
TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

D I P L O M A R B E I T

# Complexity results and algorithms for Multicut on graphs of bounded clique-width

Ausgeführt am

Institut für Informationssysteme,

Abteilung für Datenbanken und Artificial Intelligence,

der Technischen Universität Wien

unter der Anleitung von

Prof. Dr. Reinhard Pichler

und

Univ.Ass. Dipl.-Ing. Stefan Rümmele

durch

Martin Lackner

1020 Wien

9. September 2010

# DECLARATION

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

*Wien, September 2010*

Martin Lackner

# COMPLEXITY RESULTS AND ALGORITHMS FOR MULTICUT ON GRAPHS OF BOUNDED CLIQUE-WIDTH

MARTIN LACKNER

lackner@dbai.tuwien.ac.at

To Dr. Johann Wruß,
who introduced me to the beauty of mathematics
and showed me that mathematics is a truly creative activity

## ABSTRACT

Multicut is an extensively studied problem in the area of algorithms on graphs. It plays an important role in different fields such as circuit design or network theory. A Multicut problem is given by a graph $G$ and the so-called terminal set which contains pairs of vertices. The aim is to find a minimal cut that separates all terminal pairs. However, even on simple graphs such as trees, Multicut is NP-complete.

Often it is not just the size of the input that makes a problem computationally hard, but certain properties of the input. These properties are used as parameters for a more detailed analysis of hard problems. Such a parameterized complexity analysis sometimes leads to *fixed parameter tractable* (FPT) algorithms, which are especially efficient when a certain parameter is small. A number of recent results have found tractable fragments of Multicut. Especially tree-width has proven to be a useful parameter. However there is a clear drawback of FPT algorithms via tree-width: the graph has to be sparse.

The goal of this thesis is to systematically study Multicut on graphs of bounded clique-width. Clique-width is a graph complexity measure similar to tree-width, but it can be small for both sparse and dense graphs. We present an efficient, fixed-parameter tractable algorithm with the size of the terminal set and the clique-width of $G$ as parameter. Furthermore an extensive complexity analysis of Multicut on graphs of bounded clique-width establishes boundaries of this approach.

We also present an extension of a metatheorem about graphs of bounded clique-width by Courcelle et al. Our extension is applicable to arbitrary structures where the clique-width of their incidence graphs is bounded. Finally we prove that a class of graphs has bounded tree-width if and only if their incidence graphs have bounded clique-width.

# ZUSAMMENFASSUNG

Multicut ist ein viel untersuchtes Problem aus dem Gebiet der Algorithmen auf Graphen. Es hat große Bedeutung in verschiedensten Gebieten, wie zum Beispiel beim Schaltungsentwurf oder in der Netzwerktheorie. Ein Multicut-Problem ist durch einen Graphen $G$ und die so genannte Terminalmenge gegeben, die Paare von Knoten enthält. Das Ziel des Multicut-Problems ist es, alle Knotenpaare in der Terminalmenge durch Schnitte im Graphen zu trennen. Dies ist ein Problem mit hoher Rechenkomplexität, da Multicut schon auf Bäumen NP-vollständig ist.

In vielen Fällen ist es nicht nur die Größe der Eingabe, die ein Problem rechnerisch komplex macht, sondern spezielle Eigenschaften der Eingabe. Diese Eigenschaften der Eingabe werden als Parameter für eine detailliertere Untersuchung von Problemen mit hoher Rechenkomplexität verwendet. Solch eine parametrisierte Komplexitätsanalyse führt manchmal zu *parametrisierbaren Algorithmen* (kurz: FPT-Algorithmen), die besonders effizient sind, wenn bestimmte Parameter klein sind. In mehreren Publikation wurden bereits FPT-Algorithmen für Multicut gefunden. Hierbei hat sich die Baumweite als besonders geeigneter Parameter herausgestellt. Allerdings haben auf Baumweite basierende FPT-Algorithmen einen klaren Nachteil: Sie funktionieren nur für Graphen mit wenigen Kanten.

Das Ziel dieser Arbeit ist es, für Multicut eine systematische Untersuchung in Bezug auf Graphen mit beschränkter Cliquenweite durchzuführen. Cliquenweite ist ein Komplexitätsmaß für Graphen ähnlich zur Baumweite mit dem bedeutenden Unterschied, dass sie auch klein für dichte Graphen sein kann. In dieser Arbeit präsentieren wir einen effizienten FPT-Algorithmus mit der Kardinalität der Terminalmenge und der Cliquenweite von $G$ als Parameter. Darüber hinaus zeigen wir mit einer umfangreichen Komplexitätsanalyse Grenzen dieses Ansatzes auf.

Wir präsentieren auch eine Erweiterung des Cliquenweite-Metatheorems von Courcelle et al. über Graphen mit beschränkter Cliquenweite. Abschließend beweisen wir noch, dass eine Klasse von Graphen genau dann beschränkte Baumweite hat, wenn ihre Inzidenzgraphen beschränkte Cliquenweite haben.

# ACKNOWLEDGMENTS

First and foremost I want to thank my advisors Reinhard Pichler, Stefan Rümmele and Stefan Woltran. I am deeply grateful for their help, guidance and inspiration. I especially want to thank them for their countless ideas how to improve the content and presentation of this thesis. It is certainly due to them that I really enjoyed working on this thesis.

I owe my deepest gratitude to my parents. Their support in the last five years gave me the possibility to focus on my studies and concentrate on pursueing my goals.

To Mimi Bruner a big "Tack!" for helping me improve this thesis, convincing me to really write the story "Ganz ohne Orakel" and for 62 other fundamentals.

I am indebted to Dr. Roswitha Wruß and Dr. Johann Wruß. Their spontaneous and devoted support significantly improved this thesis.

I especially want to thank Andreas Pfandler for helping me with the dark LaTeX arts in many ways.

Finally I would like to thank the following people for proof-reading the story "Ganz ohne Orakel" and supporting me in my decision to publish it: Johann Lackner, Peter Lackner, Robert Luh and Peter Regner.

This work was supported by the Austrian Science Fund (FWF), project P20704-N18.

# CONTENTS

# 1 INTRODUCTION

Multicut is an extensively studied problem in the area of algorithms on graphs [9, 43]. It plays an important role in different fields such as circuit design or network theory. However, even on simple graphs such as trees, Multicut is NP-complete [6, 15]. Since this is a rare property, Multicut is also very interesting from a theoretical point of view.

A Multicut problem is given by a graph $G$ and the so-called terminal set $H$ which contains pairs of vertices. The aim is to find a minimal cut that separates all terminal pairs. There are different possibilities which kind of cuts is allowed. For Edge Multicut edges may be cut, for Vertex Multicut vertices. Vertex Multicut can be restricted to disallow cutting terminal vertices. All these variations are in general NP-complete.

But what makes Multicut computationally so hard? Parameterized complexity theory, a subdiscipline of complexity theory, is studying questions of that sort. Often it is not just the size of the input that makes a problem computationally hard, but certain properties of the input. These properties are used as parameters for a more detailed analysis of hard problems. Examples of often useful parameters for graphs are the maximal vertex degree, the size of the solution, etc. Such an analysis sometimes leads to algorithms that are especially efficient when a certain parameter is small. Therefore finding such algorithms, so-called *fixed parameter tractable* (FPT) algorithms, is a major strategy for tackling NP-complete problems.

The hope for Multicut FPT algorithms has led to a search for suitable parameters. In [30] an FPT algorithm has been found, with the cardinality of the cut and the cardinality of $H$ as parameters. However, many other parameters do not yield FPT algorithms. Examples are the cardinality of $H$ alone or the tree-width of the graph. The tree-width of a graph is a measure for the sparseness and acyclicity of a graph. For many intractable problems there are FPT algorithms on graphs of bounded tree-width [10]. However, again Multicut remains stubborn. Only bounded tree-width of $G$ together with bounded cardinality of $H$ allows for an FPT algorithm [31]. A recent result by Gottlob and Lee [26] has just found an FPT algorithm for a single parameter. This parameter is the tree-width of the primal graph $G \cup H$. The primal

graph $G \cup H$ is the graph $G$ with an edge between each terminal pair. This modification introduces more cycles into the graph and hence potentially increases the tree-width. While this is a triumph over Multicut, there is a clear drawback of tree-width FPT algorithms: the graph has to be sparse.

Clique-width is a complexity measure similar to tree-width, but it can be small for both sparse and dense graphs. For example the tree-width of trees is 1, the clique-width of trees is 3. However, the clique-width of cliques is 2, whereas the tree-width is unbounded. In general bounded tree-width implies bounded clique-width, but not the other way round. This makes clique-width a stronger notion than tree-width. However, so far clique-width has not been studied in the context of Multicut.

The goal of this thesis is to do an analysis of the Multicut problem with respect to clique-width. If tractable classes can be identified, this would go beyond previous results, since it extends previous results to dense graphs. Of course larger classes also increase the possibility of intractability. Hence, FPT algorithms will be harder to obtain, but if found, they will be usable in a more general setting. The following questions arise in this context:

- Gottlob and Lee [26] have used as parameter the tree-width of the primal graph $G \cup H$. Do we get an FPT algorithm if we bound the clique-width of this primal graph?
- Since intractability for classes of bounded tree-width implies intractability for classes of bounded clique-width, we know that it will not be enough to bound only the clique-width of $G$. But is there a second parameter, such that we can obtain an FPT algorithm?
- Courcelle et al. [14] have proved a meta-theorem about graphs of bounded clique-width, guaranteeing FPT algorithms for optimization problems that are definable in the logic $MSO_1$. This theorem is not directly applicable to Multicut. Are there any ways to circumvent its limitations?

## 1.1 SUMMARY OF THE RESULTS

Surprisingly Multicut behaves very differently on classes of bounded tree- and clique-width. Since clique-width is a more general concept, it

is much harder to get FPT algorithms. This is shown with an extensive complexity analysis. In detail the results of this thesis are:

- The approach of Gottlob and Lee [26] does not work for clique-width. Vertex and Edge Multicut are NP-complete for bounded clique-width of the primal graph $G \cup H$. However, there is one interesting exception: For Vertex Multicut, if the removal of terminal vertices is disallowed and the clique-width of $G \cup H$ is 2, an efficient algorithm has been found.

- The search for an tractable fragments of the Multicut problem have been successful. For Vertex Multicut we have found an efficient FPT algorithm with clique-width of $G$ and the number of terminal pairs as parameters. The running time of the algorithm is linear with respect to the input size and single-exponential with respect to the parameters.

- We present an extension of Courcelle's theorem about graphs of bounded clique-width. Our extension is no longer limited to graphs, but can be used for optimization problem on arbitrary structures. Also a significantly larger class of optimization problems can be dealt with, namely those definable with $MSO_2$ logic. This is achieved by requiring the clique-width of the incidence graph of the structure to be bounded. $MSO_2$ is strong enough to capture all Multicut problem. Hence, we get single-parameter FPT algorithms for Vertex Multicut and even Edge Multicut.

  It is surprising that the clique-width metatheorem can be extended to $MSO_2$, a logic that is closely related to tree-width. This is explained by showing that a class of graphs with bounded clique-width of their incidence graphs has bounded tree-width. As a direct consequence we get that a class of graphs has bounded tree-width if and only if their incidence graphs have bounded clique-width.

This thesis also contains a narrative. This is an attempt to explain some of the concepts and results in this thesis in an entertaining way. It is written with a non-scientific audience in mind. The idea behind this story is to find a way to explain the intrinsic motivation of theoretical research to a broader audience. The language of this story is German.

## 1.2 ORGANIZATION

This thesis is structured as follows: Chapter 2 introduces basic concepts which appear throughout the thesis. It contains an introduction to Multicut, graph decompositions and to parameterized complexity theory. Chapter 3 contains a detailed complexity analysis of Multicut with regard to clique-width. In Chapter 4 we present a dynamic programming FPT algorithm for Vertex Multicut. Chapter 5 contains all results regarding the clique-width metatheorem. In Chapter 6 we put the results in perspective of known results and give possible directions for future work. The appendix contains the story "'Ganz ohne Orakel"', which explains some of the concepts and results in this thesis for a non-scientific audience.

# 2

# PRELIMINARIES

This chapter provides basic definitions, already known results and an introduction to concepts that will appear throughout this thesis. The first section covers basic definitions, the second introduces Multicut, the third is about graph decompositions, especially clique-width and $k$-expressions, and the fourth section is a short introduction to parameterized complexity theory.

## 2.1 BASIC DEFINITIONS

This section contains basic definitions about sets, graphs and logic.

### 2.1.1 *Sets*

**Definition 2.1.** Let $S$ be a set. The *cardinality* or *size* of a set is denoted by $|S|$. All sets in this thesis are finite. ⊣

**Definition 2.2.** For any $n \in \mathbb{N}$, $S^{[n]}$ denotes the set of all subsets of $S$ with cardinality $n$, formally $S^{[n]} \coloneqq \{S' \subseteq S : |S'| = n\}$. ⊣

**Definition 2.3.** Let $S$ be a set of sets. Then $\bigcup S$, the *union of* $S$, is defined as

$$\bigcup S \coloneqq \{a : \exists b \in S \text{ with } a \in b\}.$$ ⊣

### 2.1.2 *Graphs*

In this thesis we only consider finite simple undirected graphs. For easier notation we do not define the edge set as a subset of $V \times V$, but as a set consisting of subsets of $V$ with cardinality 2.

**Definition 2.4.** A *simple, undirected graph* $G$ is a tuple $(V, E)$. $V$ is the set of vertices. $E$ is a subset of $V^{[2]}$, i.e. $E$ contains subsets of $V$ of size 2. The intended meaning of $E$ is that the graph contains an edge between $a$ and $b$ if $\{a, b\} \in E$.

By *graph* we always mean a simple, undirected graph. ⊣

**Definition 2.5.** Let $G := (V, E)$ be a graph and $W \subseteq V$. $W^c$ is the *complementary set of vertices*, i.e. $W^c := V \backslash W$. $E^c$ denotes the *edge complement*, i.e. $E^c = V^{[2]} \backslash E$. $G^c$ is the *complement graph*, i.e. $G^c := (V, E^c)$. ⊣

**Definition 2.6.** Let $G := (V, E)$ be a graph. A tuple $(a_1, ..., a_n)$ is a *path* if for all $i \in \{1, ..., n-1\}$, $\{a_i, a_{i+1}\} \in E$ and $a_1, ..., a_n$ are pairwise distinct. The length of a path $(a_1, ..., a_n)$ is $n - 1$.

We define the *distance* $d(a, b)$ between two vertices $a$ and $b$ in a graph as the length of the shortest path between them. If there is no path between two vertices, the distance is $\infty$.

The *diameter* of a graph is defined as $\max_{a,b \in V} d(a, b)$.

A *connected component* in a graph is a set of vertices $S$ such that for each $a \in S$,

$$d(a, b) = \begin{cases} \text{an integer} & b \in S \\ \infty & b \notin S. \end{cases}$$

A graph is *connected* if it has exactly one connected component.

The *degree* of a vertex $a$ is $|\{b : d(a, b) = 1\}|$, i.e. the number of all adjacent vertices. ⊣

**Definition 2.7.** Let $G := (V, E)$ be a graph and $V' \subseteq V$. A *$V'$-induced subgraph* is a graph $(V', E')$ with $E' := \{\{a, b\} \in E : a \in V' \wedge b \in V'\}$. We denote the $V'$-induced subgraph by $G[V']$ or $(V, E)[V']$. ⊣

**Definition 2.8.** Let $G_1 := (V_1, E_1)$ and $G_2 := (V_2, E_2)$. If $V_1 \cap V_2 = \emptyset$, then the *disjoint union* $G_1 \cup G_2$ is defined as the graph $(V_1 \cup V_2, E_1 \cup E_2)$. ⊣

**Definition 2.9.** A *clique* or *complete graph* is a graph of the form $(V, V^{[2]})$, i.e. a graph where any pair of vertices is connected by an edge. ⊣

An important graph class in this thesis are cographs. The following definitions and characterizations can be found for example in [5].

**Definition 2.10.** A *cograph* is a graph constructed by the following rules:

- A single vertex graph is a cograph.
- If $G$ is a cograph, then the complement graph $G^c$ is also a cograph.
- If $G_1$ and $G_2$ are cographs, then their disjoint union $G_1 \cup G_2$ is a cograph.

There are other characterizations of cographs. We list some of them:

- Cographs are exactly those graphs that are $P_4$-free. That means that there are no four vertices $a$, $b$, $c$ and $d$, such that the induced subgraph $G[\{a, b, c, d\}]$ is isomorphic to $P_4$, the graph consisting of a path with 4 vertices.

  Another formulation is that whenever there is a path on four vertices $(a, b, c, d)$, then either $(a, c)$ or $(b, d)$ or $(a, d)$ have to be an edge in the graph.

- Cographs are all graphs whose connected induced subgraphs have diameter at most 2. ⊣

### 2.1.3 *Logic*

Two logics play an important role in this thesis, the *monadic second-order* logics $MSO_1$ and $MSO_2$. Both are fragments of second-order logic. Second-order logic is an extension of first-order logic, where quantification over relations is allowed.

**Definition 2.11.** An $MSO_1$-formula is a second-order formula, where second-order quantification is only allowed for unary variables, i.e. quantification over sets. $MSO_1[\tau]$ denotes the class of $MSO_1$-formulas over $\tau$-structures. ⊣

$MSO_2$ is usually defined for graphs as $MSO_1$ with the additional possibility to quantify over subsets of the edge relation. We use a more general notion of $MSO_2$ for arbitrary structures. All structures we consider in this thesis are relational structures, i.e. their signature does not contain functions.

**Definition 2.12.** Let $\tau$ be a signature and $R \in \tau$. Every $MSO_1$-formula is also an $MSO_2$-formula. Additionally if $\phi$ is an $MSO_1$- or $MSO_2$-formula, then $\forall X \subseteq R\, \phi$ and $\exists X \subseteq R\, \phi$ are $MSO_2$-formulas. This means it is allowed to quantify over subsets of relations. As before $MSO_2[\tau]$ denotes the class of $MSO_2$-formulas over $\tau$-structures. ⊣

In the special case when talking about graphs, i.e. the signature is $\{E\}$, $MSO_1$ is first order logic where additionally quantification over sets of vertices is allowed. $MSO_2$ extends $MSO_1$ with the option to quantify also over subsets of $E$.

In Example 2.36 we will see an exemplary $MSO_1$ characterization of an algorithmic problem.

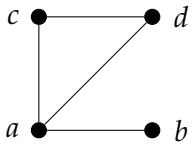


Figure 1: The primal graph of the example structure $\mathcal{S}_{Ex}$.

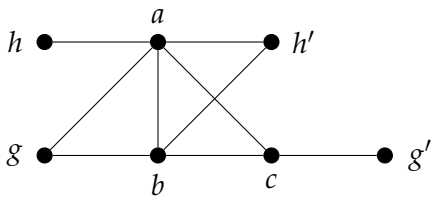


Figure 2: The graph $G_{Ex}$ will serve as a running example throughout this thesis.

**Definition 2.13.** Let $\mathcal{S} \coloneqq (U, R_1, ..., R_n)$ be a structure. The *primal graph* (or Gaifman graph) of $\mathcal{S}$ contains a vertex for each element in the domain $U$. Let $v$ and $w$ be vertices. The set $\{v, w\}$, $v \neq w$, is an edge in the primal graph if and only if one of the relations $\{R_1, ..., R_n\}$ contains a tuple $(a_1, ..., a_r)$ with $v, w \in \{a_1, ..., a_r\}$. ⊣

**Example 2.14.** Let $\mathcal{S}_{Ex} \coloneqq (U, R_1, R_2)$ be a structure with the domain $U \coloneqq \{a, b, c, d\}$, $R_1 \coloneqq \{(a, b), (b, a), (a, c)\}$ and $R_2 \coloneqq \{(a, c, d)\}$. Its primal graph can be seen in Figure 1. ◇

## 2.2 MULTICUT PROBLEMS

This section explains and defines the Multicut problem. We start with an exemplary Multicut problem, which is followed by formal definitions.

**Example 2.15.** Given is the graph $G_{Ex}$ shown in Figure 2 and the so-called terminal set $H_{Ex} \coloneqq \{\{g, g'\}, \{h, h'\}\}$. $G_{Ex}$ and $H_{Ex}$ will serve as a running example throughout this thesis.

In general it is our aim to disconnect all terminal pairs, i.e. $\{g, g'\}$ and $\{h, h'\}$. That means that there must not be a path from $g$ to $g'$ and no path from $h$ to $h'$.

We start with the Vertex Multicut problem, or more precisely: the *Unrestricted Vertex Multicut (UVMC)* problem. Here we are allowed to

remove arbitrary vertices to disconnect terminal pairs. It is our aim to remove as few vertices as possible. One can easily find a *cut* consisting of two vertices. For example $\{a, b\}$ and $\{a, c\}$ are cuts, but also $\{g, h\}$ or $\{g, a\}$. We can easily see here that there is no cut of cardinality 1.

The second variant of Vertex Multicut is *Restricted Vertex Multicut* (RVMC). Here we are no longer allowed to remove terminal vertices, i.e. we are just allowed to remove $a$, $b$ and $c$. There are still cuts of cardinality 2: $\{a, b\}$ and $\{a, c\}$. $\{a, b, c\}$ is of course also a valid cut, but has larger cardinality. We observe that every RVMC cut is a UVMC cut.

The last Multicut variant is *Edge Multicut (EMC)*. Now we are allowed to remove edges to disconnect terminal pairs. In our example there is exactly one cut of cardinality 2: cutting the edges $\{h, a\}$ and $\{c, g'\}$. All other cuts have larger cardinality. ◇

We continue with a formal definition of Multicut problems.

**Definition 2.16.** A Multicut instance is given by the triple $(G, H, m)$. $G := (V_G, E_G)$ is a graph. $H$ is the terminal set and consists of pairs of vertices. These pairs are called terminal pairs. $m$ is the greatest allowed cardinality of the cut. We further define $V_H := \bigcup H$, the set of all terminal vertices.

