. Trying to add a PS-set to a `trie` already containing an equivalent PS-set will not alter the content of the `trie`, so our `trie`'s will only contain distinct PS-sets. As retrieval of an element in our `trie` takes $\mathcal{O}(|\mathtt{cla}(F)|)$ time, by assigning a distinct integer to each PS-set at the time it is added to the `trie`, we have a $\mathcal{O}(|\mathtt{cla}(F)|)$-time mapping from PS-sets to distinct integers. This will be used implicitly in our algorithms when we say we index by PS-sets; when implementing the algorithm we will instead index by the according distinct integer the PS-set is mapped to.

In a pre-processing step we will need the following which, for each node v in T computes the sets of projection satisfiable subsets of clauses $\mathtt{PS}'(F_v)$ and $\mathtt{PS}'(F_{\overline{v}})$ of the two crossing subformulas F_v and $F_{\overline{v}}$.

Theorem 8.2. *Given a CNF formula F with a branch decomposition (T,δ) of* ps*-width k, we can in time $\mathcal{O}(k^2 m(m+n))$ compute the sets $\mathtt{PS}'(F_v)$ and $\mathtt{PS}'(F_{\overline{v}})$ for each v in T.*

Proof. We notice that for a node v in T with children c_1 and c_2, we can express $\mathtt{PS}'(F_v)$ as

$$\mathtt{PS}'(F_v) = \left\{ (C_1 \cup C_2) \cap \mathtt{cla}(F_v) : \begin{array}{l} C_1 \in \mathtt{PS}'(F_{c_1}), \text{ and} \\ C_2 \in \mathtt{PS}'(F_{c_2}) \end{array} \right\} .$$

Similarly, for sibling s and parent p of v in T, the set $\texttt{PS}'(F_{\overline{v}})$ can be expressed as

$$\texttt{PS}'(F_{\overline{v}}) = \left\{ (C_p \cup C_s) \cap \texttt{cla}(F_{\overline{v}}) : \begin{array}{l} C_p \in \texttt{PS}'(F_{\overline{p}}), \text{ and} \\ C_s \in \texttt{PS}'(F_s) \end{array} \right\} .$$

By transforming these recursive expressions into a dynamic programming algorithm, as done in Procedure 1 and Procedure 2 below, we are able to calculate all the desired sets as long as we can compute the sets for the base cases $\texttt{PS}'(F_l)$ when l is a leaf of T, and $\texttt{PS}'(F_{\overline{r}})$ for the root r of T. However, these formulas contain at most one variable, and thus we can easily construct their set of projection satisfiable clauses in linear amount of time for each of the formulas. For the rest of the formulas, we construct the formulas using Procedure 1 and Procedure 2. As there are at most twice as many nodes in T as there are clauses and variables in F, the procedures will run at most $\mathcal{O}(|\texttt{cla}(F)|+|\texttt{var}(F)|)$ times. In each run of the algorithms, we iterate through at most k^2 pairs of projection satisfiable sets, and do a constant number of set operations that might take $\mathcal{O}(|\texttt{cla}(F)|)$ time each. This results in a total runtime of $\mathcal{O}(k^2|\texttt{cla}(F)|(|\texttt{cla}(F)|+|\texttt{var}(F)|)) = \mathcal{O}(k^2m(m+n))$ for all the nodes of T combined. □

```
Procedure 1: Generating $\texttt{PS}'(F_v)$
input:  $\texttt{PS}'(F_{c_1})$ and $\texttt{PS}'(F_{c_2})$ for children $c_1$ and $c_2$ of $v$
        in branch decomposition
output: $\texttt{PS}'(F_v)$

$L \leftarrow$ empty trie of projection satisfiable clause-sets
for each $(C_1, C_2) \in \texttt{PS}'(F_{c_1}) \times \texttt{PS}'(F_{c_2})$ do
  add $(C_1 \cup C_2) \cap \texttt{cla}(F_v)$ to $L$
return $L$
```

```
Procedure 2: Generating $\texttt{PS}'(F_{\overline{v}})$
input:  $\texttt{PS}'(F_s)$ and $\texttt{PS}'(F_{\overline{p}})$ for sibling $s$ and parent $p$ of $v$
        in branch decomposition
output: $\texttt{PS}'(F_{\overline{v}})$

$L \leftarrow$ empty trie of projection satisfiable clause-sets
for each $(C_s, C_p) \in \texttt{PS}'(F_s) \times \texttt{PS}'(F_{\overline{p}})$ do
  add $(C_s \cup C_p) \cap \texttt{cla}(F_{\overline{v}})$ to $L$
return $L$
```

We now move on to the dynamic programming proper. We first give the algorithm for MaxSAT and then briefly describe the changes necessary for solving weighted MaxSAT and #SAT.

Our algorithm uses the technique of 'expectation' introduced in [21, 22]. Some partial solutions might be good when combined with certain partial solutions, but bad when combined with others. In the technique of 'expectation' we categorize how partial solutions can interact, and then optimize our selection of partial solutions based on the 'expectation' that this interaction occurs. In our dynamic programming algorithm for MaxSAT, we apply this technique by making expectations on each cut regarding what set of clauses will be satisfied by variables of the opposide side of the cut.

For a node v in the decomposition of F and PS-sets $C \in \mathtt{PS}'(F_v)$ and $C' \in \mathtt{PS}'(F_{\overline{v}})$, we say that an assignment τ of $\mathtt{var}(F)$ *meets the expectation C and C'* if $\mathtt{sat}'(F_v, \tau|_v) = C$ and $\mathtt{sat}'(F_{\overline{v}}, \tau|_{\overline{v}}) = C'$. For each node v of the branch decomposition, our algorithm uses a table $\mathtt{Tab}_v$ that for each pair $(C, C') \in \mathtt{PS}'(F_v) \times \mathtt{PS}'(F_{\overline{v}})$ stores in $\mathtt{Tab}_v(C, C')$ the maximum number of clauses in $\delta(v)$ that are satisfied, over all assignments meeting the expectation of C and C'. As the variables in $\mathtt{var}(F) \setminus \delta(v)$ satisfy exactly C', for any assignment that meets this expectation, an equivalent formulation of the content of $\mathtt{Tab}_v(C, C')$ is that it must satisfy the following constraint:

$$\begin{aligned} &\text{Over all assignments } \tau \text{ of } \mathtt{var}(F) \cap \delta(v) \text{ such that } \mathtt{sat}'(F_v, \tau) = C \ , \\ &\qquad \mathtt{Tab}_v(C, C') = \max_{\tau} \left\{ \left| \left(\mathtt{sat}'(F, \tau') \cap \delta(v) \right) \cup C' \right| \right\} \end{aligned} \tag{8.1}$$

By bottom-up dynamic programming along the tree T we compute the tables of each node of T. For a leaf l in T, generating $\mathtt{Tab}_l$ can be done easily in linear time since the formula F_v contains at most one variable. For an internal node v of T, with children c_1, c_2, we compute $\mathtt{Tab_v}$ by the algorithm described in Procedure 3. There are 3 tables involved in this update, one at each child and one at the parent. A pair of entries, one from each child table, may lead to an update of an entry in the parent table. Each table entry is indexed by a pair, thus there are 6 indices involved in a single potential update. A trick first introduced in [22] allows us to loop over triples of indices and for each triple compute the remaining 3 indices forming the 6-tuple involved in the update, thereby reducing the runtime.

```
Procedure 3: Computing $\mathtt{Tab_v}$ for inner node $v$ with children $c_1, c_2$
input:  $\mathtt{Tab_{c_1}}, \mathtt{Tab_{c_2}}$
output: $\mathtt{Tab_v}$
1. initialize $\mathtt{Tab_v} : \mathtt{PS}'(F_v) \times \mathtt{PS}'(F_{\overline{v}}) \to \{-1\}$
2. for each $(C_{c_1}, C_{c_2}, C'_v)$ in $\mathtt{PS}'(F_{c_1}) \times \mathtt{PS}'(F_{c_2}) \times \mathtt{PS}'(F_{\overline{v}})$ do
3.     $C'_{c_1} \leftarrow (C_{c_2} \cup C'_v) \cap \delta(c_1)$
4.     $C'_{c_2} \leftarrow (C_{c_1} \cup C'_v) \cap \delta(c_2)$
5.     $C_v \leftarrow (C_{c_1} \cup C_{c_2}) \setminus \delta(v)$
6.     $t \leftarrow \mathtt{Tab_{c_1}}(C_{c_1}, C'_{c_1}) + \mathtt{Tab_{c_2}}(C_{c_2}, C'_{c_2})$
7.     if $\mathtt{Tab_v}(C_v, C'_v) < t$ then $\mathtt{Tab_v}(C_v, C'_v) \leftarrow t$
8. return $\mathtt{Tab_v}$
```

