Multivariate Algorithmics in Biological Data Analysis

Vorgelegt von
Diplom-Informatiker (Bioinformatik)
Johannes Gabriel Uhlmann
geboren in Bad Soden am Taunus

Von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
Dr. rer. nat.

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Uwe Nestmann
Gutachter:	Prof. Dr. Rolf Niedermeier
Gutachter:	Prof. Dr. Peter Damaschke
Gutachter:	Prof. Dr. Till Tantau

Tag der wissenschaftlichen Aussprache: 17. Juni 2011

Berlin 2011
D83

ISBN: 978-3-7983-2351-3 (Druckausgabe)
ISBN: 978-3-7983-2352-0 (Online-Version)

Berlin 2011

Druck/Printing: docupoint GmbH Magdeburg
Otto-von-Guericke-Allee 14
D-39179 Barleben

Vertrieb/Publisher: Universitätsverlag der TU Berlin
Universitätsbibliothek
Fasanenstrasse 88 (im VOLKSWAGEN-Haus)
D-10623 Berlin
Tel. +49 (0)30 314 761 31 / Fax: +49 (0) 30 314 761 33
Email: publikationen@ub.tu-berlin
http://www.univerlag.tu-berlin.de

Preface

This thesis covers parts of my research on fixed-parameterized algorithms for NP-hard problems that arise in the context of biological data analysis.

My research was funded by the Deutsche Forschungsgemeinschaft (DFG) since October 2007 until February 2011 within the project PABI, NI 369-7. From October 2007 until December 2010, I stayed with the Friedrich-Schiller-Universität Jena and most results of this thesis were established within this period. Following my supervisor Rolf Niedermeier, I moved to TU Berlin in January 2011. I wish to express my sincere thanks to Rolf Niedermeier for giving me the opportunity to work in his group.

Furthermore, I want to thank my (partially former) colleagues Nadja Betzler, René van Bevern, Robert Bredereck, Michael Dom, Jiong Guo, Sepp Hartung, Falk Hüffner, Christian Komusiewicz, Hannes Moser, Rolf Niedermeier, André Nichterlein, and Mathias Weller for fruitful discussions and the positive work atmosphere.

Moreover, I owe sincere thanks to my coauthors Nadja Betzler, Robert Bredereck, Britta Dorn (Universität Ulm), Michael R. Fellows (Charles Darwin University, Australia), Rudolf Fleischer (Fudan University, China), Jiong Guo (Universität des Saarlandes), Sepp Hartung, Falk Hüffner, Iyad A. Kanj (DePaul University, USA), Christian Komusiewicz, Rolf Niedermeier, Dominikus Krüger (Universität Ulm), André Nichterlein, Yihui Wang (Fudan University, China), Mathias Weller , and Xi Wu (Fudan University, China) for the interesting collaborations.

Last but not least, I am indebted to several anonymous referees from various conferences and journals for comments that have improved the presentation of our results.

This thesis emerges from collaborations with various research partners. In the following, I describe my specific contributions and point to the publications establishing the basis for this thesis. In addition, I contributed to the publications [18, 20, 21, 62, 92, 95, 96, 98, 99, 136, 155, 156], which are not part of my thesis.

Part II: Fitting Biological Data with Combinatorial Structures. Chapter 4 is concerned with the investigation of Cluster Editing and Cluster Deletion for several alternative parameters. The results in Chapter 4 were obtained in close cooperation with Christian Komusiewicz. I did the main work on the fixed-parameter algorithms with respect to the cluster vertex deletion number. The results of Chapter 4

were presented at the *37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)* [122].

Chapter 5 introduces a generalization of Cluster Editing that allows for overlap between the clusters. These results were presented at the *15th Annual International Computing and Combinatorics Conference (COCOON '09)* [70]. The journal version appeared in *Discrete Optimization* [71]. Most of the results were obtained in group discussions while all authors stayed with the Friedrich-Schiller-Universität Jena.[1] I worked out several details and finished the NP-hardness proofs. Moreover, I was significantly involved in the research that lead to the two kernelization results in Section 5.5. In particular, I substantially contributed to the polynomial-size problem kernel for 2-Vertex-Overlap Deletion.

Chapter 6 focusses on the M-Hierarchical Tree Clustering problem. Suggested by Jiong Guo, the examination of M-Hierarchical Tree Clustering was initiated in a "Studienarbeit" by Sepp Hartung (also co-supervised by me). Sepp Hartung devised an $O(3^k)$-size search tree and an $O(k^2)$-element problem kernel. I came up with the basic idea that lead to the $O(Mk)$-element problem kernel. For the conference version [90], which appeared in the *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*, Christian Komusiewicz simplified the proof for the correctness of Reduction Rule 6.2. While preparing this thesis I further simplified the presentation. In particular, I came up with Lemma 6.1 which makes the proof of Reduction Rule 6.2 trivial and yields the basis for the $O(2.56^k)$ search tree algorithm. A full version of the conference paper containing these new results has been submitted to the *Journal of Classification* [91]. In addition, I want to thank Sepp Hartung who mainly accomplished the implementation and experimental work.

In Chapter 7, the main results are a cubic-vertex kernel and an improved $O(3.68^k)$-size search tree for Minimum Flip Consensus Tree. I had the basic idea for Reduction Rule 7.4 and the analysis of the kernel-size. The details of the kernelization were worked out in close cooperation with Christian Komusiewicz. The kernelization results were presented at the *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'08)* [121]. Moreover, I devised the improved $O(3.68^k)$-size search tree for Minimum Flip Consensus Tree that is not contained in the conference paper. We submitted a full version combining the conference paper with the search tree algorithm to *Algorithmica* [123].

Part III: Constrained Search Problems Chapter 8 investigates the parameterized complexity of Interval Constrained Coloring with respect to several parameters. The parameter identification by "deconstructing intractability" were performed in meetings with all authors. My technical main contribution are the dynamic programming algorithms. The results were presented at the *20th Annual Symposium on Combinatorial Pattern Matching (CPM '09)* [119]. Chapter 8 follows the journal paper that appeared in the *Journal of Discrete Algorithms* as part of the *20th Anniversary Edition of the Annual Symposium on Combinatorial Pattern Matching* [120]. I want to thank Michael R. Fellows for general discussions about the deconstructing intractability approach and Nadja Betzler for pointing us to the Interval Constrained Coloring problem. I am grateful to Ernst Althaus for providing us with real-world data. Moreover, I am indebted to our student Sven Thiel for his great job

[1] Michael R. Fellows stayed in Jena as a recipient of a Humboldt Research Award.

concerning implementation and experimentation.

Chapter 9 is concerned with the design of improved fixed-parameter algorithms for Haplotype Inference by Parsimony (HIP) and Constrained Haplotype Inference by Parsimony (CHIP). I came up with the observation that not all inference graphs must be considered in the algorithms which is decisive for the improved running time bound. I worked out the details of the algorithms in close cooperation with Mathias Weller. The results appeared in the *Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching (CPM '10)* [76]. The paper [76] also identifies an interesting polynomial-time special case (called Induced Haplotype Inference by Parsimony) that is not considered in this thesis.

Contents

Part I

Introduction

Chapter 1

Introduction

This thesis is concerned with the development of fixed-parameter algorithms for solving NP-hard combinatorial problems arising in algorithmic bioinformatics. More specifically, we consider problems that model tasks in data clustering, construction of phylogenetic trees, predicting information on the tertiary structure of proteins, and inferring haplotype information from given genotypes. Some of the problems also find applications in other areas.

We start with an introduction into the field of parameterized algorithmics including some recent developments followed by an overview of the problems and results in Section 1.2.

1.1 Algorithmic Approach

The problems considered in this thesis are NP-hard. It is commonly believed that, in general, there are no efficient (that is, polynomial-time) algorithms for optimally solving NP-hard problems. Pioneered by Downey and Fellows [63], parameterized algorithmics has established within the last 20 years as one of the main approaches to tackle NP-hard problems [77, 137]. Classically, the computational complexity of problems is measured only with respect to the overall input size n. Parameterized complexity aims at a more fine-grained complexity analysis. It provides a two-dimensional framework for studying the computational complexity of problems. One dimension is the input size n and the other one is the *parameter* k (usually a positive integer), capturing an additional aspect of the input instances. The basic idea is to restrict the nonpolynomial part of the running time to the parameter k. A problem is called *fixed-parameter tractable (fpt)* if it can be solved in $f(k) \cdot \text{poly}(n)$ time, where f is a computable function only depending on k. In settings where the considered parameter is small, fixed-parameter algorithms may provide efficient solving strategies despite general NP-hardness.

Although a parameter can basically be everything, so far the majority of fixed-parameter tractability results is given with respect to only few "standard parameterizations". For example, the cost of the solution is the standard parameter when considering optimization problems. As an other example, the treewidth (measur-

ing the "tree-likeness" of a graph) is another standard graph parameter, allowing for fixed-parameter tractability for many relevant graph problems[1]. Moreover, most fixed-parameter tractability results are only with respect to a single parameter. However, for a specific problem there may be several meaningful parameterizations and also the combination of two or more parameters may be relevant. In particular, fixed-parameter algorithms for new parameterizations can help to extend the range of instances that can be solved in practice. This might be the case when the standard parameter is not really small for several real-world instances, or the problem is intractable when parameterized by a single parameter. Thus, *multivariate algorithmics* (cf. [69, 138]) proposes a systematic study of several parameters and the combination of two or more single parameters. To this end, the identification of meaningful parameters is a crucial and nontrivial step in multivariate algorithmics. This thesis provides improved fixed-parameter algorithms for standard parameterizations for some problems and starts a systematic multivariate algorithmics approach for other problems. Before stating our concrete results in Section 1.2, we highlight two approaches used in this thesis that concern multivariate algorithmics and "nonstandard parameterizations".

First, we consider parameters that typically have smaller values than standard parameters. Such parameters are denoted as "refined parameters". It is interesting from a theoretical as well as from a practical point of view to investigate whether a problem is still fixed-parameter tractable with respect to a refined parameter. In particular, for problems whose fixed-parameter tractability has been intensively investigated with respect to a standard parameter, the investigation of its parameterized complexity with respect to refined parameters might help to obtain further improved algorithms. In Section 2.3, we provide more details and point to recent work in this direction.

Second, we propose the approach of "deconstructing intractability" to identify relevant parameterizations. The basic idea is to look at the NP-hardness proofs asking why the produced instances might be artificial. Parameters capturing these "artificial aspects" of the produced instances are natural candidates for a parameterized complexity study. Regarding this natural approach, our main contribution is to "deconstruct intractability" systematically and in combination with a multivariate complexity analysis. In particular, for problems for which there are no obvious standard parameterizations a systematic deconstructing-intractability approach can be fruitful. See Chapter 8 for more details and a concrete case study.

Finally, we want to highlight one of the key techniques used in this thesis. Up to now, several concepts and techniques for establishing fixed-parameter tractability have been developed. Besides depth-bounded search trees, enumerative strategies, and dynamic programming based approaches, one main focus of this thesis is on *kernelization*. Here, the basic idea is to transform the input instance by the application of one or more polynomial-time *data reduction rules* into an "equivalent" instance whose size is bounded by a function of the parameter. If this function is a polynomial, we say that the problem admits a polynomial-size problem kernel. Kernelization is a polynomial-time preprocessing algorithm that can be combined with different solving strategies. It has been recognized as one of the theoretically and practically most interesting techniques of parameterized algorithmics [68, 97, 29].

[1] Often, the term "standard parameter" is exclusively used to refer to the cost or quality of a solution. In this thesis, we also denote classical structural parameters such as the treewidth as "standard parameters".

1.2 Organization and Results

The thesis consists of four parts. The first part provides an introduction into the employed theoretical concepts and algorithmic methods. Moreover, it introduces basic notations used throughout this thesis. The new results are presented in Part II and Part III. The last part concludes the thesis. In the following, we briefly describe the investigated problems and corresponding contributions.

Part II (Chapters 3 to 7) This part of the thesis investigates problems arising in the context of clustering of data and the construction of phylogenetic trees. To this end, we investigate four combinatorially similar problems. Two of them have been introduced in the context of data clustering. The third problem is relevant for data clustering as well as for phylogenetics. The fourth problem arises in phylogenetics. In Chapter 3, we introduce the considered problems, summarize our results, and present a universal data reduction rule needed for several kernelization results in this part.

Data clustering is one of the most fundamental problems in data analysis. The task is to group together similar data items into clusters such that the data items in each cluster are more closely related to each other than to data items of other clusters. There is a vast amount of models and methods for data clustering. We consider graph-based data clustering, where the main goal is to partition the vertex set of a graph into clusters such that there are many edges within each cluster but only few edges between the clusters. This leads to the first of our problems.

Chapter 4 concentrates on the Cluster Editing problem, one of the most intensively investigated problems in graph-based data clustering. Given an undirected graph, the task is to modify the edge set of the graph as little as possible to obtain a disjoint union of cliques. The main contribution in Chapter 4 is to show that Cluster Editing is fixed-parameter tractable with respect to a refined parameter, namely the cluster vertex deletion number. This number is typically smaller than the standard parameter "solution size". This answers an open question of Dehne [58]. Moreover, we briefly discuss other alternative parameterizations for Cluster Editing. Chapter 4 is based on [122].

Chapter 5 introduces a new model for graph-based data clustering with overlaps, generalizing the model used for Cluster Editing. This model allows a certain amount of overlap of the clusters that can be specified by an overlap number s. We obtain a computational complexity dichotomy (polynomial-time solvable versus NP-hard) for the underlying edge modification problems. Moreover, we study the parameterized complexity with respect to the number of allowed edge modifications, achieving fixed-parameter tractability in case of constant overlap values and parameterized intractability for unbounded overlap values. Moreover, we present polynomial-size problem kernels for two problems in this context. Chapter 5 follows [71].