The Multicut decision problems are:

| **UNRESTRICTED VERTEX MULTICUT (UVMC)** | |
|---|---|
| *Instance:* | The Multicut graph $G := (V_G, E_G)$, the terminal set $H$, the maximal cut size $m$. |
| *Question:* | Is there a cut set $C \subseteq V_G$ with $\|C\| \leq m$, such that in the induced subgraph $G[V_G \setminus C]$ no terminal pair is connected? |

| **RESTRICTED VERTEX MULTICUT (RVMC)** | |
|---|---|
| *Instance:* | The Multicut graph $G := (V_G, E_G)$, the terminal set $H$, the maximal cut size $m$. |
| *Question:* | Is there a cut set $C \subseteq V_G \setminus V_H$ containing no terminal vertices with $\|C\| \leq m$, such that in the induced subgraph $G[V_G \setminus C]$ no terminal pair is connected? |

| **EDGE MULTICUT (EMC)** | |
|---|---|
| *Instance:* | The Multicut graph $G \coloneqq (V_G, E_G)$, the terminal set $H$, the maximal cut size $m$. |
| *Question:* | Is there a cut set $C \subseteq E_G$ with $\|C\| \leq m$, such that in the graph $(V_G, E_G \backslash C)$ no terminal pair is connected? |

Multicut *search* problems are to output a cut set of cardinality $\leq m$. ⊣

We continue with the definitions of *(minimal) cut set*, *(maximal) solution set* and *(maximal) solution graph*.

**Definition 2.17.** For UVMC and RVMC a *cut set* is a set of vertices, $C \subseteq V_G$, such that the induced graph $G[V_G \backslash C]$ contains no path between terminal pairs. For EMC a cut set is a set of edges, $C \subseteq E_G$, such that the graph $(V_G, E_G \backslash C)$ contains no path between terminal pairs. A cut set is called *minimal cut set* if there is no cut set with smaller cardinality.

A *solution set* is the complement of a cut set. For UVMC and RVMC a solution set is a set of vertices, $S \subseteq V_G$, such that the induced graph $G[S]$ contains no path between terminal pairs. For EMC a solution set is a set of edges, $S \subseteq E_G$, such that the graph $(V_G, S)$ contains no path between terminal pairs. A solution set is called *maximal solution set* if there is no solution set with larger cardinality.

For UVMC and RVMC a graph $G'$ is a *solution graph* if there is a solution set $S \subseteq V_G$ such that $G' = G[S]$. For EMC a graph $G'$ is a solution graph if there is a solution set $S \subseteq E_G$ such that $G' = (V_G, S)$. A solution graph is a *maximal solution graph* if the corresponding solution set is maximal. ⊣

Note that neither minimal cut sets, maximal solution sets nor maximal solution graphs have to be unique.

**Definition 2.18.** The predicate $UVMC(G, H, C)$ is true if $C$ is a cut set of the UVMC problem $(G, H, |C|)$.

The predicate $RVMC(G, H, C)$ is true if $C$ is a cut set of the RVMC problem $(G, H, |C|)$.

The predicate $EMC(G, H, C)$ is true if $C$ is a cut set of the EMC problem $(G, H, |C|)$. ⊣

Note that the size of $C$ does not matter for these predicates.

**Definition 2.19.** Let $G$ be a graph and $H$ a terminal set. Then $G \cup H$ is a short notation for the primal graph of the structure $(V_G, E_G, H)$, i.e. $G \cup H$ is the graph $(V_G, E_G \cup H)$. ⊣

**Theorem 2.20.** *UVMC, RVMC and EMC are NP-complete.*

This follows directly from the following stronger statements. The definition of a series-parallel graph can be found for example in [5].

**Theorem 2.21.** *UVMC is NP-complete on series-parallel graphs. [6]*

**Theorem 2.22.** *RVMC is NP-complete on trees. [6]*

**Theorem 2.23.** *EMC is NP-complete on trees. [15]*

Theorem 2.20 also follows from results proved in this thesis: NP-completeness for Vertex Multicut follows from Theorem 3.1 and from Theorem 3.5 and for Edge Multicut from Theorem 3.2 and from Theorem 3.15.

## 2.3 GRAPH DECOMPOSITIONS

Graph decompositions play an important role in parameterized complexity theory. For this thesis tree decompositions (and the corresponding parameter tree-width) and $k$-expressions (clique-width) are of special interest. However, there are many other decomposition methods and corresponding parameters: e.g. branch-width [41], path-width [42], local tree-width [23], rank-width [38], NLC-width [44], etc. For a survey paper on width-parameters see [33].

### 2.3.1 *Clique-width*

Clique-width is a complexity measure for graphs. It was originally introduced in terms of graph grammars by Courcelle, Engelfriet and Rozenberg [13]. Its name is a bit misleading. The clique-width of a graph is not only small if the graph is dense (e.g. a clique), but also if it is sparse. For example cographs have clique-width at most 2 (which include cliques), trees have clique-width at most 3 and cycle graphs have clique-width at most 4.

To define clique-width we first define $k$-expressions, which are descriptions of how to construct a graph with four operations. Each $k$-expression has a corresponding (labeled) graph, which is obtained by constructing the graph according to the $k$-expression.

**Definition 2.24.** Let $k \in \mathbb{N}$. *k-expressions* and their corresponding labeled graphs are defined recursively:

- *(Adding a new vertex)* Let $v$ be a vertex and the label $i \in \{1, ..., k\}$. Then $i(v)$ is a $k$-expression. The corresponding graph consists of the vertex $v$, which is labeled with $i$.
- *(Renaming labels)* Let the labels $i$ and $j$ be in $\{1, ..., k\}$ with $i \neq j$ and let $s$ be a $k$-expression. Then $\rho_{j \leftarrow i}(s)$ is a $k$-expression. The corresponding graph is the graph generated by $s$, where each $i$-labeled vertex is now labeled with $j$.
- *(Connecting vertices)* Let the labels $i$ and $j$ be in $\{1, ..., k\}$ with $i \neq j$ and let $s$ be a $k$-expression. Then $\eta_{i,j}(s)$ is a $k$-expression. The corresponding graph is the graph generated by $s$, where every $i$-labeled vertex is connected with every $j$-labeled vertex.
- *(Disjoint union)* Let $s$ and $t$ be $k$-expressions that have no vertices in common. Then $s \oplus t$ is a $k$-expression. The corresponding graph is the disjoint union of both graphs.

We define $G(s)$ as the corresponding labeled graph generated by the $k$-expression $s$. ⊣

**Definition 2.25.** A graph $G$ has clique-width $k$ (short: $cw(G) = k$) if $k$ is the smallest number such that there is a $k$-expression $s$ for which the unlabeled version of $G(s)$ is equal to $G$. With a slight abuse of notation, we will from now on denote such an equivalence with $G = G(s)$. ⊣

To illustrate these concepts we give a 3-expression $\kappa_{Ex}$ for our example graph $G_{Ex}$ in Figure 2. For easier readability we do not give the actual 3-expression but give the parse tree of $\kappa_{Ex}$ in Figure 3. Each node in the parse tree contains one operation and hence each subtree corresponds to a subexpression. Note that each subexpression is again a 3-expression and also has a corresponding labeled graph. In Figure 4 we show the graphs generated by the subtrees rooted in the double-framed tree nodes.

The last operation $\eta_{2,3}$ of $\kappa_{Ex}$ adds the remaining edges to the graph. This is shown in Figure 5. The new edges are the dashed lines.

This shows that we have really found a 3-expression for $G_{Ex}$. But is there also a 2-expression? The answer is no, since graphs of clique-width 2 are cographs [11] and $G_{Ex}$ is not a cograph. In order to see this

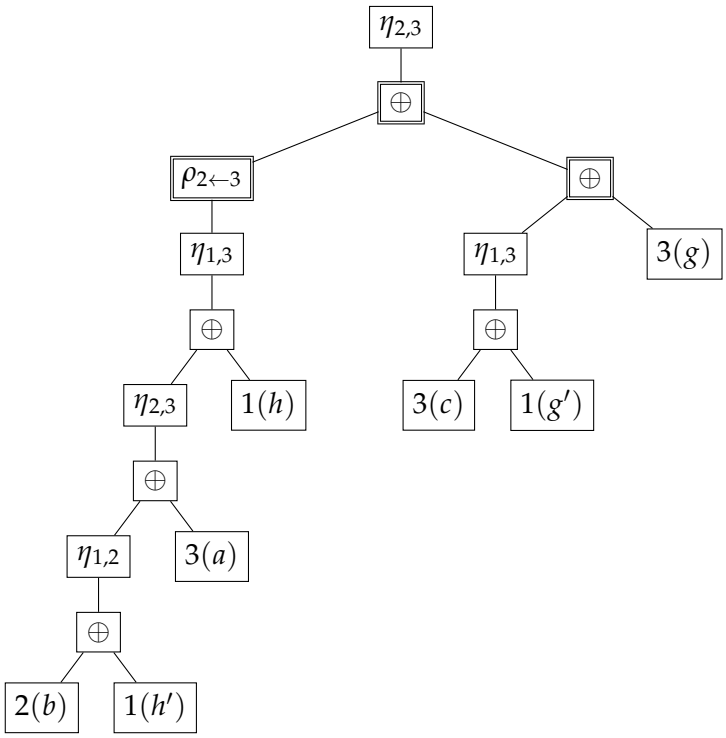


Figure 3: The 3-expression $\kappa_{Ex}$ displayed as a parse tree.

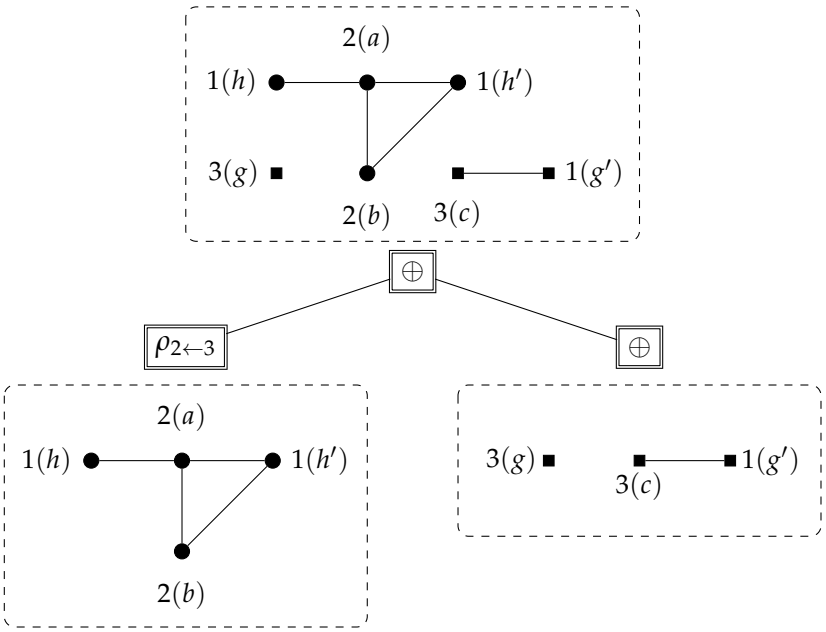


Figure 4: The graphs generated by the subtrees rooted in the double-framed tree nodes of the parse tree in Figure 3.

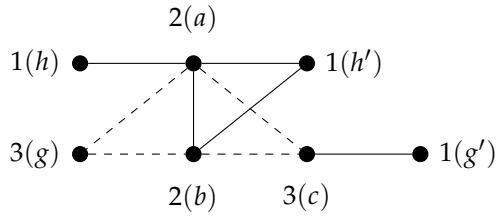


Figure 5: The example graph $G_{Ex}$ generated by $\kappa_{Ex}$. The dashed lines show the edges that have been added by the last operation $\eta_{2,3}$.

observe that $G_{Ex}[\{g, b, c, g'\}]$ is isomorphic to $P_4$ (see Definition 2.10 for details).

We conclude this introduction to clique-width with the question of how actually to find a k-expression. In general this is a hard problem. In [19] it was shown that it is NP-complete just to determine if a graph has clique-width $k$. However, it is feasible to find a non-optimal k-expression. [37] contains an algorithm which, for a given $k$, either concludes that $cw(G) > k$ or outputs a $(2^{3k+2} - 1)$-expression of the graph. Its running time is $O(|V|^4)$.

### 2.3.2 *Tree-width*

Tree-width is a notion similar to clique-width. However, contrary to clique-width, the tree-width of a graph is small only for sparse graphs. Intuitively it captures the "tree-likeness" and acyclicity of the graph. Surveys on this topic are for example [2, 3, 4].

**Definition 2.26.** A *tree decomposition* of a graph $G := (V, E)$ is a pair $(X, T)$, where $X := \{X_1, ..., X_n\}$ are subsets of $V$. $T$ is a tree, where the tree nodes are the so-called bags $X_1, ..., X_n$. Furthermore, the following properties have to hold:

1. Each $v \in V$ has to be element of at least one $X_i$, $i \in \{1, ..., n\}$.
2. Each edge $e \in E$ has to be a subset of at least one $X_i$, $i \in \{1, ..., n\}$.
3. (*Connectedness condition*) If $X_i$ and $X_j$ both contain $v \in V$, then each tree node on the path between node $X_i$ and $X_j$ has to contain $v$. ⊣

The width of a tree decomposition is the size of its largest bag minus 1. The *tree-width* of a graph $tw(G)$ is the minimum width over all possible tree decompositions. ⊣

**Example 2.27.** To illustrate this concept, we give a tree decomposition of the example graph $G_{Ex}$ in Figure 6. The width of this tree decomposition is 2, i.e. the maximal size of a bag is 3.

Corneil and Rotics [7] proved the following important connection between clique-width and tree-width, which is an improvement over the original bound proved by Courcelle and Olariu [11].

**Theorem 2.28.** ***(Corneil and Rotics [7])***

*For every graph $G$, $cw(G) \leq 3 \cdot 2^{tw(G)-1} + 1$.*

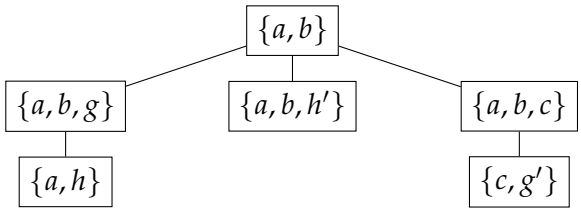


Figure 6: A tree decomposition of the example graph $G_{Ex}$

## 2.4 PARAMETERIZED COMPLEXITY THEORY

Often it is not just the size of the input that makes a problem computationally hard, but certain properties of the input. In parameterized complexity theory such properties are being used as parameters for a more detailed analysis of hard problems. Such an analysis sometimes leads to algorithms that are especially efficient when a certain parameter is small. Therefore finding such algorithms, so-called *fixed parameter tractable (FPT)* algorithms, is a major strategy for tackling NP-complete problems.

### 2.4.1 *Fixed parameter-tractability (FPT)*

In parameterized complexity theory, FPT is the most basic complexity class. It can be seen as the parameterized equivalent to PTIME. To define FPT we first introduce the notion of parameterized problems.

**Definition 2.29.** A *parameterized (decision) problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where $\Sigma$ is a finite alphabet. The second component of the input is called the *parameter*. ⊣

**Definition 2.30.** A parameterized decision problem $L$ with parameter $k$ is *fixed-parameter tractable* (FPT) if there is a deterministic Turing machine $M$ and a computable function $f$, such that $M$ decides in at most $f(k) \cdot n^{O(1)}$ steps whether the input $(x, k) \in L$.

$L$ is *fixed-parameter linear* (FPL) if this can be done in $f(k) \cdot n$ time. ⊣

There are other parameterized complexity classes, even hierarchies. These are not mentioned here, since they do not appear in this thesis. An extensive treatment can be found in [16, 21].

### 2.4.2 *Metatheorems*

A central result concerning the usefulness of tree-width in parameterized complexity theory is Courcelle's theorem [10]. It shows that every $MSO_2$ definable problem is in FPL with regard to the tree-width of the input instance. Such a result is called a *metatheorem*. For surveys on metatheorems and their applications see [29, 34].

In detail Courcelle's theorem states:

**Theorem 2.31.** ***(Courcelle [10])*** *Let $\tau$ be a signature, $C$ a set of $\tau$-structures and $\Phi$ an $MSO_2[\tau]$-formula. Then there is a fixed-parameter linear algorithm with regard to the tree-width of the primal graph of $\mathcal{A} \in C$, which decides $\mathcal{A} \models \Phi$.*

There are many extensions of this theorem, most notably with respect to counting, enumeration and optimization problems [1, 25].

Since this thesis is mostly concerned with clique-width, a similar result by Courcelle, Makowsky and Rotics for clique-width is of greater importance here. We will state the optimization version. For optimization problems we have to specify which results are "optimal" for us. To give this a precise meaning we introduce linear evaluation functions.

**Definition 2.32.** Let $U$ be the domain of the underlying structure. A $k$-ary *linear evaluation function* is a function from $U^k$ to $\mathbb{Z}$. Let $f$ be a $k$-ary evaluation function and $R$ a set of $k$-tuples, i.e. $R$ is a relation. Then $f(R) := \sum_{r \in R} f(r)$. ⊣

We furthermore have to define the notion of $p$-graphs.

**Definition 2.33.** A *$p$-graph* $\mathcal{G}$ is a relational structure $(V, E, P_1, ..., P_p)$, such that the pair $(V, E)$ is a graph and the relations $P_1, ..., P_p$ are unary. Furthermore, let $\tau_p$ denote the signature of $\mathcal{G}$. ⊣

Note that a $p$-graph can be seen as a vertex-colored graph, which allows more than one color per vertex.

As opposed to tree-width, the clique-width metatheorem only allows $MSO_1$ characterizations of problems. This weakens this theorem regarding the expressiveness, but on the other hand clique-width is a more general notion and hence the theorem guarantees FPT results for a larger class of graphs. The actual connection between $MSO_1$ and clique-width is an open question, however partial results have been found e.g. in [12].

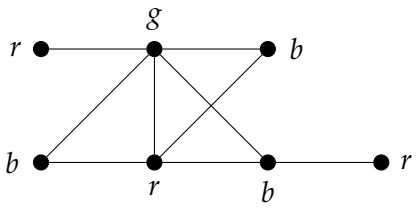


Figure 7: A 3-coloring of the example graph $G_{Ex}$.

**Definition 2.34.** For a fixed $p \in \mathbb{N}$ let $\Phi$ be an $MSO_1[\tau_p]$-formula with free unary second-order variables $S_1, ..., S_n$. Then $LinEMSO_1(\Phi)$ is defined as

| $LinEMSO_1(\Phi)$ | |
|---|---|
| *Instance:* | A $p$-graph $\mathcal{G} := (V, E, P_1, ..., P_p)$, a $k$-expression for the graph $(V, E)$, evaluation functions $g_1, ..., g_n$ for the free variables of $\Phi$ and $opt \in \{min, max\}$. |
| *Parameter:* | $k$. |
| *Output:* | Sets of vertices $T_1, ..., T_n$ such that $\mathcal{G} \models \Phi[T_1, ..., T_n]$ and $\sum_{i=1}^{n} g_i(T_i)$ is maximal or minimal (depending on $opt$). |

⊣

**Theorem 2.35.** ***(Courcelle, Makowsky, Rotics [14])*** *Let $\Phi$ be an arbitrary but fixed $MSO_1$-formula. Then $LinEMSO_1(\Phi)$ is fixed-parameter linear with regard to $k$.*

We continue with an example that illustrates the usefulness of Theorem 2.31 and Theorem 2.35.

**Example 2.36.** We want to apply Theorem 2.35 to the 3-colorability problem:

| **3-COLORABILITY** | |
|---|---|
| *Instance:* | A graph $G := (V, E)$ and its $k$-expression $\kappa$. |
| *Parameter:* | $k$ |
| *Question:* | Is there a vertex coloring with red, green and blue, such that no two adjacent vertices have the same color? |

The 3-colorability problem is in general NP-complete [24]. Therefore it would be desirable to find FPT algorithms. If we find an $MSO_1$ description of 3-colorability, Theorem 2.35 will directly yield an FPT result with the clique-width of $G$ as parameter. One way to formulate 3-colorability with $MSO_1$ is the following formula:

$$\exists R\, \exists G\, \exists B\, \forall x\, \forall y\ \underbrace{\big(R(x) \vee G(x) \vee B(x)\big)}_{*} \wedge \underbrace{\big(E(x,y) \rightarrow \mathit{Diff}(x,y)\big)}_{**},$$

where $\mathit{Diff}(x,y)$ is an abbreviation for

$$\neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \wedge \neg(B(x) \wedge B(y)).$$

The part of the formula marked with * checks that every node has (at least) one color. It is not a problem that vertices can have more than one color, since this makes the problem just harder. The part marked with ** checks that no two adjacent vertices have the same color.

This $MSO_1$ characterization of the problem cannot only be used for the clique-width metatheorem, but also for the tree-width metatheorem, Theorem 2.31. Thus we also get that 3-colorability is FPT with regard to $tw(G)$. ◇

However, these metatheorems have a critical drawback. Even if the parameter (tree-width or clique-width) is small, the running time of the algorithms one gets from these theorems have huge multiplicative constants. This makes these algorithms useless for practical purposes [28]. Therefore from a practical point of view these metatheorems can only be used to establish that a problem is fixed-parameter tractable. In the next section we will discuss how efficient implementations of FPT algorithms can be found with the help of dynamic programming.

### 2.4.3 *Fixed-parameter tractable algorithms via dynamic programming*

Dynamic programming is a major strategy for finding efficient fixed-parameter tractable algorithms. Dynamic programming can be used for problems that inhibit the *principle of optimality*. This means that the problem can be broken down into subproblems, which can be optimally solved independently of the sequence of operations that lead to this subproblem. Hence, the "history" of a subproblem can be discarded. The monograph *Invitation to Fixed-Parameter Algorithms* [36] offers an excellent introduction to dynamic programming and its application to parameterized complexity.

Dynamic programming is also the main strategy to solve problems, where a graph decomposition is available (or can efficiently be computed). For Multicut this has been used to get FPT counting and enumeration algorithms [39]. In this thesis we use dynamic programming for Vertex Multicut on graphs of bounded clique-width. In order to introduce this concept the following example shows how dynamic programming can be used for 3-colorability on graphs of bounded clique-width.

**Example 2.37.** The basic idea of the algorithm is to traverse the parse tree of the $k$-expression bottom-up. At each step all possible colorings of the corresponding subgraph are calculated. In order to compute these colorings for a node in the parse tree only the possible colorings of each child node are required.

**The data structure**

The underlying data structure is a set $S$ of k-tuples. This set has to be calculated for each subexpression of $\kappa$. Let $s$ be a subexpression of $\kappa$. Then we denote the set at the node $s$ in the parse tree with $S_s$. Each $k$-tuple in $S_s$ describes possible colorings of the graph $G(s)$. Each component of such a k-tuple $(a_i)_{1\leq i\leq k}$ is a subset of $\{r, g, b\}$. $a_i$ contains all colors (red, green, blue) that are being used to color i-labeled vertices. If there are no i-labeled vertices, $a_i$ is empty.

**A dynamic programming FPT algorithm for the decision problem**

The algorithm traverses the parse tree bottom-up. At the leaves of the parse tree, only vertex introductions can occur, since this is the only 0-ary operation. Internal nodes contain the other three operations $\rho$, $\oplus$ and $\eta$.