Lemma 8.3. *For a CNF formula F of m clauses and an inner node v, of a branch decomposition (T, δ) of* ps*-width k, Procedure 3 computes* $\mathtt{Tab_v}$ *satisfying Constraint (8.1) in time $\mathcal{O}(k^3 m)$.*

Proof. We assume $\mathtt{Tab}_{c_1}$ and $\mathtt{Tab}_{c_2}$ satisfy Constraint (8.1). Procedure 3 loops over all triples in $\mathtt{PS}'(F_{c_1}) \times \mathtt{PS}'(F_{c_2}) \times \mathtt{PS}'(F_{\overline{v}})$. From the definition of ps-width of (T, δ) there are at most k^3 such triples. Each operation inside an iteration of the loop take $\mathcal{O}(m)$ time and there is a constant number of such operations. Thus the runtime is $\mathcal{O}(k^3 m)$.

Before we show the correctness of the output, let us look a bit at the workings of Procedure 3. For any assignment τ over $\mathtt{var}(F)$, and cut, the assignment τ will only meet the expectation of a single pair of PS-sets. Let (X_1, X'_1), (X_2, X'_2) and (X_v, X'_v) be the pairs an assignment τ meets the expectation for with respect to the cuts induced by c_1, c_2, and v, respectively. We notice that

$$\begin{aligned}
X_v &= \mathtt{sat}'(F_v, \tau|_v) \\
&= \mathtt{sat}'(F_v, \tau|_{c_1} \uplus \tau|_{c_2}) \\
&= \mathtt{sat}'(F_v, \tau|_{c_1}) \cup \mathtt{sat}'(F_v, \tau|_{c_2}) \\
&= (\mathtt{sat}'(F_{c_1}, \tau|_{c_1}) \setminus \delta(v)) \cup (\mathtt{sat}'(F_{c_2}, \tau|_{c_2}) \setminus \delta(v)) \\
&= (X_1 \setminus \delta(v)) \cup (X_2 \setminus \delta(v)) \\
&= (X_1 \cup X_2) \setminus \delta(v).
\end{aligned} \tag{8.2}$$

This can also be seen from Figure 8.3. By symmetry, we find similar values for X'_1 and X'_2; namely $X'_1 = (X_2 \cup X'_v) \cap \delta(c_1)$ and $X'_2 = (X_1 \cup X'_v) \cap \delta(c_2)$. So, these latter three sets will be implicit based on the three former sets with respect to the cuts induced by v, c_1 and c_2. We will therefore, for convenience of this proof, say that an assignment τ *meets the expectation of a triple* (C_1, C_2, C') of PS-sets, when τ meets the expectation of the implicit three pairs on each of their respective cuts. We notice that for each choice of triples of PS-sets (C_{c_1}, C_{c_2}, C'_v) Procedure 3 computes the implicit three other sets and names them C'_{c_1}, C'_{c_2} and C_v accordingly.

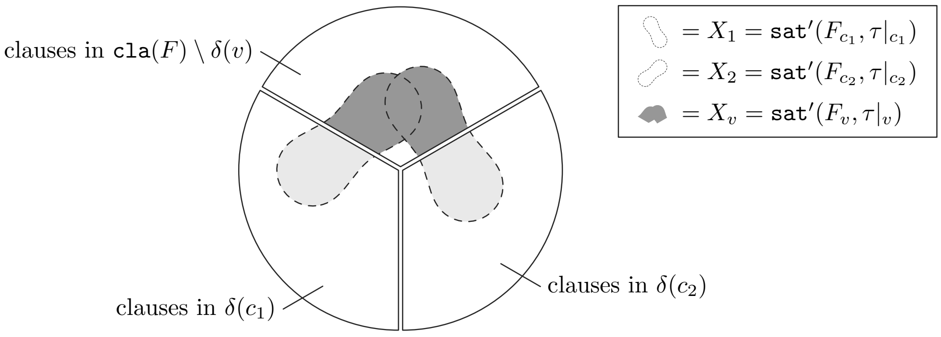

clauses in $\mathtt{cla}(F) \setminus \delta(v)$
clauses in $\delta(c_1)$
clauses in $\delta(c_2)$
$= X_1 = \mathtt{sat}'(F_{c_1}, \tau|_{c_1})$
$= X_2 = \mathtt{sat}'(F_{c_2}, \tau|_{c_2})$
$= X_v = \mathtt{sat}'(F_v, \tau|_v)$

Figure 8.3: As shown by the chain of equalities in (8.2) in the proof of Lemma 8.3, the clauses in $\mathtt{sat}'(F_v, \tau|_v)$ are precisely the clauses in $(\mathtt{sat}'(F_{c_1}, \tau|_{c_1}) \cup \mathtt{sat}'(F_{c_2}, \tau|_{c_2})) \setminus \delta(v)$.

We will now show that for all pairs $(C, C') \in \mathtt{PS}'(F_v) \times \mathtt{PS}'(F_{\overline{v}})$ the value of $\mathtt{Tab}_v(C, C')$ is correct. Let τ_0 be an assignment over $\mathtt{var}(F)$ that satisfies the maximum number of clauses, while meeting the expectation of C and C'. Thus, the value of $\mathtt{Tab}_v(C, C')$ is correct if and only if it stores exactly the number of clauses from $\delta(v)$ that τ_0 satisfies.

Let (C_1, C'_1) and (C_2, C'_2) be the pairs of PS-sets that τ_0 meet the expectation of for the cut $(\delta(c_1), \overline{\delta(c_1)})$ and $(\delta(c_2), \overline{\delta(c_2)})$, respectively. As τ_0 meets these expectations, the value of $\mathtt{Tab}_{c_1}(C_1, C'_1)$ and $\mathtt{Tab}_{c_2}(C_2, C'_2)$ must be at least as large as the number of clauses τ_0 satisfies in $\delta(c_1)$ and $\delta(c_2)$, respectively. Thus, the number of clauses τ_0 satisfies in both $\delta(c_1)$ and $\delta(c_2)$ is at most as large as the sum of these two entries. Since Procedure 3, in the iteration where $C'_v = C'$, $C_{c_1} = C_1$ and $C_{c_2} = C_2$, ensures that $\texttt{Tab}_v(C,C')$ is at least the sum of $\texttt{Tab}_{c_1}(C_1,C'_1)$ and $\texttt{Tab}_{c_2}(C_2,C'_2)$, we know $\texttt{Tab}_v(C,C')$ is at least as large as the correct value.

Now assume for contradiction that the value of the cell $\texttt{Tab}_v(C,C')$ is too large. That means that at some iteration of Procedure 3 it is being assigned the value $\texttt{Tab}_{c_1}(C_{c_1},C'_{c_1}) + \texttt{Tab}_{c_2}(C_{c_2},C'_{c_2})$ when this sum is too large. Let τ_1 and τ_2 be the assignments of $\texttt{var}(F)$ meeting the expectation of C_{c_1} and C'_{c_1} and meeting the expectation of C_{c_2},C'_{c_2}, respectively, where the number of clauses of $\delta(c_1)$ and $\delta(c_2)$, respectively, equals the according table entries of $\texttt{Tab}_{c_1}$ and $\texttt{Tab}_{c_2}$. If we now take the assignment $\tau_x = \tau_1|_{c_1} \uplus \tau_2|_{c_2} \uplus \tau_0|_{\overline{v}}$, we have an assignment that meets the expectation of C and C', and who satisfies more clauses in $\delta(v)$ than τ_0, contradicting the choice of τ_0. So $\texttt{Tab}_v(C,C')$ can be neither smaller nor larger than the number of clauses in $\delta(v)$ τ_0 satisfies, so it is exactly the same. $\square$

Theorem 8.4. *Given a formula F over n variables and m clauses, and a branch decomposition (T,δ) of F of* ps*-width k, we solve* MaxSAT*,* #SAT*, and weighted* MaxSAT *in time $\mathcal{O}(k^3 m(m+n))$.*

Proof. To solve MaxSAT, we first compute $\texttt{Tab}_r$ for the root node r of T. This requires that we first compute $\texttt{PS}'(F_v)$ and $\texttt{PS}'(F_{\overline{v}})$ for all nodes v of T, and then, in a bottom up manner, compute $\texttt{Tab}_v$ for each of the $\mathcal{O}(m+n)$ nodes in T. The former part we can do in $\mathcal{O}(k^2 m(m+n))$ time by Theorem 8.2, and the latter part we do in $\mathcal{O}(k^3 m(m+n))$ time by Lemma 8.3.