Chapter 6 studies the parameterized complexity of the M-Hierarchical Tree Clustering problem with respect to the standard parameter "cost of the solution". Given pairwise dissimilarity data on pairs of elements to be clustered, the task is to find a hierarchical representation of the input data, that is, to build a rooted tree of depth $M + 1$ such that similar objects (with respect to the input data) are close to each other in the tree. More specifically, in M-Hierarchical Tree Clustering so-called ultrametric trees are considered, where each leaf has the same distance to the root.

M-Hierarchical Tree Clustering is also closely related to the (re)construction of phylogenetic trees. In phylogenetics, the evolutionary relationship between species is usually depicted by arranging the species in a phylogenetic tree. Phylogenetic trees are usually inferred based on dissimilarities in the physical or genetic characteristics for a given set of species, reflecting their evolutionary distances. In an idealized model, evolutionary data is ultrametric. In this context, M-Hierarchical Tree Clustering can be seen as the problem to correct the input data as little as possible to obtain an ultrametric tree. The results for M-Hierarchical Tree Clustering comprise a search tree and two kernelizations with respect to different parameterizations. More precisely, we present an $O(k^2)$-element and an $O(M \cdot k)$-element kernel and a size-$O(2.562^k)$ search tree. Chapter 6 is based on [90].

There are several methods that can be used to construct evolutionary trees. Hence, it is an important task to combine the information from several trees. The central problem in Chapter 7 is the Minimum-Flip Consensus Tree problem that arises in the task to combine several rooted phylogenetic trees in one consensus tree. The NP-hard combinatorial problem that has to be solved is to destroy all so-called induced M-graphs in a bipartite graph by at most k edge modifications. We improve previously known fixed-parameter algorithms by presenting an $O(3.68^k)$-size search tree algorithm. The previously known search tree algorithm has size $O(4.42^k)$ [27]. Our main contribution is a cubic-vertex kernel with respect to k. This is the first nontrivial kernelization result for Minimum-Flip Consensus Tree. Chapter 7 is based on [121].

Part III (Chapters 8 and 9) Part III investigates two further combinatorial problems arising in molecular biology. The first problems is useful to obtain information about the 3-dimensional structure of a protein based on mass spectrometry data. The second problem is concerned with haplotype inference.

In Chapter 8, we focus on the Interval Constrained Coloring problem. Interval Constrained Coloring (ICC) appears in the interpretation of experimental data in biochemistry dealing with protein fragments. More specifically, ICC models the task to predict information about the tertiary structure of a protein based on hydrogen/deuterium exchange rates for its fragments, which can be obtained by mass spectrometry experiments. For a protein consisting of n amino acids, the input of ICC consists of a set of m integer intervals in the range 1 to n and each interval is associated with a multisets of colors. It asks whether there is a "consistent" coloring for all integer points from $\{1, \ldots, n\}$ that complies with the constraints specified by the color multisets. In the biochemical application, the different colors correspond to different hydrogen/deuterium exchange rates and a solution for ICC gives information about the location of an amino acid residue. Our main contribution is to identify several natural parameters for Interval Constrained Coloring based on a systematic "deconstructing intractability" approach. For the obtained parameterizations we present several fixed-parameter tractability results. We substantiate the usefulness of this "multivariate algorithmics approach" by presenting experimental results with real-world data. Chapter 8 follows [120].

In Chapter 9, we investigate the Parsimony Haplotyping problem. Parsimony Haplotyping is the problem of finding a smallest-size set of haplotypes that can explain a given set of genotypes. We also consider a "constraint version" of Parsimony

HAPLOTYPING where the explaining haplotypes must be chosen from a given pool of plausible haplotypes [72]. Haplotyping is important for the investigation of genetic mutations and diseases. We propose improved fixed-parameter tractability results with respect to the parameter "size of the target haplotype set" k by presenting algorithms with exponential running time factor k^{4k}. The previously known algorithms had running time factors k^{k^2+k} [151] and $k^{O(k^2)}$ [72]. Chapter 9 is based on [76].

Chapter 2

Basic Concepts and Notation

In this chapter, we give a short introduction into theoretical concepts and algorithmic methods used in this work. Moreover, we introduce basic notation employed throughout this thesis.

2.1 Computational Complexity and NP-Hardness

In complexity theory, usually decision problems are considered. Formally, a decision problem is encoded by a language $L \subseteq \Sigma^*$ over a finite alphabet Σ and the task is to decide, for a given $x \in \Sigma^*$, whether $x \in L$.[1] The computational complexity of a problem is measured by the resources needed to solve it. The running time and the space consumption are the two main measures. In this thesis, we mainly deal with time complexity. In complexity theory, the goal is to classify problems into classes of similar complexity. The most prominent complexity classes are P and NP. A problem is in P if it can be solved in polynomial time by a deterministic Turing machine and in NP if it can be solved in polynomial time by a nondeterministic Turing machine [81, 142].

It is commonly believed that the "hardest" problems in NP cannot be solved by deterministic algorithms in polynomial time and, hence, P $\neq$ NP. In order to define the hardest problems in NP, the concept of *polynomial-time many-one reduction* was introduced to show that a problem A is at least as hard as a problem B: A problem $B \subseteq \Sigma^*$ reduces to a problem $A \subseteq \Sigma^*$ (abbreviated by $B \leq_p A$) if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in B$ if and only if $f(x) \in A$ for all $x \in \Sigma^*$. A problem A is called *NP-hard* if $B \leq_p A$ for all problems $B \in$ NP. Moreover, an NP-hard problem A with $A \in$ NP is called *NP-complete*. In other words, an NP-complete problem is at least as hard as any problem in NP.

The central observation to substantiate the assumption P $\neq$ NP is that if one NP-hard problem is polynomial-time solvable, then all problems in NP are polynomial-time solvable. However, since thousands of NP-complete problems have been investigated without finding polynomial-time algorithms, it is commonly believed that there are no polynomial-time algorithms for NP-hard problems.

[1] Indeed, most results in this thesis refer to the decision version of a problem. We stress that the presented algorithms can easily be adapted to actually construct an optimal solution.

In summary, for an NP-complete problem, a running time of the form $O(2^{n^c})$ for some constant $c > 0$ seems to be unavoidable when the running time is measured only in the input size n.

2.2 Parameterized Complexity and Multivariate Algorithmics

Introduced by Downey and Fellows [63], parameterized complexity has established itself within the last 20 years as one of the main approaches for coping with the computational intractability of NP-hard problems. We refer to the textbooks [63, 77, 137] for a comprehensive introduction.

Parameterized complexity is a two-dimensional framework for studying the computational complexity of problems, one dimension is the input size n (as in classical complexity theory), and the other one is the *parameter* k. That is, in parameterized complexity, a problem always comes with a parameter.

Definition 2.1. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \Sigma^*$, where Σ is a finite alphabet. The second component is called the *parameter* of the problem.

All parameters considered in this thesis are nonnegative integers or tuples of nonnegative integers. We refer to a parameter consisting of a tuple of nonnegative integers as a *combined parameter*. The central notion in parameterized complexity is that of *fixed-parameter tractability*. Here, the basic idea is to restrict the combinatorial explosion that seems unavoidable for any exact solving strategy for NP-hard problems to a function that depends only on the parameter.

Definition 2.2. A parameterized problem L is *fixed-parameter tractable* (fpt) if there is an algorithm that decides in $f(k) \cdot n^{O(1)}$ time whether $(x, k) \in L$, where f is an arbitrary computable function depending only on k. The complexity class containing the fixed-parameter tractable problems is called FPT.

An algorithm with a running time bound as in Definition 2.2 is called *fixed-parameter algorithm*, or, synonymously, *parameterized algorithm*.

Observe that in the definition of fixed-parameter tractability the degree of the polynomial does not depend on k. Hence, the concept of fixed-parameter tractability is stronger than the notion of "polynomial-time solvability for constant parameter values". Indeed, parameterized complexity theory can be viewed as driven by contrasting the two function classes $f(k) \cdot n^{O(1)}$ (the "good" functions) and $O(n^{g(k)})$ (the "bad" functions) [64]. The problems that can be solved in the running time $O(n^{g(k)})$ form the parameterized complexity class XP.

For many parameterized problems fixed-parameter algorithms have been found. However, for many parameterized problems there is strong evidence that they are not fixed-parameter tractable. Downey and Fellows [63] developed a formal framework for showing *fixed-parameter intractability* by means of *parameterized reductions*. A parameterized reduction from a parameterized problem L to another parameterized problem L' is a function defined as follows. Given an instance (x, k), it computes in $f(k) \cdot n^{O(1)}$ time (where f is a computable function) an instance (x', k') such that

- (x, k) is a yes-instance of problem L if and only if (x', k') is a yes-instance of problem L' and

- k' only depends on a computable function in k.

The basic complexity class for fixed-parameter intractability is called W[1] and there is good reason to believe that W[1]-hard problems are not fixed-parameter tractable [63, 77, 137]. In this sense, W[1]-hardness is the parameterized complexity analog of NP-hardness. The next level of parameterized intractability is covered by the complexity class W[2] with W[1]⊆W[2].

2.2.1 Multivariate Algorithmics

A multivariate algorithm analysis extends a parameterized algorithm analysis in the sense that it *systematically* investigates the influence of *several parameters* on the computational complexity of a problem [69, 138]. Up to now, the majority of fixed-parameter tractability results in the literature have been obtained with respect to only few single "standard parameters" such as the size or cost of a minimum solution (the standard parameter when considering optimization problems) or the treewidth of a graph. Clearly, for a specific problem there may be several meaningful parameterizations and also the combination of two or more parameters is relevant. For example, it may happen that the standard parameter is not really small in practice or that a problem is W[1]-hard when parameterized only by a single parameter. In such cases, looking at several parameters or the combination of two or more parameters can help to find efficient algorithms for relevant special cases. Hence, multivariate algorithmics should be seen as an effort to systematically investigate the influence of several parameterizations and, in particular, the combination of two or more single parameterizations on the computational complexity of a problem.

Many interesting questions arise when extending the investigation from one to several parameters. Here, we give two concrete examples.

First, consider a problem for which two parameters p_1 and p_2 have been identified. It may turn out that the problem is NP-hard even for constant parameter values of p_1 or p_2. In a multivariate framework, this directly leads to the following questions.

- Is the problem fixed-parameter tractable with respect to the combined parameter (p_1, p_2)?

- Is the problem fixed-parameter tractable with respect to parameter p_1 for constant parameter values of p_2 (or vice versa)?

- Is the problem NP-hard for constant parameter values of both parameters?

Second, assume that a problem is fixed-parameter tractable for a combined parameter (p_1, p_2). This directly raises the question whether there are fixed-parameter algorithms with qualitatively different combinatorial explosions in their running times. For example, algorithms coming with the incomparable combinatorial explosions $p_1^{p_2}$ and $p_2^{p_1}$, respectively, can both be useful for solving specific real-world instances.

2.3 Parameter Identification

Parameterized algorithmics or, more generally, multivariate algorithmics aims at a fine-grained complexity analysis of problems by investigating the influence of parameters on the computational complexity of a problem. Thus, the identification of meaningful parameters is a fundamental and nontrivial step in multivariate algorithmics [69, 137, 138]. In this section, we discuss some aspects of parameter identification.

When considering optimization problems a standard parameterization refers to the size or cost of the solution set of the underlying problem. For many problems there exist sophisticated fixed-parameter algorithms employing this standard parameterization, providing efficient algorithms for small parameter values. Also most contributions of the thesis are with respect to this standard parameterization. However, for some important problems (as for example Cluster Editing, see Chapter 4), it has been observed that, in many real-world instances, the standard parameter is not really small. This motivates the investigation of parameterizations different from the solution size in order to extend the range of solvable instances

One approach is to consider "refined" parameters. Here, the basic idea is to consider parameters that are typically smaller than the standard parameter. More specifically, we say that a parameter is a *refined parameter* if it is bounded from above by the standard parameter. For example, in Chapter 4, we investigate the parameterized complexity of Cluster Editing. Cluster Editing is the problem of transforming a graph by a minimum number of edge modifications into a cluster graph. Herein, a cluster graph is a disjoint union of cliques. The standard parameter in case of Cluster Editing is the number of required edge deletions and insertions. A refined parameter is the "cluster vertex deletion number," denoting the minimum number of vertex deletions whose removal leaves a cluster graph: the deletion of one arbitrarily chosen endpoint of every deleted or inserted edge of a Cluster Editing solution also leads to a cluster graph. Moreover, it is easy to construct examples in which the refined parameter is significantly smaller than the solution size.

A second example is the Two-Layer Planarization problem [155]. Here, the task is to transform a graph into a forest of caterpillar trees[2] by a minimum number of edge deletions. Clearly, this requires to break all cycles in the graph (in order to obtain a forest). A set of edges whose deletion leaves an acyclic graph is called a feedback edge set. Thus, in case of Two-Layer Planarization the feedback edge set number is a refined parameter of the standard parameter "solutions size". A linear-size problem kernel for Two-Layer Planarization parameterized by the feedback edge set number has recently been presented [155].

As a third example consider the Vertex Cover problem: Given an undirected graph and an integer $k \geq 0$, compute a vertex cover of size at most k, that is, a set of at most k vertices covering all edges. That is, deleting the vertices of a vertex cover yields a graph that is the union of isolated vertices. This requires to break all cycles by deleting vertices. Thus, the "feedback vertex set number" (denoting the minimum number of vertex deletions needed to destroy all cycles of a graph) is a refined parameter for Vertex Cover. Very recently, Jansen and Bodlaender [113] devised a cubic-vertex kernel for Vertex Cover parameterized by the feedback vertex set number. They also used the term "refined parameter".

[2] A caterpillar tree is a tree where each internal vertex has at most two internal vertices as neighbors.

Finally, we mention that the notion of refined parameter also makes sense with respect to parameters other than the standard parameter "solution size". For example, the degeneracy of a graph can be seen as a refined parameter compared to the treewidth of a graph [7]. Moreover, the requirement for a refined parameter to be smaller for every instance is very strict. Also parameters which are typically smaller than a standard parameter for a large range of instances are of both theoretical and practical interest.