- $i(v)$: *(Adding a new vertex)* At this step only a single vertex is present in the graph $G(i(v))$, which has to have exactly one color. Hence, $S_{i(v)}$ contains three tuples:

  $$(\emptyset, ..., \{r\}, ..., \emptyset),\ (\emptyset, ..., \{b\}, ..., \emptyset) \text{ and } (\emptyset, ..., \{b\}, ..., \emptyset),$$

  where the non-empty set is at the $i$-th position.

- $\rho_{i\leftarrow j}$: *(Renaming labels)* Let $s$ be a subexpression of $\kappa$. $S_{\rho_{i\leftarrow j}(s)}$ is calculated by replacing each element $(a_1, ..., a_i, ..., a_j, ..., a_k)$ in $S_s$ with $(a_1, ..., a_i \cup a_j, ..., a_{j-1}, \emptyset, a_{j+1}, ..., a_k)$.

- $s \oplus t$: *(Disjoint union)* Let $s$ and $t$ be two $k$-expressions and $S_s$ and $S_t$ the corresponding sets of k-tuples. For each element

$(a_1, ..., a_k) \in S_s$ and $(b_1, ..., b_k) \in S_t$ the set $S_{s \oplus t}$ contains the tuple $(a_1 \cup b_1, ..., a_k \cup b_k)$.

- $\eta_{i,j}$: *(Connecting vertices)* Let $s$ be a subexpression of $\kappa$. $S_{\eta_{i,j}(s)}$ contains each tuple $(a_l)_{0 \leq l \leq k} \in S_s$ for which $a_i \cap a_j = \emptyset$. For tuples where this is not the case, an edge between two vertices of the same color has been introduced. Therefore this tuple no longer represents a valid 3-coloring.

There is no valid 3-coloring if and only if $S = \emptyset$ after the algorithm finishes. ◇

# 3 COMPLEXITY RESULTS

This section contains a number of complexity results about Multicut. We show that obvious choices for parameters are not suitable, i.e. Multicut remains NP-complete even if these parameters are bounded. For two special cases PTIME algorithms are presented, too.

## 3.1 GRAPHS OF BOUNDED CLIQUE-WIDTH

A natural starting point to investigate the usefulness of clique-width for Multicut problems is to ask the question:

If we bound the clique-width of $G$, is Multicut still NP-hard?

We get a rough answer directly from tree-width results. In [6] it was shown that UVMC is NP-hard on series-parallel graphs. A definition of series-parallel graphs can be found for example in [18]. These are graphs of tree-width at most 2. Because of Theorem 2.28 we know that series-parallel graphs have clique-width at most 7. Hence, UVMC is NP-hard for input instances where the clique-width of $G$ is bounded by a number $\geq 7$. This argument gives no information about cases where the clique-width is bounded by a number from 2 to 6. Note that a graph has clique-width 1 if and only if it has no edges, thus all Multicut problems are trivially tractable for input instances where $cw(G) = 1$.

For RVMC and EMC we know that both problems are NP-hard on trees. This was proven for RVMC in [6] and EMC in [25]. Trees have clique-width at most 3 and hence RVMC and EMC are NP-hard for input instances where the clique-width of $G$ is bounded by a number $\geq 3$. For graphs of clique-width 2, which are cographs, there have been no results so far.

This section fills these gaps and gives a complete complexity analysis for Multicut on graphs of bounded clique-width. We show that all non-trivial Multicut problems regardless of their bounds on clique-width are NP-complete. Surprisingly there is one exception: RVMC for graphs of clique-width $\leq 2$ is in PTIME.

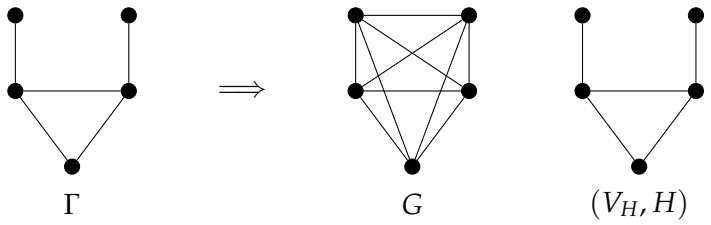


Figure 8: An example for the reduction from INDEPENDENT SET to UVMC

### 3.1.1 *UVMC*

**Theorem 3.1.** *UVMC is NP-complete for input instances where $cw(G)$ is bounded by a number $\geq 2$.*

We show this by a reduction from the following problem:

| **INDEPENDENT SET (IS)** | |
|---|---|
| *Instance:* | A graph $\Gamma := (V_\Gamma, E_\Gamma)$, an integer $l$ |
| *Question:* | Is there a subset $S \subseteq V_\Gamma$ of size at least $l$, such that the induced subgraph $\Gamma[S]$ contains no edges? |

INDEPENDENT SET was one of the early examples of NP-complete problems. CLIQUE, a very similar problem, was even one of Karp's 21 original NP-complete problems [40].

*Proof.* Let $\Gamma := (V_\Gamma, E_\Gamma)$ be an arbitrary graph for which we want to find an independent set of size at least $k$. Our input for UVMC is $G := (V_\Gamma, V_\Gamma^{[2]})$, i.e. the complete graph on $V_\Gamma$, and $H := E_\Gamma$. Figure 8 shows an example. Since $G$ is a clique, $cw(G) = 2$. We will show that $UVMC(G, H, |V_\Gamma| - l)$ is a YES-instance if and only if $IS(\Gamma, l)$ is a YES-instance. This proves NP-hardness for UVMC for a class of input instances with $cw(G) = 2$ and therefore for input instances where $cw(G)$ is bounded by a number $\geq 2$.

Assume that $UVMC(G, H, |V_\Gamma| - l)$ is a YES-instance. That means there is a solution set $S$ of size at least $l$. In the solution graph $G[S]$ no terminal pair is connected. Since each terminal pair is also an edge in $G$, at least one vertex of each terminal pair is removed. Therefore no edges are present in $\Gamma[S] = (V_G, H)[S]$ and $|S| \geq l$.

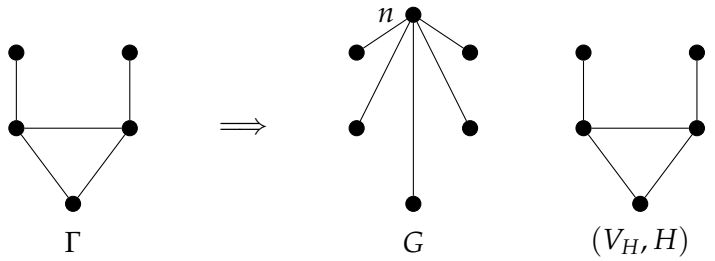


Figure 9: Example for the reduction from INDEPENDENT SET to EMC

Now let us assume that $IS(\Gamma, l)$ is a YES-instance. That means there is a set $S$ of size at least $l$ such that $\Gamma[S]$ contains no edges. We show that $S$ is a solution set. We have to check that for any terminal pair $\{a, b\} \in H$, there is no path from $a$ to $b$ in $G[S]$. However, this is trivially true because either $a \notin S$ or $b \notin S$, since $\Gamma[S]$ contains no edges. Therefore we have found a solution set of size at least $l$ and its complement is a cut set of size at most $|V_\Gamma| - l$. □

### 3.1.2 *EMC*

**Theorem 3.2.** *EMC is NP-complete for input instances where $cw(G)$ is bounded by a number $\geq 2$.*

*Proof.* Let $\Gamma := (V_\Gamma, E_\Gamma)$ be an arbitrary graph for which we want to find an independent set of size at least $k$. The input graph for EMC is $G := (V_\Gamma \cup \{n\}, E_G)$, where $n$ is an additional vertex not present in $V_\Gamma$ and $E_G := \{\{a, n\} : a \in V_\Gamma\}$. Figure 9 shows an example. Assume that $V_\Gamma := \{v_1, ..., v_g\}$. $G$ can be constructed by the following 2-expression using labels 1 and 2: $\eta_{1,2}\big(1(v_1) \oplus ... \oplus 1(v_g) \oplus 2(n)\big)$. Therefore $cw(G) = 2$. $H$, the second part of the input for EMC, is $E_\Gamma$. The intended meaning is that the vertex $v_i$ is element of the independent set if and only if the edge $\{v_i, n\}$ is in the EMC solution set. This relation between EMC and IS will allow us to show that $EMC(G, H, |V_\Gamma| - k)$ is a YES-instance if and only if $IS(\Gamma, k)$ is a YES-instance. This proves NP-hardness for EMC for a class of input instances with $cw(G) = 2$ and therefore for input instances where $cw(G)$ is bounded by a number $\geq 2$.

Assume that $EMC(G, H, |V_\Gamma| - k)$ is a YES-instance. That means there is a solution set $S \subseteq E_G$ of size at least $k$. In the solution graph $(V_G, S)$

no terminal pair is connected. We will show that $I \coloneqq \{v : \{v, n\} \in S\}$ is an independent set. Since each terminal pair $\{a, b\}$ is connected by the path $(a, n, b)$ in $G$, either the edge $\{a, n\}$ or $\{n, b\}$ or both have to be removed. This guarantees that in $\Gamma$ for each edge $\{e_1, e_2\} \in E_\Gamma$ either $e_1 \notin I$ or $e_2 \notin I$ or both. Therefore $I$ is an independent set. Furthermore, $|I| = |S| \geq k$.

Now we assume that $IS(\Gamma, k)$ is a YES-instance. That means there is a set $I \subseteq V_\Gamma$ of size at least $k$ such that $\Gamma[I]$ contains no edges. We show that $S \coloneqq \{\{v, n\} : v \in I\}$ is a solution set for the EMC problem. For this purpose we have to check that for any terminal pair $\{a, b\} \in H$, there is no path from $a$ to $b$ in $(V_G, S)$. Because each edge in $G$ contains $n$, $(a, n, b)$ is the only path from $a$ to $b$. Furthermore, we know that $H = E_\Gamma$ that means that for any terminal pair $\{a, b\}$ either $a \notin I$ or $b \notin I$ or both. Hence, either $\{a, n\} \notin S$ or $\{b, n\} \notin S$. Since $|S| = |I| \geq k$ we have found a solution set of size at least $k$ and its complement is a cut set of size at most $|V_\Gamma| - k$. □

### 3.1.3 *RVMC*

RVMC for input instances where $cw(G)$ is bounded by a number $\geq 3$ is NP-complete. This follows directly from the fact that RVMC is NP-complete on trees [6] and that trees have clique-width 3. However, for clique-width 2 RVMC becomes tractable. We give a PTIME algorithm for cographs, which are exactly those graphs with clique-width $\leq 2$ [11].

**Theorem 3.3.** *There is a PTIME algorithm for RVMC on cographs. The running time is $O(|H| \cdot |V_G|) = O(n^2)$, where $n$ is the size of the input.*

Cographs are exactly those graphs that are $P_4$-free. That means whenever there is a path on four vertices $(a, b, c, d)$, then either $\{a, c\}$ or $\{b, d\}$ or $\{a, d\}$ have to be an edge in the graph. For the proof of the theorem, we need the following observation about cographs.

**Lemma 3.4.** *Let $G \coloneqq (V, E)$ be a cograph and $a$ and $c$ two vertices with $(a, c) \notin E$. If $G$ contains a path from $a$ to $c$ of length $\geq 2$, then there is a vertex $b$ in this path such that $(a, b, c)$ is a path.*

*Proof.* We prove this by induction on the length of the path. For paths of length 2 it is trivial. Let there be a path of length $n + 1$ with the vertices $a_1, ..., a_n, a_{n+1}$. The induction hypothesis guarantees the existence of $b \in \{a_1, ..., a_n\}$ with $\{a_1, b\} \in E$ and $\{b, a_n\} \in E$. Therefore $(a_1, b, a_n, a_{n+1})$

is a path. Since cographs are $P_4$-free and by assumption $\{a_1, a_{n+1}\} \notin E$ either $(a_1, b, a_{n+1})$ or $(a_1, a_n, a_{n+1})$ is a path. □

*Proof.* (of Theorem 3.3): We give a PTIME algorithm for the RVMC problem on cographs. The algorithm first checks if there is a terminal pair $\{a, c\}$ such that $\{a, c\} \in E_G$. In this case there is no solution for the Multicut problem and the algorithm terminates. Otherwise the algorithm constructs the set

$$C := \{b \in V_G : \exists (a, c) \in H \text{ s.t. } (a, b, c) \text{ is a path}\}.$$

$C$ contains vertices we have to remove. Therefore if $C$ contains a terminal vertex, there is no solution for the Multicut problem and the algorithm terminates. Otherwise $C$ is the minimal cut.

To prove the correctness, we first show that $C$ is a cut set. We know that in the solution graph $G[V_G \backslash C]$ there is no path between a terminal pair of length 2. We consider the case of longer paths. Let $\{a, c\} \in H$ and let $(a, b_1, ..., b_l, c)$ be a path in $G$ of length $\geq 3$. Here we know because of Lemma 3.4 that this path contains a vertex $b \in \{b_1, ..., b_l\}$, such that $(a, b, c)$ is a path in $G$. Therefore $b \in C$ and $(a, b_1, ..., b_l, c)$ is not a path in the solution graph $G[V_G \backslash C]$.

It remains to show that $C$ is a minimal cut set. We show that if $C'$ is a cut set then $C \subseteq C'$. Towards a contradiction assume the existence of $b \in C$ with $b \notin C'$. Then there is a terminal pair $\{a, c\}$ such that $(a, b, c)$ is a path in $G$. $(a, b, c)$ is also a path in $G[V_G \backslash C']$ and therefore $C'$ cannot be a cut set.

For the running time analysis we observe that the algorithm consists of two steps: checking if there exists a set $\{a, c\} \in H \cap E_G$ and constructing the set $C$. If we assume linear time for checking if an edge exists, the first step takes $O(|H|)$ time. In the second step we check for each terminal pair $\{a, c\}$ and vertex $b$, if $(a, b, c)$ is a path. This takes $O(|H| \cdot |V_G|)$ time, which is quadratic in the size of the input. □

## 3.2 CLIQUE-WIDTH OF THE PRIMAL GRAPH

Gottlob and Lee [26] showed that Multicut is fixed-parameter tractable with regard to the tree-width of the primal graph $G \cup H$. $G \cup H$ is the graph $G$ together with an edge between each terminal pair (see Definition 2.19 for details).

The analogous result for clique-width does not hold. When the clique-width of $G \cup H$ is bounded, UVMC and EMC are NP-complete.

For RVMC, if the clique-width of $G \cup H$ is bounded by 2, i.e. $G \cup H$ is a cograph, there exists a PTIME algorithm. Since a graph has clique-width 1 if and only if it has no edges, all Multicut problems are trivially tractable for input instances where $cw(G \cup H) = 1$.

When we compare these results with those of the previous section, we see that bounded clique-width of $G$ and bounded clique-width of $G \cup H$ yield the same complexity results.

$G \cup H$ can be seen as the primal graph of the structure $(V_G, E_G, H)$. Whereas the tree-width of the primal graph is a very useful parameter (see Theorem 2.31), the following proofs show that bounded clique-width of the primal graph does not allow for better results than bounded $cw(G)$. Hence, these NP-hardness results also show that clique-width and tree-width are fundamentally different concepts.

### 3.2.1 *Vertex Multicut*

**Theorem 3.5.** *UVMC is NP-complete for input instances where $cw(G \cup H)$ is bounded by a number $\geq 2$ .*

Before starting the proof of Theorem 3.5, we present a large class of NP-complete problems, which was found by Yannakakis [45]. Let $\Pi$ be a property of graphs that fulfills the following requirements:

- $\Pi$ has to hold for arbitrarily large graphs.
- Whenever $\Pi$ holds for a graph, it has to hold for each induced subgraph, too.
- $\Pi$ must not hold for every graph.

| **INDUCED SUBGRAPH WITH PROPERTY** $\Pi$ | |
|---|---|
| *Instance:* | A graph $G$, a positive integer $n \leq \|V\|$. |
| *Question:* | Is there a subset $V' \subseteq V$ with $V' \geq n$ such that the subgraph of $G$ induced by $V'$ has property $\Pi$? |

**Theorem 3.6.** ***(Yannakakis [45])*** *INDUCED SUBGRAPH WITH PROPERTY $\Pi$ is NP-hard.*

This theorem allows us to prove NP-completeness for the following problem, which will appear in the proof of Theorem 3.5.

| **INDUCED SUBGRAPH OF DISJOINT CLIQUES** | |
|---|---|
| *Instance:* | A graph $G$, a positive integer $n \leq \|V\|$. |
| *Question:* | Is there a subset $V' \subseteq V$ with $V' \geq n$ such that the subgraph of $G$ induced by $V'$ consists only of disjoint cliques? |

**Corollary 3.7.** *The INDUCED SUBGRAPH OF DISJOINT CLIQUES problem is NP-complete.*

*Proof.* This problem is obviously in NP. NP-hardness can be shown by using the fact that INDUCED SUBGRAPH WITH PROPERTY $\Pi$ is NP-hard. Our property $\Pi$ is "consists of disjoint cliques". That $\Pi$ is a valid property can easily be checked. □

*Proof.* (of Theorem 3.5): We show that UVMC is NP-complete for a class of input instances of the form $(G, H, m)$ with $cw(G \cup H) = 2$.

Let the class of input instances contain tuples $(G, H, m)$ with $G := (V_G, E_G)$, where for every set of vertices $V_G$, $E_G$ can be any subset of $V^{[2]}$ and $H := V^{[2]} \setminus E_G$. Since $G \cup H$ is a complete graph, $cw(G \cup H) = 2$.

Such a tuple $(G, H, m)$ has the special property that whenever two vertices $a$ and $b$ are not connected in $G$, the pair $\{a, b\} \in H$. This reduces the problem of finding a subset $S$ (of size at least $|V| - m$), such that the induced subgraph consists only of disjoint cliques. This is because if the $S$-induced subgraph of $G$ did not consist only of disjoint cliques, it would contain vertices $a$ and $b$ such that there is a path from $a$ to $b$ but $\{a, b\} \notin E_G$. If $(a, b) \notin E_G$ then $\{a, b\} \in H$. We have found a connected terminal pair. This contradicts the assumption that $S$ is a solution set. On the other hand let the $S$-induced subgraph consist only of disjoint cliques. Let $a$ and $b$ be two vertices. We check that $S$ is a solution of the UVMC problem. If $a$ and $b$ are both in the same clique, then $\{a, b\} \in E_G$ and hence $\{a, b\} \notin H$. If $a$ and $b$ are in different cliques, then there is no path from $a$ to $b$ because all cliques are disjoint. Therefore no terminal pair is connected.

With this equivalent formulation of the problem we can use Corollary 3.7 with input $(G, |V| - m)$. Corollary 3.7 yields that our special case of the UVMC problem is NP-complete. Hence, we have found a class of inputs with $cw(G \cup H) = 2$ for which the UVMC problem is NP-complete. This implies that it is also NP-complete for instances, where $cw(G \cup H)$ is bounded by a number $\geq 2$. □

The aim for the rest of this section is to show that there is a PTIME algorithm for RVMC if $G \cup H$ is a cograph, i.e. for input instances where $cw(G \cup H) = 2$.

We already know from Theorem 3.3 that RVMC is in PTIME for input instances where $cw(G) = 2$. However, this does not imply that RVMC is in PTIME when $cw(G \cup H) = 2$, since $cw(G)$ can be larger than $cw(G \cup H)$. Even more than that - for any graph $G$ there is a terminal set $H$, such that $cw(G \cup H) = 2$ (for example the complement of $E_H$). That means that for an input class with unbounded $cw(G)$, $cw(G \cup H)$ can be bounded. In the other direction there is no correlation either. If for an input instance $cw(G \cup H)$ is arbitrarily large, $V_G$ can be empty and hence $cw(G) = 1$. This shows that the class of input instances $(G, H, m)$ where $cw(G) = 2$ is entirely different from the class where $cw(G \cup H) = 2$.

We start with a lemma about cographs.

**Lemma 3.8.** *A cograph does not contain induced subgraphs isomorphic to $C_n$, the cycle on $n$ vertices, with $n \geq 5$.*

*Proof.* $C_n$ with $n \geq 5$ contains a path $(a, b, c, d)$ where $(a, d)$, $(b, d)$ and $(a, c)$ are not edges. This contradicts the assumption that the graph is $P_4$-free. □

It is easy to check that $C_1$,...,$C_4$ are cographs and can be subgraphs of a cograph.

We have already encountered the NP-complete decision problem INDEPENDENT SET in Theorem 3.1. The analogue search problem MAXIMAL INDEPENDENT SET will play an important role for RVMC when $cw(G \cup H) = 2$.

| **MAXIMAL INDEPENDENT SET** | |
|---|---|
| *Instance:* | A graph $\Gamma := (V_\Gamma, E_\Gamma)$. |
| *Output:* | A maximal subset $S \subseteq V_\Gamma$, such that the induced subgraph $\Gamma[S]$ contains no edges. |

**Proposition 3.9.** *The search problem MAXIMAL INDEPENDENT SET for cographs is in PTIME.*

The algorithm relies on the cotree representation of the given cograph. Details about cotrees and cographs can be found for example in [32].

**Definition 3.10.** A *cotree* is a binary tree in which the internal nodes are labeled with 0 and 1. Every cotree defines a cograph. The leaves in the cotree represent the vertices in the cograph. Furthermore, each subtree of a cotree represents an induced subgraph of the cograph. Inductively we define:

- A subtree consisting of a single leaf represents the subgraph consisting of a single vertex.
- A subtree with a 0-labeled root represents the disjoint union of the subgraphs represented by its two children.
- A subtree with a 1-labeled root represents the disjoint union of the subgraphs represented by its two children. However, also edges are added. Let $G_a$ and $G_b$ be the two children subgraphs. Then an edge is added between each vertex from $G_a$ and each vertex from $G_b$. ⊣

Note that this definition expresses exactly the graphs that can be generated by a 2-expression, which are exactly cographs. We will use this cotree representation to give a MAXIMAL INDEPENDENT SET algorithm for cographs.

*Proof.* (of Proposition 3.9) Since the algorithm follows almost immediately from the cotree representation of the cograph, we just sketch it. A cotree representation for $\Gamma$ can be found in linear time [8]. The algorithm traverses the cotree bottom-up. At each node we construct a maximal independent set with regard to the subgraph. The root set is a maximal independent set.

At leaf nodes the maximal independent sets consist of the vertex represented by that leaf. At 0-labeled nodes the set is the union of the children sets. At 1-labeled nodes we take the larger of the children sets as our next set. This algorithm requires linear time in the input size. □

**Theorem 3.11.** *RVMC is in PTIME if $G \cup H$ is a cograph, i.e. for input instances where $cw(G \cup H) = 2$.*

*Proof.* The algorithm is based on the algorithm of Theorem 3.3. However, we no longer know that $G$ is a cograph, just that $G \cup H$ is a cograph. As before we want to find the minimal cut.