At the root r of T we have $\delta(r) = \texttt{var}(F) \cup \texttt{cla}(F)$. Thus $F_r = \emptyset$ and $F_{\overline{r}}$ does not have any variables, so that $PS(F_r) \times PS(F_{\overline{r}})$ contains only $(\emptyset,\emptyset)$. As all assignments over $\texttt{var}(F)$ meet the expectation of $\emptyset$ and $\emptyset$ on the cut $(\delta(r),\overline{\delta(r)})$, and $\texttt{cla}(F) \cap \delta(r) = \texttt{cla}(F)$, by Constraint (8.1) the value of $\texttt{Tab}_r(\emptyset,\emptyset)$ is the maximal number of clauses in F any assignment of $\texttt{var}(F)$ satisfies. And hence, this number is the solution to MaxSAT.

For a weight function $w : \texttt{cla}(F) \to \mathbb{N}$, by redefining Constraint (8.1) for $\texttt{Tab}_v$ to maximize $w(\texttt{sat}'(F,\tau) \cap \delta(v))$ instead of $|\texttt{sat}'(F,\tau) \cap \delta(v))|$, we are able to solve the more general problem weighted MaxSAT in the same way.

For the problem #SAT, we care only about assignments satisfying all the clauses of F, and we want to decide the number of distinct assignments doing so. This requires a few alterations. Firstly, alter the definition of the contents of $\texttt{Tab}_v(C,C')$ in Constraint (8.1) to be the number of assignments τ over $\texttt{var}(F) \cap \delta(v)$ where $\texttt{sat}'(F_v,\tau) = C$ and all clauses in $\delta(v)$ is either in C' or satisfied by τ. Secondly, when computing $\texttt{Tab}_l$ for the leaves l of T, we set each of the entries of $\texttt{Tab}_l$ to either zero, one, or two, according to the definition. Thirdly, we alter the algorithm to compute $\texttt{Tab}_v$ (Procedure 3) for inner nodes. We initialize $\texttt{Tab}_\texttt{v}(C,C')$ to be zero at the start of the algorithm, and substitute lines 6 and 7 of Procedure 3 by the following line which increases the table value by the product of the table values at the children

$$\texttt{Tab}_\texttt{v}(C_v,C_{\overline{v}}) \leftarrow \texttt{Tab}_\texttt{v}(C_v,C_{\overline{v}}) + \texttt{Tab}_{\texttt{c}_1}(C_{c_1},C_{\overline{c_1}}) \cdot \texttt{Tab}_{\texttt{c}_2}(C_{c_2},C_{\overline{c_2}})$$

This will satisfy our new constraint of $\texttt{Tab}_v$ for internal nodes v of T. The value of $\texttt{Tab}_r(\emptyset,\emptyset)$ at the root r of T will be exactly the number of distinct assignments satisfying all clauses of F. □

The bottleneck giving the cubic factor k^3 in the runtime of Theorem 8.4 is the number triples in $\texttt{PS}'(F_{\overline{v}}) \times \texttt{PS}'(F_{c_1}) \times \texttt{PS}'(F_{c_2})$ for any node v with children c_1 and c_2. When (T,δ) is a linear branch decomposition, it is always the case that either c_1 or c_2 is a leaf of T. In this case either $|\texttt{PS}'(F_{c_1})|$ or $|\texttt{PS}'(F_{c_2})|$ is a constant. Therefore, for linear branch decompositions $\texttt{PS}'(F_{\overline{v}}) \times \texttt{PS}'(F_{c_1}) \times \texttt{PS}'(F_{c_2})$ will contain no more than $\mathcal{O}(k^2)$ triples. Thus we can reduce the runtime of the algorithm by a factor of k.

Theorem 8.5. *Given a formula F over n variables and m clauses, and a linear branch decomposition (T,δ) of F of* ps*-width k, we solve* #SAT*,* MaxSAT*, and weighted* MaxSAT *in time $\mathcal{O}(k^2 m(m+n))$.*

8.4 CNF formulas of polynomial ps-width

In this section we investigate classes of CNF formulas having decompositions with ps-width polynomially bounded in the total size s of the formula. In particular, we show that this holds whenever the incidence graph of the formula has constant mim-width (see Section 1.1 for the definition of mim-width). We also show that a large class of bipartite graphs, using what we call bigraph bipartizations, have constant mim-width.

In order to lift the upper bound of Lemma 8.1 on the ps-value of F, i.e $|\texttt{PS}(F)|$, to the ps-width of F, we use mim-width of the incidence graph $I(F)$, which is defined using branch decompositions of graphs. A branch decomposition of the formula F, as defined in Section 8.2, can also be seen as a branch decomposition of the incidence graph $I(F)$. Therefore, from Lemma 8.1, we immediately get the following corollary.

Corollary 8.6. *For any CNF formula F over m clauses, with no clause containing more than t literals, the* ps*-width of F is at most $\min\{m^k+1, 2^{tk}\}$ for $k = \text{mim-w}(I(F))$.*

Many classes of graphs have intersection models, meaning that they can be represented as intersection graphs of certain objects, i.e. each vertex is associated with an object and two vertices are adjacent iff their objects intersect. The objects used to define intersection graphs usually consist of geometrical objects such as lines, circles or polygons. Many well known classes of intersection graphs have constant mim-width, as in the following which lists only a subset of the classes proven to have such bounds in [6, 107].

Theorem 8.7 ([6, 107])**.** *Let G be a graph. If G is a:*
interval graph then $\text{mim-w}(G) \leq 1$.
circular arc graph then $\text{mim-w}(G) \leq 2$.
k-trapezoid graph then $\text{mim-w}(G) \leq k$.
Moreover there exist linear decompositions satisfying the bound, that can be found in polynomial time (for k-trapezoid assume the intersection model is given).

Let us briefly mention the definition of these graph classes. A graph is an interval graph if it has an intersection model consisting of intervals of the real line. A graph is a circular arc graph if it has an intersection model consisting of arcs of a circle. To build a k-trapezoid we start with k parallel line segments $(s_1,e_1),(s_2,e_2),...,(s_k,e_k)$ and add two non-intersecting paths s and e by joining s_i to s_{i+1} and e_i to e_{i+1} respectively by straight lines for each $i \in \{1,...,k-1\}$. The polygon defined by s and e and the two line segments $(s_1,e_1),(s_k,e_k)$ forms a k-trapezoid. A graph is a k-trapezoid graph if it has an intersection model consisting of k-trapezoids. See [18] for information about graph classes and their containment relations.

Combining Corollary 8.6 and Theorem 8.7 we get the following

Corollary 8.8. *Let F be a CNF formula containing m clauses with maximum clause-size t. If $I(F)$ is a:*
interval graph then $\mathtt{psw}(F) \leq \min\{m+1, 2^t\}$.
circular arc graph then $\mathtt{psw}(F) \leq \min\{m^2+1, 4^t\}$.
k-trapezoid graph then $\mathtt{psw}(F) \leq \min\{m^k+1, 2^{tk}\}$.
Moreover there exist linear decompositions satisfying the bound, that can be found in polynomial time (for k-trapezoid assume the intersection model is given).

The incidence graphs of formulas are bipartite graphs, which is not the case for the majority of graphs in the above-mentioned graph classes. In the following we show how to extend the results of Corollary 8.8 to large classes of bipartite graphs. For a graph G and subset of vertices $A \subseteq V(G)$ the bipartite graph $G[A,\overline{A}]$ is the subgraph of G containing all edges of G with exactly one endpoint in A. For any graph G and $A \subseteq V(G)$ we call $G[A,\overline{A}]$ a *bigraph bipartization* of G, and note that G has a bigraph bipartization for each subset of vertices. For a graph class X we define the class of X *bigraphs* as the bipartite graphs H for which there exists $G \in X$ such that H is isomorphic to a bigraph bipartization of G. For example, a bipartite graph H is an *interval bigraph* if there is some interval graph G and some $A \subseteq V(G)$ with H isomorphic to $G[A,\overline{A}]$.