A further generic approach for parameter identification, namely "deconstructing intractability", is systematically performed in Chapter 8. Here, the basic idea is to analyze known NP-hardness (or W[t]-hardness) proofs to find meaningful parameters. For example, if a known NP-hardness proof requires that for some parameter the parameter values are unbounded, then we directly arrive at the question whether the problem is fixed-parameter tractable with respect to this parameter. Parameterizations with such parameters might lead to fixed-parameter tractability results. For more details and a concrete example concerning the "deconstructing intractability" approach, we refer to Chapter 8.

2.4 Kernelization

A core tool in the development of fixed-parameter algorithms is *kernelization*. Roughly speaking, kernelization is *polynomial-time* preprocessing by *data reduction* with provable performance guarantee. Formally, kernelization is defined as follows.

Definition 2.3. Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. A *reduction to a problem kernel* or *kernelization* for L is a polynomial-time executable transformation $f : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ each of the following statements is true:

- $(x', k') := f((x, k))$ is a yes-instance if and only if (x, k) is a yes-instance,
- $k' \leq k$, and
- $|x'| \leq g(k)$ for a computable function $g : \mathbb{N} \to \mathbb{N}$.

The reduced instance (x', k') is called a *problem kernel*. Its size is $g(k)$. If there is a kernelization for L, then we say that L *admits a problem kernel* of size $g(k)$. If $g(k)$ is a polynomial one speaks of a *polynomial (size) problem kernel*.

In summary, a kernelization yields an equivalent instance whose size can provably be bounded from above by a function only depending on the parameter. As a consequence, kernelization is useful as polynomial-time preprocessing prior to any solving strategy be it exact, approximative, or heuristic. Thus, the relevance of kernelization is not restricted to the field of parameterized algorithmics. Indeed, kernelization is one of the most active research areas in parameterized algorithmics and is considered as one of the theoretically and practically most interesting methods of parameterized algorithmics [68, 97, 110, 29].

It is folklore in parameterized algorithmics that a problem admits a problem kernel if and only if it is fixed-parameter tractable [37]. However, devising small problem kernels (in particular problem kernels of polynomial size) might be a highly nontrivial task. This should also be seen in the light of recent breakthrough results on methods to

prove the "non-existence" of polynomial-size kernels; Bodlaender et al. [30] and Fortnow and Santhanam [78] developed a framework to show that a problem does not admit a polynomial-size problem kernel unless an unexpected complexity-theoretic collapse takes place. Based on this framework, several non-existence results of polynomial-size problem kernels have been established (see for example [32, 61, 124]).

Finally, we introduce some notation used for the presentation of our problem kernels. A kernelization is usually achieved by the application of several *data reduction rules*. A data reduction rule is a polynomial-time executable function that replaces an instance (x, k) with an instance (x', k'). A data reduction rule is called *correct* if the new instance (x', k') after an application of this rule is a yes-instance if and only if the original instance (x, k) is a yes-instance. An instance is called *reduced* with respect to a set of data reduction rules if a further application of any of the reduction rules does not modify the instance. In this case, we also say that the data reduction rules have been applied *exhaustively*.

2.5 Depth-Bounded Search Trees

Depth-bounded search trees are a fundamental and well-established algorithm design technique in parameterized algorithmics [137, Chapter 8]. A search tree based algorithm works in a recursive manner by creating several subinstances and calling itself for each of the created subinstances. To obtain fixed-parameter algorithms it is decisive that the parameter value is decreased for each created subinstance and hence the total size of the "recursion tree" can be bounded from above by a function of the parameter.

We use the concept of branching rules for the presentation of our search tree algorithms. Given an instance (G, k), a *branching rule* creates $\ell \geq 2$ subinstances $(G_1, k_1), \ldots, (G_\ell, k_\ell)$. A branching rule is *correct* if (G, k) is a yes-instance if and only if (G_i, k_i) is a yes-instance for some $1 \leq i \leq \ell$. Branching rules lead to a search algorithm by solving each of the created subinstances recursively, terminating the recursion when $k \leq 0$ or none of the branching rules applies. For a branching rule creating $\ell \geq 2$ subinstances, the *branching vector* is the ℓ-tuple describing how the parameter is decreased in each subinstance. That is, for a branching rule creating $\ell \geq 2$ subinstances $(G_1, k_1), \ldots, (G_\ell, k_\ell)$, the branching vector is $(k - k_1, \ldots, k - k_\ell)$. A branching vector describes the recurrence $T_k = T_{k_1} + \ldots + T_{k_\ell}$ for the asymptotic size of the search tree. Using standard branching analysis tools, a *branching number* can be computed from the branching vector [137, Chapter 8]. The branching number describes the base of the (exponential) search tree size. For example, if the branching number of a given branching rule is 3.68, then the above recursion leads to a search tree size of $O(3.68^k)$. If several branching rules are used, then the size of the overall search tree is determined by the largest branching number over all branching rules.

Combining search tree based algorithms with kernelization algorithms is one of the most successful approaches for efficient fixed-parameter algorithms. Suppose that we have a search tree algorithm with running time $O(\xi^k \cdot q(n))$ and a kernelization with running time $p(n)$ yielding a problem kernel of size $s(k)$, where q and p are polynomials. If a given instance is first reduced by applying the kernelization and then the search tree algorithm is applied, one obtains an algorithm with "additive fpt running time" $O(p(n)+q(s(k))\xi^k)$. Furthermore, search tree algorithms invite to apply

the kernelization at each search tree node; an approach known as *interleaving* [139]. Indeed, Niedermeier and Rossmanith [139] have shown that by interleaving search trees with kernelizations one can improve the worst-case running time of a search tree algorithm to $O(p(n) + \xi^k)$ if s is a polynomial.

2.6 Basic Graph Notation

We use the following notation for graphs throughout this thesis. For a comprehensive introduction into graph theory we refer to [60, 117, 157].

An *undirected graph* G is a pair (V, E), where V is a finite set of *vertices* and E is a finite set of *edges*. Herein, an edge is defined as unordered pair of vertices.[3] For an undirected graph G, we also use $V(G)$ and $E(G)$ to denote its vertex and edge sets, respectively. We refer to the cardinality of $V(G)$ as the *order* of G. Two vertices $v \in V$ and $w \in V$ are called *adjacent* if $\{v, w\} \in E$. Moreover, an edge $e \in E$ is *incident* to a vertex $v \in V$ if $v \in e$. For a set X of vertices, $\mathcal{P}_2(X)$ denotes the set of all possible edges on X.

The *open neighborhood* $N_G(v)$ of a vertex v is the set of vertices that are adjacent to v, and the *closed neighborhood* $N_G[v] := N_G(v) \cup \{v\}$. For a vertex set $S \subseteq V$, let $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$. The *closed neighborhood* of S is denoted by $N_G[S] := S \cup N_G(S)$. With $N_G^2(S) := N_G(N_G(S)) \setminus N_G[S]$ we denote the *second neighborhood* of a vertex set S. The *degree* of a vertex v, denoted by $\deg_G(v)$, is the cardinality of $N_G(v)$. If G is clear from the context, we omit the subscript G.

A *subgraph* of $G = (V, E)$ is a graph $G' := (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. We use $G[S]$ to denote the subgraph of G *induced* by $S \subseteq V$, that is, $G[S] := (S, \mathcal{P}_2(S) \cap E\})$. Moreover, let $G - v := G[V \setminus \{v\}]$ for a vertex $v \in V$ and $G - e := (V, E \setminus \{e\})$ for an edge $e = \{u, v\}$.

For a graph $G = (V, E)$ the *complement graph* of G is $\overline{G} := (V, \overline{E})$ with $\overline{E} := \mathcal{P}_2(V) \setminus E$. A *path* is a graph $P = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}\}$; the vertices v_1 and v_n are the *endpoints* of P. The *length* of a path P is given by $|E(P)|$. A *cycle* is the graph consisting of a path on at least three vertices and the edge between its endpoints. With P_n we denote the path on n vertices.

Two vertices v and w of a graph G are called *connected* if G contains a path with endpoints v and w as a subgraph. A graph is called *connected* if any two of its vertices are connected. The *connected components* of a graph are its maximal connected subgraphs.

A graph is called *cyclic* if it contains a cycle as subgraph; otherwise it is called *acyclic*. Acyclic graphs are called *forests*. An acyclic and connected graph is called a *tree*. A *rooted tree* is a tree where one vertex is marked as the root of the tree. The *depth* of a vertex v in a rooted tree is the length of the path from v to the root. The depth of a tree is the maximum depth over all vertices. The *ancestors* of a vertex v are the vertices of the path from v to the root. For two vertices u and v, the *least common ancestor* is the maximum-depth vertex that is an ancestor of u and v.

An undirected graph $G = (V, E)$ is called *bipartite* if V can be partitioned into two sets V_1 and V_2 such that for every edge $\{u, v\} \in E$ it holds that $\{u, v\} \cap V_1 \neq \emptyset$

[3] In Chapter 9, we allow self-loops, that is, edges of the form $\{v, v\}$. If not explicitly stated otherwise, we only consider simple undirected graphs without self-loops.

and $\{u, v\} \cap V_2 \neq \emptyset$. It is folklore that a graph is bipartite if and only if it does not contain an odd-length cycle. We sometimes refer to graphs containing an odd-length cycle as non-bipartite graphs. A *biclique* is a bipartite graph $(V_1 \cup V_2, E)$ with $E := \{\{v, w\} \mid v \in V_1, w \in V_2\}$. We also use the term biclique to refer to a vertex set inducing a biclique.

For an undirected graph (V, E) a *matching* denotes a subset $M \subseteq E$ such that for all $e, e' \in M$ with $e \neq e'$ it holds that $e \cap e' = \emptyset$. A *maximum matching* is a matching with maximum cardinality. Moreover, given a weight function $\omega : E \to \mathbb{N}$ a *maximum-weight matching* is a matching with maximum total edge weight. A maximum-weight matching in a bipartite graph with n vertices and m edges can be computed in $O(n(m + n \log n))$ time [80].

For an undirected graph (V, E) a set of pairwise adjacent vertices is called a *clique* and a set of pairwise nonadjacent vertices is called an *independent set*. A clique K is called a *maximal clique* if $K \cup \{v\}$ is not a clique for every $v \in V \setminus K$.

For our investigations in Part II cliques and independent sets where all vertices have an identical neighborhood are of particular interest.

Definition 2.4. A clique K is a *critical clique* if all its vertices have an identical *closed* neighborhood and K is maximal under this property.

Definition 2.5. An independent set I is a *critical independent set* if all its vertices have an identical *open* neighborhood and K is maximal under this property.

All critical independent sets of a graph can be found in linear time [108]. Given a graph $G = (V, E)$ and the collection $\mathcal{I} = \{I_1, I_2, \ldots, I_q\}$ of its critical independent sets, where $q \leq n$, the *critical independent set graph* of G is the undirected graph $(\mathcal{I}, \mathcal{E})$ with $\{I_i, I_j\} \in \mathcal{E}$ if and only if $\forall u \in I_i, v \in I_j : \{u, v\} \in E$. That is, the vertices of the critical independent set graph represent the critical independent sets and two vertices are adjacent if and only if the corresponding critical independent sets together form a biclique. The critical independent set graph is defined in analogy to the *critical clique graph* which plays a decisive role in a kernelization of Cluster Editing [89].

Part II

Fitting Biological Data with Combinatorial Structures

This part of the thesis investigates four problems arising in the context of clustering of data or the construction of phylogenetic trees. For example, a goal is to "fit" genomic data with the combinatorial structure of a phylogenetic tree or a cluster graph.

Chapter 3 provides an introduction to this part. Besides introducing the considered problems it sheds light on the relationship between them and gives an overview of our results.

Chapter 4 shows that Cluster Editing and Cluster Deletion are fixed-parameter tractable with respect to the refined parameter "cluster vertex deletion number."

Chapter 5 introduces a generalization of Cluster Editing that allows a certain amount of overlap between the clusters that can be specified by an overlap number s. We perform a basic complexity study of the corresponding problems, devise a forbidden subgraph characterization for generalized cluster graphs, and present two polynomial-size problem kernels for two of the problems.

Chapter 6 focusses on the M-Hierarchical Tree Clustering problem, where the goal is to fit dissimilarity data on pairs of elements with an ultrametric tree of depth $M + 1$. We present an $O(k^2)$-element and an $O(M \cdot k)$-element problem kernel as well as an $O(2.56^k)$-size search tree algorithm, where k denotes the cost of the solution.

Chapter 7 is concerned with the investigation of the parameterized complexity of the Minimum Flip Consensus Tree problem with respect to the parameter "solution size". We improve previously known fixed-parameter algorithms by presenting a refined search tree of size $O(3.68^k)$ and a cubic-vertex kernel.

Chapter 3

Introduction to Part II

This introductory chapter is organized as follows. In Section 3.1, we provide a brief introduction to the problems considered in this part of the thesis. Here, the focus is on comparing the different problems and the underlying models and to shed light on the relationships between the problems. A summary and comparison of the results presented in this part is given in Section 3.2. Most problems considered in this part are so-called edge modification problems. The basics concerning edge modification problems are summarized in Section 3.3. Finally, in Section 3.4 we give a universal version of several data reduction rules used in several kernelizations presented in this part.

3.1 The Considered Problems

In the following, we introduce the problems that are considered in this part of the thesis, compare the different underlying models, and shed light on the relationships between the problems.

