Multicut on Graphs of Bounded Clique-Width[*][†]

Martin Lackner, Reinhard Pichler, Stefan Rümmele, Stefan Woltran

{lackner,pichler,ruemmele,woltran}@dbai.tuwien.ac.at

Vienna University of Technology

Several variants of Multicut problems arise in applications like circuit and network design. In general, these problems are NP-complete. The goal of our work is to investigate the potential of clique-width for identifying tractable fragments of Multicut. We show for several parameterizations involving clique-width whether they lead to tractability or not. Since bounded tree-width implies bounded clique-width, our tractability results extend previous results via tree-width, in particular to dense graphs.

1 Introduction

Multicut problems are graph problems with many applications to circuit and network design, telecommunication, and recently even databases [15]. An instance of a Multicut problem is given by an undirected graph G and a set H of pairs of so-called *terminal* vertices. The aim is to find a minimum *cut* that separates all terminal pairs. Different kinds of cuts are considered. For the Edge Multicut (EMC) problem, the cut is a set of edges whose removal disconnects each terminal pair. In case of the Restricted (resp. Unrestricted) Vertex Multicut problem (RVMC resp. UVMC), the cut is a set of non-terminal (resp. arbitrary) vertices. All three variants of Multicut problems are intractable, i.e. the corresponding decision problems (asking if a cut of a given cardinality exists) are NP-complete. RVMC and EMC remain NP-hard even on trees [2, 7].

An important approach in dealing with intractable problems is to search for *fixed-parameter tractability* in order to confine the combinatorial explosion to certain problem *parameters*. More formally, we say that a problem is in the class FPT with respect to

[*]This work was supported by the Austrian Science Fund (FWF): P20704-N18.

[†]This paper is going to appear in the proceedings of the 6th Annual International Conference on Combinatorial Optimization and Applications, COCOA 2012.

Table 1: (Parameterized) Complexity of Multicut problems for various parameters

Graph classes	**UVMC**	**RVMC**	**EMC**
Interval graphs	NP-c [13]	P [13]	NP-c [11]
Trees	P [13]	NP-c [2]	NP-c [11]
Cographs	NP-c (Thm 3.1)	P (Thm 3.4)	NP-c [11][3]
Parameters[1]	**UVMC**	**RVMC**	**EMC**
m, $\|H\|$	FPT [19]	FPT [19]	FPT [19]
m	FPT [1, 20]	FPT [1, 20]	FPT [1, 20]
$tw(G)$	NP-c [2]	NP-c [2]	NP-c [11]
$tw(G \cup H)$	FPT [12]	FPT [12]	FPT [12]
$tw(G)$, $\|H\|$	FPT [13]	FPT [13]	FPT [13]
$\|H\|$	NP-c [19]	NP-c [13]	NP-c [7]
$cw(G)$	NP-c (Thm 3.1)	NP-c [2][4]	NP-c [11][3]
$cw(G \cup H)$	NP-c (Cor 3.2)	FPT[2] (Thm 4.5)	NP-c (Thm 3.3)
$cw(G)$, $\|H\|$	FPT (Thm 4.3)	FPT (Thm 4.3)	XP (Section 4.3)

[1] NP-c refers here to NP-completeness even if the parameter value is fixed by a constant.
[2] For graphs without edges between terminal vertices (cf. Section 4.2).
[3] Follows from NP-hardness of EMC on stars (trees of height 1) [11].
[4] Follows from NP-hardness of RVMC on trees [2].

a parameter k, if the problem is solvable in time $f(k) \cdot n^{O(1)}$, where n denotes the size of the input instance. The function f is usually exponential but only depends on k. The related complexity class XP contains the problems solvable in time $\mathcal{O}(n^{g(k)})$ where function g depends only on the parameter k. In general algorithms with FPT runtime are clearly preferable to those with XP runtime. In Table 1 we recall previous complexity results on various parameters of Multicut problems. For several parameters, like the size m of the cut plus cardinality $|H|$ [19] and very recently even the size m alone [1, 20], FPT membership could be shown. Also several parameterizations involving tree-width (a parameter which measures the “tree-likeness” of a graph), like the tree-width of the structure representing both G and H (denoted by $tw(G \cup H)$) [12] and the tree-width of G plus $|H|$ [13], lead to FPT membership. In contrast, for some other parameters it was shown that the Multicut problems remain NP-complete even if the parameters are bounded by a constant. This is the case for a parameterization with $tw(G)$ alone [2] and with $|H|$ alone [7, 13, 19].