The following result will allow us to lift the results of Corollary 8.8 from the given graphs to the bigraph bipartizations of the same graphs.

Theorem 8.9. *Assume that we are given a CNF formula F of m clauses and maximum clause-size t, a graph G, a subset $A \subseteq V(G)$, and (T,δ_G) a (linear) branch decomposition of G of* mim*-width k. If $I(F)$ is connected and isomorphic to $G[A,\overline{A}]$ (thus $I(F)$ a bigraph bipartization of G) then we can in linear time produce a (linear) branch decomposition (T,δ_F) of F having* ps*-width at most* $\min\{m^k+1, 2^{tk}\}$

Proof. Since each variable and clause in F has a corresponding node in $I(F)$, and each node in $I(F)$ has a corresponding node in G, by defining δ_F to be the function mapping each leaf l of T to the variable or clause in F corresponding to the node $\delta_G(l)$, we get that (T,δ_F) is a branch decomposition of F. Consider a cut $(B,\overline{B})$ induced by a node of (T,δ_F). Note that the mim-value of $G[B,\overline{B}]$ is at most k. $I(F)$ is connected which means that we have either A or $\overline{A}$ corresponding to the set of variables of F. Assume wlog the former. Thus $C = \overline{A} \cap B \subseteq \mathtt{cla}(F)$ are the clauses in B, with $\overline{C} = \mathtt{cla}(F) \setminus C$ and $X = A \cap B \subseteq \mathtt{var}(F)$ are the variables in B, with $\overline{X} = \mathtt{var}(F) \setminus X$. The mim-values of $G[C, \overline{X}]$ and $G[\overline{C}, X]$ are at most k, since these are induced subgraphs of $G[B, \overline{B}]$, and taking induced subgraphs cannot increase the size of the maximum induced matching. Hence by Lemma 8.1, we have $|\mathtt{PS}(F_{C,\overline{X}})| \leq |\mathtt{cla}(F)|^k + 1$, and likewise we have $|\mathtt{PS}(F_{\overline{C},X})| \leq |\mathtt{cla}(F)|^k + 1$, with the maximum of these two being the ps-value of this cut. Since the ps-width of the decomposition is the maximum ps-value of each cut the theorem follows. □

Combining Theorems 8.9 and 8.7 we immediately get the following.

Corollary 8.10. *Let F be a CNF formula containing m clauses with maximum clause-size t. If $I(F)$ is a:*
interval bigraph then $\mathtt{psw}(F) \leq \min\{m+1, 2^t\}$.
circular arc bigraph then $\mathtt{psw}(F) \leq \min\{m^2+1, 4^t\}$.
k-trapezoid bigraph then $\mathtt{psw}(F) \leq \min\{m^k+1, 2^{tk}\}$.
Moreover there exist linear decompositions satisfying the bound.

In the next section we address the question of finding such linear decompositions in polynomial time. We succeed in the case of interval bigraphs, but for circular arc bigraphs and k-trapezoid bigraphs we must leave this as an open problem.

8.5 Interval bigraphs and formulas having interval orders

We will in this section show that for formulas whose incidence graph is an interval bigraph we can in polynomial time find linear branch decompositions having small ps-width. Let us recall the definition of interval ordering. A CNF formula F has an interval ordering if there exists a linear ordering of variables and clauses such that for any variable x occurring in clause C, if x appears before C then any variable between them also occurs in C, and if C appears before x then x occurs also in any clause between them. See Figure 8.4 for an example.

By a result of Hell and Huang [53] it follows that a formula F has an interval ordering if and only if $I(F)$ is a interval bigraph.

Theorem 8.11. *Given a CNF formula F over n variables and m clauses each of at most t literals. In time $\mathcal{O}((m+n)mn)$ we can decide if F has an interval ordering (yes iff $I(F)$ is an interval bigraph), and if yes we solve* #SAT *and weighted* MaxSAT *with an additional runtime of* $\mathcal{O}(\min\{m^2, 4^t\}(m+n)m)$.

Proof. Using the characterization of [53] and the algorithm of [84] we can in time $\mathcal{O}((m+n)mn)$ decide if F has an interval ordering and if yes, then we find it. From this interval ordering we build an interval graph G such that $I(F)$ is a bigraph bipartization of G, and construct a linear branch decomposition of G having mim-width 1 [6]. From such a linear branch decomposition we get from Theorem 8.9 that we can construct another linear branch decomposition of F having ps-width $\mathcal{O}(m)$. We then run the algorithm of Theorem 8.5. □

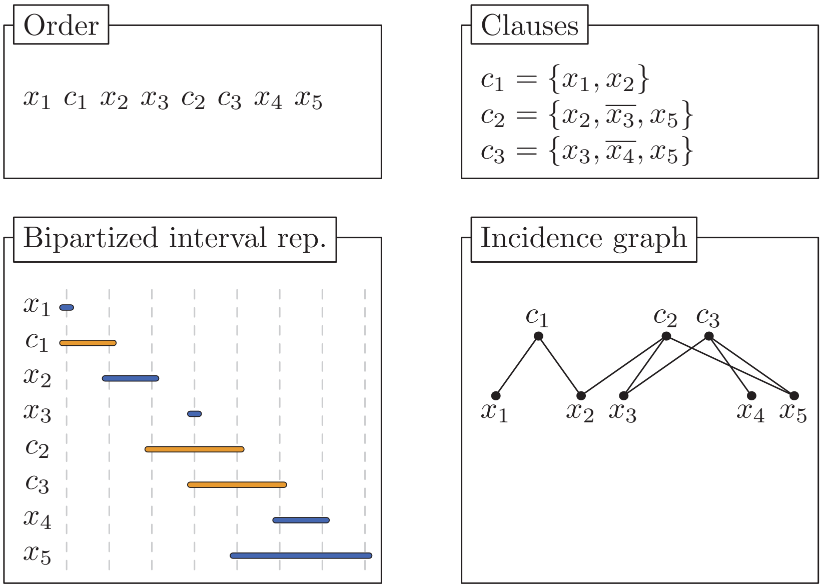

Order
$x_1\ c_1\ x_2\ x_3\ c_2\ c_3\ x_4\ x_5$
Clauses
$c_1 = \{x_1, x_2\}$
$c_2 = \{x_2, \overline{x_3}, x_5\}$
$c_3 = \{x_3, \overline{x_4}, x_5\}$
Bipartized interval rep.
x_1 c_1 x_2 x_3 c_2 c_3 x_4 x_5
Incidence graph
c_1 c_2 c_3
x_1 x_2 x_3 x_4 x_5

Figure 8.4: A CNF formula having an interval ordering. Its incidence graph is an interval bigraph, since it is isomorphic to the bigraph bipartization, defined by the blue intervals, of the interval graph with intersection model on the left.

8.6 Experimental results

We present some simple experimental results, intended as proof of concept. It is our belief that some of the ideas behind our algorithms, like the notion of ps-value, are useful in practice, but it will require a thorough investigation to confirm such a belief. Our results indicate that the worst-case runtime bounds of the dynamic programming, Theorems 8.4 and 8.5, are probably higher than what would commonly be seen in practice.

In the past decade, SAT solvers have become very powerful, and are currently able to handle very large practical instances. Techniques from these SAT solvers have been applied to develop relatively powerful MaxSAT and #SAT solvers [9]. In our experiments we compare implementations of our algorithms against state-of-the-art MaxSAT and #SAT solvers. We do not enhance our implementations with any other techniques, not even simple pre-processing, and on the vast majority of instances our implementations fall far behind in a comparison. However, when focusing on formulas with a certain linear order our implementations compare favorably.