All problems in this part adhere to the following general setting. The input consists of objects (also called "data items" or "taxa" in phylogenetics[1]) and some information about the (dis)similarity or relationship between the objects. This information is for example provided by a matrix containing the pairwise similarities or by a graph where the vertices represent the objects and two vertices are considered similar if they are adjacent (a so-called similarity graph). Moreover, each problem comes with a model that can be used to represent the relationships between the objects in an easy to interpret fashion. For example in graph-based data clustering, clusterings are sometimes represented by disjoint unions of cliques, so-called *cluster graphs*, implying a partition of the objects into disjoint subsets. As a second example, in phylogenetics rooted trees are widely used to represent the evolutionary relationships between the taxa. Here, the leaves of the tree are one-to-one labeled with the taxa and the distance between two taxa is proportional to the length of the path between the two taxa in the tree. That is, a model is given by a special class of combinatorial structures (like

[1] In phylogenetics, a group of species is called a taxon.

cluster graphs or trees) and the task is to find the structure of this class that best represents the relationships given in the input, where the quality is measured by some problem-specific distance function. Thus, in summary, for all problems the task is to fit combinatorial structures with given similarity data in the best possible way.

Next, we introduce in more detail the problems considered in this part. For formal definitions, we refer to the respective chapters. The order of presentation of the problems is based on the model used to represent the relationships between the objects in the solution. Roughly speaking, we start with the simplest model, namely that of a cluster graph (demanding the partition of the objects into disjoint subsets). In this simple model, we have no overlap between the clusters. Then, we allow some degree of overlap between the clusters, resulting in more complex cluster models. Finally, we consider problems where rooted trees (that is, hierarchical structures) are used for representing the relationships between the objects in the solution, allowing to display subclusters of the clusters.

The standard task in clustering is to group together a set of objects into several clusters such that the objects inside a cluster are highly similar to each other, whereas objects not occurring in a common cluster have low or no similarity. There are numerous approaches to clustering and "there is no clustering algorithm that can be universally used to solve all problems" [162]. One prominent line of attack is to use methods based on graph theory [148, 150]. Graph-based data clustering is an important tool in exploratory data analysis [148, 150, 162]. The applications range from bioinformatics [15, 152] over document clustering and agnostic learning [12] to image processing [161]. The formulation as a graph-theoretic problem relies on the notion of a *similarity graph* where vertices represent data items and an edge between two vertices expresses high similarity between the corresponding data items. Then, the computational task is to group the vertices into clusters, where a *cluster* is nothing but a dense subgraph (typically, a clique) such that there are only few edges between the clusters. Following Ben-Dor et al. [15], Shamir et al. [150] initiated a study of graph-based data clustering in terms of *edge modification problems*. Here, the task is to modify (delete or insert) as few edges of an input graph as possible to obtain a *cluster graph*, that is, a *vertex-disjoint* union of cliques. The corresponding problem is referred to as Cluster Editing (see Definition 4.1). Numerous recent publications build on this concept of cluster graphs [24, 25, 28, 46, 53, 54, 57, 73, 83, 89, 145]. Indeed, the NP-hard Cluster Editing problem is among the best-studied parameterized problems. Cluster Editing is considered in Chapter 4. The model in case of Cluster Editing is that of a *partition* of the set of objects. That is, the clusters in a clustering obtained by solving Cluster Editing are disjoint. In this work, we refer to such clusterings as *non-overlapping*.

To uncover the *overlapping* community structure of complex networks in nature and society [141], the concept of a partition of the set of objects (as one has for cluster graphs) fails to model that clusters may overlap. Consequently, the concept of cluster graphs has been criticized explicitly for this lack of overlaps [57]. In Chapter 5, we introduce a graph-theoretic relaxation of the concept of cluster graphs by allowing, to a certain extent, overlaps between the clusters (which are cliques). We distinguish between "vertex-overlaps" and "edge-overlaps" and provide a first thorough study of the corresponding cluster graph modification problems. The two core concepts we introduce in Chapter 5 are *s-vertex-overlap* and *s-edge-overlap*, where in the first case we demand that every vertex in the cluster graph is contained in at most s maximal

cliques and in the second case we demand that every edge is contained in at most s maximal cliques (see Definition 5.1). By definition, 1-vertex-overlap means that the cluster graph is a vertex-disjoint union of cliques (that is, there is no overlap of the clusters and, thus, the corresponding graph modification problem is Cluster Editing). Based on these definitions, we study a number of edge modification problems (addition, deletion, editing) in terms of the two overlap concepts, generalizing and extending previous work that mainly focussed on non-overlapping clusters.

The problems considered in Chapters 4 and 5 are clustering problems relying on graph classes as models for representing the clusterings. In (hierarchical) clustering and phylogeny, often trees are used as an easy to interpret model for representing the relationship between objects or taxa. For the problems considered in Chapters 6 and 7 of this part, the task is to construct *rooted* trees from the given (dis)similarity data.

In Chapter 6, we investigate M-Hierarchical Tree Clustering. Roughly speaking, given dissimilarity data on pairs of objects, the task is to fit a rooted tree to this data, where the tree gives a hierarchical representation of the data. Hierarchical representations of data play an important role in biology, the social sciences, and statistics [5, 55, 106, 125]. The basic idea behind hierarchical clustering is to obtain a recursive partitioning of the input data in a tree-like fashion such that the leaves one-to-one represent the single items and all inner points represent clusters of various granularity degrees. Let X be the input set of elements to be clustered. The dissimilarity of the elements is expressed by a symmetric function $D : X \times X \to \{0, \ldots, M+1\}$, briefly called *distance function*. Herein, the constant $M \in \mathbb{N}$ specifies the depth of the clustering tree to be computed. In Chapter 6, we focus on the case to find a closest *ultrametric tree* that fits the given data. An ultrametric tree is a rooted tree where all leaves are equally distant from the root and the leaves are bijectively labeled with the elements of X. Let $U(i, j)$ denote half the length of the path between leaves corresponding to the elements i and j. It is not hard to see that the distances $U : X \times X \to \{1, 2, \ldots\}$ fulfill the strong triangle inequality, namely,

$$U(i, j) \leq \max\{U(i, l), U(j, l)\}$$

for all $i, j, l \in X$. Indeed, it is well-known that a distance function can be represented by an ultrametric tree if and only if it fulfills the strong triangle inequality [5]. Thus, distance functions fulfilling the strong triangle inequality are called ultrametrics, and constitute an equivalent representation of ultrametric trees. The M-Hierarchical Tree Clustering problem considered in Chapter 6 can be formulated as follows. Given a set X of elements, a distance function $D : X \times X \to \{0, \ldots, M+1\}$, and $k \geq 0$, is there a distance function $D' : X \times X \to \{0, \ldots, M+1\}$ such that D' is an ultrametric and $||D - D'||_1 \leq k$? Herein, $||D - D'||_1 := \sum_{\{i,j\} \subseteq X} |D'(i, j) - D(i, j)|$ (also see Definition 6.2). In other words, given any distance function D, the goal is to modify D as little as possible to obtain an ultrametric D'. This problem is closely related to the reconstruction of phylogenetic trees [67, 5]. Moreover, 1-Hierarchical Tree Clustering is the same as Cluster Editing. Here, the crucial observation is that a cluster graph can be represented by an ultrametric tree of depth two, see Figure 3.1. This will be discussed in more detail in Chapter 6.

The Minimum-Flip Consensus Tree problem, considered in Chapter 7, arises in an approach to aggregate the information of several trees in a consensus tree [45]. Given rooted phylogenetic trees $T_1, \ldots, T_\ell$ (all on the same set of taxa), a consensus

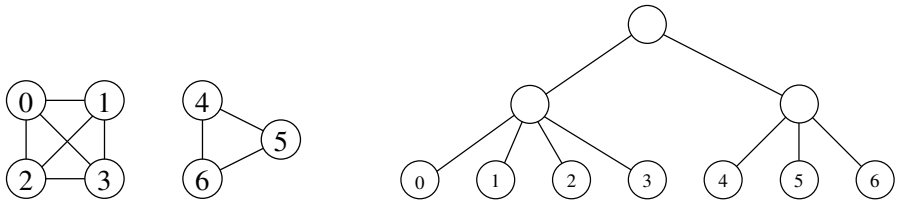


Figure 3.1: A cluster graph and a corresponding ultrametric tree of depth two.

tree is constructed in three phases. In a first phase, the information of all input trees is represented in a bipartite graph $G = (V_c, V_t)$, where V_t contains a vertex for each taxon and V_c contains a vertex for each inner vertex of each tree. If the input trees do not contain any conflicting information, then G is *M-free*, meaning that G does not contain an induced path on five vertices whose degree-one vertices belong to V_t. M-free graphs "admit a perfect phylogeny", meaning that one can construct a rooted phylogenetic tree from an M-free graph G. If, however, the input trees contain contradicting information, then G contains induced M-graphs. Hence, in a second "error correction" phase, the task is to destroy all M-graphs by a minimum number of edge modifications. This is the NP-hard combinatorial problem considered in Chapter 7. Given a bipartite graph $G = (V_c, V_t, E)$ and an integer $k \geq 0$, the task is to decide whether G can be transformed by up to k edge modifications into an M-free graph, that is, a graph without an induced M-graph (see Definition 7.1). In a third phase, the consensus tree is then inferred from the M-free graph obtained by the error correction phase.

From a graph-theoretic point of view, all problems considered in this part except for M-Hierarchical Tree Clustering are so-called edge modification problems. The task in the case of *edge modification problems* is to modify the edge set of a given graph as little as possible to obtain a graph fulfilling a desired graph property. For example, in Cluster Editing (Chapter 4) the desired graph property is to be a disjoint union of cliques, that is, each vertex is contained in exactly one maximal clique. To allow (Chapter 5) overlapping clusterings the desired graph property is that each vertex (respective edge) is contained in at most s maximal cliques (for some fixed positive integer s). In case of Minimum Flip Consensus Tree the desired graph property is to be M-free (Chapter 7).

Next, we provide an overview of our results. Then, in Section 3.3, we give a formal introduction to edge modification problems and graph properties.

3.2 Summary of Results

While the parameterized complexity of some of the considered problems has extensively been investigated in literature, for others there are hardly any known results. For example, Cluster Editing (see Chapter 4) is one of the best studied problems in parameterized algorithmics, whereas we newly introduced the models for graph-based data clustering with overlaps considered in Chapter 5. This is also reflected in our contributions to the respective problems.

All problems in this part of the thesis are related to Cluster Editing which is

studied in Chapter 4. The parameterized complexity of Cluster Editing has intensively been studied [24, 28, 53, 46, 73, 83, 89, 145]. Moreover, experiments with fixed-parameter algorithms for Cluster Editing have been performed [25, 57]. Several algorithmic improvements have led to impressive theoretical results, for example, a problem kernel consisting of at most $2k$ vertices [46] and a search-tree algorithm with running time $O(1.76^k + n^3)$ [28]. So far the proposed fixed-parameter algorithms for Cluster Editing almost exclusively examine the parameter solution size k. However, it has been observed that the parameter k is often not really small for real-world instances [25]. Still, the fixed-parameter algorithms can solve many of these instances [25]. This raises the question whether there are "hidden parameters" that are implicitly exploited by these algorithms. In the spirit of multivariate algorithmics (see Section 2.2.1 and Niedermeier [138]), Chapter 4 aims at identifying promising new parameterizations for Cluster Editing that help to separate easy from hard instances. Thus, our main contribution in Chapter 4 is the investigation of parameterizations of Cluster Editing different from the solution size k. In this thesis the focus is on the parameter "size of a minimum cluster vertex deletion set of G", also called the "cluster vertex deletion number of G". A *cluster vertex deletion set* is a minimum-cardinality set of vertices whose removal results in a cluster graph. The cluster vertex deletion number is bounded from above by the minimum number of edge modifications needed to transform a graph into a cluster graph and, thus, is a typically much smaller parameter than k. Our technical main results in Chapter 4 are that Cluster Editing and its edge deletion version, Cluster Deletion, are fixed-parameter tractable with respect to the cluster vertex deletion number. In summary, in Chapter 4 we initiate the study of Cluster Editing with respect to "refined parameters" (where a refined parameter is a parameter bounded from above by a standard parameter, see Section 2.3). Although the presented algorithms have huge combinatorial explosions, the hope is that this investigation initiates the development of new algorithms to extend the range of instances that can be solved in practice.

In Chapter 5, we introduce overlap cluster graph modification problems where, other than in most previous work, the clusters of the target graph may overlap. More precisely, the studied graph problems ask for a minimum number of edge modifications such that the resulting graph has the *s-vertex-overlap* property or the *s-edge-overlap* property, respectively. Our results are as follows. First, we provide a complexity dichotomy (polynomial-time solvable versus NP-hard) for the underlying edge modification problems, see Table 5.1. Second, we develop forbidden subgraph characterizations of "cluster graphs with overlaps". More specifically, we show that, for every fixed value of s, the class of graphs having the *s-vertex-overlap* or the *s-edge-overlap* property can be characterized by a finite set of forbidden induced subgraphs. Third, we study the parameterized complexity in terms of the number of allowed edge modifications. In case of constant s-values we achieve fixed-parameter tractability, whereas in case of unbounded s-values the respective problems are shown to be W[1]-hard. Finally, we present polynomial-time kernelization algorithms for the two most basic NP-hard clustering problems with nontrivial overlap. Altogether, in Chapter 5 we introduce new problems with applications in graph-based data clustering, generalizing Cluster Editing by allowing overlaps and perform basic complexity studies of the corresponding problems.

In Chapter 6, we initiate the study of the parameterized complexity of the NP-hard M-Hierarchical Tree Clustering problem, that is, the problem to find a closest

ultrametric tree for given dissimilarity data on pairs. This is a central problem in the area of hierarchical clustering, where so far only polynomial-time approximation algorithms were known. In contrast, we develop efficient kernelization algorithms and a simple search tree algorithm. More precisely, we develop a search tree of size $O(2.562^k)$ and two problem kernels; an $O(k^2)$-element kernel (that is, the kernel size is independent of M) and an $O(kM)$-element kernel (here, the kernel size depends on M). Recall that 1-Hierarchical Tree Clustering is exactly the same as Cluster Editing. The presented kernelization algorithms generalize kernelizations for Cluster Editing [83, 89]. Moreover, we perform experiments on synthetic and real-world biological data. On the biological data, we also compare our algorithm with an approximation algorithm due to Ailon and Charikar [5] and an integer linear program for this problem.