To begin with, the algorithm checks for any terminal pair $\{a, b\} \in H$ if $\{a, b\} \in E_G$. If that is the case, there is no solution to the RVMC problem and the algorithm terminates.

Now the algorithm constructs a set $C$, which consists of vertices that "trivially" have to be cut. We call $C$ the *trivial set*.

First we take a look at terminal pairs. Let $\{a, b\} \in H$. If there is no path from $a$ to $b$, we can ignore this pair. Let $(a, n_1, ..., n_l, b)$ be a path in $G$. Because $\{a, b\} \in H$, $(a, n_1, ..., n_l, b)$ forms a cycle in $G \cup H$. The algorithm just looks at induced cycles in $G \cup H$, i.e. cycles $(a, n_1, ..., n_l, b)$ such that the induced subgraph has no diagonals. Because of Lemma 3.8 we know that there are only two cases: $(a, n_1, b)$ and $(a, n_1, n_2, b)$.

We start with the case $(a, n_1, n_2, b)$, where both $n_1$ and $n_2$ are terminal vertices. We cannot cut any of those vertices. Hence, if $\{n_1, n_2\} \in E_G$ then there is no solution to the Multicut problem. The algorithm terminates. If on the other hand $\{n_1, n_2\} \in H$ but $\{n_1, n_2\} \notin G$, then $(a, n_1, n_2, b)$ is not a path in $G$ and we can ignore this cycle.

Next we consider the case $(a, n_1, n_2, b)$, where either $n_1$ or $n_2$ is a terminal vertex. Without loss of generality let $n_2$ be a terminal vertex. Because $a$, $b$ and $n_2$ are terminal vertices, $n_1$ has to be cut. We add $n_1$ to the trivial set $C$.

Now we take a look at the cycles on three vertices, i.e. the case $(a, n_1, b)$. If $n_1$ is a terminal vertex it must not be cut. Hence, there is no solution to the Multicut problem and the algorithm terminates. If $n_1$ is not a terminal vertex, we cut it and add $n_1$ to $C$.

There are no more cases which allow us to find trivial cuts. This means that the trivial set $C$ can either be built by the algorithm or formally be defined as

$$C \coloneqq \big\{ n \in V_G \backslash V_H : \exists \{a, b\} \in H \text{ s.t. } (a, n, b) \text{ is a path in G} \vee \exists c \in V_H : (a, c, n, b) \text{ is a path in G} \big\}. \tag{3.1}$$

At last we consider the case $(a, n_1, n_2, b)$, where both $n_1$ and $n_2$ are not terminal vertices. It may be unclear at this point whether $n_1$ or $n_2$ or both have to be cut. One of them has certainly to be cut, because for RVMC it is not allowed to cut $a$ or $b$. If either $n_1$ or $n_2$ has been cut in previous steps, we can disregard the cycle $(a, n_1, n_2, b)$. If that is not the case, let us call such a pair $\{n_1, n_2\}$ a *nondistinctive pair*. We have to consider all nondistinctive pairs to decide which of the remaining vertices have to be cut. We build a graph $(V_N, E_N)$ that consists of all nondistinctive pairs. We call this graph *nondistinctive graph*. Formally we can define

$$E_N \coloneqq \big\{ \{n_1, n_2\} : n_1 \notin V_H \cup C \wedge n_2 \notin V_H \cup C \wedge \exists \{a, b\} \in H \text{ s.t. } (a, n_1, n_2, b) \text{ is a path in } G \big\} \tag{3.2}$$

and

$$V_N := \bigcup E_N. \tag{3.3}$$

At this point we have considered all induced cycles in $G \cup H$ but have encountered some nondistinctive pairs. The nondistinctive pairs are represented by the graph $(V_N, E_N)$. For each edge $\{n_1, n_2\}$ in $E_N$, we have to remove either $n_1$ or $n_2$. This corresponds with finding a maximal subset $D \subseteq V_N$, such that the $D$-induced subgraph of $(V_N, E_N)$ contains no edges. Therefore at least one element of each nondistinctive pair is not element of $D$. Then we cut all vertices in $V_N \backslash D$ and have hereby disconnected all paths of the form $(a, n_1, n_2, b)$. Since $D$ is a maximal subset, $V_N \backslash D$ is a minimal cut.

Now we actually have to find this maximal subset $D$. $D$ is a maximal independent set. In general the problem of finding a maximal independent set is NP-complete. However, by Proposition 3.9 finding a maximal independent set is in PTIME for cographs. We will show in Lemma 3.12 that actually $(V_N, E_N)$ is a cograph. Therefore we can find a maximal independent set $D$ in polynomial time.

Here the algorithm terminates successfully. The minimal cut set of the RVMC problem is $C \cup (V_N \backslash D)$.

In order to prove the correctness of the algorithm, we first note that every cut we make is necessary. We just have to check that we have actually disconnected all terminal pairs. Let $(a, n_1, ..., n_l, b)$ be an arbitrary path between a terminal pair $\{a, b\}$. In $G \cup H$, $(a, n_1, ..., n_l, b)$ is a cycle because $\{a, b\} \in H$. Because of Lemma 3.8 we know that there are indices $i$ and $j$ in $\{1, ..., l\}$ such that either $(a, b)$, $(a, n_i, b)$ or $(a, n_i, n_j, b)$ are induced cycles in $G \cup H$. In all these cases we have disconnected $a$ and $b$ - if this is possible at all. Therefore $(a, n_1, ..., n_l, b)$ is also disconnected. □

**Lemma 3.12.** *The nondistinctive graph $(V_N, E_N)$, as defined by Formula (3.2) and (3.3), is a cograph.*

*Proof.* To show this we will use the following characterization of a cograph:

> Cographs are all graphs whose connected induced subgraphs have diameter at most 2.

It follows immediately from this characterization that induced subgraphs of cographs are also cographs. Let us consider

$$E_N^{ind} := \{\{n_1, n_2\} \in E_G : n_1 \in V_N \wedge n_2 \in V_N,$$

where $C$ is the trivial set defined by Formula (3.1). $E_N^{ind}$ is a superset of $E_N$ and $(V_N, E_N^{ind})$ is the induced subgraph $G[V_N]$. Therefore $(V_N, E_N^{ind})$ is a cograph.

We observe that $E_N = E_N^{ind} \setminus X$ with

$$X := \{\{n_1, n_2\} \in E_G : \nexists \{a, b\} \in H \text{ s.t. } (a, n_1, n_2, b) \text{ is a path}\}.$$

Hence, we just have to show that we do not lose the cograph property by removing all edges $\{n_1, n_2\} \in X$ from $E_N^{ind}$. We do this by removing one edge at a time and by showing at each step that we still have a cograph. Let $\{n_1, n_2\}$ be an edge in $X$. We have to check that the remaining subgraph has no connected induced subgraphs with diameter $> 2$. There are three cases how this could happen:

1. A vertex $v_0$ was connected with $n_2$ by the path $(v_0, n_1, n_2)$. Now $v_0$ and $n_2$ are still connected but the shortest path is $(v_0, v_1, .., v_l, n_2)$ where $l \geq 2$.

2. A vertex $v_0$ was connected with $n_1$ by the path $(n_1, n_2, v_0)$. Now $v_0$ and $n_1$ are still connected but the shortest path is $(v_0, v_1, .., v_l, n_1)$ where $l \geq 2$.

3. Now $n_1$ and $n_2$ are still connected but the shortest path between $n_1$ and $n_2$ is $(n_1, v_1, .., v_l, n_2)$ where $l \geq 2$.

We will show that neither of those cases is possible.

**Case 1:** Because of Lemma 3.4 there has to exist $b \in \{v_1, .., v_l\}$ such that $(v_0, b, n_2)$ is a path. Hence, the distance of $v_0$ and $n_2$ is 2. This contradicts the assumption that $(v_0, v_1, .., v_l, n_2)$ was the shortest path.

**Case 2** is symmetrical to case 1.

**Case 3:** For the vertex $n_1$ to be element of $V_N$, it is required that there is a non-terminal vertex $n_3$ and terminal vertices $a$ and $b$, such that $(a, n_1, n_3, b)$ is a path in $G$. Since the edge $\{n_1, n_2\} \notin E_N$, certainly $n_3 \neq n_2$. $(n_2, n_1, n_3, b)$ is a path on four vertices in $G \cup H$ and therefore either $\{n_2, n_3\}$, $\{n_1, b\}$ or $\{n_2, b\}$ have to be edges. Figure 10 shows these cases. Since none of these edges is a pair of terminal vertices, they have to be edges in $G$ (and not elements of $H$). We will show now that all these three cases are contradictory.

- If $\{n_2, n_3\}$ was an edge in $G$, then the shortest path from $n_1$ to $n_2$ would be $(n_1, n_3, n_2)$ which contradicts the assumption that the shortest path from $n_1$ to $n_2$ has length $\geq 3$.

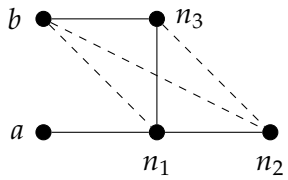


Figure 10: A sketch for case 3 from the proof of Lemma 3.1.

- If $\{n_1, b\}$ was an edge in $G$, then $(a, n_1, b)$ is a path in $G$. However, in this case $n_1$ would be element of $C$ and hence not element of $V_N$.
- If $\{n_2, b\}$ was an edge in $G$, then $(a, n_1, n_2, b)$ was a path in $G$ and therefore $\{n_1, n_2\} \in E_N$. This is a contradiction, since $\{n_1, n_2\} \in X$.

This concludes the proof. We have shown that after removing all edges in $X$, all connected induced subgraphs of $(V_N, E_N)$ still have diameter $\leq 2$. Hence, $(V_N, E_N)$ is a cograph. □

### 3.2.2 *EMC*

A result like Theorem 3.5 also holds for EMC. In the proof we use a corollary, which is a special case of the following theorem by El-Mallah and Colbourn [17].

**Theorem 3.13.** ***(El-Mallah and Colbourn [17])*** *The $P_l$-EDGE DELETION PROBLEM is NP-hard.*

| ***$P_l$-EDGE DELETION PROBLEM*** | |
|---|---|
| *Instance:* | *A graph $G := (V, E)$, positive integers $n \leq \|E\|$ and $l \geq 3$* |
| *Question:* | *Is there a set of edges $E'$ with $\|E'\| \leq n$ such that the graph $(V, E \setminus E')$ has no induced subgraph isomorphic to $P_l$?* |

$(V, E \setminus E')$ can be seen as the graph $G$, where all edges in the set $E'$ were deleted.

**Corollary 3.14.** *The DISJOINT CLIQUES EDGE DELETION PROBLEM is NP-complete.*

| ***DISJOINT CLIQUES EDGE DELETION PROBLEM*** | |
|---|---|
| *Instance:* | *A graph* $G := (V, E)$ *and positive integers* $n \leq \lvert E \rvert$ |
| *Question:* | *Is there a set of edges* $E'$ *with* $\lvert E' \rvert \leq n$ *such that the graph* $(V, E \setminus E')$ *consists of disjoint cliques?* |

*Proof.* A graph consists of disjoint cliques if and only if it does not contain $P_3$ as an induced subgraph. Hence, by Theorem 3.13 this problem is NP-hard. It is easy to see that the DISJOINT CLIQUES EDGE DELETION PROBLEM is in NP. □

**Theorem 3.15.** *EMC is NP-complete for input instances where* $cw(G \cup H)$ *is bounded by a number* $\geq 2$.

*Proof.* The proof is similar to the proof of the analogous result for UVMC (Theorem 3.5). We show that EMC is NP-hard for problem instances where $G \cup H$ is a clique. As before, we can show that this problem is equivalent to finding a subgraph consisting only of disjoint cliques. Since we are considering the EMC problem here, we do not delete vertices but edges. The question is: Is it possible to get a subgraph consisting only of disjoint cliques by removing $k$ edges. By Corollary 3.14 this problem is NP-hard and hence EMC is also NP-complete for input instances where $cw(G \cup H)$ is bounded by a number $\geq 2$. □

# 4

# AN FPT ALGORITHM FOR VERTEX MULTICUT

We have seen in the previous chapter that it is much harder to obtain tractable algorithms for graphs of bounded clique-width than for graphs of bounded tree-width. Ideas that work for tree-width, e.g. looking at the primal graph, do not work for clique-width. The aim of this chapter is to give an efficient algorithm for Vertex Multicut for graphs of bounded clique-width. The clear advantage of such an algorithm is that contrary to tree-width algorithms it can be used for both dense and sparse graphs.

More precisely speaking we give a two-parameter, dynamic programming FPT algorithm for UVMC and RVMC. The parameters are $cw(G)$ and $|H|$. Only the combination of these two parameters allows for an FPT algorithm. In the previous chapter we have seen that $cw(G)$ alone is not enough. For bounded $|H|$, Multicut also remains NP-complete [15, 35, 31].

It is reasonable to assume that both parameters are small in many situations. $cw(G)$ is small for both dense and sparse graphs and the size of $H$ does usually not depend on the input size. Therefore a typical area of application for this algorithm would be a dense graph with a small number of terminal pairs. Tree-width algorithms are not applicable in this situation, since the tree-width of dense graphs is large.

We start by introducing the notion of *connected component sets*, which will play a central role in the algorithm.

## CONNECTED COMPONENT SETS

The central idea of the Multicut algorithm presented here is to keep track of the connected components of $G$, while $G$ is built accordingly to its $k$-expression. Especially the connected components that contain terminal vertices are important. As long as no terminal pair is in a single connected component, the graph under consideration is a solution graph. The following definition captures the idea to store all information about "important" connected components.

**Definition 4.1.** Let $G$ be a graph labeled with $1, ..., k$. We define $ccs(G)$, the *connected component set* (CCS) of $G$. For this let $c_1 \subseteq V_H$ and $c_2 \subseteq \{1, ..., k\}$. The set $c_1 \cup c_2$ is an element of $ccs(G)$ if and only if there is a connected component in $G$

1. that contains exactly the terminal vertices in $c_1$ and
2. whose vertices are labeled exactly with the numbers in $c_2$. ⊣

Intuitively a CCS describes all connected components in a graph. Each set in a CCS corresponds to at least one connected component. If an element of a CCS contains an element of $V_H$, there is only one corresponding connected component. This is because a terminal vertex can be present in only one connected component. However, if an element of a CCS does not contain an element of $V_H$, there may be many corresponding connected components.

It is worth noting that every graph $G$ has exactly one corresponding CCS, although, of course, different graphs can have the same CCS.

We now want to extend this concept to Vertex Multicut.

**Definition 4.2.** Let $s$ be a $k$-expression and $G(s)$ its corresponding graph. Furthermore, let $G$ be the unlabeled version of $G(s)$. Then $CCS(G(s), H, m)$ is defined as the set of all $ccs(G(s)[S])$, where $S$ is a solution set for the UVMC/RVMC problem $(G, H, m)$. ⊣

Of course, $CCS(G(s), H, m)$ differs for a UVMC problem and for a RVMC problem. However, we do not make a clear distinction, since this notion will only be used if the statements hold for both UVMC and RVMC.

### THE ALGORITHM

We give an algorithm which is FPT with regard to $cw(G)$ and $|H|$. The algorithm relies heavily on the $k$-expression of $G$. Since we do not want to determine how the $k$-expression is found, we consider it as a part of the input.

The algorithm solves the following two problems:

**UNRESTRICTED VERTEX MULTICUT (UVMC)**

| | |
|---|---|
| *Instance:* | The Multicut graph $G := (V_G, E_G)$, its $k$-expression $\kappa$, the terminal set $H$, the maximal cut size $m$. |
| *Parameters:* | $k$ and $\lvert H\rvert$ |
| *Question:* | Is there a cut set $C \subseteq V_G$ with $\lvert C\rvert \leq m$, such that in the induced subgraph $G[V_G \backslash C]$ no terminal pair is connected? |

**RESTRICTED VERTEX MULTICUT (RVMC)**

| | |
|---|---|
| *Instance:* | The Multicut graph $G := (V_G, E_G)$, its $k$-expression $\kappa$, the terminal set $H$, the maximal cut size $m$. |
| *Parameters:* | $k$ and $\lvert H\rvert$ |
| *Question:* | Is there a cut set $C \subseteq V_G \backslash V_H$ containing no terminal vertices with $\lvert C\rvert \leq m$, such that in the induced subgraph $G[V_G \backslash C]$ no terminal pair is connected? |

The underlying data structure is a set $S$ of CCS together with a function *cuts* from $S$ to $\mathbb{N}$. The algorithm traverses the parse tree of the $k$-expression bottom-up, i.e. starting at the leaves. The subtree rooted in each tree node represents a subexpression of $\kappa$. Let $s$ be such a subexpression of $\kappa$. Then $S_s$, the set $S$ built by the subexpression $s$, contains all possible CCS of solution graphs for the UVMC or RVMC problem $(G(s), H, m)$, i.e. $S_s = CCS(G(s), H, m)$. For each CCS $\Delta \in S_s$, $cuts_s(\Delta)$ is the minimal number of cuts required to obtain a graph represented by $\Delta$ from the original graph $G(s)$. The function *cuts* is essential to discard CCS that have size greater than $m$. Since $\Delta \in S_s = CCS(G(s), H, m)$, $cuts_s(\Delta)$ is always $\leq m$.

Once the algorithm reaches the root node, which corresponds to the $k$-expression $\kappa$, $S_\kappa$ contains all possible CCS of solution graphs for the UVMC or RVMC problem $(G, H, m)$, i.e. $S_\kappa = CCS(G(\kappa), H, m)$. There is a cut set of size $\leq m$ if and only if $S_\kappa$ is not empty.

We now give a detailed description of the algorithm. We start by introducing the functions *ren* and *con* that will allow us to give a succinct description of the algorithm. *ren* is closely related to the $\rho$-operation.

**Definition 4.3.** $r_{i\leftarrow j}$ is a function from subsets of $V_H \cup \{1,...,k\}$ to subsets of $V_H \cup \{1,...,k\}\backslash\{j\}$. Let $c \subseteq V_H \cup \{1,...,k\}$. Then it is defined by

$$r_{i\leftarrow j}(c) \coloneqq \begin{cases} c \cup \{i\}\backslash\{j\} & \text{if } j \in c, \\ c & \text{if } j \notin c. \end{cases}$$

$ren_{i\leftarrow j}$ is a function that maps a CCS to another CCS. Let $\Delta$ be a CCS. Then $ren_{i\leftarrow j}(\Delta) \coloneqq \{r_{i\leftarrow j}(c) : c \in \Delta\}$. ⊣

**Example 4.4.** Let $\Delta \coloneqq \{\{a,1,2\},\{2\},\{3\}\}$ be a CCS and $a \in V_H$. Then $ren_{1\leftarrow 2}(\Delta) = \{\{a,1\},\{1\},\{3\}\}$. ◇

The other function we want to define is *con*. It is closely related to the $\eta$-operation, which often connects connected components. *con* captures this "merging" of connected components in a CCS.

**Definition 4.5.** $con_{i,j}$ is a function that maps a CCS to another CCS. Let $\Delta$ be a CCS. Then

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

⊣

**Example 4.6.** Let $\Delta \coloneqq \{\{1\},\{1,2\},\{1,3\},\{2\},\{3\}\}$ be a CCS. Then $con_{1,2}(\Delta) = \{\{1,2,3\},\{3\}\}$ and $con_{2,3}(\Delta) = \{\{1\},\{1,2,3\}\}$. ◇

Furthermore, we want to introduce the predicates *Valid* and *Vert*. Both predicates are defined for CCS. *Valid* checks if an $\eta$-operation leads again to a solution graph. This is the case if after the connect operation no terminal pair is in the same connected component.

**Definition 4.7.** Let $\Delta$ be a CCS. $Valid_{i,j}$ is defined by

$$Valid_{i,j}(\Delta) \coloneqq \begin{cases} \mathit{false} & \text{if } h \subseteq \bigcup\{c \in \Delta : i \in c \vee j \in c\} \text{ for an } h \in H, \\ \mathit{true} & \text{otherwise.} \end{cases}$$

⊣

**Example 4.8.** Let $\Delta \coloneqq \{\{1,a\},\{2,b\},\{2,3\}\}$ be a CCS, where $\{a,b\}$ is a terminal pair. Then $Valid_{1,2}(\Delta) = \mathit{false}$ and $Valid_{2,3}(\Delta) = \mathit{true}$. ◇

$Vert_i$ checks if there is an i-labeled vertex in the graphs represented by the CCS. This is the case if $i$ is an element of an element in the CCS.

**Definition 4.9.** Let $\Delta$ be a CCS. $Vert_i(\Delta)$ is defined as

$$Vert_i(\Delta) \coloneqq \begin{cases} true & \text{if } \exists c \in \Delta : i \in c, \\ false & \text{otherwise.} \end{cases}$$

⊣

**Example 4.10.** Let $\Delta \coloneqq \{\{1, a\}, \{1, 3, b\}\}$ be a CCS. Then $Vert_1(\Delta) = true$ whereas $Vert_2(\Delta) = false$. ◇

We can now describe what the algorithm does at each node in the parse tree of the $k$-expression $\kappa$, depending on the operation at this node (adding a new vertex, renaming labels, connecting vertices and disjoint union). Since we are only interested in cut sets of size at most $m$, we can discard CCS that require more than $m$ cuts. To be precise let $s$ be a subexpression of $\kappa$. If $S_s$ contains a CCS $\Delta$ with $cuts_s(\Delta) > m$, $\Delta$ is discarded before the algorithm proceeds to the parent tree node.

At the leaves of the parse tree, only vertex introductions can occur, since this is the only 0-ary operation. Internal nodes contain the other three operations $\rho$, $\oplus$ and $\eta$.