We recall that all previous FPT algorithms based on tree-width are only applicable to sparse graphs. To identify FPT fragments of Multicut which additionally apply to dense graphs, we study here the parameter *clique-width*. Clique-width generalizes the class of cographs similarly as tree-width generalizes trees. A formal definition of clique-width is given in Section 2. Many graph classes are known to have bounded clique-width, such as cliques, cographs, trees, tree-cographs, probe cographs, distance-hereditary, P_4-reducible, and series-parallel graphs. Hence, all fixed-parameter tractability results w.r.t. clique-width immediately yield tractability for these graph classes. Moreover, recall the following important connection between clique-width and tree-width (shown in [3], improving a result from [6]). For every graph G, $cw(G) \leq 3 \cdot 2^{tw(G)-1} + 1$ holds. Hence, our tractability results strictly extend previous ones that are based on tree-width. Furthermore, by the relationship between clique-width and rank-width proved in [21], both our NP-completeness results and our tractability results immediately carry over to rank-width.

Our main results, as summarized in the lower part of Table 1, are as follows:

• *NP-completeness results.* The NP-completeness of Multicut parameterized by tree-width $tw(G)$ carries over to parameter $cw(G)$ by the relationship of $tw(G)$ and $cw(G)$ recalled above. However, this leaves a gap for Multicut instances with small clique-width. We fill this gap by extending NP-completeness to these cases, with the notable exception of RVMC, which we show to be tractable on graphs with $cw(G) \leq 2$. Completely new NP-hard cases arise for UVMC and EMC with respect to the parameter $cw(G \cup H)$, as opposed to the parameter $tw(G \cup H)$ for which both problems are in FPT.

• *FPT results.* We prove the FPT membership of UVMC and RVMC with respect to the parameter $cw(G) + |H|$. Under a weak additional assumption, RVMC is also shown to be in FPT with respect to $cw(G \cup H)$. It is noteworthy that these FPT results can also be obtained by exploiting the Monadic Second-Order logic (MSO) characterizations of UVMC and RVMC and by applying the clique-width metatheorem from [5]. However, by designing concrete algorithms we are able to show better upper bounds for the runtime. The presented algorithms have a runtime that is double-exponential in $cw(G)$ but only single-exponential in $|H|$. Furthermore, the runtime depends only linearly on the input size. These algorithms are therefore tailored for large instances with small clique-width and a moderate number of terminal pairs. Note that graphs of small clique-width can have unbounded tree-width. This makes our algorithms more versatile than those based on tree-width.

2 Preliminaries

For a set S and $n \in \mathbb{N}$, we write $S^{[n]}$ to denote the set of all subsets of S with cardinality n, formally $S^{[n]} := \{S' \subseteq S : |S'| = n\}$. For a set of sets S, the *union of* S, denoted $\bigcup S$, is defined as $\{a : \exists B \in S \text{ with } a \in B\}$. We consider here finite simple undirected graphs and refer to edges via two-element subsets of V. An *induced subgraph* (by a vertex set V') is denoted as $G[V']$.

An instance of a Multicut problem is given by a triple (G, H, m), where $G := (V_G, E_G)$ is a graph, $H \subseteq V_G^{[2]}$ is a set of *terminal pairs*, and m is a non-negative integer. We write $V_H := \bigcup H \subseteq V_G$ to denote the *terminal vertices* or *terminals* in such a problem. The Multicut problem asks if a *cut* of size at most m exists, which separates all terminal pairs. In case of the {UVMC, RVMC, EMC} problem, a *cut* is a set of {arbitrary vertices, non-terminal vertices, edges}. A *solution graph* is a valid graph that remains

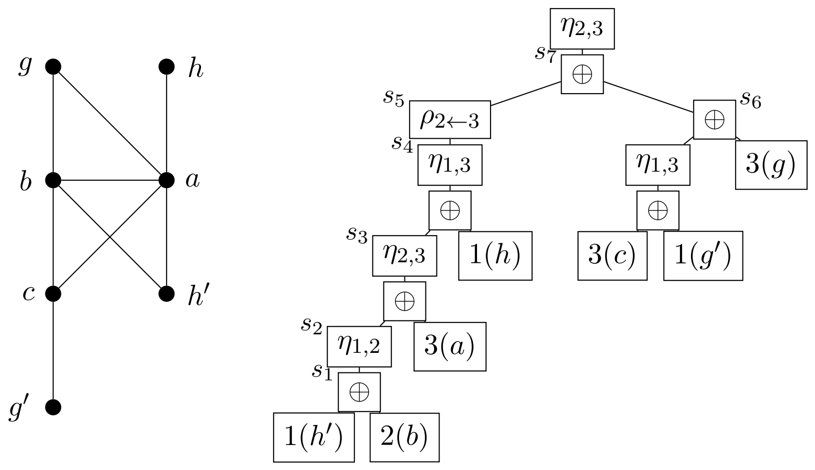

Figure 1: The graph G_{Ex} and its 3-expression.

after "cutting".