The Minimum Flip Consensus Tree problem considered in Chapter 7 arises in computational phylogenetics in the context of consensus and super tree construction. Chen et al. [45] showed that Minimum-Flip Consensus Tree is NP-complete and presented a fixed-parameter algorithm based on an $O(6^k)$-size search tree. Subsequently, Böcker et al. [27] presented a refined branching leading to a $O(4.42^k)$-size search tree. We improve these results by an $O(3.68^k)$-size search tree. The improvement is based on the identification of polynomial-time solvable special cases. Moreover, we complement these results by polynomial-time executable data reduction rules yielding a problem kernel with $O(k^3)$ vertices. Altogether, for Minimum Flip Consensus Tree our contribution is the improvement of the parameterized algorithms with respect to the standard parameter k denoting the number of allowed edge modifications, leading to the currently fastest fixed-parameter algorithm for Minimum Flip Consensus Tree with running time $O(3.68^k + |V_c|^2 \cdot |V_t| \cdot |E|)$.

Finally, in Section 3.4 we present a "universal" data reduction rule that unifies some data reduction rules used for the kernelizations in Part II. This rule applies for all edge modification problems where the desired graph properties can be characterized by forbidden induced subgraphs.

3.3 Edge Modification Problems

In the case of edge modification problems the task is to modify the edge set of a given graph as little as possible to obtain a graph fulfilling a desired graph property such as, for example, being a cluster graph. Herein, an edge modification is either the deletion or the insertion of an edge. Edge modification problems naturally arise in the interpretation of experimental data sets, when graph models are used to represent the data. For example, in graph-based data clustering the similarity between the objects to be clustered is represented by a so-called similarity graph. One of the most intensively studied problems in parameterized algorithmics, Cluster Editing, asks whether a given similarity graph can be transformed into a *cluster graph* by applying at most k edge modifications in order to reveal the assumed cluster structure hidden in the input data.

In addition to finding some hidden combinatorial structure, edge modification problems also play an important role for error correction or conflict resolution, respectively. For instance, in Minimum Flip Consensus Tree (see Chapter 7) the information of several input trees is combined in a single bipartite graph and the task is to resolve all conflicts in this graph such that from the resulting graph a consensus tree can be

constructed.

The guiding principle in case of edge modification problems is the idea that fewer edge modifications mean that we introduce fewer "errors" into our final solution, and, hence, the solution requiring a minimum number of edge modifications is preferred. This is in accordance with the natural hypothesis that the less one perturbs the input graph the more robust and plausible the achieved solution is. This is known as maximum parsimony principle (see Böcker et al. [24] for making this point in terms of Cluster Editing).

There is a long history of research dealing with edge modification problems. A lot of work has been put into classifying Π-Edge Modification problems with respect to their classical complexity [34, 135, 163]. Recently, parameterized algorithmics —in particular kernelizations—for Π-Edge Modification problems have attracted special attention. For instance, there is a series of papers studying the kernelizability of Cluster Editing and some of its variations [40, 46, 71, 73, 83, 89, 93, 96, 145]. Also see [59, 16, 17, 33, 88, 99, 121, 155, 156] for kernelization results for edge modification and related problems. All these works present polynomial-size problem kernels. However, Kratsch and Wahlström [124] showed that there is a graph H on seven vertices such that H-free Editing[2] does not admit a polynomial-size problem kernel unless an unexpected complexity-theoretic collapse takes place. This contrasts the case of vertex deletion where for every fixed forbidden subgraph H the problem to destroy all occurrences of H by deleting at most k vertices admits a polynomial-size problem kernel (this follows from the fact that for each fixed H with d vertices H-free vertex deletion can be reduced to d-Hitting Set which admits a kernel of order $k^{O(d)}$ [1, 2, 140]). Hence, for every forbidden induced subgraph it is a challenging task to prove the (non)existence of a polynomial-size problem kernel for the corresponding edge modification problem.

Next, we introduce basic notations regarding edge modification problems. Then, in Section 3.4, we present data reduction rules unifying several data reduction rules used for the kernelizations presented in Chapters 5 and 7. Moreover, in Section 3.4, we establish a structural observation that is central for some of our kernelizations.

3.3.1 Basic Notation for Edge Modification Problems

Formally, a *graph property* is defined as a nonempty proper subset Π of the set of graphs. We say that a graph G has property Π (abbreviated by $G \in \Pi$) if G is isomorphic to a member of Π. For a desired graph property Π, Π-Editing is defined as follows.

Definition 3.1 (Π-Editing, Π-Deletion, Π-Addition)**.**
Input: An undirected graph G and a nonnegative integer k.
Question: Can G be modified by up to k edge deletions and insertions into a graph with property Π?
If only edge deletions or edge insertions are allowed, then the corresponding problems are called Π-Deletion and Π-Addition, respectively.

All graph properties considered in this work are so-called hereditary graph properties. A hereditary graph property Π is a graph property closed under vertex deletion.

[2]For a fixed graph H, H-free Editing is the problem to destroy all induced occurrences of H by a minimum number of edge modifications.

Formally, the definition reads as follows.

Definition 3.2. A graph property Π is *hereditary* if the property Π holds for every induced subgraph of G whenever it holds for G.

Hereditary graph properties can be characterized by a (possibly infinite) set of forbidden induced subgraphs [86]. We use the following notation. Let $\mathcal{F}$ denote a set of graphs. We say that a graph G is *$\mathcal{F}$-free* if G contains no induced subgraph isomorphic to a member of $\mathcal{F}$. For each hereditary graph property there exists a possibly infinite set of graphs $\mathcal{F}_\Pi$ such that, for each graph G, G has property Π if and only if G is $\mathcal{F}_\Pi$-free. Furthermore, one can assume that all forbidden subgraphs in $\mathcal{F}_\Pi$ characterizing a hereditary graph property Π are minimal in the sense that, for each $F \in \mathcal{F}_\Pi$, all proper subgraph of F have property Π. That is, a graph F is a *minimal forbidden induced subgraph* for a hereditary graph property Π if $F \notin \Pi$ but every induced proper subgraph of F has property Π.

For a formal description of the modification of the edge set of a graph, we use the following notation. For two sets E' and E'' let $E' \Delta E'' := (E' \setminus E'') \cup (E'' \setminus E')$ denote the symmetric difference of E' and E''. Recall that for a set X of vertices $\mathcal{P}_2(X)$ denotes the set of all possible edges on X (see Section 2.6). Furthermore, for a graph $G = (V, E)$ and a set $S \subseteq \mathcal{P}_2(V)$, let $G \Delta S := (V, E \Delta S)$ denote the graph that results from modifying G according to S. With this notation, for a desired graph property Π, the definition of Π-Editing reads as follows.

> Given a graph $G = (V, E)$ and a nonnegative integer k, is there a set $S \subseteq \mathcal{P}_2(V)$ with $|S| \leq k$ such that $G \Delta S$ has property Π ?

This set S is called a solution for Π-Editing for (G, k). The elements of S are called *edge modifications*. Furthermore, we refer to the edge modifications in $S \cap E$ by *edge deletions* and to the edge modifications in $S \setminus E$ as *edge insertions* or *edge additions*. We say that an edge modification $e \in S$ *involves* a vertex v if $v \in e$. Finally, a vertex v is called *affected* by S if S contains an edge modification involving v.

In several of our proofs we compare two graphs G_1 and G_2 (with property Π) with respect to the number of edge modification required to transform the input graph G to G_1 or G_2, respectively. The number of edge modifications to transform a graph G into a graph G' is called edit distance. More formally, the *edit distance* between two graphs G' and G'' on the same vertex set is $|E(G') \Delta E(G'')|$. Note that for $S := E(G') \Delta E(G'')$ it holds that $G' = G'' \Delta S$ and $G'' = G' \Delta S$. Given three graphs G, G_1, and G_2, we say that G_1 is *closer* to G than G_2 if the edit distance between G and G_1 is strictly smaller than the edit distance between G and G_2.

3.4 Universal Data Reduction Rules and Structural Observations for Edge Modification Problems

The problems considered in Chapters 4, 5, and 7 belong to the class of edge modification problems (see Section 3.3). In this section, the goal is to unify some of the data reduction rules that are used for all kernelizations in Chapters 5 and 7. To this end, we describe polynomial-time data reduction rules for parameterized edge modification problems that apply to hereditary graph properties and which are generalizations

of rules that were developed for Cluster Editing and Bicluster Editing [145]. These rules are based on the modular decomposition of a graph and yield problem kernels with $O(k^2)$ vertices for Cluster Editing and Bicluster Editing [145]. More specifically, we present two data reduction rules whose applicability depends only on the size of the largest critical independent set or largest critical clique of the forbidden induced subgraphs characterizing the desired graph property. Moreover, we present a structural observation concerning the solutions of edge modification problems for graph properties that can be described by forbidden subgraphs whose largest critical independent sets or critical cliques have size one. This unifies two structural observations used for the kernelizations in Section 5.5.2 and Section 7.4.

We use the data reduction rules introduced in this section (in combination with problem-specific data reduction rules) for obtaining polynomial-size problem kernels for two problems arising in the context graph-based data clustering with overlaps (see Chapter 5) and for the Minimum-Flip Consensus Tree problem (see Chapter 7), but we believe that they can be useful for other edge modification problems as well.

The data reduction rules introduced in this section are based on the concepts of critical independent sets and critical cliques. Recall that a critical independent set (critical clique) is an independent set (a clique) that is maximal with respect to the property that all its vertices have an identical open neighborhood (closed neighborhood) (see Definitions 2.5 and 2.4).

The basic idea of the data reduction rules is to show that, for some graph properties, the vertices of large critical independent sets or critical cliques are not affected by any optimal edge modification set. Therefore, large critical independent sets and critical cliques can be shrunken. First, we describe the corresponding graph properties.

Definition 3.3. Let Π be a hereditary graph property and let r denote a positive integer. We call Π *r-critical independent set preserving* (*r-cisp*) whenever for each forbidden induced subgraph F of Π the largest critical independent set of F has size at most r. Analogously, we call Π *r-critical clique preserving* (*r-clip*) whenever for each forbidden induced subgraph F of Π the largest critical clique of F has size at most r.

For example, the only forbidden induced subgraph of *cluster graphs* is a path on three vertices (a so-called P_3) [150]. Obviously, the largest critical independent set of a P_3 has size two and the largest critical clique of a P_3 has size one. Thus, the property of being a cluster graph is a 2-cisp and 1-clip graph property.

Next, we present the two data reduction rules. Informally speaking, the data reduction rules remove vertices from a critical independent set (or critical clique) as long as its size exceeds $k + r$. Recall that k denotes the number of allowed edge modifications.

Reduction Rule 3.1. Let Π be an r-cisp graph property and let (G, k) denote an instance of Π-Editing, Π-Deletion, or Π-Addition. If G contains a critical independent set I with $|I| > k + r$, then delete $|I| - (k + r)$ arbitrary vertices from I.

Reduction Rule 3.2. Let Π be an r-clip graph property and let (G, k) denote an instance of Π-Editing, Π-Deletion, or Π-Addition. If G contains a critical clique K with $|K| > k + r$, then delete $|K| - (k + r)$ arbitrary vertices from K.

In the following, we prove the correctness of Reduction Rules 3.1 and 3.2. To this end, we first show that r-cisp and r-clip graph properties are closed under a certain vertex-addition operation.

Lemma 3.1. *Let $G = (V, E)$ be a graph fulfilling an r-cisp (r-clip) graph property Π and let I denote a critical independent set (critical clique) of G with $|I| \geq r$. Let G' be the graph that results by adding to G a new vertex $x \notin V$ and making it adjacent to each vertex in $N_G(I)$ ($N_G[I]$). Then, G' also fulfills Π.*

Proof. First, we prove the lemma for r-cisp graph properties. To this end, we show by contradiction that G' does not contain a forbidden induced subgraph. Note that $I \cup \{x\}$ forms a critical independent set in G'.

Assume towards a contradiction that there is a vertex subset $X \subseteq V(G')$ inducing a forbidden subgraph in G'. Since G has property Π and Π is hereditary, it follows that $x \in X$. Moreover, since $I \cup \{x\}$ forms a critical independent set in G' and since the largest critical independent set in $G'[X]$ contains at most r vertices, $I \setminus X \neq \emptyset$. Let $v \in I \setminus X$ be arbitrarily chosen. Thus, since x and v have an identical open neighborhood, $(X \setminus \{x\}) \cup \{v\}$ induces a forbidden subgraph in G' not containing x; a contradiction to the fact that G has property Π.

For r-clip graph properties the proof follows by replacing "critical independent set" by "critical clique" and "open neighborhood" by "closed neighborhood". □

The crucial observation to show the correctness of Reduction Rules 3.1 and 3.2 is the fact that critical independent sets and critical cliques of size at least $k + r$ are not affected by optimal solutions for Π-Editing, Π-Deletion, or Π-Addition.

Lemma 3.2. *Let Π be an r-cisp (r-clip) graph property and let (G, k) denote an instance of Π-Editing, Π-Deletion, or Π-Addition. For every critical independent set (critical clique) I with at least $k + r$ vertices, every optimal solution of size at most k for Π-Editing (Π-Deletion, Π-Addition) does not affect any vertex of I.*

Proof. First, we prove the lemma for Π-Editing, where Π is a r-cisp graph property. Assume towards a contradiction that there is an optimal solution S of size at most k that contains an edge modification involving a vertex from I. Let $G_S := G \,\Delta\, S$ denote the graph that results by applying S to G. Moreover, let $S_{out} := \{\{x, y\} \in S \mid x \in I, y \in V \setminus I\}$ denote the set of edge modifications of S involving exactly one vertex of I. Furthermore, let I_{out} denote the vertices of I involved in any edge modification from S_{out}. Let $I' := I \setminus I_{out}$. Clearly, $|I_{out}| \leq k$.