- $i(v)$: *(Adding a new vertex)* There are two possibilities - either remove the vertex or add it to the graph. Depending on whether $v$ is a terminal vertex there are two cases:
  - $v$ is a terminal vertex. This is the only part of the algorithm which differs for UVMC and RVMC. For UVMC $S_{i(v)} \coloneqq \{\emptyset, \{\{i, v\}\}\}$. $cuts_{i(v)}(\emptyset) = 1$ and $cuts_{i(v)}(\{\{i, v\}\}) = 0$. The CCS $\{\{i, v\}\}$ represents the connected component that contains the i-labeled terminal vertex $v$, the empty CCS represents the empty graph. For RVMC removing $v$ is not an option and hence $S_{i(v)} \coloneqq \{\{\{i, v\}\}\}$.
  - If $v$ is not a terminal vertex, then $S_{i(v)} \coloneqq \{\emptyset, \{\{i\}\}\}$. Again $cuts(\emptyset) \coloneqq 1$ and $cuts(\{\{i\}\}) \coloneqq 0$.
- $\rho_{i \leftarrow j}$: *(Renaming labels)* Let $s$ be the $k$-expression of the subtree below the current node $\rho_{i \leftarrow j}$. Then

  $$S_{\rho_{i \leftarrow j}(s)} \coloneqq \{ren_{i \leftarrow j}(\Delta) : \Delta \in S_s\},$$

  i.e. $S_{\rho_{i \leftarrow j}(s)}$ is generated by replacing each symbol $j$ in the data structure $S_s$ with $i$. For each $\Delta_{new} \in S_{\rho_{i \leftarrow j}(s)}$ the cuts function is defined as

  $$cuts_{\rho_{i \leftarrow j}(s)}(\Delta_{new}) \coloneqq min\left\{cuts_s(\Delta) : \Delta \in S_s \wedge ren_{i \leftarrow j}(\Delta) = \Delta_{new}\right\}.$$

- $s \oplus t$: *(Disjoint union)* Let $s$ and $t$ be the $k$-expression of the subtree below the current $\oplus$-node. Furthermore, let $S_s$ and $S_t$ denote both data structures and $cuts_s$ and $cuts_t$ their cuts-mappings. Then

$$S_{s\oplus t} := \{\Delta_s \cup \Delta_t : \Delta_s \in S_s \wedge \Delta_t \in S_t \wedge \\ cuts_s(\Delta_s) + cuts_t(\Delta_t) \leq m\} \text{ and}$$

$$cuts_{s\oplus t}(\Delta) := min\{cuts_s(\Delta_s) + cuts_t(\Delta_t) : \\ \Delta_s \in S_s \wedge \Delta_t \in S_t \wedge \Delta_s \cup \Delta_t = \Delta\}.$$

- $\eta_{i,j}$: *(Connecting vertices)* Let $s$ be the $k$-expression of the subtree below the current node $\eta_{i\leftarrow j}$. For each $\Delta \in S_s$ there are two cases:
  - $\{i,j\} \not\subseteq \bigcup \Delta$, i.e. there is no i-labeled or no j-labeled vertex in the graphs that are represented by $\Delta$. Therefore no connections are introduced and hence nothing has to be done.
  - Otherwise $\Delta$ is replaced by $con_{i,j}(\Delta)$. However, edges might have been added such that a terminal pair was connected. That is the case if $Valid_{i,j}(\Delta)$ is false. Then $con_{i,j}(\Delta)$ has to be discarded, because $\Delta$ represents graphs with a connected terminal pair.

  These two cases can be summarized by

$$S_{\eta_{i,j}(s)} := \big\{\Delta : \Delta \in S_s \wedge \neg(Vert_i(\Delta) \wedge Vert_j(\Delta))\big\} \cup \\ \big\{con_{i,j}(\Delta) : \Delta \in S_s \wedge Vert_i(\Delta) \wedge \\ \wedge Vert_j(\Delta) \wedge Valid_{i,j}(\Delta)\big\}.$$

  The cuts function $cuts_{\eta_{i,j}(s)}$ is defined for each $\Delta_{new} \in S_{\eta_{i,j}(s)}$ as the minimum of the following set:

$$\Big\{cuts_t(\Delta) : \Delta \in S_s \wedge \ \neg(Vert_i(\Delta) \wedge Vert_j(\Delta)) \wedge \Delta = \Delta_{new}\Big\} \cup \\ \Big\{cuts_t(\Delta) : \Delta \in S_s \wedge Vert_i(\Delta) \wedge Vert_j(\Delta) \wedge \\ Valid_{i,j}(\Delta) \wedge con_{i,j}(\Delta) = \Delta_{new}\Big\}.$$

  Intuitively $cuts_{\eta_{i,j}(s)}(\Delta_{new})$ is the minimum number of cuts of all CCS that lead to $\Delta_{new}$ if i-labeled and j-labeled vertices are connected.

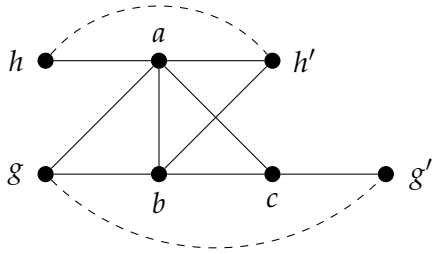


Figure 11: The example graph $G_{Ex}$ together with the example terminal set $H_{Ex}$. Solid lines depict edges whereas dashed lines represent terminal pairs.

If the algorithm has traversed the parse tree of $\kappa$ up to the root node, the algorithm terminates. There is a cut set for the RVMC/UVMC problem of size at most $m$ if and only if $S_\kappa \neq \emptyset$.

**Theorem 4.11.** *The running time of the algorithm is* $\mathcal{O}\left(2^{cw(G)+2\cdot|H|} \cdot \|\kappa\|\right)$, *where* $\|\kappa\|$ *denotes the number of operations in* $\kappa$. *Therefore the algorithm is fixed-parameter linear.*

*Proof.* The algorithm basically operates on CCS. CCS are subsets of $\{1,...,k\} \cup V_H$ and hence there are $2^{cw(G)+2\cdot|H|}$ possible CCS. The operations for each CCS can be done in constant time. Since the data structure $S$ is calculated for each subexpression of $\kappa$, we have $\|\kappa\|$ as a factor. □

### EXAMPLE FOR RVMC

This section contains an example computation of the algorithm for the RVMC instance $(G_{Ex}, H_{Ex}, 2)$. $G_{Ex}$ and $H_{Ex}$ are shown in Figure 11. Solid lines depict edges whereas dashed lines represent terminal pairs. The terminal set is $H_{Ex} := \{\{g, g'\}, \{h, h'\}\}$.

Figure 12 shows the parse tree of the 3-expression $\kappa_{Ex}$ for the example graph $G_{Ex}$. Additionally certain tree nodes are marked with $s_1, ..., s_7$. Let $s_1, ..., s_7$ denote the subexpressions of $\kappa_{Ex}$ corresponding to the subtrees rooted at these nodes. For these nodes we give the data structure $S$ and the cuts function.

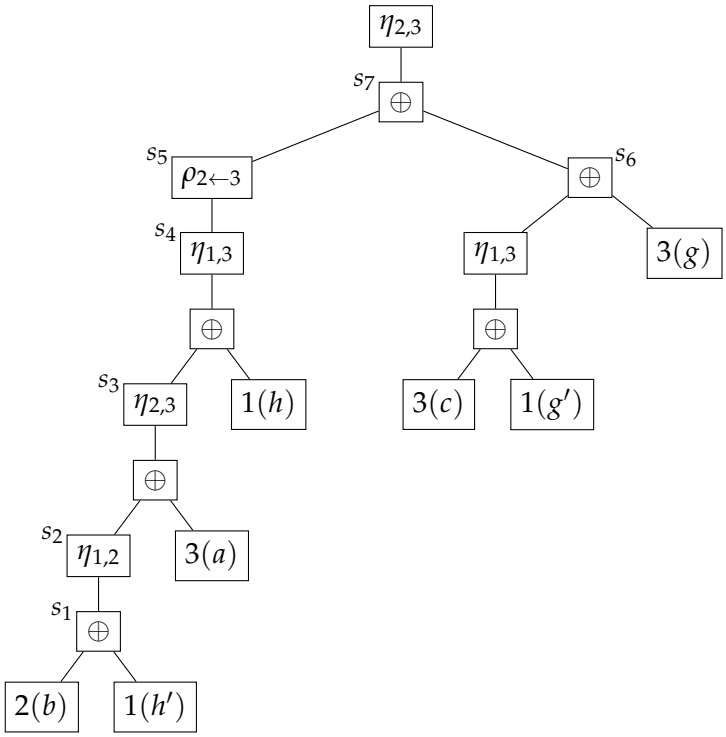


Figure 12: The parse tree of the 3-expression $\kappa_{Ex}$. $s_1, ..., s_7$ denote the subexpressions rooted in these nodes.

1. Subexpression $s_1$. Operation in this node: $\oplus$

| CCS | cuts |
|---|---|
| $\{\{1,h'\}\} \cup \{\{2\}\}$ | 0 |
| $\{\{1,h'\}\} \cup \{\,\}$ | 1 |

Observe that the vertex $h'$ is present in every CCS, since it is a terminal vertex.

2. Subexpression $s_2$. Operation in this node: $\eta_{1,2}$

| CCS | cuts |
|---|---|
| $\{\{1,2,h'\}\}$ | 0 |
| $\{\{1,h'\}\}$ | 1 |

The second CCS does not contain the label 2 and hence nothing has to be done here.

3. Subexpression $s_3$. Operation in this node: $\eta_{2,3}$

| CCS | cuts |
|---|---|
| $\{\{1,2,h'\} \cup \{3\}\}$ | 0 |
| $\{\{1,h'\},\{3\}\}$ | 1 |
| $\{\{1,2,h'\}\}$ | 1 |
| $\{\{1,h'\}\}$ | 2 |

These four rows correspond to (in this order): no vertex cut, vertex $b$ cut, vertex $a$ cut and both $a$ and $b$ cut.

4. Subexpression $s_4$. Operation in this node: $\eta_{1,3}$

| CCS | cuts |
|---|---|
| ~~$\{\{1,2,3,h'\} \cup \{1,h\}\}$~~ | 0 |
| ~~$\{\{1,h'\} \cup \{3\} \cup \{1,h\}\}$~~ | 1 |
| $\{\{1,2,h'\},\{1,h\}\}$ | 1 |
| $\{\{1,h'\},\{1,h\}\}$ | 2 |

Row 1 and 2 are not a valid CCS, since the terminal pair $\{h,h'\}$ is in one connected component. In row 3 and 4 the 3-labeled vertex $a$ has been cut and therefore no edges have been added.

5. Subexpression $s_5$. Operation in this node: $\rho_{2\leftarrow 3}$

| CCS | cuts |
|---|---|
| $\{\{1,2,h'\},\{1,h\}\}$ | 1 |
| $\{\{1,h'\},\{1,h\}\}$ | 2 |

Since no rows contain the label 3, all CCS remain unchanged.

6. Subexpression $s_6$. Operation in this node: $\oplus$

| CCS | cuts |
|---|---|
| $\{\{1,3,g'\}\} \cup \{\{3,g\}\}$ | 0 |
| $\{\{1,g'\}\} \cup \{\{3,g\}\}$ | 1 |

In the first row no cuts have been made. In the second row the 3-labeled vertex $c$ has been cut.

7. Subexpression $s_7$. Operation in this node: $\oplus$

| CCS | cuts |
|---|---|
| $\{\{1,2,h'\},\{1,h\}\} \cup \{\{1,3,g'\},\{3,g\}\}$ | 1 |
| $\{\{1,h'\},\{1,h\}\} \cup \{\{1,3,g'\},\{3,g\}\}$ | 2 |
| $\{\{1,2,h'\},\{1,h\}\} \cup \{\{1,g'\},\{3,g\}\}$ | 2 |
| ~~$\{\{1,h'\},\{1,h\}\} \cup \{\{1,g'\},\{3,g\}\}$~~ | 3 |

The CCS in the last row would require more than 2 cuts and is therefore discarded.

8. The root node. Operation in this node: $\eta_{2,3}$

| CCS | cuts |
|---|---|
| ~~$\{\{1,2,3,g,g'h'\},\{1,h\}\}$~~ | 1 |
| $\{\{1,h'\},\{1,h\},\{1,3,g'\},\{3,g\}\}$ | 2 |
| $\{\{1,2,3,g,h'\},\{1,h\},\{1,g'\}\}$ | 2 |

The CCS in the first row is not valid, since the terminal pair $\{g,g'\}$ is in one connected component. The two other rows are CCS of solution graphs. The algorithm terminates here and yields two solutions of the RVMC problem. Both have a cut set of size 2. The solution in row three corresponds to the cut set $\{a,b\}$ and the solution in the last row with the cut set $\{a,c\}$. These are exactly the RVMC solutions we found in the Multicut Example 2.15.

### PROOF OF CORRECTNESS

The proof of correctness has two parts. The first part shows that a UVMC or RVMC instance $(G, H, m)$ is a YES-instance if and only if the set $CCS(G(\kappa), H, m)$ is not empty. The second part shows that the algorithm indeed calculates $CCS(G(\kappa), H, m)$.

For the following proofs we fix a UVMC or RVMC instance $(G, H, m)$.

**Theorem 4.12.** *There is a cut set of size at most $m$ if and only if the set $CCS(G(\kappa), H, m)$ is not empty.*

*Proof.* Let $C$ be a cut set of size $\leq m$ and $G := (V_G, E_G)$. Then the graph $G[V_G \backslash C]$ is a solution graph. By Definition 4.2

$$ccs\left(G[V_G \backslash C]\right) \in CCS(G(\kappa), H, m)$$

and therefore $CCS(G(\kappa), H, m) \neq \emptyset$.

On the other hand let $\Delta \in CCS(G(\kappa), H, m)$. That means that there has to be a solution graph $G' := (V_{G'}, E_{G'})$, such that $ccs(G') = \Delta$ and $|V_G \backslash V_{G'}| \leq m$. Hence, $V_G \backslash V_{G'}$ is a cut set. □

**Theorem 4.13.** *The algorithm computes $S_\kappa = CCS(G(\kappa), H, m)$.*

*Proof.* We show per structural induction on $\kappa$ that for each subexpression $s$ of $\kappa$

$$S_s = CCS(G(s), H, m)$$

and especially $S_\kappa = CCS(G(\kappa), H, m)$.

In the leaf nodes of the parse tree there are only vertex introductions $i(v)$. A vertex can either be removed or not. This results in the possible graphs $(\emptyset, \emptyset)$ and $(\{v\}, \emptyset)$. The connected component set for the first graph is $\{\}$; for the second one $\{\{i\}\}$, or $\{\{i, v\}\}$ if $v$ is a terminal vertex. That is exactly what $S_{i(v)}$ consists of.

Now assume that $S_s = CCS(G(s), H, m)$. In the induction step there are three cases: $\rho_{i \leftarrow j}$, $s \oplus t$ and $\eta_{i,j}$.

First we show that $S_{\rho_{i \leftarrow j}(s)} = CCS(G(\rho_{i \leftarrow j}(s)), H, m)$. This is because

$$\begin{aligned} S_{\rho_{i \leftarrow j}(s)} &= \left\{ren_{i \leftarrow j}(\Delta) : \Delta \in S_s\right\} \\ &= \left\{ren_{i \leftarrow j}(\Delta) : \Delta \in CCS(G(s), H, m)\right\} \\ &= CCS\left(G(\rho_{i \leftarrow j}(s)), H, m\right). \end{aligned}$$

The last equality is due to the fact that renaming labels in the graph and renaming labels directly in the CCS leads to the same CCS.

Furthermore, we have to show that if $S_s = CCS(G(s), H, m)$ and $S_t = CCS(G(t), H, m)$, for subexpressions $s$ and $t$ of $\kappa$, then $S_{s\oplus t} = CCS(G(s \oplus t), H, m)$. This is because

$$\begin{aligned} S_{s\oplus t} = \{&\Delta_s \cup \Delta_t : \Delta_s \in S_s \wedge \Delta_t \in S_t \wedge \\ &cuts_s(\Delta_s) + cuts_t(\Delta_t) \leq m\} \\ = \{&\Delta_s \cup \Delta_t : \Delta_s \in CCS(G(s), H, m) \wedge \\ &\Delta_t \in CCS(G(t), H, m) \wedge \\ &cuts_s(\Delta_s) + cuts_t(\Delta_t) \leq m\} \\ = &\, CCS(G(s) \dot\cup G(t), H, m) \\ = &\, CCS(G(s \oplus t), H, m). \end{aligned}$$

Finally we show that $S_{\eta_{i,j}(s)} = CCS\left(G(\eta_{i,j}(s)), H, m\right)$. Using the induction hypothesis we get

$$\begin{aligned} S_{\eta_{i,j}(s)} = &\left\{\Delta : \Delta \in S_s \wedge \neg(Vert_i(\Delta) \wedge Vert_j(\Delta))\right\} \cup \\ &\left\{con_{i,j}(\Delta) : \Delta \in S_s \wedge Vert_i(\Delta) \wedge Vert_j(\Delta) \wedge Valid_{i,j}(\Delta)\right\} \\ = &\{\Delta : \Delta \in CCS(G(s), H, m) \wedge \\ &\quad \neg(Vert_i(\Delta) \wedge Vert_j(\Delta))\} \cup \qquad (4.1a) \\ &\{con_{i,j}(\Delta) : \Delta \in CCS(G(s), H, m) \wedge \\ &\quad Vert_i(\Delta) \wedge Vert_j(\Delta) \wedge Valid_{i,j}(\Delta)\}. \qquad (4.1b) \end{aligned}$$

Let us call the set in Formula (4.1a) $S_1$ and the set in Formula (4.1b) $S_2$, i.e. $S_{\eta_{i,j}(s)} = S_1 \cup S_2$. We have to show that

$$S_1 \cup S_2 = CCS\left(G(\eta_{i,j}(s)), H, m\right).$$

First, let $\Delta \in S_1$. Since $Vert_i(\Delta) \wedge Vert_j(\Delta)$ is false, there are either no i-labeled or no j-labeled vertices in the graphs represented by $\Delta$. Hence, $\eta_{i,j}$ does not change these graphs and $\Delta \in CCS\left(G(\eta_{i,j}(s)), H, m\right)$.

On the other hand if $\Delta \in CCS\left(G(\eta_{i,j}(s)), H, m\right)$ and either $Vert_i(\Delta)$ is false or $Vert_j(\Delta)$ is false, then $\Delta \in CCS\left(G(s), H, m\right)$ and therefore $\Delta \in S_1$.

Now let $con_{i,j}(\Delta) \in S_2$. Then $\eta_{i,j}$ adds edges in the graphs represented by $\Delta$. Each component containing $i$ is connected with each component

containing $j$. This leads to one big connected component that consists exactly of the connected components in $\Delta$ that contain either $i$ or $j$. This transformation is captured by the function $con_{i,j}$. Therefore $con_{(i,j)}(\Delta)$ is an element of $CCS\left(G(\eta_{i,j}(s)), H, m\right)$.

In the opposite direction let $\Delta \in CCS(G(\eta_{i,j}(s)), H, m)$ such that both $Vert_i(\Delta)$ and $Vert_j(\Delta)$ are true. Then there is a $\Delta' \in CCS(G(s), H, m)$ with $con_{i,j}(\Delta') = \Delta$. Obviously $Vert_i(\Delta)$ and $Vert_j(\Delta)$ also hold for $\Delta'$. $Valid_{i,j}(\Delta')$ is also true, because otherwise $Valid_{i,j}(\Delta)$ would also be false. Therefore $\Delta \in S_2$.

This concludes the induction. □

### APPLYING THIS ALGORITHM TO EMC

Since this algorithm works perfectly well for UVMC and RVMC, the question arises if it can be extended to EMC. However, this is not possible. In principle the CCS concept can be used for EMC. The problem is that in case of an $\eta$-operation, we have to know how many edges are introduced. Otherwise we would not know how many edges have to be cut to keep two connected components apart, which is essential to maintain solution graphs.

In order to have this information available we have to store the number of vertices for each label and each connected component. The problem is that this count of vertices is not unique for a CCS. A CCS can have many possible counts of vertices.

**Example 4.14.** Figure 13 shows an example to highlight this problem. Here the terminal set is $\{g, g'\}$. In the left graph the edge $\{g, a\}$ has been removed, in the right graph edge $\{a, b\}$. Both graphs have the CCS $\big\{\{1\}, \{1, g\}, \{2, g'\}\big\}$. However, they lead to a different number of cuts. Assume that the next operation is $\eta_{1,2}$. The dashed lines show the edges that are added with this operation. Then the right graph requires two cuts to remain valid, whereas the left graph requires only one. In retrospective the left graph should have been preferred to the right graph. However, this was not known before the $\eta_{1,2}$-operation. Therefore the number of vertices for both the middle and the right graph has to be stored. ◇

If we look at this data structure, we notice that we can no longer bound the number of CCS by $k$ and $|H|$, since the number of i-labeled vertices in one connected component is unbounded with regard to these two parameters. A more detailed analysis shows that the running

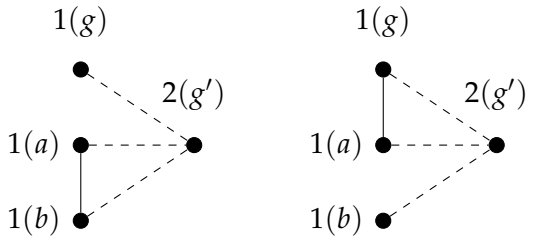


Figure 13: These two graphs have the same CCS but lead to different results.

time of the algorithm would contain a factor $n^m$, where $n$ is the input size and $m$ is the maximal number of cuts. Therefore the algorithm is no longer fixed-parameter tractable. One could, of course, add $m$ as a parameter, but it is already known that $|H|$ and $m$ suffice as parameters for fixed parameter tractability [35].

The next section shows another reason why $cw(G)$ might be the wrong parameter to tackle EMC. On the other hand we will see that the clique-width of the incidence graph of the structure $(V_G, E_G, H)$ allows for a single-parameter FPT algorithm.

# 5

# METATHEOREMS

This chapter contains results about metatheorems and their application to Multicut. In the first section we show how the clique-width metatheorem can be used to obtain FPT results for Multicut. In the second section we present an extension of Courcelle's theorem about graphs of bounded clique-width. The third section further explains this extension by showing that bounded clique-width of the incidence graph implies bounded tree-width of the graph.

## 5.1 THE CLIQUE-WIDTH METATHEOREM AND MULTICUT

The metatheorem for graphs of bounded clique-width, Theorem 2.35, would be an easy way to prove that Multicut is fixed-parameter tractable. However, it is not directly applicable since it deals with $p$-graphs and not about general structures. This section discusses how this problem can be circumvented. We will see that the FPT results from Chapter 4 also follow from the metatheorem. As remarked in Section 2.4.3, metatheorems are constructive but do not yield algorithms feasible for actual implementations. Therefore the results in this section do not make the FPT algorithm of Chapter 4 unnecessary, but show an straightforward way how to obtain fixed-parameter tractability for a problem, specifically for Multicut.

A Multicut instance consists of the triple $(G, H, m)$. We therefore have to code $G$ and $H$ into a $p$-graph (for a fixed $p$). In general this is not directly possible, since we would need colors for edges instead of colors for vertices. However, if the number of terminal vertices is bounded, we can circumvent this problem by assigning each terminal pair a color. Recall that colors of $p$-graphs are considered as unary predicates. Furthermore, let $p := |H|$. We then have a $p$-graph with a unary predicate for each terminal pair. Let $H_1, ..., H_p$ be these unary predicates.