Clique-width is a graph property introduced in [4]. For a formal definition of clique-width we require *k-expressions.* Each k-expression s has a corresponding (labeled) graph $G(s)$, which is obtained by the following construction ($k \in \mathbb{N}$):

- *Adding a new vertex.* Let v be a vertex and $i \in \{1, ..., k\}$ a label. Then $i(v)$ is a k-expression and $G(i(v))$ consists of the vertex v labeled with i.
- *Renaming labels.* Let $i, j \in \{1, ..., k\}$ be labels with $i \neq j$ and let s be a k-expression. Then $\rho_{j \leftarrow i}(s)$ is a k-expression and $G(\rho_{j \leftarrow i}(s))$ is obtained by re-labeling each i-labeled vertex in $G(s)$ with j.
- *Connecting vertices.* Let $i, j \in \{1, ..., k\}$ be labels with $i \neq j$ and s a k-expression. Then $\eta_{i,j}(s)$ is a k-expression and $G(\eta_{i,j}(s))$ is obtained from $G(s)$ by connecting every i-labeled vertex with every j-labeled vertex.
- *Disjoint union.* Let s, t be k-expressions with no vertices in common. Then $s \oplus t$ is a k-expression and $G(s \oplus t)$ is the disjoint union of $G(s)$ and $G(t)$.

A graph G has *clique-width k*, written $cw(G) = k$, if k is the smallest number such that there is a k-expression s for which the unlabeled version of $G(s)$ is G. In Figure 1 we illustrate these concepts by graph G_{Ex} and the parse tree of a possible 3-expression of G_{Ex}.

In general, finding a k-expression is hard: [9] reports NP-completeness for determining whether a graph has clique-width k. However, a (possibly) sub-optimal k-expression can be computed efficiently: [21] contains an FPT algorithm which, for a given k, either concludes that $cw(G) > k$ or outputs a $(2^{3k+2} - 1)$-expression of the graph. It is an open problem whether finding a k-expression is fixed-parameter tractable with respect to k. A more detailed introduction to width parameters and graph decompositions can be found in [18].

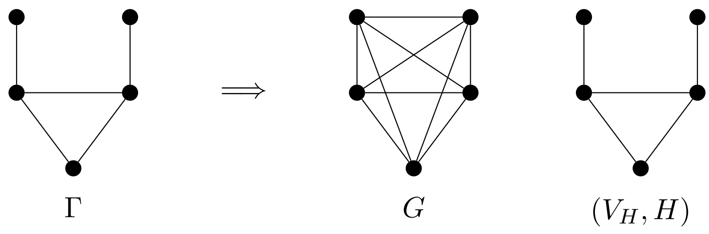

Figure 2: An example for the reduction in the proof of Theorem 3.1.

In this paper, we want to focus on the complexity of several variants of Multicut rather than the complexity of computing a k-expression. We therefore assume that an appropriate k-expression is given as part of the input. The runtime estimates in this paper are performed under the assumption that basic set operations (union, set difference, intersection) can be computed in linear time with respect to the cardinality of the sets.

3 Complexity Results

In this section we address the complexity of the problems UVMC, RVMC, and EMC with regard to the parameters $cw(G)$ and $cw(G \cup H)$. Let $G \cup H$ denote the graph $(V_G, E_G \cup H)$, i.e. the primal graph of the structure (V_G, E_G, H). Whereas some results for $cw(G)$ follow from previous work, no complexity analysis has been done for $cw(G \cup H)$-bounded graphs yet. Since graphs of clique-width 1 do not contain edges, we do not mention them in this chapter.

UVMC. UVMC is known to be NP-complete on series-parallel graphs [2], i.e. for graphs of clique-width 4. To close the gap for graphs of clique-width 2 and 3, we show that UVMC is NP-complete on cographs (and even on cliques). The analogous result for $cw(G \cup H)$-bounded graphs follows immediately.

Theorem 3.1. *UVMC remains NP-complete if G is restricted to cliques.*

Proof sketch. We use a reduction from VERTEX COVER. Let $\Gamma := (V_\Gamma, E_\Gamma)$ be an arbitrary graph for which we want to find a vertex cover of size at most k, i.e. a set of vertices that covers every edge. Our input for UVMC is $G := (V_\Gamma, V_\Gamma^{[2]})$, i.e. the complete graph on V_Γ, and $H := E_\Gamma$. An example is given in Figure 2. One can show that a set of vertices is a cut in G iff it is a vertex cover of Γ. $\square$

Corollary 3.2. *UVMC remains NP-complete if $G \cup H$ is restricted to cliques.*

These results prove NP-hardness for UVMC for a class of input instances with $cw(G) = 2$ (resp. $cw(G \cup H) = 2$) and therefore for input instances where $cw(G)$ (resp. $cw(G \cup H)$) is bounded by any number ≥ 2.

EMC. In [11] it is shown that EMC is NP-complete on trees of depth 1, i.e. stars. We show that EMC also remains NP-complete if $G \cup H$ is restricted to cliques. Since both stars and cliques are cographs and hence their clique-width is 2, it follows that EMC remains NP-complete even if either G or $G \cup H$ is restricted to graphs of clique-width 2.