In the following, we use the observation that there is a set $A \subseteq I'$ of at least r vertices such that A forms an independent set in G_S and $N_{G_S}(x) \setminus I = N_G(x)$ for all $x \in A$.

To show the existence of such a set A observe the following. First, by definition of I', there are no edge modifications between vertices in I' and $V \setminus I$ which implies that $N_G(x) = N_{G_S}(x) \setminus I$ for all $x \in I'$. Second, $|I'| \geq k + r - |I_{out}|$ since $|I| \geq k + r$. Third, because each vertex in I_{out} is involved in an edge modification of S_{out}, there are at most $k - |I_{out}|$ edges in $G_S[I']$. Thus, we can cover all edges of $G_S[I']$ with at most $k - |I_{out}|$ vertices (by arbitrarily choosing for each edge in $G_S[I']$ one endpoint). Fix one such cover C arbitrarily. The vertices in $A := I' \setminus C$ form an independent set in $G_S[I']$ and, hence, in G_S. Finally, note that $|A| \geq |I'| - |C| \geq k + r - |I_{out}| - |C| \geq r$. Thus, A fulfills all above requirements.

Let $I'' := I \setminus A$. Since G_S has property Π, so does $G_S - I''$. Observe that A forms a critical independent set in $G_S - I''$ such that $N_G(A) = N_{G_S - I''}(A)$. Thus, by Lemma 3.1, one obtains a graph G' with property Π by adding the vertices in I'' step by step to $G_S - I''$ making each vertex adjacent to $N_G(A)$. Finally, we show that G' is closer to G than G_S. By assumption, S contains edge modifications involving vertices from I. Each vertex in I, however, has the same neighborhoods in G' and G. Thus, G' is closer to G than G_s: a contradiction to the optimality of S.

This concludes the proof for Π-Editing and r-cisp graph properties. It is straightforward to verify that every step of the proof holds for Π-Deletion and Π-Addition when Π is r-cisp.

For the proof for r-clip graph properties observe the following. A critical clique in a graph G clearly is a critical independent set in the complement graph $\overline{G}$ of G, and vice versa. Moreover, let $\mathcal{F}$ denote the set of forbidden induced subgraphs of a r-cisp graph property Π. Let $\overline{\mathcal{F}} := \{\overline{F} \mid F \in \mathcal{F}\}$ and let $\Pi_{\overline{\mathcal{F}}}$ denote the $\overline{\mathcal{F}}$-free graphs. Clearly, Π is r-cisp if and only $\Pi_{\overline{\mathcal{F}}}$ is r-clip. Thus, the correctness of Reduction Rule 3.2 follows by the observation that, for a graph $G = (V, E)$ and $S \subseteq \mathcal{P}_2(V)$, it clearly holds that $G \,\Delta\, S$ has property Π if and only if $\overline{G} \,\Delta\, S$ has property $\Pi_{\overline{\mathcal{F}}}$. □

With Lemma 3.2 we can show the correctness of Reduction Rules 3.1 and 3.2.

Lemma 3.3. *Reduction Rule 3.1 (Reduction Rule 3.2) is correct for* Π-Editing, Π-Deletion, *and* Π-Addition *for every r-cisp (r-clip) graph property* Π. *Moreover, Reduction Rules 3.1 and 3.2 can be exhaustively applied in $O(|V| + |E|)$ time.*

Proof. Let (G, k) denote an input instance for Π-Editing, Π-Deletion, or Π-Addition for an r-cisp (r-clip) graph property Π. Furthermore, let I denote a critical independent set (critical clique) of G with $|I| > k + r$, let I' denote a set of $k + r$ arbitrarily chosen vertices from I, and let $G' := G - (I \setminus I')$ (that is, G' is the graph that results from applying Reduction Rule 3.1 (Reduction Rule 3.2) to I). For the correctness, we show that (G, k) is a yes-instance of Π-Editing(/Deletion/Addition) if and only if (G', k) is a yes-instance of Π-Editing(/Deletion/Addition). The "$\Rightarrow$"-direction follows directly from the fact that Π is hereditary. For the "$\Leftarrow$"-direction note the following. By Lemma 3.2, if (G', k') is a yes-instance, then there is a solution for (G', k) that does not involve any vertex of I'. Hence, I' is a critical independent set (critical clique) of $G' \,\Delta\, S$. Thus, by Lemma 3.1 one obtains a graph G'' with property Π by first adding the vertices $I \setminus I'$ to G' and then making each $x \in I \setminus I'$ adjacent to each vertex in $N_G(x)$ ($N_G[x]$). Observe that since no vertex in I' is affected, $G'' = G \,\Delta\, S$. Thus, S is a solution for G.

The running times of Reduction Rule 3.1 and Reduction Rule 3.2 follow from the fact that all critical cliques and all critical independent sets of a graph can be computed in $O(|V| + |E|)$ time [108, 133, 145]. □

Reduction Rules 3.2 and 3.1 turned out to be very useful for several kernelizations. For Cluster Editing and Bicluster Editing, where the desired graph properties are 1-clip and 1-cisp, respectively, Reduction Rules 3.2 and 3.1 lead to quadratic–vertex kernels that can be computed in linear time [145]. We use Reduction Rule 3.2 for 1-Edge Overlap Deletion (here the desired graph property is 2-clip) in Section 5.5.1 and for 2-Vertex Overlap Deletion (here the desired graph property is 1-clip) in Section 5.5.2. Moreover, Reduction Rule 3.1 is used for the cubic-vertex kernel for

Minimum Flip Consensus Tree in Chapter 7, where the desired graph property is 1-cisp. Moreover, Bessy et al. [17] used Reduction Rule 3.2 for a kernelization of Closest 3-Leaf Power.

Next, we focus on 1-cisp and 1-clip graph properties. Some of the graph properties introduced in Chapter 5 in the context of graph-based data clustering with overlaps and the graph property that characterizes the target graphs in the case of Minimum Flip Consensus Tree (see Chapter 7) are 1-cisp and 1-clip, respectively.

Using Lemma 3.1, we will show that for edge modification problems for 1-cisp or 1-clip graph properties there is an optimal solution treating the vertices of a critical independent set or critical clique equally. More specifically, we will show that for a 1-cisp (1-clip) graph property Π there is an optimal solution S for Π-Editing such that if two nonadjacent (adjacent) vertices have an identical open neighborhood (closed neighborhood) in the input graph G, then these two vertices have an identical open neighborhood (closed neighborhood) in the final target graph $G \Delta S$. The kernelizations in Section 5.5.2 and Section 7.4 rely on this observation. The advantage of this observation is that since an optimal solution applies the same edge modifications to all vertices in the same critical independent set (critical clique) we can treat these vertices as one "super vertex". First, we show the correctness of this structural observation for the case of 1-cisp graph properties.

Lemma 3.4. *Let Π be a 1-cisp graph property and let $G = (V, E)$ denote an undirected graph. There exists a minimum-cardinality solution for Π-Editing (Π-Deletion, Π-Addition) such that every critical independent set I of G is part of a critical independent set in $G \Delta S$.*

For the proof of Lemma 3.4, we show the following more general statement, used in Section 7.4 for the correctness proof of a data reduction rule for Minimum Flip Consensus Tree.

Lemma 3.5. *Let Π be a 1-cisp graph property and let $G = (V, E)$ denote an undirected graph. Moreover, let S denote a solution for Π-Editing (Π-Deletion, Π-Addition) on G and let X denote the set of vertices affected by S. Then, there exists a solution S^* such that*

- *$|S^*| \leq |S|$,*
- *every critical independent set I of G is part of a critical independent set in $G \Delta S^*$, and*
- *$X^* \subseteq X$, where X^* denotes the set of vertices affected by S^*.*

Proof. Assume that there exists a critical independent set I of G that is not contained in a critical independent set in $G \Delta S$. We show that, by a local modification, one can find a graph $G' = (V, E') \in \Pi$ such that

1. the edit distance of G' to G is at most the edit distance of $G \Delta S$ to G,
2. I is contained in a critical independent set in G',
3. for each critical independent set I' of G, the number of critical independent sets of G' intersecting with I' is at most the number of critical independent sets of $G \Delta S$ intersecting with I', and

4. the set of affected vertices in G' (i.e., the set of vertices affected by $E(G) \Delta E(G')$) is a subset of X.

By Condition 3, this local modification step can be applied iteratively until every critical independent set of G is contained in a critical independent set of the resulting graph. Moreover, by Condition 4 the set of affected vertices in the resulting graph is a subset of X.

First, we formally specify the modification. Then, we show that the above conditions are fulfilled. Let $I_1, I_2, \ldots, I_\ell$ denote the critical independent sets in $G \Delta S$ with $I_i \cap I \neq \emptyset$, $1 \leq i \leq \ell$. Observe that $\ell > 1$. For a vertex $w \in I$ let S_w denote the set of edge modifications from S involving w and vertices in $V \setminus I$, that is, $S_w := \{\{w, x\} \in S \mid x \in V \setminus I\}$. Let $v \in I$ such that $|S_v|$ is minimal and assume without loss of generality that $v \in I_1$. Build a graph G' as follows from $G \Delta S$. First, remove all vertices in $I \setminus \{v\}$ from $G \Delta S$. Then, add the vertices in $I \setminus \{v\}$ step by step, making each vertex adjacent to the vertices in the current open neighborhood of v. Let $S' := E(G) \Delta E(G')$ and note that $G' = G \Delta S'$.

By Lemma 3.1, G' fulfills Π. To prove Condition 1, we show that $|S'| \leq |S|$. Note that in the construction above we "change" only edges incident to the vertices in I and, hence, $G'[V \setminus I]$ is identical to $G \Delta S[V \setminus I]$. Moreover, since every vertex in $w \in I \setminus I_1$ gets the same closed neighborhood as v (more specifically, the vertices in $N_{G \Delta S}(v) \setminus I$), we have to spend at most $|S_v|$ edge modifications for every $w \in I \setminus \{v\}$ (instead of at least $|S_w|$). Since $|S_v| \leq |S_w|$ by the choice of v, it follows that $|S'| \leq |S|$.

Condition 2 follows directly from the fact that by construction all vertices in I have the same open neighborhood in G' (namely, $N_{G \Delta S}(v) \setminus I$).

Next, we show that Condition 3 is fulfilled. To this end, note that deleting a vertex does not increase the number of critical independent sets of a graph. Moreover, observe that two nonadjacent vertices with an identical neighborhood have an identical neighborhood after adding a vertex and making it adjacent to the neighbors of an existing vertex. Hence, for each critical independent set I' of G the number of critical independent sets of G intersecting with I' is at most the number of critical independent sets of G' intersecting with I'.

Clearly, by construction a vertex w not affected by S is not affected by S' implying Condition 4. □

Chapter 7 is concerned with an edge modification problem on bipartite graphs. For edge modification problems on bipartite graphs, a solution does not contain edge insertions between the vertices of a critical independent set. Since in none of the proofs edge insertions between the vertices of a critical independent set are applied, the presented results for r-cisp graph properties also hold for edge modification problems on bipartite graphs.

For 1-clip graph properties an analogous result to Lemma 3.4 can be shown by a straightforward adaption of the proof. Indeed, Bessy et al. [17] used an analogous result for 1-clip graph properties. They considered graph properties that are closed under "true twin addition": A graph property Π is closed under *true twin addition* if for any graph $G \in \Pi$ adding a vertex and making it adjacent to each vertex in $N[v]$ for some vertex $v \in V(G)$ yields a graph with property Π. Clearly, by Lemma 3.1 a 1-clip graph property Π is closed under true twin addition. Moreover, we can show the following.

Lemma 3.6. *A hereditary graph property* Π *is 1-clip if and only if* Π *is closed under true twin addition.*

Proof. The "$\Rightarrow$"-direction follows directly by Lemma 3.1.

For the "$\Leftarrow$"-direction we show that if Π is not 1-clip, then Π is not closed under true twin addition. To this end, consider a graph property Π not being 1-clip and let F denote a minimal forbidden induced subgraph for Π containing a critical clique K with $|K| > 1$. Let $x \in K$ be arbitrarily chosen. Since F is a minimal forbidden induced subgraph, $(F - x) \in \Pi$. However, F results from $F - x$ by a "true twin addition" (that is, by adding x to F and making it adjacent to each vertex in $N_G[v]$, for an arbitrarily chosen vertex $v \in K \setminus \{x\}$). Hence, Π is not closed under true twin addition. □

Bessy et al. [17, Lemma 1.4] have shown that for graph properties that are closed under true twin addition, two vertices that have an identical closed neighborhood in the input graph, have an identical closed neighborhood in the final target graph. Thus, by Lemma 3.6 one arrives at the following.

Lemma 3.7. *[17, Lemma 1.4] Let* Π *be a* 1*-clip graph property and let* $G = (V, E)$ *denote an undirected graph. There exists a minimum-cardinality solution* S *for* Π-Editing *(*Π-Deletion*,* Π-Addition*) such that every critical clique* K *of* G *is part of a critical clique in* $G \Delta S$*.*

Guo [89, Lemma 2] uses a similar observation for a kernelization of Cluster Editing. It says that for a specific critical clique K of the input graph G there exists an optimal solution $S \subseteq E$ such that K is part of a critical clique in $G \Delta S$. Thus, Lemma 3.7 is a stronger claim than Guo's corresponding result.

In this section, we presented a universal data reduction rule that is used for several kernelizations in this work. This is a first effort to generalize data reduction rules for edge modification problem. As also discussed in the concluding section of the thesis (Chapter 10), the "unification" of other data reduction rules used for edge modification problems and the design of new data reduction rules that work for whole classes of edge modification problems are desirable.

Chapter 4

Cluster Editing and Cluster Deletion

4.1 Introduction

The NP-hard Cluster Editing problem is among the best-studied parameterized problems. It has applications in bioinformatics [15, 152, 158], document clustering, and agnostic learning [12]. Let a *cluster graph* be a graph where every connected component is a clique. Then, the problem is defined as follows.