To apply the metatheorem we have to find $MSO_1$ characterizations for the Multicut problems. Gottlob and Lee [26] have already found $MSO_1$formulas for UVMC and RVMC. These contain the abbreviation

$$\begin{aligned} connects(S,x,y) \equiv S(x) \wedge S(y) \wedge \forall P\big( & \\ & (P(x) \wedge \neg P(y)) \rightarrow \exists v \exists w( \\ & S(v) \wedge S(w) \wedge P(v) \wedge \neg P(w) \wedge E(v,w))\big). \end{aligned}$$

For a set of vertices $S \subseteq V$ $connects(S,x,y)$ is true if and only if there is a path from $x$ to $y$ that lies entirely in $S$. A detailed explanation of *connects* can be found in [26]. Now UVMC can be characterized with the following $MSO_1$formula, where $X$ is the cut set we are looking for:

$$\begin{aligned} uvmc(X) \equiv \forall x \forall y\big(H(x,y) \rightarrow & \\ & \forall S\left(connects(S,x,y) \rightarrow \exists v\left(X(v) \wedge S(v)\right)\right)\big). \end{aligned}$$

However, we cannot use this formula, since it contains $H$. We therefore have to replace $H(x,y)$ by

$$\left(H_1(x) \wedge H_1(y)\right) \vee \ldots \vee \left(H_h(x) \wedge H_h(y)\right).$$

This modification allows us to use the clique-width metatheorem. Our evaluation function is the cardinality function $|\cdot|$. We therefore want to minimize $|X|$, which is the size of the cut set. By Theorem 2.35 we get a fixed-parameter linear algorithm with regard to $k$ if $|H|$ is constant. This means that the running time of that algorithm is $\mathcal{O}\left(f(k,|H|) \cdot n\right)$, where $n$ is the size of the input. Therefore the algorithm is also FPL with regard to $k$ and $|H|$.

Exactly the same argumentation works for RVMC. Here we use the formula

$$rvmc(X) \equiv uvmc \wedge \left(\forall x\left(X(x) \rightarrow \neg \exists y\ H(x,y)\right)\right),$$

where we replace $H(x,y)$ as before.

EMC, however, has no (obvious) $MSO_1$ characterization. EMC deals with finding a cardinality-minimal set of edges. Quantification over sets of edges is only allowed for $MSO_2$. Gottlob and Lee [26] give an $MSO_2$ characterization of EMC:

$$\begin{aligned} emc(X) \equiv \forall x \forall y\big(H(x,y) \rightarrow & \\ & \forall S(connects(S,x,y) \rightarrow \\ & \exists v \exists w(X(v,w) \wedge S(v) \wedge S(w)))\big). \end{aligned}$$

$X$ is again the cut set, but in this case it is a set of edges. Therefore it is not an $LinEMSO_1$ problem and we cannot use the clique-width metatheorem. That the natural logical formula for EMC requires $MSO_2$ is another indication that bounded $cw(G)$ might not help to solve EMC.

## 5.2 A CLIQUE-WIDTH METATHEOREM FOR INCIDENCE GRAPHS

The aim of this section is to extend the clique-width metatheorem, Theorem 2.35, to structures, where the clique-width of the incidence graph is bounded. This will allow that an $MSO_2$-formula - instead of $MSO_1$ - may be used to describe the optimization problem.

**Definition 5.1.** Let $U$ be the domain of a $\tau$-structure with $R \in \tau$. $R$ is an n-ary *symmetric relation* if $R(x_1, ..., x_n)$ is true implies that for each permutation $\pi$ on $\{1, ..., n\}$, $R(x_{\pi(1)}, ..., x_{\pi(n)})$ is also true. We will consider symmetric relations as sets that contain subsets of $U$ with cardinality $n$ or formally $R \subseteq U^{[n]}$. $R(x_1, ..., x_n)$ is true if $\{x_1, ..., x_n\} \in R$.

A *symmetric structure* is a relational structure, consisting only of symmetric relations. ⊣

In this chapter we only consider symmetric structures. An example for a symmetric structure is an undirected graph.

**Definition 5.2.** Let $\mathcal{S} := (U, R_1, ..., R_l)$ be a symmetric structure. The arities of the relations are $r_1, ..., r_l$. Its *incidence graph*

$$\mathcal{I}(\mathcal{S}) := (V_\mathcal{I}, E_\mathcal{I}, P_U, P_1, ..., P_l)$$

can be seen as a colored graph. Precisely speaking $(V_\mathcal{I}, E_\mathcal{I})$ is a graph and $P_U, P_1, ..., P_l$ are unary predicates.

$V_\mathcal{I}$ contains vertices for each element in $U$ and for each set in the relations $R_1, ..., R_l$. Hence,

$$V_\mathcal{I} := \{v_u : u \in U\} \cup \bigcup_{i=1}^{l} \{v_r : r \in R_i\}.$$

The edge set is defined as

$$E_\mathcal{I} := \bigcup_{i=1}^{l} \{\{v_r, v_x\} : x \in r, r \in R_i\},$$

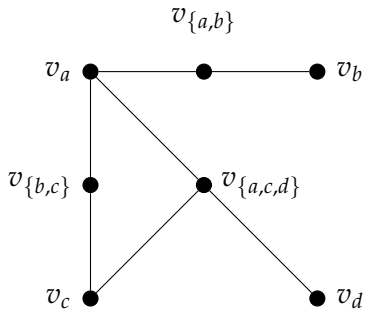


Figure 14: The incidence graph of the example structure $\mathcal{S}_{sym}$.

i.e. each vertex representing a set in a relation is connected with the vertices representing the elements of that set. The unary relations $P_i$, $i \in \{1, ..., l\}$, are defined as $P_i = \{v_r : r \in R_i\}$ and $P_U = \{v_u : u \in U\}$.

$\tau' \coloneqq \{E_{\mathcal{I}}, P_U, P_1, ..., P_l\}$ is the signature of $\mathcal{I}(\mathcal{S})$. ⊣

**Definition 5.3.** In order to give a better description of the relation between $\mathcal{S}$ and $\mathcal{I}(\mathcal{S})$, we define for each relation $R_i$ in $\mathcal{S}$ a function $f_i : R_i \to \{v_r : r \in R_i\}$ with $\{x_1, ..., x_n\} \mapsto v_{\{x_1,...,x_n\}}$. $f_U$ is a function from $U$, the domain of $\mathcal{S}$, to $\{v_u : u \in U\}$ that maps $u \mapsto v_u$. Clearly all $f_i$, $i \in \{1, ..., l\}$, and $f_U$ are bijective. For $T \subseteq R_i$ with $i \in \{1, ..., l\}$, $f_i(T)$ is defined as $\{f_i(x) : x \in T\}$.

For notational reasons we sometimes refer to $f_i$ as $f_{R_i}$. ⊣

**Example 5.4.** Let $\mathcal{S}_{sym} \coloneqq (U, R_1, R_2)$ be a symmetric structure with $U \coloneqq \{a, b, c, d\}$, $R_1 \coloneqq \{\{a, b\}, \{a, c\}\}$ and $R_2 \coloneqq \{\{a, c, d\}\}$. Then its incidence graph $\mathcal{I}(\mathcal{S}_{Ex})$ consists of the vertices

$$\{v_a, v_b, v_c, v_d, v_{\{a,b\}}, v_{\{b,c\}}, v_{\{a,c,d\}}\}.$$

The edges between them can be seen in Figure 14. ◇

We can now define *LinEMSO*$_2$, which is similar to *LinEMSO*$_1$ but allows more complex optimization problems.

**Definition 5.5.** Let $\tau$ be a signature and let $\Phi$ be an $MSO_2[\tau]$-formula with free second-order variables $S_1, ..., S_n$. Furthermore, let $\pi$ be a mapping from $\{1, ..., n\}$ to $\{0, 1, ..., l\}$. Then *LinEMSO*$_2[\Phi]$ is defined as

| $LinEMSO_2[\Phi]$ | |
|---|---|
| *Instance:* | A symmetric $\tau$-structure $\mathcal{S} \coloneqq (U, R_1, ..., R_l)$, a $k$-expression defining $\mathcal{I}(\mathcal{S})$, evaluation functions $g_1, ..., g_n$ for the free variables of $\Phi$ and $opt \in \{min, max\}$. |
| *Parameter:* | $k$. |
| *Output:* | Sets $T_1 \subseteq R_{\pi(1)}, ..., T_n \subseteq R_{\pi(n)}$, where $R_0 \coloneqq U$ with arity $r_0 \coloneqq 1$, such that $\mathcal{S} \models \Phi[T_1, ..., T_n]$ and $\sum_{i=1}^{n} g_i(T_i)$ is maximal or minimal (depending on *opt*). |

**Theorem 5.6.** *Let $\Phi$ be an arbitrary but fixed $MSO_2$-formula. Then the problem $LinEMSO_2[\Phi]$ is fixed-parameter linear with regard to $k$.*

We show this by applying theorem 2.35 to the incidence graph of $\mathcal{S}$. Since $\Phi$ is a $MSO_2[\tau]$-formula, we have to translate $\Phi$ in a $MSO_1[\tau']$-formula.

**Definition 5.7.** We define a translation function *tr*, which maps arbitrary $MSO_2[\tau]$-formulas to $MSO_1[\tau']$-formulas. *tr* is defined by using structural recursion on $\Phi$. To allow a shorter definition we assume that all existential quantifiers in $\Phi$ have been replaced by negated universal quantifiers in the usual way.

- Let $x_1$ and $x_2$ be free first-order variables. Then

$$tr(x_1 = x_2) \equiv (x_1 = x_2).$$

- Let $x_1, ..., x_{r_i}$ be free first-order variables. Then

$$tr\left(R_i(x_1, ..., x_{r_i})\right) \equiv \exists v \left( P_i(v) \wedge \bigwedge_{j=1}^{r_i} E_{\mathcal{I}}(x_j, v) \right).$$

- Let $S$ be a free $n$-ary second-order variable and let $x_1, ..., x_n$ be free first-order variables.

$$tr\left(S(x_1, ..., x_n)\right) \equiv \exists v \left( S(v) \wedge \bigwedge_{j=1}^{n} E_{\mathcal{I}}(x_j, v) \right).$$

- Let $\phi$ and $\psi$ be $MSO_2$-formulas. $P_U(S)$ is an abbreviation for $\forall x \left(S(x) \rightarrow P_U(x)\right)$ and $P_i(S)$ for $\forall x \left(S(x) \rightarrow P_i(x)\right)$. Then

$$tr\left(\phi \wedge \psi\right) \equiv tr(\phi) \wedge tr(\psi),$$

$$
\begin{aligned}
&tr(\neg\phi) \equiv \neg tr(\phi),\\
&tr(\forall x\phi) \equiv \forall x\,(P_U(x) \to tr(\phi)),\\
&tr(\forall S \subseteq R_i\phi) \equiv \forall S\,(P_i(S) \to tr(\phi)) \text{ and}\\
&tr(\forall S \subseteq U\phi) \equiv \forall S\,(P_U(S) \to tr(\phi)).
\end{aligned}
$$

⊣

One can easily check that for any $MSO_2[\tau]$-formula $\Psi$, $tr(\Psi)$ is indeed an $MSO_1[\tau']$-formula.

For the rest of this section let $\Phi' := tr(\Phi)$. To begin with, we will prove the following lemma.

**Lemma 5.8.** *Let $\tau$ be a signature and let $\mathcal{S} := (U, R_1, ..., R_l)$ be a symmetric $\tau$-structure. Let $\Phi$ be a $MSO_2[\tau]$-formula with free second-order variables $S_1, ..., S_n$ and free first-order variables $x_1, ..., x_m$. Furthermore, let $\pi$ be a mapping from $\{1, ..., n\}$ to $\{0, 1, ..., l\}$. Again $R_0 := U$ with arity $r_0 := 1$. Then $\forall T_1 \subseteq R_{\pi(1)}$ ... $\forall T_n \subseteq R_{\pi(n)}$ and $\forall u_1 \in U$ ... $\forall u_m \in U$*

$$
\begin{aligned}
&\mathcal{S} \models \Phi[T_1, ..., T_n, u_1, ..., u_m] \iff\\
&\mathcal{I}(\mathcal{S}) \models \Phi'(f_{\pi(1)}(T_1), ..., f_{\pi(n)}(T_n), f_U(u_1), ..., f_U(u_m)).
\end{aligned}
$$

*Proof.* We prove this by structural induction on $\Phi$. First we prove the left-to-right direction of the equivalence:

Base cases:

- $x_1 = x_2$
  Since $f_U$ is a bijection, $u_1 = u_2$ implies $f_U(u_1) = f_U(u_2)$ for any $u_1, u_2 \in U$.
- $R_i(x_1, ..., x_{r_i})$ with $i \in \{1, ..., l\}$
  Let $u_1, ..., u_{r_i} \in U$ such that $R_i(u_1, ..., u_{r_i})$. Then there exists $v := f_i(\{u_1, ..., u_{r_i}\}) \in V_\mathcal{I}$. By Definition 5.2 $v$ satisfies $P_i(v)$ and $E_\mathcal{I}(f_U(u_j), v)$ for any $j \in \{1, ..., r_i\}$.
- $S(x_1, ..., x_n)$
  Let $\{u_1, ..., u_n\} \in T$ and $T \subseteq R_i$, $i \in \{0, 1, ..., l\}$, such that $r_i = n$. Then there exists $v := f_i(\{u_1, ..., u_n\}) \in V_\mathcal{I}$. Again by Definition 5.2 $v$ satisfies $E_\mathcal{I}(f_U(u_j), v)$ for any $j \in \{1, ..., n\}$. Recall that $f_i(T) = \{f_i(t) : t \in T\}$ and therefore $v \in f_i(T)$. This proves

$$
\mathcal{I}(\mathcal{S}) \models \exists v \left( f_i(T)(v) \wedge \bigwedge_{j=1}^{n} E_\mathcal{I}(f_U(u_j), v) \right).
$$

Induction steps:

- Immediate for $\wedge$ and $\neg$.
- $\forall x\phi$, where $x$ occurs freely in $\phi$
  We know that $\mathcal{S} \models \forall x\phi$ and hence $\mathcal{S} \models \phi[u]$ for all $u \in U$. By the induction hypothesis $\mathcal{I}(\mathcal{S}) \models tr(\phi)[f_U(u)]$ for any $u \in U$. $P_U = \{f_U(u) : u \in U\}$ and hence

  $$\mathcal{I}(\mathcal{S}) \models \forall x(P_U(x) \rightarrow tr(\phi)).$$

- $\forall S \subseteq R_i\ \phi$, where $S$ occurs freely in $\phi$ and $i \in \{1, ..., l\}$
  We know that $\mathcal{S} \models \forall S \subseteq R_i\ \phi$ and hence $\mathcal{S} \models \phi[T]$ for all $T \subseteq R_i$. By the induction hypothesis

  $$\mathcal{I}(\mathcal{S}) \models tr(\phi)[f_i(T)] \text{ for any } T \subseteq R_i.$$

  Since $P_i(S)$ guarantees $S \subseteq P_i$ and $P_i = f_i(R_i)$, we have

  $$\mathcal{I}(\mathcal{S}) \models \forall S(P_i(S) \rightarrow tr(\phi)).$$

- The proof for the case $\forall S \subseteq U\ \phi$, where $S$ occurs freely in $\phi$, is analogous to the previous case.

We continue with the base cases of the right-to-left direction from the equivalence:

- $x_1 = x_2$
  Since $f_U$ is a bijection, $f_U(u_1) = f_U(u_2)$ implies $u_1 = u_2$ for any $u_1, u_2 \in U$.
- $R_i(x_1, ..., x_{r_i})$ with $i \in \{1, ..., l\}$
  Let $v \in V_\mathcal{I}$ such that $P_i(v)$ and $E_\mathcal{I}(f_U(u_j), v)$ for any $j \in \{1, ..., r_i\}$. Since there are edges between $v$ and $f_U(u_j)$ and $f_i$ is bijective, $f_i(\{u_1, ..., u_{r_i}\}) = v$. Furthermore, $P_i = f_i(R_i)$. Hence, $\mathcal{S} \models R_i(\{u_1, ..., u_{r_i}\})$.
- $S(x_1, ..., x_n)$, where $S$ is an $n$-ary second-order variable
  Let $T \subseteq R_i$ and $u_1, ..., u_n \in T$. Furthermore, let $v \in V_\mathcal{I}$ such that $f_i(T)(v)$ and $E_\mathcal{I}(f_U(u_j), v)$ for any $j \in \{1, ..., n\}$. Since there are edges between $v$ and $f_U(u_j)$ and $f_i$ is bijective, $f_i(\{u_1, ..., u_n\}) = v$. Hence, $\mathcal{S} \models T(\{u_1, ..., u_n\})$.

Induction steps:

- Immediate for $\wedge$ and $\neg$.
- $\forall x\phi$, where $x$ occurs freely in $\phi$
  We know that $\mathcal{I}(\mathcal{S}) \models \forall x\,(P_U(x) \rightarrow tr(\phi))$ and hence $\mathcal{I}(\mathcal{S}) \models tr(\phi)[f_U(u)]$ for all $u \in U$. By the induction hypothesis $\mathcal{S} \models \phi[u]$ for all $u \in U$. Hence, $\mathcal{S} \models \forall x\phi$.
- $\forall S \subseteq R_i\ \phi$, where $S$ occurs freely in $\phi$ and $i \in \{1, ..., l\}$
  We know that $\mathcal{I}(\mathcal{S}) \models \forall S\ \ (P_i(S) \rightarrow tr(\phi))$ and hence $\mathcal{I}(\mathcal{S}) \models tr(\phi)[f_i(T)]$ for all $T \subseteq R_i$. By the induction hypothesis $\mathcal{S} \models \phi[T]$ for any $T \subseteq R_i$ and therefore $\mathcal{S} \models \forall S \subseteq R_i\ \phi$.
- Again the proof for the case $\forall S \subseteq U\ \phi$, where $S$ occurs freely in $\phi$, is analogous to the previous case.

□

The proof of Theorem 5.6 is a direct consequence of Lemma 5.8.

*Proof.* (of Theorem 5.6) We transform $\Phi$ with the translation function $tr$ and get $\Phi' := tr(\Phi)$. $\Phi'$ is an $MSO_1$-formula. Furthermore, we construct the incidence graph $\mathcal{I}(\mathcal{S})$. Thus we have translated the $LinEMSO_2$ problem into an $LinEMSO_1$ problem. Its input is $\mathcal{I}(\mathcal{S})$, a k-expression defining $\mathcal{I}(\mathcal{S})$ and evaluation functions $g_1', ..., g_n'$, where $g_i'$, $i \in \{1, ..., n\}$, is defined as $g_i \circ f_{r_i}$. One can easily check that the size of this input is linear with regard to the original input. We use Theorem 2.35 to solve it in linear time. The output can be translated back into $\mathcal{S}$ by applying the inverted $f$-functions. In this way we have found a solution to the $LinEMSO_2$ problem in linear time. □

This theorem can now directly be used for Multicut.

**Theorem 5.9.** *UVMC, RVMC and EMC are FPL with $cw(\mathcal{I}(G, H))$ as parameter, where $(G, H)$ is an abbreviation for the structure $(V, E, H)$.*

*Proof.* In Section 5.1 $MSO_1$ characterizations for UVMC and RVMC and an $MSO_2$ characterization for EMC have been presented. Together with those Theorem 5.6 immediately yields the FPT result. □

As a final remark we answer the question if it is enough to consider $cw(\mathcal{I}(G))$ as the parameter instead of $cw(\mathcal{I}(G, H))$. This is not the case:

**Theorem 5.10.** *RVMC and EMC are NP-complete for input instances where $cw(\mathcal{I}(G))$ is bounded by a number $\geq 3$.*

*UVMC is NP-complete for input instances where $cw(\mathcal{I}(G))$ is bounded by a number $\geq 4$.*

*Proof.* It is known that RVMC and EMC are NP-complete on trees [6, 25]. The incidence graph of a tree is again a tree. Since trees have clique-width at most 3, RVMC and EMC are NP-complete for input instances where $cw(\mathcal{I}(G))$ is bounded by a number $\geq 3$.

UVMC is NP-complete on series-parallel graphs [6]. Again the incidence graph of a series-parallel graph remains a series-parallel graph. Since series-parallel graphs have clique-width at most 4, UVMC is NP-complete for input instances where $cw(\mathcal{I}(G))$ is bounded by a number $\geq 4$. □

## 5.3 CLIQUE-WIDTH OF INCIDENCE GRAPHS AND TREE-WIDTH

The main result of this section is that a class of graphs with bounded clique-width of their incidence graphs has bounded tree-width. As a direct consequence we get that a class of graphs has bounded tree-width if and only if their incidence graphs have bounded clique-width.

We start with the definition of incidence graphs, however this time not for arbitrary structures but only for graphs. This is a special case of Definition 5.2, but for better readability we fully state the definition again:

**Definition 5.11.** Let $G := (V, E)$ be a graph. Its *incidence graph* $\mathcal{I}(G)$ is defined as the pair $(V_\mathcal{I}, E_\mathcal{I})$. $V_\mathcal{I}$ contains a vertex for each element in $V$ and a vertex for each element in $E$. The first type of vertices in $\mathcal{I}(G)$ is called *v-vertices* and the second type *e-vertices*. Formally

$$V_\mathcal{I} := \{v_w : w \in V\} \cup \{v_e : e \in E\}.$$

The edge set $E_\mathcal{I}$ is defined as

$$E_\mathcal{I} := \big\{\{v_w, v_e\} : w \in V \wedge e \in E \wedge \exists x \in V \text{ s.t. } e = \{w, x\}\big\},$$

i.e. in $\mathcal{I}(G)$ there is an edge between a v-vertex and an e-vertex if and only if the corresponding edge in $G$ connects the corresponding vertex in $G$ with another vertex. There is never an edge between v-vertices in $\mathcal{I}(G)$ and never between e-vertices either. ⊣

**Lemma 5.12.** *Let $G$ be a graph and $\kappa$ a $k$-expression for incidence graph, i.e. $\mathcal{I}(G) = G(\kappa)$. Furthermore, let $\eta_{i,j}(s)$ be a subexpression of $\kappa$. Without loss of generality let there be an $i$-labeled v-vertex. Then in $G(\eta_{i,j}(s))$ exactly one of those statements is true: (The expressions in the brackets denote the names of the statements.)*

1. ***($v^1$-$e^1$)*** *There is exactly one $i$-labeled vertex, which is a v-vertex, and exactly one $j$-labeled vertex, which is an e-vertex.*
2. ***($v^1$-$e^{\geq 2}$)*** *There is exactly one $i$-labeled vertex, which is a v-vertex, and two or more $j$-labeled vertices, which are all e-vertices.*
3. ***($v^2$-$e^1$)*** *There are exactly two $i$-labeled vertices, which are both v-vertices, and exactly one $j$-labeled vertex, which is an e-vertex.*

*Proof.* First, observe that it is correct to assume without loss of generality that there is an $i$-labeled v-vertex, since there are no edges between e-vertices. Furthermore, in case of an $\eta_{i,j}$-operation it is not possible that there are both v-vertices and e-vertices with the same label ($i$ or $j$). If we take these remarks into account, there are exactly two cases not listed in the lemma:

4. **($\mathbf{v^2}$-$\mathbf{e^{\geq 2}}$)** There are exactly two $i$-labeled vertices, which are both v-vertices, and two or more $j$-labeled vertices, which are all e-vertices.
5. **($\mathbf{v^{\geq 3}}$-$\mathbf{e^{\geq 1}}$)** There are three or more $i$-labeled vertices, which are both v-vertices, and one or more $j$-labeled vertices, which are all e-vertices.