Theorem 3.3. *EMC is NP-complete even if $G \cup H$ is a clique.*

Proof sketch. The NP-complete P_3-EDGE DELETION PROBLEM [8] asks if it is possible to obtain a graph containing no induced P_3, i.e. a path on 3 vertices, by deleting a given number of edges. This problem can be reduced to UVMC where $G \cup H$ is a clique. To do so, let G be the given graph and H the edge complement. Now observe that a subgraph of G contains no induced P_3 iff it consists of disjoint cliques, which holds iff no terminal pair is connected. □

RVMC. RVMC for input instances where $cw(G)$ is bounded by a number ≥ 3 is NP-complete. This follows directly from the fact that RVMC is NP-complete on trees [2] and that trees have clique-width 3. However, Theorem 3.4 shows that RVMC becomes tractable on graphs of clique-width 2. For the parameter $cw(G \cup H)$ a new situation arises, which is analyzed in Section 4.3.

Theorem 3.4. *RVMC can be solved on cographs in $\mathcal{O}(|H| \cdot |V_G|)$ time.*

Proof sketch. First note that if a cograph contains a path from a to c of length ≥ 2 then either there is a vertex b in this path such that (a, b, c) is itself a path or $\{a, c\}$ is an edge. Now, if there is a connected terminal pair in the cograph, the algorithm rejects the input. Otherwise we construct $C := \{b \in V_G : \exists (a, c) \in H \text{ s.t. } (a, b, c) \text{ is a path}\}$. Because of the aforementioned fact, C contains the vertices that have to be removed. Therefore if C contains a terminal vertex, the algorithm rejects the input. Otherwise C is the minimal cut. □

4 Algorithms

We now present FPT and XP results with several parameterizations related to clique-width. Recall from Section 2 that when dealing with clique-width, we consider a k-expression (referred to as κ) as part of the input. Furthermore, $|\kappa|$ denotes the number of operations in κ.

4.1 Vertex Multicut with $cw(G) + |H|$ as Parameter

We give an FPT algorithm based on dynamic programming for UVMC and RVMC with $cw(G) + |H|$ as parameter. The central idea of our algorithm is to keep track of the connected components of G, while G is built according to its k-expression and potential cuts are performed. Especially the connected components that contain terminal vertices are important. For this purpose, we will use the concept of *connected component sets* (CCSs), which only hold the "relevant" information on each connected component V_C, namely: (i) all terminal vertices occurring in V_C and (ii) all labels of the vertices in V_C. Observe that as long as no terminal pair is in a single connected component, the graph under consideration is a solution graph. Variations of the concept of CCSs will be used later in all following algorithms.

Definition 4.1. *Let $G = (V, E)$ be a labeled graph, let $V_H \subseteq V$ and let $L(V')$ denote the set of labels of $V' \subseteq V$. The* connected component set (CCS) *of the pair (G, H) is*

$$ccs(G, V_H) := \{(V_C \cap V_H) \cup L(V_C) : V_C \subseteq V \textit{ forms a connected component}\}.$$

Observe that in case an element of a CCS contains a terminal vertex, it corresponds to exactly one connected component. However, if an element of a CCS does not contain a terminal, several corresponding components may exist.

In the following we will give a detailed description of the algorithm. We start by describing the *data structure*, which for each subexpression s of the input k-expression κ consists of a set S_s of CCSs and a function $cuts_s$ from S_s to $\mathbb{N}$. The algorithm traverses the parse tree of the k-expression bottom-up manipulating this data structure (details are given below). The main idea of the algorithm is that for each subexpression s of κ,

$$S_s = \{ccs(G(s)[V_G \setminus C], V_H) : C \text{ is a cut w.r.t. } G(s) \wedge |C| \leq m\}.$$

This means that S_s contains all possible CCSs of solution graphs for the UVMC or RVMC problem $(G(s), H, m)$. Furthermore for each CCS $\Delta \in S_s$, $cuts_s(\Delta)$ is the minimum number of cuts required to obtain a graph represented by Δ from the original graph $G(s)$. The function *cuts* is essential to discard CCSs that have size greater than m. Since we are only interested in cuts of size $\leq m$, $cuts_s(\Delta)$ is always $\leq m$. Once the algorithm reaches the root node, which corresponds to the k-expression κ, S_κ contains all possible CCSs of solution graphs for the UVMC or RVMC problem (G, H, m). Hence there is a cut set of size $\leq m$ if and only if S_κ is not empty.

The functions *ren* and *con* will allow us to give a succinct description of the algorithm. *ren* is closely related to the ρ-operation, *con* to the η-operation.