As explained in Section 8.1 and Chapter 2, there are two steps involved: `[P1]` find a good decomposition of the input CNF formula F, and`[P2]` perform DP (dynamic programming) along the decomposition. Let us start by describing a very simple heuristic for step `[P1]`. It takes as input the bipartite graph $I(F)$ with vertex set $\texttt{cla}(F) \cup \texttt{var}(F)$, and outputs a linear order σ on the vertex set. The below heuristic GreedyOrder is a greedy algorithm that for increasing values of i chooses $\sigma(i)$ to be a vertex having the highest number of already chosen neighbors, and among these choosing one with fewest non-chosen neighbors. This defines a linear branch decomposition (T,δ) of the CNF formula F, with non-leaf nodes of the binary tree T inducing a path, with T rooted at one end of this path, and with δ mapping the ith leaf encountered by a breadth-first search starting at the root of T to the clause or variable $\sigma(i)$, for all $1 \leq i \leq |\texttt{cla}(F) \cup \texttt{var}(F)|$.

```
**Algorithm** GreedyOrder
**input**: $G=(V,E)$, a (bipartite) graph
**output**: $\sigma$, a linear ordering of $V$

$L=\emptyset, R=V$, $i=1$
**for all** $v \in V$ set $Ldegree(v)=0$
**while** $R$ is not empty **do**
    **choose** $v$: from vertices in $R$ with max $Ldegree$ take one of smallest degree
    set $\sigma(i)=v$, increment $i$, add $v$ to $L$ and remove $v$ from $R$
    **for all** $w \in R$ with $vw \in E$ increment $Ldegree(w)$
```

All our implementations can be found online [1]. We have implemented Greedy-Order in Java, together with a straight-forward implementation of the DP algorithm of Theorem 8.5.

Given a CNF formula, this allows us to solve MaxSAT and #SAT by first running GreedyOrder and then the DP. We compare our implementation to the best solvers we could find online, respectively CCLS-TO-AKMAXSAT [70] which was among the best solvers of the MaxSAT Evaluation competition in 2014 [2], and the latest version of the #SAT solver called SHARPSAT developed by Marc Thurley [102, 103]. These solvers handily beat our implementation on most inputs. We have therefore generated some CNF formulas having interval orderings, as in Theorem 8.11, to check if at least on these instances we do better. Note that for step `[P1]` we have not implemented the polynomial-time algorithm recognizing formulas having interval orders, relying instead on the GreedyOrder heuristic.

8.6.1 Generation of instances

Before presenting our results, let us describe the generation of the set of instances, which are of three types. We start with type 1. The generation of these formulas is based on the definition of interval orderings given by the interval bigraph definition, see e.g. the left side of Figure 8.4. To generate a formula of type 1 with n variables and m clauses, we generate $n+m$ intervals of the real line by iterating through points i from 1 to $2(n+m)$ as left and right endpoints of the intervals:

- At step i, check which of the 4 cases below are legal (e.g. 3 is legal if there exists a live variable, i.e. with left endpoint $< i$ and no right endpoint) and randomly make one of those legal choices:

1. start interval of new variable with left endpoint i
2. start interval of new clause with left endpoint i
3. end interval of randomly chosen live variable by right endpoint i
4. end interval of randomly chosen live clause by right endpoint i

Towards the end of the process boundary conditions are enforced to reach exactly m clauses, with n expected to be slightly smaller than m. For each clause interval we randomly choose each variable having overlapping interval as being either positive or negative in this clause. The resulting CNF formula will have an interval ordering given by the rightmost endpoints of intervals. To hide this ordering the clauses and variables are randomly permuted to make the final CNF formula.

The formulas of type 2 are generated in a very similar fashion as type 1, except we guarantee that all clauses have the same size t, which by Lemma 8.1 could be of big help. The only change is to case 4 above which instead of being a choice becomes enforced for a live clause that at step i has accumulated exactly t overlapping variable intervals. We also let each clause interval represent 4 clauses over the same variable set but on randomly chosen literals, at the aim of increasing the probability of each instance not being satisfiable.

The formulas of type 3 are the CNF-representation of a conjunction of XOR functions where each XOR has a fixed number t of literals and the variables of the XOR functions overlap in such a way that the incidence graph will be the bipartization of a circular arc graph.

A formula of type 3 is generated from three input parameters n,t,s. It has n variables represented by successive points 1 to n on the circle. The first XOR function has interval from 1 to t thus containing variables with points 1 to t, the second has interval $s+1$ to $s+t$, and in general the ith has interval $i*s+1$ to $i*s+t$, with appropriate modulo addition and some boundary condition at the end to ensure n/s XOR functions. Variables are chosen randomly to appear positive or negative in each XOR. Each XOR is then transformed in the standard way to a CNF formula with 2^{t-1} clauses to give us a resulting CNF formula with $n/s*2^{t-1}$ clauses. Again, variables and clauses are randomly permuted to hide the ordering giving the circular arc bigraph representation.

Note that all the resulting formulas have a quite simple structure, and that a state-of-the-art SAT solver, like `lingeling` [8], handles all generated instances within a few seconds.

8.6.2 Results

We are now ready to present our results. We ran all the solvers on a Dell Optiplex 780 running Ubuntu 12.04 64-Bit. The machine has 8GB of memory and an Intel Core 2 Quad Q9650 processor with OpenJDK java 6 (IcedTea6 1.13.5).

For instances of type 1 the GreedyOrder heuristic fails terribly and becomes a huge bottleneck. The greedy choice based on degrees of vertices in $I(F)$ is too simple. However, when given the correct interval order to our solver(s) they performed better.

Instances of type 2 are generated similar to those of type 1 but all clauses have small size, which by Lemma 8.1 could be of help. In this case the number of clauses is approximately four times the number of variables, and as a consequence a great number of the instances were not satisfiable, making the work of the #SAT-solvers easier than that of the MaxSAT solvers. All generated instances of type 2 were solved within seconds by `sharpSAT`, see Figure 8.5. As the size of the instances grow, we see a clear tendency for the runtimes of `CCLS_to_akmaxsat` to increase much more rapidly than both our solvers. The runtimes of our two solvers were almost identical. The GreedyOrder heuristic on these instances seems to produce decompositions/orders of low ps-width.

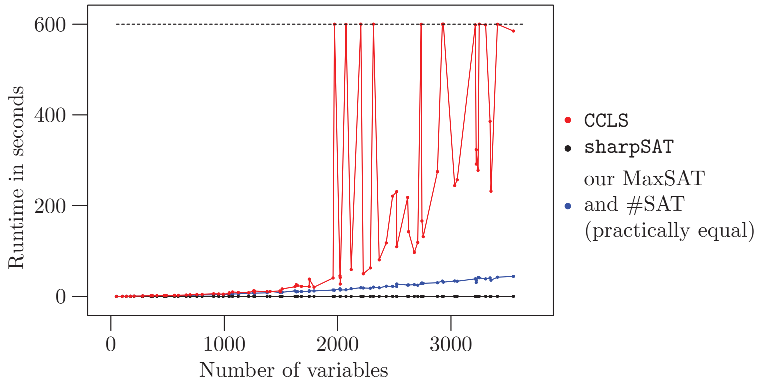


Figure 8.5: Runtimes of instances of type 2. Here our MaxSAT solver is clearly faster than `CCLS_to_akmaxsat`. The vertical axis represents time in seconds. Runs taking more than 600 seconds were stopped before completion and are drawn on the dotted line.

The type 3 instances shown in Figure 8.6 were generated with $k = 5$ and $s = 3$. All instances are satisfiable, which may explain why `CCLS_to_akmaxsat` is very fast. Choosing $k = 3$ and $s = 2$ there will be some not satisfiable instances and `CCLS_to_akmaxsat` would then often spend more than 600 seconds and time out. As the size of the instances grow, we see a clear tendency for the runtimes of `sharpSAT` to increase much more rapidly than our solvers. The runtimes of our two solvers were almost identical.

8.7 Conclusion

In this chapter we have proposed a structural parameter of CNF formulas, called ps-width or projection-satisfiable-width. We showed that weighted MaxSAT and #SAT can be solved in polynomial time if given a decomposition of the formula of polynomially bounded ps-width. Using the concept of interval bigraphs we also showed a polynomial time algorithm that actually finds such a decomposition, for formulas having an interval ordering. Could one devise such an algorithm also for the larger class of circular arc bigraphs, or maybe even for the even larger class of k-trapezoid bigraphs? In other words, is the problem of

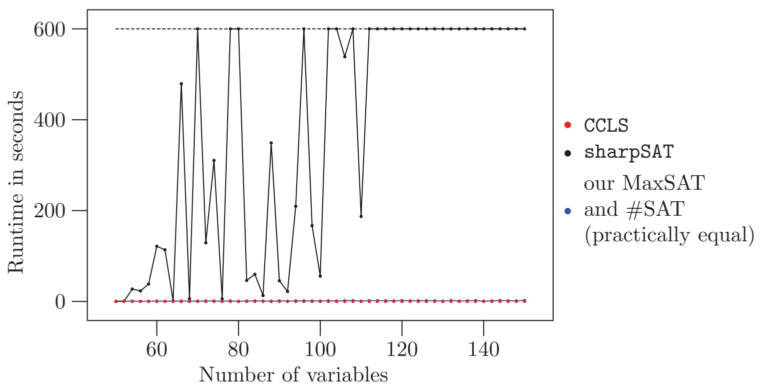


Figure 8.6: Runtimes of instances of type 3. Here our #SAT solver is clearly faster than `sharpSAT`. The vertical axis represents time in seconds. Runs taking more than 600 seconds were stopped before completion and are drawn on the dotted line.

recognizing if a bipartite input graph is a circular arc bigraph, or a k-trapezoid bigraph, polynomial-time solvable?