Definition 4.1. Cluster Editing (CE)
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Can G be transformed into a cluster graph by applying at most k edge modifications?

An illustration is given in Figure 4.1. Cluster Deletion (CD) is defined analogously except that only edge deletions are allowed.

In the field of parameterized algorithmics, Cluster Editing has almost exclusively been studied parameterized by the solution size k. In a nutshell, in this chapter we show that Cluster Editing and Cluster Deletion are fixed-parameter tractable with respect to a refined parameter (see Section 2.3), namely the cluster vertex deletion number, which is typically smaller than the standard parameter k. Furthermore, we briefly discuss other alternative parameterizations for Cluster Editing. In the remainder of this section, we review previous literature of Cluster Editing, point to related problems, and motivate the considered parameterization.

4.1.1 Previous Work

Cluster Editing has been introduced independently by several publications. We start by describing the "historical development" including different applications followed by an overview of algorithmic results.

To the best of our knowledge, Cluster Editing (under a different name) was introduced by Zahn, Jr. in 1964 in a paper entitled "Approximating symmetric relations by equivalence relations" [164]. In other words, Zahn introduced Cluster Editing as the problem to fit symmetric relations (that is, graphs) with equivalence

Figure 4.1: An example for Cluster Editing. On the left, the input graph is shown. Inserting an edge between 1 and 3 and deleting the edge between 4 and 5 results in the cluster graph shown on the right.

relations (that is, cluster graphs). In this paper, Cluster Editing was motivated by application scenarios "in which an interconnected structure or organization must be partitioned (perhaps for cataloging or formal administrative purposes) in a way which reflects the actual interconnections as well as possible". Zahn describes a solving strategy for Cluster Editing on a special graph class [164]. Referring to the work of Zahn, Křivánek and Morávek [125] were the first to show the NP-hardness of Cluster Editing. In addition to previous described applications, they investigated Cluster Editing in the context of hierarchical clustering. We refer to Chapter 6 for more details concerning the relationship between Cluster Editing and hierarchical clustering.

In 1999, Ben-Dor et al. [15] "reinvented" Cluster Editing. More specifically, they developed an algorithm for a problem in clustering gene expression patterns leading to a heuristic for Cluster Editing. Finally, the name "Cluster Editing" was introduced by Shamir et al. [150], who started the investigation of Cluster Editing formalized as an edge modification problem. In this work [150] also Cluster Deletion was introduced. Furthermore, Cluster Editing is exactly the same as the Correlation Clustering problem on complete graphs in its original formulation introduced by Bansal et al. [12], which is motivated by document clustering and agnostic learning. Moreover, very recently Cluster Editing has been employed for clustering biological data such as protein similarity data [158, 25, 159, 160].

In the following, we provide an overview of the algorithmic results for Cluster Editing. Note that some general results on edge modification problems have been provided in Section 3.3. The NP-hardness of Cluster Editing has been shown several times [125, 47, 150, 12], given in chronological order. The NP-hardness of Cluster Deletion has been showed by Shamir et al. [150]. Moreover, the problem that asks for a cluster graph consisting of at most two clusters by a minimum number of edge modifications is NP-hard [150].

The polynomial-time approximability and the parameterized complexity of Cluster Editing have intensively been investigated. As to approximability, Shamir et al. [150] showed that there exists some constant $\epsilon > 0$ for which it is NP-complete to approximate Cluster Deletion within a factor of $1 + \epsilon$. The analogous result was shown by Charikar et al. [43] for Cluster Editing. On the positive side, Charikar et al. [43] presented a polynomial-time factor-4 approximation algorithm for Cluster Editing. A randomized factor-2.5 polynomial-time approximation algorithm was given by Ailon [6]. Finally, van Zuylen and Williamson [166] devised a deterministic factor-2.5 polynomial-time approximation for Cluster Editing.

Several studies of Cluster Editing investigate the parameterized complexity with respect to the solution size k. The first nontrivial fixed-parameter tractability

results for Cluster Editing and Cluster Deletion are due to Gramm et al. [83]. After a series of improvements [73, 145, 89, 24, 46, 28], the currently fastest fixed-parameter algorithm for Cluster Editing for this parameter is due to Böcker and Damaschke and has running time $O(1.76^k+|V|^3)$ [28]. Moreover, the currently smallest problem kernel is due to Chen and Meng and contains at most $2k$ vertices [46]. Several experimental studies on the application of fixed-parameter algorithms have been performed [57, 25] demonstrating that fixed-parameter algorithms can be successfully applied to solve Cluster Editing on real-world instances. Damaschke [54] investigated Cluster Editing in the context of enumeration. He showed that a concise enumeration of all inclusion-minimal solutions of size at most k can be accomplished in $O(2.27^k + k^2|V| + |E|)$ time.

The parameterized complexity of Cluster Deletion with respect to the parameter k has first been investigated by Gramm et al. [82]. They presented a search tree algorithm with running time $O(1.53^k + |V|^3)$. Based on a characterization of graphs where each edge is contained in at most two induced P_3's, Damaschke [53] devised a refined search tree algorithm with running time $O(1.47^k + |V|^3)$. Very recently Böcker and Damaschke [28] presented a further improved search tree algorithm with running time $1.415^k \cdot |V|^{O(1)}$.

4.1.2 Related Problems

The problem to transform a bipartite graph into a graph where every connected component forms a biclique (complete bipartite graph) by at most k edge modifications is called Bicluster Editing. Bicluster Editing can be solved in $3.24^k|V|^{O(1)}$ time and admits a problem kernel with $6k$ vertices [93, 145]. Ailon et al. [4] devised a randomized factor-4 approximation algorithm for Bicluster Editing.

The "vertex deletion version"of Cluster Editing, that is, to transform a graph into a cluster graph by a minimum number of vertex deletions, is called Cluster Vertex Deletion. Hüffner et al. [109] presented an $O(2^k k^9 + |V| \cdot |E|)$-time iterative compression algorithm for Cluster Vertex Deletion.

Correlation Clustering has originally been defined as follows [12]. Given a *complete* graph with edge-labels "+" and "−", where a "+"-edge stands for high similarity and a "−"-edge stands for low similarity of the entities represented by the respective vertices, Correlation Clustering asks for a partition of the vertices into clusters such that the number of "−"-edges inside the clusters plus the number of "+"-edges between the clusters is minimized. In this sense, Correlation Clustering is the problem to find a partition into clusters minimizing the disagreements. Correlation Clustering is hence equivalent to Cluster Editing on the graph containing only the "+"-edges, a "−"-edge inside a cluster corresponding to an edge insertion and a "+"-edge between two clusters corresponding to an edge deletion. Thus, Cluster Editing is exactly the same as Correlation Clustering on complete graphs.

The version of Correlation Clustering for general (*noncomplete*) graphs is also called Fuzzy Cluster Editing [31]. In the Cluster Editing setting this means that some edges of the input graph are "undecided" in the sense that the insertion or deletion of an undecided edge does not contribute to the editing cost. Bodlaender et al. [31] presented a problem kernel with $O(k^2 + r)$ vertices for Fuzzy Cluster Editing, where k is the editing cost and r is the minimum number of vertices needed to cover all undecided edges. Until recently, it was an open question

whether Fuzzy Cluster Editing or equivalently Correlation Clustering is fixed-parameter tractable with respect to the editing cost alone. Recently, Marx and Razgon [132] answered this question to the positive by proving that Multicut is fixed-parameter tractable with respect to the cut size and using a parameterized reduction from Correlation Clustering to Multicut.

Further studies deal with the parameterized complexity of different generalizations of Cluster Editing [54, 71, 95, 96]. Basically, Cluster Editing has been generalized in three ways.

First, Cluster Editing has been generalized by replacing the clique requirement in the cluster graph with other models for dense graphs [96, 95]. The fact that the clique concept has been criticized to be overly restrictive in some application scenarios [48, 149] motivated the investigation of combining clique relaxations and graph-based data clustering. In the case of s-Plex Cluster Editing [96], it is required that the connected components of the cluster graph form so-called s-plexes (instead of cliques). An s-plex is a graph where every vertex is adjacent to all but s vertices [149]. Guo et al. [96] have shown that s-Plex Cluster Editing is fixed-parameter tractable for the combined parameter (s, k), where k denotes the number of allowed edge modifications. In particular, data reduction rules and a polynomial-size kernelization result have been presented. Similarly, Guo et al. [95] considered other clique relaxations in combination with graph-based data clustering.

Second, Cluster Editing has been generalized by allowing overlapping clusters in the cluster graph [54, 71]. In Chapter 5 the focus is on cluster models allowing some degree of overlap between the clusters. There, we will introduce a model for graph-based data clustering with overlaps and provide an overview on related work.

Third, Heggernes et al. [107] introduced a generalization of cluster graphs, so called (p, q)-*cluster graphs*. A graph G is called (p, q)-cluster graph if its vertex set can be partitioned into subsets with each subset missing at most p edges from being a clique and having at most q edges going to other subsets. Observe that $(0, 0)$-cluster graphs are exactly cluster graphs. Heggernes et al. [107] showed that the recognition problem for (p, q)-cluster graphs is NP-hard in general and presented polynomial-time algorithms for recognizing $(0, q)$-cluster, $(p, 1)$-cluster, $(p, 2)$-cluster, and $(1, 3)$-cluster graphs. They leave open the parameterized complexity of (p, q, k)-Cluster Graph Editing, the problem to transform a given graph into a (p, q)-cluster graph by inserting and deleting at most k edges. Lokshtanov and Marx [129] considered a further generalization of (p, q)-cluster graphs. From their results it follows that the recognition of (p, q)-cluster graphs is fixed-parameter tractable parameterized by p or by q.

4.1.3 Our Results

So far, the proposed fixed-parameter algorithms for Cluster Editing and Cluster Deletion concentrate on the parameter solution size k. It has been observed that the parameter k is often not really small for real-world instances [25, 58]. Although the fixed-parameter algorithms can still solve many of these instances [25], this lead to the call for "better parameterizations" [58]. In particular, this raises the question whether there are "hidden parameters" that are implicitly exploited by these algorithms. Hence, this work aims at identifying promising new parameterizations for Cluster Editing and Cluster Deletion that help to separate easy from hard

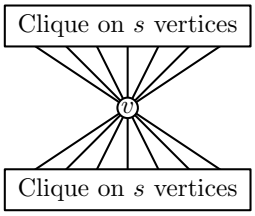


Figure 4.2: A graph consisting of two cliques of order s that are connected by a vertex v adjacent to all vertices in the two cliques. Clearly, for every $s \geq 1$, the cluster vertex deletion number is one. In contrast, since there are s P_3's intersecting only in v, at least s edge modifications are necessary to transform this graph into a cluster graph.

instances. We mainly focus on the parameterization *cluster vertex deletion number* c of graph G, that is, the minimum number of vertex deletions required to transform a graph into a cluster graph. The cluster vertex deletion number is at most the size k of a minimum-cardinality edge modification set: deleting for each edge modification one of the two vertices (arbitrarily chosen) clearly results in a cluster graph. Moreover, as illustrated in Figure 4.2, there is an unbounded number of instances such that the cluster vertex deletion number is constant but k depends on the number of vertices. Thus, the cluster vertex deletion number is a refined parameter for Cluster Editing as well as for Cluster Deletion (see Section 2.3). Answering an open question of Dehne [58], we show that Cluster Editing and Cluster Deletion are fixed-parameter tractable with respect to the parameter cluster vertex deletion number of the input graph. Finally, we briefly discuss further alternative parameterizations. For example, we discuss that Cluster Editing and Cluster Deletion remain NP-hard for graphs with maximum vertex degree six.

Preliminaries. For general notation concerning graphs and edge modification problems see Sections 2.6 and 3.3.1, respectively. We briefly recall the concepts needed here. A graph where every connected component is a clique is called *cluster graph*. The cliques of a cluster graph are referred to as *clusters*. Clearly, the property of being a cluster graph is hereditary (see Definition 3.2). The only forbidden induced subgraph for cluster graphs is a path on three vertices (a so-called P_3) [150]. Throughout this chapter, let c denote the cluster vertex deletion number of a graph G, that is, the minimum number of vertex deletions to end up with a cluster graph. For two sets A and B with $A \cap B \neq \emptyset$, we use $A \uplus B$ to denote $A \cup B$.

A clique K is called *critical* if all vertices have an identical neighborhood and K is maximal with respect to this property (see Definition 2.4). For two graphs $G = (V, E)$ and $G' = (V', E')$ with $|V| = |V'|$ a *graph isomorphism* is a bijective function $\phi : V \to V'$ such that for all $v, w \in V$ it holds that $\{v, w\} \in E \iff \{\phi(v), \phi(w)\} \in E'$.

4.2 Cluster Vertex Deletion Number as Parameter

In this section, we present two fixed-parameter algorithms for Cluster Editing and Cluster Deletion parameterized by the cluster vertex deletion number c. Both

algorithms make use of an observation for cliques that are large in comparison to the size of their neighborhood. Basically, such a clique is "largely preserved" by any optimal solution for CE or CD since it is "cheaper" to cut all edges to its neighbors than to cut it into several small pieces. More specifically, we show the following.

Lemma 4.1. *Let K denote a clique in G of size at least $2 \cdot |N_G(K)|$. Then, for every optimal solution S for* Cluster Editing *or* Cluster Deletion *the graph $G \Delta S$ contains a cluster K' with*

$$|K \cap K'| \geq |K| - 2|N_G(K)|.$$

Proof. First, we show the lemma for the case of an optimal solution S for Cluster Editing. To this end, let $K'_1, \ldots, K'_\ell$ denote the clusters in $G \Delta S$ with $K'_i \cap K \neq \emptyset$, $1 \leq i \leq \ell$. Furthermore, for all $1 \leq i \leq \ell$, define $B_i := K'_i \cap K$. Observe that $K = \bigcup_{i=1}^{\ell} B_i$. In the case that $|K| \leq 2|N_G(K)| + 1$ the lemma trivially holds since $|B_1| \geq 1$ and $|K| - 2|N_G(K)| \leq 1$. Hence, $|K| > 2|N_G(K)| + 1$ in what follows.