Definition 4.2. *$ren_{i \leftarrow j}$ and $con_{i,j}$ are functions that map a CCS Δ to another CCS as follows:*

$$ren_{i \leftarrow j}(\Delta) := \{c \cup \{i\} \setminus \{j\} : c \in \Delta \textit{ with } j \in c\} \cup \{c : c \in \Delta \textit{ with } j \notin c\},$$

$$con_{i,j}(\Delta) := \{c \in \Delta : i \notin c \wedge j \notin c\} \cup \left\{\bigcup \{c \in \Delta : i \in c \vee j \in c\}\right\}.$$

We now describe what the algorithm does in each node of the parse tree of κ. The only distinction between UVMC and RVMC is in the leaves. Below, we let s (and possibly t) be the k-expression of the subtree(s) below the current (internal) node.

$i(v)$: If v is a non-terminal vertex, we have two CCSs. We define $S_{i(v)} := \{\emptyset, \{\{i\}\}\}$ with $cuts_{i(v)}(\emptyset) := 1$ and $cuts_{i(v)}(\{\{i\}\}) := 0$. If v is a terminal vertex, we do the same for UVMC but add the vertex to the CCS, i.e. $S_{i(v)} := \{\emptyset, \{\{i, v\}\}\}$, $cuts_{i(v)}(\emptyset) = 1$ and $cuts_{i(v)}(\{\{i, v\}\}) = 0$. For RVMC, we cannot remove v, thus only $S_{i(v)} := \{\{\{i, v\}\}\}$ with $cuts_{i(v)}(\{\{i, v\}\}) = 0$ remains.

$\rho_{i \leftarrow j}(s)$: $S_{\rho_{i \leftarrow j}(s)} := \{ren_{i \leftarrow j}(\Delta) : \Delta \in S_s\}$ and for each $\Delta' \in S_{\rho_{i \leftarrow j}(s)}$ we have

$$cuts_{\rho_{i \leftarrow j}(s)}(\Delta') := min\{cuts_s(\Delta) : ren_{i \leftarrow j}(\Delta) = \Delta'\}.$$

$s \oplus t$: Here we compute $S_{s\oplus t} := \{\Delta_s \cup \Delta_t : \Delta_s \in S_s, \Delta_t \in S_t, cuts_s(\Delta_s) + cuts_t(\Delta_t) \leq m\}$ and $cuts_{s\oplus t}(\Delta') := min\{cuts_s(\Delta_s) + cuts_t(\Delta_t) : \Delta_s \cup \Delta_t = \Delta'\}$.

$\eta_{i,j}$: For each $\Delta \in S_s$ there are two cases: (1) $\{i, j\} \not\subseteq \bigcup \Delta$, i.e. there is no i-labeled or no j-labeled vertex in the graphs represented by Δ. Therefore no connections are introduced and hence Δ is added to $S_{\eta_{i,j}(s)}$. (2) Otherwise we consider $con_{i,j}(\Delta)$. Here, edges might have been added such that a terminal pair was connected. If this is not the case, i.e. there is no set in $con_{i,j}(\Delta)$ which contains a terminal pair, $con_{i,j}(\Delta)$ is added to $S_{\eta_{i,j}(s)}$. In both cases – (1) and (2) – $cuts_{\eta_{i,j}(s)}(\Delta')$ is the minimum number of cuts of all CCSs in S_s that lead to Δ' if i-labeled and j-labeled vertices are connected.

Theorem 4.3. *UVMC and RVMC are FPT with respect to the parameters $cw(G)$ and $|V_H|$ and can be solved in time*

$$\mathcal{O}\left(4^{2^{cw(G)}} \cdot (|V_H| + 2^{cw(G)})^{2|V_H|} \cdot (|V_H| \cdot (cw(G) + 1) + cw(G) \cdot 2^{cw(G)}) \cdot |\kappa|\right).$$

Proof. The algorithm described above operates on the parse tree of κ. We therefore analyze the cost of each of the four operations. The operation $i(v)$ requires constant time. The operations $\eta_{i,j}(s)$ and $\rho_{i\leftarrow j}(s)$ perform basic set operations on each set in each CCS in S_s. A CCS has size at most $|V_H| \cdot (cw(G) + 1) + cw(G) \cdot 2^{cw(G)}$. That is for each element of V_H a set with at most $cw(G) + 1$ elements plus at most $2^{cw(G)}$ subsets of $cw(G)$ each having size at most $cw(G)$. In order to bound the size of S_s, we have to bound the number of possible CCSs. Recall that each $h \in V_H$ appears in at most one set in a CCS. Therefore, we can estimate the number of possible CCSs as follows. For the first $h_1 \in V_H$ there are $1 + 2^{cw(G)}$ many possibilities, since it is either not contained or it occurs together with an arbitrary subset of $cw(G)$. For the second $h_2 \in V_H$ there are at most $2 + 2^{cw(G)}$ many possibilities, since it is either not contained, or it occurs in the set of h_1, or it occurs together with an arbitrary subset of $cw(G)$. This scheme repeats for the other elements from V_H. After fixing these, we still have $2^{2^{cw(G)}}$ many possibilities for choosing arbitrary subsets of $cw(G)$. This results in less than $2^{2^{cw(G)}} \cdot (|V_H| + 2^{cw(G)})^{|V_H|}$ CCSs in S_s. In total this yields a runtime of at most $\mathcal{O}\left(2^{2^{cw(G)}} \cdot (|V_H| + 2^{cw(G)})^{|V_H|} \cdot (|V_H| \cdot (cw(G) + 1) + cw(G) \cdot 2^{cw(G)})\right)$ for η- and ρ-operations. In order to compute $s \oplus t$ we have to consider pairs of CCSs, i.e. at most $4^{2^{cw(G)}} \cdot (|V_H| + 2^{cw(G)})^{2|V_H|}$ combinations. Computing the union is bounded by the maximum size of a CCS. $\square$