It could be of practical interest to design a heuristic algorithm which given a formula finds a decomposition of relatively low ps-width, as has been done for boolean-width in [55]. One could then check if benchmarks covering real-world SAT instances have low ps-width, and perform a study on the correlation between low ps-width and their practical hardness by MaxSAT and #SAT solvers, as has been done for treewidth and SAT solvers [71]. We presented some simple experimental results, but it will require a thorough investigation to check if ideas from our algorithms could be useful in practice. Finally, we hope the essential combinatorial result enabling the improvements in this chapter, Lemma 8.1, may have other uses as well.

Chapter 9

Conclusion

In this thesis, we have looked at four cases where we find alternative structural parameters solving a number of parameterized problems. From these case studies many natural open problems arise. We now look at some of these problems, case by case.

In the first case study, we showed that we can get a faster algorithm for Dominating Set by using branch decompositions of low mm-width when the treewidth of the input graph is at least 1.55 larger than the mm-width. One of the improvements of the mm-width approach over the treewidth approach is that we can get a better approximation than we do for treewidth in the same amount of time. However, we still only get a 3-approximation of mm-width. This was using the general approximation algorithm of Oum and Seymour [79] for branch decompositions (Theorem 3.1). By looking in particular at mm-width, and making use of properties unique for its cut function; can we get a better approximation of mm-width, both in terms of approximation ratio, and in terms of runtime? And what is the complexity of computing mm-width? Since the cut function $\mathrm{mm}(A)$ is polynomial time computable, it is clear that deciding mm-width is in NP, but is it NP-hard? And if so, is it also W[1]-hard, or is it FPT?

In the second case study, we defined the parameter split-matching-width (sm-width), which is weaker than clique-width and stronger than treewidth, and and showed that four problems that are W[1]-hard parameterized by clique-width are FPT parameterized by sm-width. We selected our four NP-hard problems based on them being expressible in MSO_2 but not in MSO_1, and being W[1]-hard parameterized by clique-width. Are there more problems like this which are FPT parameterized by sm-width and treewidth, but not by clique-width? Can such a class of problems be defined by expressibility in some logical language? For [P1] of our FPT algorithms we used an approximation that used FPT time, yet computing a split decomposition can be done in polynomial time. What is the complexity of computing sm-width?

In the third case, we looked at using an alternative rank-like parameter for solving many domination-type problems. One of the implications of our result was that we improved the worst case runtime for solving Dominating Set parameterized by clique-width from $\mathcal{O}^*(2^{\mathrm{cw}(G)^2})$ to $\mathcal{O}^*(\mathrm{cw}(G)^{\mathcal{O}(\mathrm{cw}(G))})$. A natural question to ask is whether this new runtime is optimal. Can we show, under some complexity theory assumption, e.g. under the Exponential Time Hypothesis, that the runtime of $\mathcal{O}^*(\mathrm{cw}(G)^{\mathrm{cw}(G)})$ is optimal for DOMINATING SET? Or can we show this for any of the other $[\sigma, \rho]$-partition problems for that matter?

In the fourth case study, we defined the parameter projection-satisfiable-width (ps-width). This parameter was a result of carefully looking at the minimal information necessary for a dynamic programming algorithm to solve MAXSAT and #SAT over a branch decomposition of the input CNF formula which. However, we did not find a good algorithm for computing a branch decomposition of low ps-width for general input formulas. We showed if the incidence graph of the formula is an interval bigraph, then we can construct a branch decomposition having ps-width upper bounded by the number of clauses in polynomial time. For what other classes of CNF formulas can we construct a branch decomposition of polynomial ps-width in polynomial time?

In Chapter 6 we showed that deciding the boolean-width of a graph is NP-hard, and deciding the mim-width of a graph is W[1]-hard and not in APX unless NP=ZPP. However, these results only show lower bounds on the complexity of these two problems. What are the complexity upper bounds of these problems? Is deciding boolean-width FPT or not? Is deciding mim-width W[1]-complete? We cannot approximate mim-width to within a constant factor in polynomial time, but can we approximate mim-width to within a constant factor in FPT time? Or can we even approximate it to within $f(mim(G))$ for any function f in FPT time? No polynomial time recognition algorithm has been found even for mim-width one graphs. Might mim-width actually be paraNP-hard[1]?

[1]Being paraNP-hard means that even when fixing the parameter to a constant, the problem is NP-hard.

Bibliography

[1] Implementation of solver, found online at `http://people.uib.no/ssa032/pswidth/`. 8.1, 8.6

[2] Ninth Max-SAT Evaluation (Max-SAT 2014), 2014. `http://www.maxsat.udl.cat/14/` accessed 16-January-2015. 8.6

[3] Amir, E. Approximation algorithms for treewidth. *Algorithmica 56*, 4 (2010), 448–479. 2.3.1, 3.1

[4] Arnborg, S., Corneil, D. G., and Proskurowski, A. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods 8*, 2 (1987), 277–284. 6.1

[5] Bacchus, F., Dalmao, S., and Pitassi, T. Algorithms and complexity results for #SAT and bayesian inference. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on* (2003), IEEE, pp. 340–351. 8.1

[6] Belmonte, R., and Vatshelle, M. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science 511* (2013), 54–65. 3, 7.4, 8.1, 8.4, 8.7, 8.5

[7] Bertossi, A. A. Dominating sets for split and bipartite graphs. *Inform. Process. Lett. 19*, 1 (1984), 37–40. 7.3.2

[8] Biere, A. Yet another local search solver and lingeling and friends entering the SAT competition 2014. *SAT Competition 2014* (2014), 39. 8.6.1

[9] Biere, A., Heule, M., and van Maaren, H. *Handbook of satisfiability*, vol. 185. IOS Press, 2009, ch. 20. 8.6

[10] Björklund, A., Husfeldt, T., Kaski, P., and Koivisto, M. Fourier meets Möbius: fast subset convolution. In *STOC'07—Proceedings of the 39th Annual ACM Symposium on Theory of Computing.* ACM, New York, 2007, pp. 67–74. 3.11

[11] Bodlaender, H., van Leeuwen, E., van Rooij, J., and Vatshelle, M. Faster algorithms on clique and branch decompositions. In *Proceedings of MFCS* (2010), vol. 6281 of *LNCS*, Springer, pp. 174–185. 7.1, 7.5

[12] Bodlaender, H. L. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (1993), ACM, pp. 226–234. 2.1, 2.3.1, 2.3.2

[13] Bodlaender, H. L., Cygan, M., Kratsch, S., and Nederlof, J. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *Proceedings ICALP*. Springer, 2013, pp. 196–207. 5.3

[14] Bodlaender, H. L., Drange, P. G., Dregi, M. S., Fomin, F. V., Lokshtanov, D., and Pilipczuk, M. An O(c^kn) 5-approximation algorithm for treewidth. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on* (2013), IEEE, pp. 499–508. 2.3.1, 1

[15] Bodlaender, H. L., Fellows, M. R., and Hallett, M. T. Beyond NP-completeness for problems of bounded width (extended abstract): hardness for the W hierarchy. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing* (1994), ACM, pp. 449–458. 1

[16] Bodlaender, H. L., van Leeuwen, E. J., van Rooij, J. M. M., and Vatshelle, M. Faster algorithms on branch and clique decompositions. In *Mathematical foundations of computer science 2010*, vol. 6281 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 2010, pp. 174–185. 3.1

[17] Booth, K. S., and Johnson, J. H. Dominating sets in chordal graphs. *SIAM J. Comput. 11*, 1 (1982), 191–199. 7.3.2