We show that Case 4 and Case 5 are not possible. For Case 4 let $a$ and $b$ be two $i$-labeled v-vertices and $e$ and $f$ two $j$-labeled e-vertices. After the $\eta_{i,j}$-operation there is a path $(a, e, b)$ and a path $(a, f, b)$. Hence, there would be two edges between $a$ and $b$ in $G$, which is not possible since we consider only simple graphs.

For Case 5 let $a$, $b$ and $c$ be v-vertices and $e$ an e-vertex. After applying the $\eta_{i,j}$-operation $(a, e, b)$, $(a, e, c)$ and $(b, e, c)$ would be paths and therefore in $G$ the vertices $a$ and $c$ would be connected by the same edge as $b$ and $c$. Hence, only the cases 1-3 are possible. □

**Theorem 5.13.** *Let $G$ be a graph. Then $tw(G) \leq 2 \cdot cw(\mathcal{I}(G)) - 1$.*

Before starting the proof, we first define *labels* for notational reasons.

**Definition 5.14.** Let $s$ be a $k$-expression and $G(s)$ its corresponding graph. Furthermore, let $v$ be a vertex in $G(s)$ and $i$ its label. Then $\mathit{labels}_s(v)$ is the number of $i$-labeled vertices in $G(s)$. ⊣

*Proof.* (of Theorem 5.13) Let $G := (V, E)$ be a graph and $\mathcal{I}(G) := (V_\mathcal{I}, E_\mathcal{I})$ its incidence graph. Furthermore, let $\kappa$ be a $k$-expression of $\mathcal{I}(G)$. We construct a tree decomposition $(X, T)$ of the graph $G$ with width $2k-1$, i.e. the largest bag has size at most $2k$. The tree $T$ is exactly the parse tree of $\kappa$. Therefore our bags will be labeled with subexpressions of $\kappa$. We define the bags in $X$ inductively:

- $i(v)$: We distinguish between the introduction of v-vertices and e-vertices. This is because since we are constructing a tree decomposition for $G$ and not for $\mathcal{I}(G)$, i.e. the bags contain only v-vertices. Actually the bags should contain elements of $V$, but we identify vertices in $V$ with their corresponding v-vertices. Hence,

$$X_{i(v)} := \begin{cases} \{v\} & \text{if } v \text{ is a v-vertex,} \\ \emptyset & \text{if } v \text{ is an e-vertex.} \end{cases}$$

- $\rho_{i \leftarrow j}(s)$: Renaming of labels does not change the graph and therefore

$$X_{\rho_{i \leftarrow j}(s)} := X_s.$$

- $s \oplus t$: Here we have to take the union of the bags $X_s$ and $X_t$. Potentially this could lead to a bag of size $4k$. However, we will put only vertices of those labels in the bag that may later appear in an $\eta$-operation. From Lemma 5.12 we know that if there are more than two v-vertices with the same label, there cannot be a $\eta$-operation using that label. Therefore we will discard all vertices that share a label with two or more vertices:

$$X_{s \oplus t} := \{v \in X_s \cup X_t : \mathit{labels}_{s \oplus t}(v) \leq 2\}.$$

- $\eta_{i,j}(s)$: Here we have to distinguish the three cases presented in Lemma 5.12.
  - **($\mathbf{v}^1$-$\mathbf{e}^1$)** Let $v$ be the v-vertex and $e$ the e-vertex. Furthermore, let $w$ be the (unique) remaining v-vertex, which is

connected with $e$. If the edge between $w$ and $e$ has already been introduced in $G(s)$, then

$$X_{\eta_{i,j}(s)} := X_s.$$

If the edge between $w$ and $e$ has not yet been introduced in $G(s)$, then

$$X_{\eta_{i,j}(s)} := X_s \cup \{w\}.$$

This rule will be essential to satisfy Condition 2 for tree decompositions (Definition 2.26).

– **($v^1$-$e^{\geq 2}$)** and **($v^2$-$e^1$)**: Here

$$X_{\eta_{i,j}(s)} := X_s.$$

This concludes the construction of the tree. It remains to show that this is indeed a tree decomposition of width $2k-1$.

We start by showing per induction on the tree that the maximal size of a bag is $2k$. At the leaves the size of a bag is 0 or 1. $\rho$-operations do not change the size of a bag. Let $s$ and $t$ be subexpressions of $\kappa$. For $\oplus$-operations note that there are $k$ different labels and hence $X_{s\oplus t}$ cannot contain more than $2k$ vertices. In the $\eta$-operation cases ($v^1$-$e^{\geq 2}$) and ($v^2$-$e^1$) no vertices are added. Only in the ($v^1$-$e^1$) case one additional vertex may be added. For the subexpression $\eta_{i,j}(s)$ let $v$ be the v-vertex and $e$ the e-vertex. Furthermore, let $w$ be the (unique) remaining v-vertex which is connected with $e$. If the edge between $w$ and $e$ has already been introduced in $G(s)$, then no vertex is added to the bag. Therefore let us assume that the edge between $w$ and $e$ has not yet been introduced in $G(s)$. There are two possibilities how this edge will be introduced later on: ($v^1$-$e^1$) and ($v^1$-$e^{\geq 2}$). In both cases $w$ is the only vertex with its label, i.e. $\mathit{labels}_s(w) = 1$. Therefore if $w$ has already been introduced in $G(s)$ but has not yet been connected to $e$, $w$ is certainly in the bag $X_s$ and hence the $\left|X_{\eta_{i,j}(s)}\right| = |X_s|$. If $w$ has not been introduced yet, there are no $j$-labeled v-vertices, just $j$-labeled e-vertices. Hence, the bag $X_s$ has size at most $2k-2$ and by adding $w$ to it the size is at most $2k-1$. Also note that until $w$ and $e$ will be connected, the only $j$-labeled vertices can be e-vertices. Furthermore, until the edge $\{e, w\}$ is introduced, $w$ will be the only vertex that will be connected to the $j$-labeled e-vertices. Hence, $w$ is the only v-vertex

that is "assigned" to the label $j$. Therefore all bags in $X$ have size at most $2k$.

It remains to check that the previously defined tree is indeed a tree decomposition. For this we have to check the three conditions in Definition 2.26.

- (Each $x \in V$ has to be element of at least one $X_s$, $s$ subexpression of $\kappa$):

  This is trivially true, since each $x \in V$ appears at least in a leaf bag, or more precisely $v_x$ appears for each $x \in V$.

- (Each edge $e \in E$ has to be a subset of at least one $X_i$, $i \in \{1, ..., n\}$):

  Let $e = \{x, y\}$ with $x, y \in V$. Without loss of generality assume that when $\mathcal{I}(G)$ is built accordingly to $\kappa$, the edge $\{v_x, v_e\}$ is introduced before the edge $\{v_y, v_e\}$. As before we do not really distinguish between $v_x$ and $x$, $v_y$ and $y$ and $v_e$ and $e$. According to Lemma 5.12 there are 3 possibilities how the edge $\{v_x, v_e\}$ is introduced:

  1. **($\mathbf{v^1}$-$\mathbf{e^1}$)** In this case both $x$ and $y$ are in the bag. Since there is no edge between $v_y$ and $v_e$, $y$ has been added to the bag. Also $x$ is in the bag since it is the only vertex with its label.
  2. **($\mathbf{v^1}$-$\mathbf{e^{\geq 2}}$)** In this case at least two e-vertices of the same label are to be connected to $v_x$. Since both e-vertices have the same label, later on both would be connected to $v_y$. This would result in at least two edges from $x$ to $y$ in $G$. Since $G$ is a simple graph, this case is not possible.
  3. **($\mathbf{v^2}$-$\mathbf{e^1}$)** Here $v_x$ and $v_y$ have the same label and there are no other vertices with this label. Hence, both $v_x$ and $v_y$ are in the bag, since only vertices are discarded if there are more than 2 vertices of the same label.

- (If $X_i$ and $X_j$ both contain $v \in V$, then each tree node on the path between node $X_i$ and $X_j$ has to contain $v$):

  The only way how this might fail is if a v-vertex is discarded because there are three or more vertices of its label and then re-introduced by a ($v^1$-$e^1$) $\eta$-operation. However, that is not possible, since a ($v^1$-$e^1$) $\eta$-operation requires that the v-vertex is the only one of its label.

□

**Theorem 5.15.** *Let $\mathcal{G}$ be a class of graphs. Then $\mathcal{G}$ has bounded tree-width if and only if the incidence graphs of $\mathcal{G}$ have bounded clique-width.*

*Proof.* Let the tree-width of all graphs $G$ in $\mathcal{G}$ be bounded by $k$. Then for all $G \in \mathcal{G}$ we know that

$$tw(\mathcal{I}(G)) \leq k + 1$$

by [27]. This implies by Theorem 2.28 that

$$cw(\mathcal{I}(G)) \leq 3 \cdot 2^k + 1.$$

The other direction follows directly from Theorem 5.13. □

# 6 CONCLUSION

## 6.1 AN OVERVIEW OF THE RESULTS

We have presented a detailed complexity analysis of the Multicut problem with regard to clique-width. Furthermore, we have found an efficient FPT algorithm for Vertex Multicut and two PTIME algorithms for RVMC on cographs. The following table summarizes these complexity results and puts them in context of previously known results. As usual the Multicut instance is given as the tuple $(G, H, m)$.

| **Parameters** | **UVMC** | **RVMC** | **EMC** |
|---|---|---|---|
| $tw(G)$ | NP-c [6] in PTIME for $tw(G) = 1$ | NP-c [6] | NP-c [25] |
| $cw(G)$ | NP-c (Thm 3.1) | NP-c (Chp 3.1.3) in PTIME for $cw(G) \leq 2$ | NP-c (Thm 3.2) |
| $tw(G \cup H)$ | FPT [26] | FPT [26] | FPT [26] |
| $cw(G \cup H)$ | NP-c (Thm 3.5) | in PTIME for $cw(G \cup H) \leq 2$ | NP-c (Thm 3.15) |
| $\|H\|$ | NP-c [35] | NP-c [31] | NP-c [15] |
| $cw(G)$, $\|H\|$ | FPT (Chp 4) | FPT (Chp 4) | |
| $tw(G)$, $\|H\|$ | FPT [31] | FPT [31] | FPT [31] |
| $m$, $\|H\|$ | FPT [35] | FPT [35] | FPT [35] |
| $cw(\mathcal{I}(G))$ | NP-c (Thm 5.10) | NP-c (Thm 5.10) | NP-c (Thm 5.10) |
| $cw(\mathcal{I}(G, H))$ | FPT (Thm 5.9) | FPT (Thm 5.9) | FPT (Thm 5.9) |

In addition to these results, we have found an extension of the clique-width metatheorem for classes of graphs, where the clique-width of the incidence graphs is bounded. We also have proved that a class of graphs has bounded tree-width if and only if their incidence graphs have bounded clique-width.

## 6.2 FUTURE WORK

So far only evidence has been collected that EMC might not be in FPT with $cw(G)$ and $|H|$ as parameters. The exact complexity of this parameterized problem is an open problem. A detailed analysis as for example in [22] might yield interesting results.

The algorithm presented in Chapter 4 uses connected component sets, which contain subsets of $V_H \cup \{1, ..., k\}$. It would be enough to consider subsets of $H \cup \{1, ..., k\}$, i.e. which decreases the size of this set from $2|H| + k$ to $|H| + k$. This would reduce the running time of the algorithm to $\mathcal{O}\left(2^{cw(G)+|H|} \cdot \|\kappa\|\right)$ (see Theorem 4.11 for notational details). Also an implementation of this algorithm would offer further information about its applicability.

Furthermore, there are some other parameters worth studying:

- Is the maximal vertex degree of the graph $(V_H, H)$ a useful parameter? Does for example $cw(G)$ together with the maximal degree allow for an FPT algorithm?
- Does $cw(G)$ and $m$, the maximal size of the cut, allow for an FPT algorithm?
- Is the diameter of $(V_H, H)$ a useful parameter?
- Rank-width is a width parameter closely related to clique-width. However, it has the advantage that decompositions can be found far more easily. Hence, it would be worthwhile to find out if the algorithm in Chapter 4 can be modified to work with rank-width or if a new algorithm is necessary.
- Signed clique-width, a notion introduced in [20], could be adapted for the Multicut problem. It seems that by encoding both $G$ and $H$ this could be used to obtain a single-parameter FPT algorithm.

# A

## APPENDIX: GANZ OHNE ORAKEL

This story is an attempt to explain some of the concepts and results in this thesis in an entertaining way. It is written for – but certainly not limited to – a non-scientific audience. The language of this story is German. It is my hope that this does not exclude potential readers. If it does, I want to apologize.

Since this is somewhat of an experiment, I am very interested in feedback of any sort.

Martin Lackner,
lackner@dbai.tuwien.ac.at

### GANZ OHNE ORAKEL

Unsere Geschichte beginnt am Olymp, der schon seit Anbeginn der Zeit ein beliebter Treffpunkt zahlreicher Götter war. Auch heute noch – nicht zuletzt wegen der akzeptablen Getränkepreise – ist er gut besucht. So sitzen auch an diesem Sonntag Vormittag zwei Stammgäste in einer wolkenverhangenen Ecke und spielen Schach. Der eine, mit hünenhafter Statur, ist Hephaistos, Gott des Feuers, der Schmiedekunst und der Vulkane. Ihm gegenüber sitzt ein Jüngling mit blonden Locken, schlank und auch sonst in starkem Kontrast zu seinem Gegenüber. Hier, in Griechenland, ist er unter dem Namen Eros bekannt, da ihm aber der Name Amor lieber ist, wollen wir ihn auch so nennen. Amor lungert auf seinem Sessel herum und rührt abwesend mit einem seiner Pfeile in einem Becher Himbeerbowle. Hephaistos hingegen starrt mit verbissener Miene auf das Schachbrett und hat noch keinen Schluck von dem Guinness genommen, das vor ihm steht.

Aber wir wollen unsere Aufmerksamkeit fürs Erste nicht auf die beiden Götter sondern auf das Schachbrett lenken, denn dort hat die

schwarze Dame, gespielt von Hephaistos, den weißen König arg in Bedrängnis gebracht. Ihre feuerroten Locken und die schwarze Rüstung verstärken ihre Bedrohlichkeit noch weiter. Der weiße König, fern all seiner Untergebenen, steht mit Schweiß auf der Stirn an den Rand des Spielbretts gedrängt, während die schwarze Dame Schritt für Schritt näherkommt. Hephaistos blickt mit grimmiger Genugtuung auf das Brett, wo seine Dame kurz davor ist, der blutrünstigen Natur des Spiels entsprechend den gegnerischen Monarchen zu erdolchen. Doch dem ist nicht so. Anstatt dem weißen König das Leben zu nehmen, beugt sie sich vor und drückt ihm einen Kuss auf den Mund. Nur wenige geflüsterte Worte später verlassen die beiden Hand in Hand das Spielfeld.

Hephaistos tobt: „WAS? Wie kannst du es wagen? Du kannst nicht einfach alles mit ... deiner verdammten Liebe lösen!"[1] Amor hört auf in seinem Getränk zu rühren und antwortet mit einem Grinsen: „Doch. Kann ich." Die Gesichtsfarbe von Hephaistos nähert sich der Farbe von Magma. „Ich werde ... Eine Götterwette. Ja, eine Götterwette. Nie und nimmer kannst du DAS mit deiner Liebe in etwas Gutes verwandeln!" Mit diesen Worten schlägt er mit seinem massiven Schmiedehammer auf den Boden. Tausende Kilometer entfernt, unter Island, beginnen sich Magmamassen zu bewegen und bahnen sich ihren Weg in Richtung Erdoberfläche. Nur wenig später beginnt ein Vulkan[2] Asche, Lava und Rauch zu spucken. Mehr und mehr und mehr. „Und jetzt zeige mir, Bürschchen, wie du da etwas mit deiner Liebe machen kannst." Mit gespielter Zuversicht grinst ihn Amor an: „Nichts leichter als das. Nichts leichter als das." Die beiden schütteln sich noch die Hand und die Wette ist besiegelt. Amor gewinnt die Götterwette, wenn er mit seinen Kräften aus dem Vulkanausbruch etwas Gutes machen kann. Zuversichtlich verlässt er den Olymp.

An dieser Stelle muss sich die Leserin oder der Leser auf einen plötzlichen Ortswechsel gefasst machen, nämlich nach Madrid. Solche Ortswechsel werden in dieser Geschichte noch oft vorkommen, weil Götter sehr flott reisen können und sich wenig um Zoll und Passkontrollen kümmern. Wir befinden uns auf dem Flughafen Madrid-Barajas. Dort

1 Man muss an dieser Stelle hinzufügen, dass Hephaistos einige Entäuschungen mit „der verdammten Liebe" erleben musste. Die Geschichte mit Aphrodite hat sich sogar bis zu den Sterblichen herumgesprochen.

2 Nach bester isländischer Tradition hat dieser Vulkan einen gänzlich unaussprechlichen Namen, der sich hauptsächlich aus ‘æ’s, ‘jö’s und ‘fj’s zusammensetzt.

lernen sich seit ein paar Minuten Myra und Thorsten kennen. Myra, sehr entspannt, wartet auf ihren Flug in ihre Heimatstadt Chicago, der erst in drei Stunden starten wird. Thorstens Flug nach Berlin hingegen geht in 45 Minuten. Das heißt, er muss spätestens in einer Viertelstunde in Richtung Sicherheitskontrolle aufbrechen. Das ist nicht viel Zeit, vor allem nicht genug Zeit um dieses Gespräch zu einem Punkt zu bringen, wo Thorsten es wagen würde, ein weiteres Treffen vorzuschlagen. Nachdem Myra über 7000 km entfernt wohnt, ist so ein Treffen nichts, was so spontan vorgeschlagen werden kann. „Ein bisschen mehr Zeit, nur ein bisschen mehr Zeit“, denkt Thorsten, während Myra von Franz Kafka schwärmt, der auch der Lieblingsautor von Thorsten ist. Im Hintergrund beobachtet Amor dies und langsam beginnt sich in seinen Gedanken eine Idee zu formen.

Wieder Ortswechsel, diesmal zum Oslo International Airport. Dort sitzen, nur fünf Sitze von einander entfernt, Luc und Clémentine. Die beiden waren zusammen in der Schule und dort enge Freunde. Fast sogar mehr als enge Freunde, doch als dann Clémentine ins Ausland ging, riss ihr Kontakt ab. Aber wie es der Zufall so will, denken die beiden gerade aneinander. Bloß gesehen haben sie sich noch nicht. Und es wirkt so, als würde das auch nicht mehr passieren, denn Clémentine macht sich gerade auf, um zu ihrem Flugzeug zu gehen. „Wenn allerdings ihr Flug ausfallen würde", murmelt Amor und sein Plan nimmt Gestalt an.

„Insgesamt habe ich auf 31 Flughäfen in Europa solche Szenen beobachtet. Alle bräuchten nur etwas mehr Zeit, genauer gesagt: ihre Flüge müssten entfallen", schließt Amor seine Erzählung. Zeus, seines Zeichens Göttervater, Herr des Himmels und des Wetters, hebt eine Augenbraue. „Eine Götterwette zwischen dir und Hephaistos ... wieder einmal. Ich nehme an, du wirst mir gleich erklären, wie ich dir helfen kann.“ Zeus seufzt schwer und lehnt sich in seinem Ledersessel zurück. Während er die letzten Wetterdaten auf dem Monitor vor sich überprüft, deutet er in Richtung Amor fortzufahren. Dieser ignoriert gekonnt Zeus' Unmut. „Also diese 31 Paare sind auf 31 verschiedenen Flughäfen in Europa. Wäre es dir nicht leicht möglich, die Aschewolke des Vulkans so zu verblasen, dass diese Flughäfen gesperrt werden müssen? Damit hätte ich genügend Zeit, mit ein paar wohlplatzierten Pfeilen für ein wenig Romantik zu sorgen.“ Amor setzt ein gewinnendes Lächeln auf.

Zeus überlegt. Eigentlich war ihm ganz und gar nicht nach solchen Spielereien zu Mute. Er hatte alle Hände voll zu tun, seit dieser Vulkan ausgebrochen war. Seine gewöhnlichen Wettermodelle waren wertlos

mit so viel Asche in der Luft. Womit er dann aber doch wieder bei dieser Wette wäre. Es würde ihm zumindest ein bisschen Genugtuung verschaffen, wenn er es Hephaistos etwas heimzahlen könnte, das Wetter so in Unordnung gebracht zu haben. „Nun gut, mal schauen was ich machen kann." Zeus startet auf seinem Computer eine Aschewolkensimulation.

Ein paar Minuten später blickt Zeus wieder von seinem Computer auf. Das entspannte Lächeln auf Amors Gesicht und der Fruchtcocktail in seiner Hand zeigen vor allem eins: Er hält sich bereits für den Sieger. Für Zeus ist das alles andere als klar: „Nun gut, hör zu. Die schlechte Nachricht: Ich kann nicht 31 Flughäfen überdecken. Dazu ist zu wenig Asche in der Luft und die Zeit ist zu kurz. Was ich kann – und das ist das absolute Maximum – ist Asche über 25 Flughäfen zu verteilen. Die gute Nachricht ist, dass das eventuell genug sein könnte. Thorsten, den du vorher erwähnt hast, hat einen Flug von Madrid über Wien nach Berlin. Und Clémentine von Oslo über Frankfurt nach Wien. Es würde also genügen, wenn Wien gesperrt ist, um beide Flüge zu verzögern. Wenn wir alle Startflughäfen, Zielflughäfen und auch alle Zwischenstopps berücksichtigen, sind insgesamt 67 Flughäfen für deine Idee relevant. Die Frage die sich also stellt: Ich kann 25 dieser 67 Flughäfen blockieren. Genügt dies um alle 31 Flüge ausfallen zu lassen?"