Note that although the runtime depends on $|V_H|$, this can also be seen as an FPT algorithm for $cw(G) + |H|$ since $|V_H| \leq 2\,|H|$. Actually, this FPT result could also be obtained via the metatheorem of [5] and an encoding of the UVMC and RVMC problem by a Monadic Second-Order (MSO) formula in the spirit of [12]. However, a precise upper bound on the runtime in terms of an $\mathcal{O}(\cdot)$-expression would remain obscure if the FPT result were proved via the metatheorem.

Example for RVMC. We apply the algorithm described above to the RVMC instance $(G_{Ex}, H_{Ex}, 2)$, where G_{Ex} and a 3-expression of G_{Ex} are shown in Figure 1. Moreover,

Table 2: The RVMC algorithm applied to the graph in Figure 1.

	$CSSs$	cuts		$CCSs$	cuts
s_1	$\{\{1, h'\}\} \cup \{\{2\}\}$	0	s_5	$\{\{1, 2, h'\}, \{1, h\}\}$	1
$(\oplus)$	$\{\{1, h'\}\} \cup \{\ \}$	1	$(\rho_{2 \leftarrow 3})$	$\{\{1, h'\}, \{1, h\}\}$	2
s_2	$\{\{1, 2, h'\}\}$	0	s_6	$\{\{1, 3, g'\}\} \cup \{\{3, g\}\}$	0
$(\eta_{1,2})$	$\{\{1, h'\}\}$	1	$(\oplus)$	$\{\{1, g'\}\} \cup \{\{3, g\}\}$	1
s_3	$\{\{1, 2, 3, h'\}\}$	0	$s7$	$\{\{1, 2, h'\}, \{1, h\}\} \cup \{\{1, 3, g'\}, \{3, g\}\}$	1
$(\eta_{2,3})$	$\{\{1, h'\}, \{3\}\}$	1	$(\oplus)$	$\{\{1, h'\}, \{1, h\}\} \cup \{\{1, 3, g'\}, \{3, g\}\}$	2
	$\{\{1, 2, h'\}\}$	1		$\{\{1, 2, h'\}, \{1, h\}\} \cup \{\{1, g'\}, \{3, g\}\}$	2
	$\{\{1, h'\}\}$	2	root	~~$\{\{1, 2, 3, g, g', h'\}, \{1, h\}\}$~~	1
s_4	$\{\{1, 2, h'\}, \{1, h\}\}$	1	$(\eta_{2,3})$	$\{\{1, h'\}, \{1, h\}, \{1, 3, g'\}, \{3, g\}\}$	2
$(\eta_{1,3})$	$\{\{1, h'\}, \{1, h\}\}$	2		$\{\{1, 2, 3, g, h'\}, \{1, h\}, \{1, g'\}\}$	2

suppose that the terminal set is $H_{Ex} := \{\{g, g'\}, \{h, h'\}\}$. In the parse tree certain nodes are marked with $s_1, ..., s_7$. These denote the subexpressions corresponding to the subtrees rooted at these nodes. For these nodes we give the data structure S_{s_i} and the cuts function in Table 2.

The operation in node s_1 is $\oplus$. Here we take the union of the CCSs from the left and right branch. Observe that h' is present in every CCS, since it is a terminal vertex. The next operations are $\eta_{1,2}$ and $\eta_{2,3}$. Both times the components in the first CCS are merged, whereas the other CCSs do not change. In s_3 the four rows correspond to (in this order): no vertex cut, vertex b cut, vertex a cut, and both a and b cut. The s_4-operation is $\eta_{1,3}$. Since both h and h' are labeled with 1, CCSs with 3-labeled vertices yield invalid CCSs and are therefore not listed anymore. Subexpression s_6 denotes the right branch of the parse tree. The first CCS required no cuts whereas in the second CCS vertex c has been cut. The root node contains two valid CCSs and therefore there are valid cuts of size 2. The first solution corresponds to the cut set $\{a, b\}$ and the second to the cut set $\{a, c\}$. Note that in general a CCS may correspond to several cuts.