[18] Brandstädt, A., Le, V. B., and Spinrad, J. P. *Graph Classes: A Survey*, vol. 3 of *Monographs on Discrete Mathematics and Applications.* SIAM Society for Industrial and Applied Mathematics, Philadelphia, 1999. 8.4

[19] Brandstädt, A., and Lozin, V. V. On the linear structure and clique-width of bipartite permutation graphs. *Ars Comb. 67* (2003). 8.1

[20] Brault-Baron, J., Capelli, F., and Mengel, S. Understanding model counting for β-acyclic CNF-formulas. *CoRR abs/1405.6043* (2014). 2.3.4, 8.1, 8.1

[21] Bui-Xuan, B.-M., Telle, J. A., and Vatshelle, M. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discrete Applied Mathematics 158*, 7 (2010), 809–819. 7.1, 8.3

[22] Bui-Xuan, B.-M., Telle, J. A., and Vatshelle, M. Boolean-width of graphs. *Theoretical Computer Science 412*, 39 (2011), 5187–5204. 1.1.3, 2.2, 8.3, 8.3

[23] Bui-Xuan, B.-M., Telle, J. A., and Vatshelle, M. Fast dynamic programming for locally checkable vertex subset and vertex partitioning

problems. *Theoretical Computer Science 511* (2013), 66–76. 1.1.3, 2.3.3, 7, 7.1, 7.2, 7.2.1, 7.2.1, 7.2.2, 7.1, 3, 7.4

[24] Cameron, K. Induced matchings. *Discrete Applied Mathematics 24*, 1 (1989), 97–102. 6.1

[25] Cao, Y., and Marx, D. Interval deletion is fixed-parameter tractable. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (2014), pp. 122–141. 3

[26] Charbit, P., de Montgolfier, F., and Raffinot, M. Linear Time Split Decomposition Revisited. *SIAM Journal on Discrete Math. 26*, 2 (2012), 499–514. 4.2

[27] Corneil, D. G., and Rotics, U. On the relationship between clique-width and treewidth. *SIAM Journal on Computing 34*, 4 (2005), 825–847. 4.1

[28] Corneil, D. G., and Rotics, U. On the relationship between clique-width and treewidth. *SIAM J. Comput. 34*, 4 (2005), 825–847 (electronic). 7.3.1

[29] Courcelle, B. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation 85*, 1 (1990), 12–75. 2.2

[30] Courcelle, B. Clique-width of countable graphs: a compactness property. *Discrete Mathematics 276*, 1-3 (2004), 127–148. 8.1

[31] Courcelle, B., Makowsky, J. A., and Rotics, U. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems 33*, 2 (2000), 125–150. 2.1, 2.3, 2.3.2, 7.1

[32] Courcelle, B., and Olariu, S. Upper bounds to the clique width of graphs. *Discrete Appl. Math. 101*, 1–3 (2000), 77–114. 1.1.3

[33] Cunningham, W. H. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods 3*, 2 (1982), 214–228. 4.2, 4.3, 4.3

[34] Cygan, M., Fomin, F. V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. *Parameterized Algorithms.* Springer International Publishing, New York, 2016. 3.4

[35] Darwiche, A. Recursive conditioning. *Artificial Intelligence 126*, 1 (2001), 5–41. 8.1

[36] Diestel, R. *Graph theory*, fourth ed., vol. 173 of *Graduate Texts in Mathematics.* Springer, Heidelberg, 2010. 3.3.1

[37] Dregi, M. S., and Lokshtanov, D. Parameterized complexity of bandwidth on trees. In *Automata, Languages, and Programming.* Springer, 2014, pp. 405–416. 1

[38] Elbassioni, K. M., Raman, R., Ray, S., and Sitters, R. On the approximability of the maximum feasible subsystem problem with 0/1-coefficients. In *SODA* (2009), SIAM, pp. 1210–1219. 6.1

[39] Fellows, M. R., Rosamond, F. A., Rotics, U., and Szeider, S. Clique-width is np-complete. *SIAM Journal on Discrete Mathematics 23*, 2 (2009), 909–939. 6.1

[40] Fischer, E., Makowsky, J. A., and Ravve, E. V. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics 156*, 4 (2008), 511–529. 8.1

[41] Fomin, F., Golovach, P., Lokshtanov, D., and Saurabh, S. Algorithmic lower bounds for problems parameterized by clique-width. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms* (2010), pp. 493–502. 2.3.2

[42] Fomin, F., Golovach, P., Lokshtanov, D., and Saurabh, S. Intractability of clique-width parameterizations. *SIAM Journal on Computing 39*, 5 (2010), 1941–1956. 2.3.2

[43] Fomin, F. V., Lokshtanov, D., and Saurabh, S. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings SODA* (2014), pp. 142–151. 5.1, 5.3, 5.3.3, 5.3.5, 5.6, 5.3.5, 5.3.5

[44] Fredkin, E. Trie memory. *Communications of the ACM 3*, 9 (1960), 490–499. 8.3

[45] Gajarskỳ, J., Lampis, M., and Ordyniak, S. Parameterized algorithms for modular-width. In *Proceedings IPEC*. Springer, 2013, pp. 163–176. 3, 2.3.2

[46] Ganian, R. Twin-Cover: Beyond Vertex Cover in Parameterized Algorithmics. In *Parameterized and Exact Computation* (2011), Springer, pp. 259–271. 2.3.2, 4.1

[47] Ganian, R., Hliněnỳ, P., Nešetřil, J., Obdržálek, J., de Mendez, P. O., and Ramadurai, R. When trees grow low: Shrubs and fast mso1. In *Mathematical Foundations of Computer Science 2012* (2012), Springer, pp. 419–430. 2.3.2

[48] Ganian, R., Hliněný, P., and Obdržálek, J. Better algorithms for satisfiability problems for formulas of bounded rank-width. *Fundam. Inform. 123*, 1 (2013), 59–76. 8.1

[49] Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 8.1

[50] Gavril, F. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combinatorial Theory Ser. B 16* (1974), 47–56. 3.3.2

[51] Golumbic, M. C., and Rotics, U. On the clique-width of some perfect graph classes. *Internat. J. Found. Comput. Sci. 11*, 3 (2000), 423–443. 7.3.1

[52] Gurski, F. A comparison of two approaches for polynomial time algorithms computing basic graph parameters. *CoRR abs/0806.4073* (2008). 7.1

[53] Hell, P., and Huang, J. Interval bigraphs and circular arc graphs. *Journal of Graph Theory 46*, 4 (2004), 313–327. 8.1, 8.5, 8.5

[54] Hopcroft, J. E., and Karp, R. M. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing 2*, 4 (1973), 225–231. 3.3.2

[55] Hvidevold, E. M., Sharmin, S., Telle, J. A., and Vatshelle, M. Finding good decompositions for dynamic programming on dense graphs. In *IPEC* (2011), D. Marx and P. Rossmanith, Eds., vol. 7112 of *Lecture Notes in Computer Science*, Springer, pp. 219–231. 8.7

[56] Impagliazzo, R., Paturi, R., and Zane, F. Which problems have strongly exponential complexity? In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on* (1998), IEEE, pp. 653–662. 1.1.5

[57] James, L. O., Stanton, R. G., and Cowan, D. D. Graph decomposition for undirected graphs. In *Proceedings of the Third Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1972)* (1972), Florida Atlantic Univ., Boca Raton, Fla., pp. 281–290. 2.3.2

[58] Jaumard, B., and Simeone, B. On the complexity of the maximum satisfiability problem for Horn formulas. *Inf. Process. Lett. 26*, 1 (1987), 1–4. 8.1

[59] Jelínek, V. The rank-width of the square grid. *Discrete Appl. Math. 158*, 7 (2010), 841–850. 7.3.1

[60] Jeong, J., Sæther, S. H., and Telle, J. A. Maximum matching width: new characterizations and a fast algorithm for dominating set. *to appear in proceedings of IPEC 2015* (2015). 2.4

[61] Johansson, Ö. *Graph decomposition using node labels.* PhD thesis, Royal Institute of Technology, 2001. 2.2

[62] Kanté, M. M., and Rao, M. The rank-width of edge-coloured graphs. *Theory Comput. Syst. 52*, 4 (2013), 599–644. 7.1, 7.3.1