Amor ist für viel bekannt – aber nicht unbedingt für seine Geduld: „Und? Geht es sich aus?" „Nun ja, da beginnen die wirklich schlechten Nachrichten. Diese Fragestellung – also ob 25 blockierte Flughäfen genug sind zum Ausfall aller 31 Flugverbindungen – ist NP-vollständig." „NP-was?" „NP-vollständig. Das sind Fragestellungen, für die es sehr leicht überprüfbar ist, ob eine potentielle Lösung korrekt ist, aber das tatsächliche Finden der Lösung ... das ist eine andere Frage. Hätten wir die Lösung für unser Problem, also eine Liste von 25 Flughäfen, wäre es kein Problem, zu überprüfen, ob diese Lösung korrekt ist. Dies kann getan werden, indem man überprüft, ob es keine Flugverbindungen mehr zwischen den Start- und Zielflughäfen gibt." „Ja, aber wie finden wir diese 25 Flughäfen?" Man merkt wie unangenehm es für Zeus ist, die folgende Antwort zu geben: „Ehrlich gesagt, am besten du befragst ein Orakel. Die Korrektheit des Orakelspruchs können wir dann ja anschließend überprüfen." Amor hat deutlich weniger Vorbehalte bei der Wahl seiner Hilfsmittel. „Nichts leichter als das. Wenn ich mich beeile, sollte das Orakel in Delphi heute eigentlich noch offen haben."

Wiederum ein plötzlicher Ortswechsel. Amor ist inzwischen in Delphi angekommen und spaziert pfeifend auf das Orakelgebäude zu. Auf dem Schild vor dem modernen Bürogebäude steht nur 'Oracle Consulting Inc.'. Als Amor gerade die Gegensprechanlage betätigen will, erscheint eine hünenhafte Gestalt direkt hinter ihm. „Amor, Amor, das war doch zu erwarten. Nachdem du keine Idee hast, wie du diese Wette gewinnen kannst, willst du nun vom Orakel eine Lösung." Hephaistos blickt verärgert auf Amor herab. „Du weißt so gut wie ich, dass die Verwendung von Orakeln bei Götterwetten verboten ist. Regel 1: Ganz ohne Orakel." Amor überlegt noch kurz, mit Hephaistos zu diskutieren, aber er kennt dieses Leuchten in Hephaistos' Augen nur zu gut – hier kann er nicht mit Nachsicht rechnen.

Als Zeus sich den Bericht von Amor angehört hat, lässt er sich nachdenklich in seinen Sessel zurücksinken. „Du erinnerst dich daran, dass es für NP-vollständige Fragestellungen einfach zu überprüfen ist, ob eine mögliche Lösung korrekt ist oder nicht. Ein Computer könnte also alle möglichen Orakelsprüche erzeugen und dann überprüfen, ob eine dieser Antworten korrekt ist. Ich fürchte nur, dass dies viel zu lange dauert. Oder genauer gesagt", Zeus lässt wieder seine Finger über die Tastatur seiner Computers fliegen, „mehr als sechs Monate." Amor runzelt verwirrt die Stirn. „Sechs Monate? So viel Zeit habe ich nicht! Ich dachte du hast ein Großrechenzentrum zur Verfügung?" Zeus antwortet ruhig: „Ja, das habe ich. Aber das Problem sitzt tiefer. Ein Großrechenzentrum rechnet schon viel schneller als ein Laptop, aber im Grunde haben sie die gleichen Möglichkeiten und Beschränkungen. Also insbesondere keinen Zugang zu Orakeln."

Amor überlegt. Dies sind nicht die Probleme, mit denen er sich normalerweise auseinandersetzt. Seiner Erfahrung nach ist es immer eine gute Idee, bei Problemen Blumen zu schenken, aber wie dies in der momentanen Situation weiterhilft, will ihm nicht einfallen. „Gibt es nicht irgendeine andere Möglichkeit? Braucht man wirklich ein Orakel? Gibt es nicht eine schnellere Möglichkeit so ein Orakel mit deinem Rechenzentrum zu simulieren?" Amor ist immer noch sehr verwundert, dass eine Halle voller Computer nicht augenblicklich jede Antwort berechnen kann. Zeus lächelt leicht. „Da bist du auf eins der größten Probleme der modernen Informatik und Mathematik gestoßen. Die Frage, ob sich Orakel schnell am Computer simulieren lassen, oder **P=NP?**, wie die Frage in der Wissenschaft genannt wird, ist nach wie vor ungelöst. Aber die meisten Wissenschaftler sind davon überzeugt,

dass es nicht möglich ist Orakel zu ersetzen. Beziehungsweise, da Menschen ohnehin keinen Zugang zu Orakeln haben, lautet die Frage, ob es wirklich notwendig ist, alle möglichen Orakelsprüche überprüfen zu lassen."

„Ich ...", Thorsten ist am Ende seiner Weisheit, „Es war ... schön dich kennen gelernt zu haben." Das ist nicht das, was er eigentlich sagen will, aber es ist das, was er gesagt hat. „Vielleicht ... sehen wir uns ja wieder", sagt Myra und Thorsten nickt ohne Überzeugung. Das war es, denkt Thorsten.

Amor will sich so schnell nicht geschlagen geben: „Da muss es doch noch eine andere Möglichkeit geben. Da Menschen überhaupt keinen Zugang zu Orakeln haben, müssen sich doch schon Leute über dieses Problem Gedanken gemacht haben!" Nach etwas Überlegen meint Zeus: „Eine Möglichkeit fällt mir noch ein. Unser Problem ist ja, dass es viel zu viele mögliche Lösungen gibt – oder anders gesagt: viel zu viele mögliche Orakelsprüche. Dies liegt daran dass sich die Anzahl der Orakelsprüche bei jedem Flughafen mehr als verdoppelt. Bei 67 Flughäfen ergibt dies eine gigantische Zahl. Aber manchmal ist die Größe des Problems, also in unserem Fall die Anzahl der involvierten Flughäfen, gar nicht so relevant. Manchmal gibt es Eigenschaften des Problems, so genannte Parameter, die viel wichtiger sind als die Größe. Wenn ein solcher Parameter klein ist, kann man ein Problem schnell lösen, obwohl es viel zu lange dauern würde, alle möglichen Lösungen durchzuprobieren. Die Disziplin, die sich mit solchen Überlegungen beschäftigt, heißt Parametrisierte Komplexitätstheorie. Vielleicht hilft uns dieser Ansatz weiter."

Amor runzelt die Stirn. „Du meinst also, wir müssen so einen Parameter finden? Und wenn der dann für unser Problem passend ist, finden wir schnell eine Lösung? Wie findet man so einen Parameter?"

„Am besten wir zeichnen unser Problem einmal auf." Zeus faltet einen großen Bogen Papier auf und macht für jeden Flughafen einen Punkt darauf. Dann beginnt er Flugverbindungen einzuzeichnen. Für Thorsten verbindet er Madrid mit Wien und Wien mit Berlin, für Clémentine Oslo mit Frankfurt und Frankfurt mit Wien. Nach kurzer Zeit bildet sich ein dichtes Netz aus Linien auf dem Papier. Amor starrt ratlos darauf. „Hier soll man einen Parameter erkennen?" „Probiere es. Das ist meine letzte Idee", antwortet Zeus und beugt sich über den Papierbogen.

Nach ein paar Minuten reibt sich Amor die Augen. „Egal wie viel ich darauf starre, das einzige was ich sehe, ist, dass fast jeder Flug-

hafen mit fast jedem anderen verbunden ist. Wenn das nicht zufällig ein Parameter ist, dann hab ich echt keine Idee.“ Zeus schüttelt den Kopf. „Nein, das ist kein Param– Moment. Vielleicht ist es das, was wir wollen. Einen Moment, ich bin gleich wieder da.“

Zeus spaziert die schier endlosen Bücherreihen in seiner Bibliothek entlang und lässt seinen Blick schweifen. So sehr er auch im Laufe des letzten Jahrhunderts die Vorzüge der modernen Technologie zu schätzen gelernt hat, seine Bibliothek zu digitalisieren, hat er noch nicht über das Herz gebracht. Es war einfach ein erbauendes Gefühl, zu sehen, wieviel Wissen und Erkenntnis die Menschheit hervorgebracht hat. So ist es seine Hoffnung, dass sich vielleicht schon jemand über Amors Problem Gedanken gemacht hat. Dank einer guten Mischung aus Fernsicht, göttlicher Intuition und einer Portion Glück bleiben seine Augen ein wenig später an einer gebundenen Diplomarbeit hängen. ‘Complexity results and algorithms for Multicut on graphs of bounded clique-width‘ist der klobige Name. An dieser Stelle macht sich Zeus’ Interesse an Theoretischer Informatik bezahlt, denn zumindest ein wenig sind ihm die Begriffe vertraut.

„Ich glaube, ich habe ‘and‘, ‘for‘, ‘on‘, ‘of‘ verstanden, aber nicht einmal da bin ich mir ganz sicher.", meint Amor, als Zeus ihm den Titel der Arbeit vorgelesen hat. „Bist du sicher, dass das irgendetwas mit Aschewolken zu tun hat?“ Zeus lächelt. „Nun ja, nicht direkt. Aber Multicut ist genau das Problem, über das wir uns Gedanken machen. Bei dem Multicut-Problem geht es um die Suche nach Punkten, die weggeschnitten werden können, um eine Anzahl von verbundenen Anfangs- und Endpunkten zu trennen. Und das ist genau das, was wir brauchen. Wir haben viele Punkte, beziehungsweise Flughäfen, die mit Linien, oder eben Flugverbindungen, verbunden sind. Darüber hinaus gibt es noch 31 Anfangs- und Endpunkte, so wie Madrid und Berlin für Thorsten oder für Clémentine Oslo und Wien. Und wir haben 25 Schnitte zur Verfügung um diese 31 Anfangs- und Endflughäfen zu trennen.“

Amor ist noch nicht überzeugt: „So weit waren wir doch schon. Ich dachte dieses Problem lässt sich ohne Orakel nicht in vernünftiger Zeit lösen?“ „Da hast du Recht. Im Allgemeinen können wir dieses Problem nicht schnell lösen, aber in manchen Spezialfällen schon. Hier kommt die Parametrisierte Komplexitätstheorie wieder ins Spiel. Diese Arbeit zeigt nämlich, wie man das Problem schnell lösen kann, wenn es entweder sehr viele oder wenige Verbindungen gibt. Das steckt in ‘graphs

of bounded clique-width' und trifft genau auf unseren Fall zu. Wie du glücklicherweise bemerkt hast, sind nahezu alle Flughäfen miteinander verbunden. Somit können wir diese Ergebnisse verwenden."

„Nun gut, nun gut", Amor spielt nervös mit seinen Locken, „Wie lange dauert das also nun? Viel Zeit ist nicht mehr." Zeus wendet sich wieder seinem Computer zu und startet die Berechnungen. Nach einiger Zeit zieht er eine goldene Taschenuhr aus seiner Jackettasche und wirft einen nachdenklichen Blick darauf. „Nun ja .. es wird knapp."

In Madrid sitzt Thorsten inzwischen im Flugzeug. Um ihn herum befüllen die Leute die Gepäckfächer. Er starrt trübsinnig durch das kleine Fenster. Auch Clémentine blickt durch ihr Fenster auf das Rollfeld des Oslo International Airport. In ihr beginnt sich das unbestimmte Gefühl zu verstärken, irgendwas vergessen zu haben. Oder verpasst? Beiden, Thorsten und Clémentine, steigt in diesem Moment der selbe Gedanke in den Kopf: Ist es möglich, dass sich das Bauchgefühl nicht irrt? Kann es sein, dass es das beste wäre, aufzustehen und dieses Flugzeug zu verlassen? Macht das Sinn? Nach kurzem Nachdenken verneinen beide die Frage. Amor hätte sich an dieser Stelle an den Kopf gegriffen, wenn er diese Gedanken gehört hätte. Schon oft hatte er feststellen müssen, dass Menschen bei kaum einer Frage mehr irrten, als bei der, ob es Sinn macht, von ihrem geplanten Weg abzuweichen. Amors Erfahrung zufolge machte das immer Sinn.

Amor geht unruhig von einer Seite des Zimmers zur anderen. Er hofft inständig, dass diese Berechnungen bald fertig sein werden. Bei dem Gedanken an Hephaistos verkrampft sich sein Magen. Es ist ein ungeschriebenes Gesetz, dass bei einer Götterwette der Verlierer dem Sieger einen Wunsch gewährt. Bei Hephaistos wäre das gewiss nichts Nettes. Zuletzt hatte er eine Woche in seiner Schmiede arbeiten müssen. Der Geruch von Rauch war tagelang nicht von ihm gewichen. Und jetzt hing alles an diesem seltsamen Multicut-Problem und Parametrisierter Komplexitätstheorie! Die Welt war manchmal schon ein überraschend seltsamer Ort, selbst für ihn als Gott.

In dem Moment tönt Zeus ruhige Stimme durch den Raum: „Fertig." Amor fährt herum. „Fertig? Das waren doch nur ..." Zeus blickt wiederum auf seine Taschenuhr. „3 Minuten." Er lächelt – bis ihn Amor stürmisch umarmt. Zeus schiebt ihn mit Nachdruck wieder von sich weg. „Schon gut, schon gut. Ich mache mich jetzt besser daran ein paar Aschewolken über Europa zu verteilen." Doch Amor lässt sich nicht

so leicht abwimmeln. „Also sag, wie hat das funktioniert? Zuerst 6 Monate und dann doch nur drei Minuten?“

„Diese Diplomarbeit enthält eine Methode, mit der die Berechnungsdauer nicht mehr wirklich von der Anzahl der Flughäfen abhängt, sondern nur mehr von der Anzahl der Paare und der ‘clique-width‘. Auch in Summe sind diese beiden Parameter deutlich geringer als die Anzahl der Flughäfen. Und schon ein minimaler Unterschied zwischen diesen beiden Werten erlaubt die Berechnung der Lösung in einem Bruchteil der Zeit. Man kann also sagen, dass Multicut relativ schnell lösbar ist, solange es nicht zu viele Paare gibt und die meisten Flughäfen verbunden sind.“

Es ist noch nicht viel Zeit vergangen, als Thorsten schnellen Schrittes zurück in die Wartehalle eilt. Die erfreuliche Mitteilung des Piloten, dass der Flug aufgrund einer Aschewolke abgesagt werden muss, klingt noch in seinen Ohren nach. Dort, wo er sie zurückgelassen hat, sitzt immer noch Myra. Ihre Blicke treffen sich. Und beide lächeln.

Clémentine geht wesentlich ruhiger zurück zur Wartehalle. So recht will sich bei ihr Ärger darüber nicht einstellen, dass ihr Flug ausgefallen ist. Unbewusst steuert sie zurück zu ihrem vorigen Sitzplatz. In diesem Moment fällt ihr Blick auf Luc. Obwohl einige Jahre vergangen sind, erkennt sie ihn sofort. Erinnerungen steigen in ihr hoch. Als würde er die Blicke spüren, blickt Luc auf. Auch er erkennt sie sofort. Und beide lächeln.

Wir befinden uns wieder am Olymp, dort wo alles seinen Anfang nahm. Der schwarze König tobt. Mit lautem Geschrei und wilden Gesten jagt er seine Untergebenen über den Spielplan. Die Schmach der letzten Partie sitzt ihm noch tief in den Knochen. Von der eigenen Dame verraten. Mehr und mehr umkreist er mit seinen Truppen den gegnerischen König. Doch die weiße Dame wehrt heldenhaft jeden Angriff ab. Man könnte fast meinen, die Angriffe zielen nicht auf den König sondern auf die Dame ab. Auf die weiße Dame mit den feuerroten Locken.

„Dir ist schon bewusst, dass du meinen König attackieren musst und nicht meine Dame?“, sagt Amor grinsend. Hephaistos knurrt verächtlich und meint: „Diese Verräterin erwische ich noch.“ „Ich glaube nicht. Schach.“ Hephaistos reißt die Augen auf. Wiederum ein Knurren von ihm, dann schleudert er das Schachbrett vom Tisch. Als er sich wieder beruhigt hat und einen großen Schluck Guinness genommen hat,

wendet er sich wieder Amor zu. „Nun gut, ich gebe mich geschlagen. Vielleicht steckt in deiner Liebe doch irgendetwas ...“ Er bricht ab. Für ein paar Sekunden starrt er in sein Glas und spricht dann weiter: „Du hast die Götterwette also gewonnen und dir steht ein Wunsch frei. Was willst du?“ Es ist offensichtlich, dass die Nachdenkpause von Amor nur gespielt ist. „31 Paare, auf Flughäfen überall in Europa, haben soeben ein wenig Zeit bekommen. Ich wünsche mir von dir 62 neu geschmiedete Pfeile für meinen Bogen.“ Hephaistos nickt. „Ach ja“, fügt Amor hinzu, „eins wollte ich dir noch sagen: Es geht auch ganz ohne Orakel.“

Für Mimi,
die mich davon überzeugt hat,
diese Geschichte tatsächlich aufzuschreiben.

# BIBLIOGRAPHY


[1] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. (Cited on page 25.)

[2] Hans Bodlaender. Treewidth: Algorithmic techniques and results. In *Mathematical Foundations of Computer Science 1997*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer Berlin / Heidelberg, 1997. (Cited on page 23.)

[3] Hans Bodlaender. Treewidth: Characterizations, applications, and computations. In *Graph-Theoretic Concepts in Computer Science*, volume 4271 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin / Heidelberg, 2006. (Cited on page 23.)

[4] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993. (Cited on page 23.)

[5] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. ISBN 0-89871-432-X. (Cited on pages 14 and 19.)

[6] Gruia Calinescu, Cristina G. Fernandes, and Bruce Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. *Journal of Algorithms*, 48(2):333 – 359, 2003. (Cited on pages 9, 19, 30, 33, 66, and 72.)

[7] Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. (Cited on page 23.)

[8] Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985. (Cited on page 38.)

[9] Marie-Christine Costa, Lucas Létocart, and Frédéric Roupin. Minimal multicut and maximal integer multiflow: A survey. *European Journal of Operational Research*, 162(1):55–69, 2005. (Cited on page 9.)

[10] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. *Handbook of theoretical computer science, volume b: formal models and sematics*, pages 193–242, 1990. (Cited on pages 9 and 25.)

[11] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77 – 114, 2000. (Cited on pages 20, 23, and 33.)

[12] Bruno Courcelle and Sang-il Oum. Vertex-minors, monadic second-order logic, and a conjecture by Seese. *J. Comb. Theory Ser. B*, 97(1): 91–126, 2007. (Cited on page 25.)

[13] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993. (Cited on page 19.)

[14] 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. (Cited on pages 10 and 26.)

[15] 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. (Cited on pages 9, 19, 44, and 72.)

[16] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. (Cited on page 24.)

[17] Ehab S. El-Mallah and Charles J. Colbourn. The complexity of some edge deletion problems. *IEEE transactions on circuits and systems*, 1988. (Cited on page 42.)

[18] David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41 – 55, 1992. (Cited on page 30.)

[19] 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. (Cited on page 23.)

[20] Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008. (Cited on page 73.)

[21] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Birkhäuser, 2006. (Cited on page 24.)

[22] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Clique-width: on the price of generality. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 825–834, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics. (Cited on page 73.)

[23] Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001. (Cited on page 19.)

[24] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0716710455. (Cited on page 27.)

[25] 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. (Cited on pages 25, 30, 66, and 72.)

[26] Georg Gottlob and Stephanie Tien Lee. A logical approach to multicut problems. *Information Processing Letters*, 103(4):136 – 141, 2007. (Cited on pages 9, 10, 11, 34, 59, and 72.)

[27] Georg Gottlob and Reinhard Pichler. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. *SIAM J. Comput.*, 33(2):351–378, 2004. (Cited on page 71.)

[28] Martin Grohe. Descriptive and parameterized complexity. In *Proc. CSL'99, Volume 1683 of LNCS*, pages 14–31. Springer-Verlag, 1999. (Cited on page 27.)

[29] Martin Grohe. Algorithmic Meta Theorems. In *Graph-Theoretic Concepts in Computer Science, 34th International Workshop, WG 2008, Durham, UK*, Lecture Notes in Computer Science. Springer, 2008. (Cited on page 25.)

[30] Jiong Guo and Rolf Niedermeier. Fixed-parameter tractability and data reduction for multicut in trees. *Networks*, 2005. (Cited on page 9.)

[31] Jiong Guo, Falk Hüffner, Erhan Kenar, Rolf Niedermeier, and Johannes Uhlmann. Complexity and exact algorithms for multicut. In *SOFSEM 2006: Theory and Practice of Computer Science, 32nd Conference on Current Trends in Theory and Practice of Computer Science, Merín, Czech Republic, January 21-27, 2006*, volume 3831 of *Lecture Notes in Computer Science*, pages 303–312. Springer, 2006. (Cited on pages 9, 44, and 72.)

[32] Frank Harary. *Graph Theory*. Westview Press, 1994. (Cited on page 37.)

[33] 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. (Cited on page 19.)

[34] Stephan Kreutzer. Algorithmic meta-theorems. *CoRR*, abs/0902.3616, 2009. (Cited on page 25.)

[35] Dániel Marx. Parameterized graph separation problems. In *Theor. Comput. Sci.*, volume 351, pages 394–406. 2006. (Cited on pages 44, 57, and 72.)

[36] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2002. (Cited on page 27.)

[37] Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5(1), 2008. (Cited on page 23.)

[38] Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Comb. Theory Ser. B*, 96(4):514–528, 2006. (Cited on page 19.)

[39] Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Multicut algorithms via tree decompositions. In *Algorithms and Complexity, 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, Lecture Notes in Computer Science, pages 167–179. Springer, 2010. (Cited on page 28.)

[40] Richard M. Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, 1972. (Cited on page 31.)

[41] Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991. (Cited on page 19.)

[42] Neil Robertson and Paul D. Seymour. Graph minors. I. Excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61, 1983. (Cited on page 19.)

[43] Zhifeng Sun. Multicut survey. Technical report, 2008. URL http://www.ccs.neu.edu/home/austin/papers/multicut.pdf. (Cited on page 9.)

[44] Egon Wanke. k-NLC graphs and polynomial algorithms. *Discrete Appl. Math.*, 54(2-3):251–266, 1994. (Cited on page 19.)

[45] Mihalis Yannakakis. Node- and Edge-Deletion NP-Complete Problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing, 1-3 May 1978, San Diego, California, USA*, pages 253–264, New York, NY, USA, 1978. ACM. (Cited on page 35.)