4.2 RVMC with $cw(G \cup H)$ as Parameter

In Section 3 we left open the complexity of RVMC with regard to $cw(G \cup H)$. Now we present an FPT algorithm for that problem on a slightly restricted class of inputs, namely those that do not contain edges (in G) between terminal vertices. If this restriction holds, edges between terminal vertices in $G \cup H$ always correspond to terminal pairs. Note that an edge between two terminal vertices which do not form a terminal pair in H, does not automatically prohibit a cut set. This restriction is not necessary if the following conjecture[1] holds.

[1]A stronger statement, namely that edge contractions do not increase the clique-width at all, is mentioned as an open problem in [14].

Conjecture 4.4. *There is a computable function f such that for every class of graphs $\mathcal{G}$, if the clique-width of $\mathcal{G}$ is bounded by k then the clique-width of $\mathcal{G}'$ is bounded by $f(k)$, where $\mathcal{G}'$ is the class obtained from $\mathcal{G}$ by closing $\mathcal{G}$ under edge contractions.*

If this conjecture holds, we can contract all edges in $G \cup H$ between terminal vertices a and b with $\{a, b\} \in E_G$. Then we use the $f(k)$-expression of this graph as input, i.e. the clique-width of the modified graph is still bounded. This modified graph has exactly the same cut sets as the original graph.

The algorithm presented here is based on the one in Section 4.1. Especially the *cuts* function is defined in exactly the same way, but we use a slightly different form of CCSs. Here, CCSs are subsets of $\mathcal{P}(\{1, \dots, k, \hat{1}, \dots, \hat{k}\})$. To define such CCSs, $L(V')$ is as before and $\hat{L}(V') := \{\hat{i} : i \in L(V')\}$. Now $\widehat{ccs}(G, V_H) := \{\hat{L}(V_C \cap V_H) \cup L(V_C \setminus V_H) : V_C \subseteq V \text{ forms a conn. comp.}\}$. Let $\hat{K} := \{\hat{1}, \dots, \hat{k}\}$. The data structure consists of a set S of CCSs, *forbidden sets* $N \subseteq \hat{K}^{[1]} \cup \hat{K}^{[2]}$ and a function *cuts* from S to $\mathbb{N}$. The intended meaning is that $\hat{i}$ is contained in an element of a CCS iff this component contains an i-labeled terminal. If $\{\hat{i}, \hat{j}\} \in N$, then there must not be an edge introduction between a component containing an i-labeled and one containing a j-labeled terminal. If $\{\hat{i}\} \in N$, then there must not be an edge introduction between a component containing an i-labeled terminal and any other component. The operations are as follows:

$i(v)$**:** If v is a non-terminal vertex, we have two CCSs $S_{i(v)} := \{\emptyset, \{\{i\}\}\}$. If v is a terminal vertex, $S_{i(v)} := \{\{\{\hat{i}\}\}\}$. In both cases $N := \emptyset$.

$\rho_{i \leftarrow j}(s)$**:** All CCSs are relabeled: $j \mapsto i$ and $\hat{j} \mapsto \hat{i}$. N is also changed accordingly; this might lead to N containing sets of size 1.

$s \oplus t$**:** Here $S_{s \oplus t}$ is calculated in exactly the same way as in the algorithm of Section 4.1 and $N_{s \oplus t} := N_s \cup N_t$.

$\eta_{i,j}$**:** If $\hat{i}$ and $\hat{j}$ are present in S_s, $N_{\eta_{i,j}(s)} := N_s \cup \{\hat{i}, \hat{j}\}$, otherwise $N_{\eta_{i,j}(s)} := N_s$. The reason for this is that there is no edge between terminal nodes in G. Consequently, there exists a terminal pair in H that has labels i and j. $S_{\eta_{i,j}(s)}$ is calculated similar to the original algorithm. However, checking if a terminal pair is contained in a component works differently. We have to distinguish four cases: both i and j appear in components in the CCS Δ, only i (only j) appears in a component and fourthly neither appears. The reason why the four cases only consider the occurence of i and j but not of $\hat{i}$ and $\hat{j}$ is again the precondition that in G there exists no edge between terminal nodes.

In the first case Δ is not valid iff there is an $h \in N_{\eta_{i,j}(s)}$ with $h \subseteq \bigcup\{c \in \Delta : i \in c \vee j \in c \vee \hat{i} \in c \vee \hat{j} \in c\}$, i.e. if the newly connected component is a superset of a forbidden set in $N_{\eta_{i,j}(s)}$. In the second case we have to check whether $h \subseteq \bigcup\{c \in \Delta : i \in c \vee \hat{j} \in c\}$. If this is the case a terminal pair has been connected and hence the CCS Δ has to be discarded. Case 3 works analogously to Case 2. In Case 4 no new edges are introduced. Nevertheless we have to check if not already an i- and j-labeled terminal pair is contained in a connected component.