[63] Kaski, P., Koivisto, M., and Nederlof, J. Homomorphic hashing for sparse coefficient extraction. In *Proceedings of the 7th international conference on Parameterized and Exact Computation* (2012), Springer-Verlag, pp. 147–158. 8.2.2

[64] König, D. Gráfok és mátrixok. *Matematikai és Fizikai Lapok 38* (1931), 116–119. 1.1.2, 3.4

[65] Lampis, M. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica 64*, 1 (2012), 19–37. 2.3.2

[66] Le Gall, F. Powers of tensors and fast matrix multiplication. In *ISSAC 2014—Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ACM, New York, 2014, pp. 296–303. 3.1

[67] Lokshtanov, D., Marx, D., and Saurabh, S. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, 2011), SIAM, pp. 777–789. 2.3.1, 5.6, 7.1

[68] Lokshtanov, D., Marx, D., and Saurabh, S. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS 105* (2011), 41–72. 5.6, 7.3.2

[69] Lokshtanov, D., Saurabh, S., and Sikdar, S. Simpler parameterized algorithm for OCT. In *Combinatorial algorithms*, vol. 5874 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 2009, pp. 380–384. 2

[70] Luo, C., Cai, S., Wu, W., Jie, Z., and Su, K. CCLS: An efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers* (2014). 8.6

[71] Mateescu, R. Treewidth in industrial SAT benchmarks. Tech. rep., Cambridge, UK: Microsoft Research, 2011. 8.7

[72] McConnell, R. M., and Spinrad, J. P. Modular decomposition and transitive orientation. *Discrete Mathematics 201*, 1 (1999), 189–241. 2

[73] Moser, H., and Sikdar, S. The parameterized complexity of the induced matching problem. *Discrete Applied Mathematics 157*, 4 (2009), 715–727. 6.1

[74] Müller, H. Recognizing interval digraphs and interval bigraphs in polynomial time. *Discrete Applied Mathematics 78*, 1-3 (1997), 189–205. 8.1

[75] Oum, S.-i. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B 95*, 1 (2005), 79–100. 4.1

[76] Oum, S.-i. Approximating rank-width and clique-width quickly. *TALG 5*, 1 (2008), 10. 6.1, 1, 7.1

[77] Oum, S.-i. Rank-width is less than or equal to branch-width. *J. Graph Theory 57*, 3 (2008), 239–244. 7.3.2

[78] Oum, S.-i., Sæther, S. H., and Vatshelle, M. Faster algorithms for vertex partitioning problems parameterized by clique-width. *Theoretical Computer Science 535* (2014), 16–24. 2.4

[79] Oum, S.-i., and Seymour, P. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B 96*, 4 (2006), 514–528. 1.1.3, 2.1, 2.2, 2.3.1, 2.3, 3.1, 3.1, 4, 3.1, 7.1, 7.3, 7.2, 7.3.1, 9

[80] Paul, C., and Telle, J. A. Edge-maximal graphs of branchwidth k: the k-branches. *Discrete Math. 309*, 6 (2009), 1467–1475. 3, 3.3.2, 3.3.2

[81] Paulusma, D., Slivovsky, F., and Szeider, S. Model counting for CNF formulas of bounded modular treewidth. In *STACS* (2013), N. Portier and T. Wilke, Eds., vol. 20 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 55–66. 2.3.2, 8.1

[82] Provan, J. S., and Ball, M. O. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing 12*, 4 (1983), 777–788. 6.1

[83] Rabinovich, Y., Telle, J. A., and Vatshelle, M. Upper bounds on boolean-width with applications to exact algorithms. In *Parameterized and Exact Computation.* Springer, 2013, pp. 308–320. 6.1

[84] Rafiey, A. Recognizing interval bigraphs by forbidden patterns. *CoRR abs/1211.2662* (2012). 8.1, 8.5

[85] Raman, V., Ravikumar, B., and Rao, S. S. A simplified NP-complete MAXSAT problem. *Inf. Process. Lett. 65*, 1 (1998), 1–6. 8.1

[86] Rao, M. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics 308*, 24 (2008), 6157–6165. 2.2, 3, 8.1

[87] Rao, M. Solving some NP-complete problems using split decomposition. *Discrete Applied Mathematics 156*, 14 (2008), 2768–2780. 4.2

[88] Robertson, N., and Seymour, P. D. Graph minors. II. Algorithmic aspects of tree-width. *Journal of algorithms 7*, 3 (1986), 309–322. 1.1.3

[89] Robertson, N., and Seymour, P. D. Graph minors. X. Obstructions to tree-decomposition. *J. Combin. Theory Ser. B 52*, 2 (1991), 153–190. 1.1.3, 4, 2.3.1

[90] Robertson, N., and Seymour, P. D. Graph minors. XIII. The disjoint paths problem. *Journal of combinatorial theory, Series B 63*, 1 (1995), 65–110. 2.1, 2.2

[91] Roth, D. A connectionist framework for reasoning: Reasoning with examples. In *AAAI/IAAI, Vol. 2* (1996), W. J. Clancey and D. S. Weld, Eds., AAAI Press / The MIT Press, pp. 1256–1261. 8.1

[92] Sæther, S. H. Solving hamiltonian cycle by an EPT algorithm for a non-sparse parameter. In *Algorithms and Discrete Applied Mathematics.* Springer International Publishing, 2015, pp. 205–216. 2.4

[93] Sæther, S. H., and Telle, J. A. Between treewidth and clique-width. *in proceedings of WG 2014* (2014). Invited to contribute to special section of Algorithmica. 2.4, 4.2, 4.2, 5.3

[94] Sæther, S. H., Telle, J. A., and Vatshelle, M. Solving MaxSAT and #SAT by dynamic programming. *To appear in Journal of Artificial Intelligence Research* (2015). 2.4, 8.1

[95] Sæther, S. H., and Vatshelle, M. Hardness of computing width parameters based on branch decompositions over the vertex set. *to appear in proceedings of EuroComb 2015* (2015). 2.4

[96] Samer, M., and Szeider, S. Algorithms for propositional model counting. *J. Discrete Algorithms 8*, 1 (2010), 50–64. 8.1

[97] Seymour, P. D., and Thomas, R. Call routing and the ratcatcher. *Combinatorica 14*, 2 (1994), 217–241. 6.1

[98] Slivovsky, F., and Szeider, S. Model counting for formulas of bounded clique-width. In *ISAAC* (2013), L. Cai, S.-W. Cheng, and T. W. Lam, Eds., vol. 8283 of *Lecture Notes in Computer Science*, Springer, pp. 677–687. 8.1, 8.1, 8.1, 8.2.2, 8.3

[99] Szeider, S. On fixed-parameter tractable parameterizations of SAT. In *SAT 2003* (2003), E. Giunchiglia and A. Tacchella, Eds., vol. 2919 of *Lecture Notes in Computer Science*, Springer, pp. 188–202. 8.1

[100] Telle, J. A., and Proskurowski, A. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proceedings of the Third Workshop on Algorithms and Data Structures* (1993), Springer-Verlag, pp. 610–621. 7.3.2

[101] Telle, J. A., and Proskurowski, A. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math. 10*, 4 (1997), 529–550. 7.2.2

[102] Thurley, M. sharpSAT. https://sites.google.com/site/marcthurley/sharpsat accessed 16-January-2015. 8.6

[103] Thurley, M. sharpSAT–counting models with advanced component caching and implicit BCP. In *Theory and Applications of Satisfiability Testing-SAT 2006*. Springer, 2006, pp. 424–429. 8.1, 8.6

[104] Tovey, C. A. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics 8*, 1 (1984), 85–89. 1

[105] van Rooij, J. M. M., Bodlaender, H. L., and Rossmanith, P. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Algorithms—ESA 2009*, vol. 5757 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 2009, pp. 566–577. 2.3.1, 3.1, 3.5

[106] van 't Hof, P., and Villanger, Y. Proper interval vertex deletion. *Algorithmica 65*, 4 (2013), 845–867. 1

[107] Vatshelle, M. *New width parameters of graphs.* PhD thesis, The University of Bergen, 2012. 1.1.3, 1.1.3, 2.3.1, 2.1, 3, 2.1, 3.2, 3.3, 3.3.2, 7.3.2, 7.4, 8.1, 8.4, 8.7

[108] Yannakakis, M., and Gavril, F. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics 38*, 3 (1980), 364–372. 1