Theorem 4.5. *RVMC is FPT with respect to $cw(G \cup H)$ on graphs that do not contain edges between terminal vertices.*

Runtime estimates can be found similarly to Theorem 4.3. It can easily be seen that Conjecture 4.4 holds for cographs. Hence,

Corollary 4.6. *RVMC is in P if $G \cup H$ is restricted to cographs.*

4.3 Edge Multicut with $cw(G) + |H|$ as Parameter

For EMC, contrary to Vertex Multicut, cuts occur during η-operations, i.e. when new edges are introduced. Therefore it is necessary to store the number of vertices in a component with a certain label. Otherwise we would not be able to calculate the number of cuts required to separate two components. However, adding this information to the data structure yields up to $n^{f(k)}$ branchings in the algorithm, where $f(k)$ is defined as k, the number of labels, times the number of components in a CCS. Clearly such an algorithm is in XP.

5 Conclusion and Future Work

We have pinpointed the parameterized complexity of the UVMC, RVMC and EMC problem for several parameterizations involving clique-width. In the literature, also *weighted versions* of Multicut have been studied where the vertices or the edges are assigned a weight and one seeks to minimize the weight rather than the cardinality of the cut. Our algorithms for UVMC and RVMC can be easily extended so as to also handle weights. In contrast, there is no obvious extension of our EMC algorithm to the weighted version of this problem. For the XP-algorithm for EMC it would be of interest to prove a corresponding W[1]-hardness result which would imply that no FPT algorithm exists.

One drawback of clique-width is that it is in general NP-complete to detect if a given graph has clique-width $\leq k$ [9]. However, there has been recent progress in identifying graph classes for which the computation of k-expression can be done in polynomial time [16, 17]. Another approach, by Oum and Seymour [21], is the notion of *rank-width*, which is strongly related to clique-width and which admits good algorithms for finding decompositions. Since $rw(G) \leq cw(G) \leq 2^{1+rw(G)}-1$ holds [21], all our NP-completeness results as well as all our FPT and XP membership results also hold for rank-width. Nevertheless, it is to be expected that custom-made algorithms for rank-width perform better than algorithms using a “detour” via clique-width. The construction of such algorithms, e.g. building on [10], is a task for future work.

Finally, the question of how much edge contractions can increase the clique-width of graphs (Conjecture 4.4) seems to be a problem worthwhile to study.

References

[1] Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 459–468, New York, 2011. ACM.

[2] Gruia Calinescu, Cristina G. Fernandes, and Bruce Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. *J. Algorithms*, 48(2):333–359, 2003.

[3] Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.

[4] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993.

[5] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

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

[7] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

[8] Ehab S. El-Mallah and Charles J. Colbourn. The complexity of some edge deletion problems. *IEEE transactions on circuits and systems*, 1988.

[9] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.

[10] Robert Ganian and Petr Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics*, 158(7):851 – 867, 2010. Third Workshop on Graph Classes, Optimization, and Width Parameters Eugene, Oregon, USA, October 2007.

[11] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and Multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[12] Georg Gottlob and Stephanie Tien Lee. A logical approach to Multicut problems. *Information Processing Letters*, 103(4):136 – 141, 2007.

[13] Jiong Guo, Falk Hüffner, Erhan Kenar, Rolf Niedermeier, and Johannes Uhlmann. Complexity and exact algorithms for Multicut. In Jirí Wiedermann, Gerard Tel,

Jaroslav Pokorný, Mária Bieliková, and Július Štuller, editors, *SOFSEM 2006: Theory and Practice of Computer Science*, volume 3831 of *LNCS*, pages 303–312. Springer Berlin / Heidelberg, 2006.

[14] Frank Gurski. Graph operations on clique-width bounded graphs. *CoRR*, abs/cs/0701185, 2007.

[15] Claudio Gutierrez, Carlos Hurtado, and Alejandro Vaisman. RDFS update: From theory to practice. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *The Semanic Web: Research and Applications*, volume 6644 of *LNCS*, pages 93–107. Springer Berlin / Heidelberg, 2011.

[16] Pinar Heggernes, Daniel Meister, and Udi Rotics. Exploiting restricted linear structure to cope with the hardness of clique-width. In Jan Kratochvíl, Angsheng Li, Jirí Fiala, and Petr Kolman, editors, *Theory and Applications of Models of Computation*, volume 6108 of *LNCS*, pages 284–295. Springer Berlin / Heidelberg, 2010.

[17] Pinar Heggernes, Daniel Meister, and Udi Rotics. Computing the clique-width of large path powers in linear time via a new characterisation of clique-width. In Alexander Kulikov and Nikolay Vereshchagin, editors, *Computer Science - Theory and Applications*, volume 6651 of *LNCS*, pages 233–246. Springer Berlin / Heidelberg, 2011.

[18] Petr Hliněný, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal*, 51(3):326–362, 2008.

[19] Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006.

[20] Dániel Marx and Igor Razgon. Fixed-parameter tractability of Multicut parameterized by the size of the cutset. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 469–478, New York, 2011. ACM.

[21] Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Comb. Theory Ser. B*, 96(4):514–528, 2006.