For the correctness of the lemma we argue that there is an i, $1 \leq i \leq \ell$, with $|B_i| \geq |K| - 2|N_G(K)|$. Assume towards a contradiction that all B_i's contain less than $|K| - 2|N_G(K)|$ vertices. This implies that—in order to separate the B_i's from each other—the solution contains at least

$$1/2 \sum_{i=1}^{\ell} |B_i|(|K| - |B_i|) > 1/2 \sum_{i=1}^{\ell} (|B_i| \cdot 2|N_G(K)|) = |N_G(K)| \cdot |K|$$

edge deletions. Hence, one obtains a cluster graph that is closer to G than $G \Delta S$ by deleting in $G \Delta S$ all edges between K and $N(K)$ (at most $|N(K)| \cdot |K|$) and undoing all edge deletions between vertices in K (at least $|N(K)| \cdot |K| + 1$); a contradiction to the fact that S is optimal. It is easy to verify that, since we apply only edge deletions, all steps of the proof hold for an optimal solution for CD, too. □

Next, we present our fixed-parameter algorithm for Cluster Editing.

4.2.1 Cluster Editing

Given an input graph G and a size-c cluster vertex deletion set Y of G, the key observation used by our algorithm is that clusters in $G - Y$ that are much larger than Y will not be split by any optimal solution for CE. The basic idea for showing this observation is as follows. Consider a cluster K in $G - Y$ of size at least $3c+1$. By Lemma 4.1 for every solution S for CE the cluster graph $G \Delta S$ contains a cluster K' that intersects with K in at least $c+1$ vertices. We can show that $K' \subseteq Y \cup K$. Thus, for a vertex $v \in K \setminus K'$ it would be cheaper to add v to K' since this would require at most c edge insertions to vertices of Y but would allow to undo at least $c+1$ edge deletions. As a consequence, $K \subseteq K'$.

Next, we prove this key observation.

Lemma 4.2. *Let Y denote a size-c cluster vertex deletion set and let K denote a cluster in $G - Y$ of size at least $3c + 1$. Then, for every optimal solution S for* CE *the graph $G \Delta S$ contains a cluster K' with*

$$K \subseteq K' \subseteq K \cup Y.$$

Proof. Let $K_1, \ldots, K_\ell$ ($\ell \geq 1$) denote the clusters in $G \Delta S$ with $K_i \cap K \neq \emptyset$. Let $B_i := K \cap K_i$ and observe that $K = \bigcup_{i=1}^{\ell} B_i$. Without loss of generality, assume that B_1 has maximum cardinality of all B_i's. Since $N_G(K) \subseteq Y$ and $|K| \geq 3|Y| + 1 > 2|Y|$, Lemma 4.1 implies that $|B_1| \geq c + 1$. For the correctness of the lemma, we show that $K \subseteq K_1 \subseteq K \cup Y$, that is, $K' = K_1$.

First, we show that $K_1 \subseteq B_1 \cup Y$, that is, $K_1 \subseteq K \cup Y$. Let $X := K_1 \setminus (B_1 \cup Y)$ and assume towards a contradiction that $X \neq \emptyset$. Since $|B_1| \geq c + 1$, one obtains a cluster graph that is closer to G than $G \Delta S$ by modifying $G \Delta S$ such that X becomes an isolated clique. This requires at most $|X| \cdot c$ edge deletions to separate X from $K_1 \cap Y$, however, allows to undo the edge insertions between X and B_1 which amount to at least $|X| \cdot (c + 1)$; a contradiction.

Next, we prove that $\ell = 1$ and, hence, $K \subseteq K_1$. Assume towards a contradiction that $\ell > 1$. We argue that one obtains a cluster graph that is closer to G than $G \Delta S$ by modifying $G \Delta S$ such that $K_1 \cup B_2$ becomes an isolated clique. More specifically, consider the cluster graph G' that results from $G \Delta S$ by:

- Deleting all edges between B_2 and $K_2 \setminus B_2$. Note that in $G[K_2]$ there are only edges between B_2 and $K_2 \cap Y$. Hence, this step requires at most $|K_2 \cap Y| \cdot |B_2|$ additional edge deletions.
- Inserting all edges between B_2 and $K_1 \setminus B_1$. Since $(K_1 \setminus B_1) \subseteq Y$ this are at most $|B_2| \cdot |K_1 \cap Y|$ edge insertions.
- Undoing the edge deletions between B_2 and B_1. These amount to $|B_2| \cdot |B_1|$ since $B_1 \cup B_2$ forms a clique in G.

Since $|B_1| > c$ this implies that $|B_2| \cdot |K_2 \cap Y| + |B_2| \cdot |K_1 \cap Y| \leq |B_2| \cdot c < |B_2| \cdot (c+1) \leq |B_2| \cdot |B_1|$. As a consequence, the resulting cluster graph is closer to G than $G \Delta S$; a contradiction to the assumption that S is optimal.

It follows that $K \subseteq K_1 \subseteq K \cup Y$, yielding the lemma for $K' = K_1$. □

Description of the Algorithm. By Lemma 4.2, a cluster in $G - Y$ of size at least $3c + 1$ will not be "split" or "merged" with any other clusters of $G - Y$ in an optimal cluster graph. We refer to clusters of $G - Y$ of size at least $3c + 1$ as *large clusters*.

Now, the basic idea to establish fixed-parameter tractability is as follows. See procedure `CEbyCVD` (Alg. 1) for an outline. Given a cluster vertex deletion set Y, in a first step, we guess the partition of Y "induced" by the clusters generated by an optimal solution for CE and apply the respective edge modifications (Lines 6 to 8). We refer to the sets of such a partition as *fixed subclusters* in the following. Then, in a second step, we guess which of these fixed subclusters will end up in a cluster together with a large cluster (Line 11). From Lemma 4.2, we know that the large clusters are not split and, since the subclusters in Y are fixed, the large clusters end up in a final cluster with at most one fixed subcluster and vice versa. We will show that the "mapping" of the large clusters to the respective fixed subclusters can efficiently be done by computing a maximum-weight matching. To this end, procedure `CEbyCVD` employs the subroutine `SolveLarge` (see Line 14). In the remaining instance all clusters from $G - Y$ have size at most $3c$. For solving these instances we devise a subroutine, called `SolveSmall`. This subroutine uses data reduction to bound the number of small

Function `CEbyCVD(`G`)`
Input: A graph $G = (V, E)$.
Output: Size of an optimal solution S for Cluster Editing.

```
1  Y = SolveCVD (G);
2  Let A_1, ..., A_p denote the clusters in G − Y of size at most 3c;
3  Let B_1, ..., B_q denote the clusters in G − Y of size at least 3c + 1;
4  m_1 = +∞;
5  Let G* := G (keep a copy of the original graph);
6  forall partitions Q_1, ..., Q_t of Y (1 ≤ t ≤ |Y|) do
7      Add all edges between vertices in Q_i, 1 ≤ i ≤ t;
8      Delete all edges between Q_i and Q_j, 1 ≤ i < j ≤ t;
9      Let c_1 denote the number of these edge modifications;
10     m_2 = +∞;
11     forall subsets I ⊆ {1, ..., t} do
12         Delete all edges between Q_i and A_j, i ∈ I and 1 ≤ j ≤ p;
13         Let c_2 denote the number of these edge deletions;
14         c_l = SolveLarge ({Q_i | i ∈ I}, B_1, ..., B_p);
15         c_s = SolveSmall ({Q_i | i ∈ {1, ..., t} \ I}, A_1, ..., A_q);
16         m_2 = min(m_2, c_2 + c_l + c_s);
17     end
18     m_1 = min(m_1, c_1 + m_2);
19     Let G := G* (undo all changes);
20 end
21 return m_1;
```

Algorithm 1: Algorithm for Cluster Editing parameterized by the cluster vertex deletion number.

clusters in $G - Y$ by a function only depending on c, thus yielding a problem kernel for this subproblem. Altogether, this implies fixed-parameter tractability of Cluster Editing parameterized by the cluster vertex deletion number.

In the following, we first consider the two subproblems for large and small clusters separately. Based on these results, we then establish the correctness and running-time bound of `CEbyCVD`.

Large Clusters. To compute an optimal solution for the subproblem that has to be solved by `SolveLarge` (Line 14 of Alg. 1), we have to find a solution to the following problem.

Fixed Clique Cluster Editing:

Input: A graph $G = (V, E)$ with $B \uplus Q = V$ such that $G[B]$ and $G[Q]$ are cluster graphs. Let $\mathcal{B} = \{B_1, \ldots, B_q\}$ be the set of clusters in $G[B]$ and let $\mathcal{Q} = \{Q_1, \ldots, Q_s\}$ be the set of clusters in $G[Q]$.

Task: Find a cluster graph G_c on V with minimum edit distance to G such that for each cluster K in G_c either

- $K = B_i$ for a $B_i \in \mathcal{B}$ or,
- $K = Q_j$ for a $Q_j \in \mathcal{Q}$, or
- $K = B_i \cup Q_j$ for a $B_i \in \mathcal{B}$ and a $Q_j \in \mathcal{Q}$.

As shown in the following, Fixed Clique Cluster Editing can be formulated as a bipartite maximum-weight matching problem and, hence, can be solved in polynomial time. The procedure `SolveLarge` in Alg. 1 solves instances of Fixed Clique Cluster Editing based on the following lemma.

Lemma 4.3. Fixed Clique Cluster Editing *can be solved in polynomial time.*

Proof. First, we describe how to build an auxiliary graph H with an edge weight function w. Then, we show that a maximum-weight matching for H corresponds to a solution cluster graph for Fixed Clique Cluster Editing and vice versa.

Let $H = (U_{\mathcal{B}}, U_{\mathcal{Q}} \cup U'_{\mathcal{B}}, F)$ denote the bipartite graph that contains two vertices $u_{B_i} \in U_{\mathcal{B}}$ and $u'_{B_i} \in U'_{\mathcal{B}}$ for every $B_i \in \mathcal{B}$, a vertex $u_{Q_j} \in U_{\mathcal{Q}}$ for every $Q_j \in \mathcal{Q}$, and with $F := \{\{u_{B_i}, u'_{B_i}\} \mid B_i \in \mathcal{B}\} \cup \{\{u_{B_i}, u_{Q_j}\} \mid 1 \leq i \leq q, 1 \leq j \leq s\}$. Moreover, let $\kappa(B_i, Q_j)$ denote the minimum number of edge insertions needed to transform $B_i \cup Q_j$ into a clique (that is, the number of "missing edges" in $G[B_i \cup Q_j]$) plus $|\{\{u, v\} \in E \mid u \in B_i, v \in \bigcup_{j' \neq j} Q_{j'}\}|$ (that is, the number of edge deletions to separate each vertex in B_i from every vertex in $Q_{j'}$ for all $j' \neq j$). Moreover, let $\kappa(B_i)$ denote the minimum number of edge deletions needed to make B_i an isolated clique. Observe that these two values can easily be computed by counting the edges between B_i and Q_j for all $1 \leq i \leq q$ and $1 \leq j \leq s$. Finally, we define a weight function $w : F \to \mathbb{N}_{\geq 0}$ as follows: $w(\{u_{B_i}, u'_{B_i}\}) := T - \kappa(B_i)$ and $w(\{u_{B_i}, u_{Q_j}\}) := T - \kappa(B_i, Q_j)$, where $T - 1$ is the maximum over all $\kappa(.)$. This ensures that all weights are at least 1. Altogether, one arrives at the following.

> *Claim.* Let m denote the value of a maximum-weight matching in H with respect to the weight function w and let d denote the minimum edit distance between the input graph G and a cluster graph G_c fulfilling the constraints in the definition of Fixed Clique Cluster Editing. Then, $d = T \cdot q - m$.

The correctness of the claim can be easily seen as follows. Given a maximum-weight matching for H it is easy to verify that all vertices in $U_{\mathcal{B}}$ are matched since every $u_B \in U_{\mathcal{B}}$ has a degree-one neighbor $u'_B \in U'_{\mathcal{B}}$ and all edge weights are positive. Hence, from a maximum-weight matching of H one can obtain a cluster graph such that for each matching edge $\{u_{B_i}, u'_{B_i}\}$, the vertices from B_i form a cluster and for each matching edge $\{u_{B_i}, u_{Q_j}\}$, the vertices from $B_i \cup Q_j$ form a cluster and *vice versa*. Due to the definition of the edge weights the claim directly follows. □

Small Clusters. Next, we focus on instances of Cluster Editing where all clusters in $G - Y$ have size at most $3c$. The procedure `SolveSmall` (Line 15 in Alg. 1) solves these instances based on the following lemma.

Lemma 4.4. *Let Y denote a cluster vertex deletion set for G of size c. If all clusters in $G - Y$ have size at most $3c$, then* CE *can be solved in $2^{2^{O(c^2)}} \cdot |V|^{O(1)}$ time.*

Proof. The basic idea to show fixed-parameter tractability is as follows. We group the clusters of $G - Y$ into $2^{O(c^2)}$ different "types". Then, we show that from each cluster type we need to keep at most c representatives and, hence, the overall input size is bounded (directly implying fixed-parameter tractability).

We say that two clusters Q_i and Q_j of $G-Y$ have the same *cluster type* if the two graphs $G[Y \cup Q_i]$ and $G[Y \cup Q_j]$ are identical, that is, if there is a graph-isomorphism ϕ between $G[Y \cup Q_i]$ and $G[Y \cup Q_j]$ such that $\forall v \in Y : \phi(v) = v$. Next, we show that the number of types of the clusters in $G - Y$ is bounded by $2^{O(c^2)}$.

To bound the number of cluster types, we first classify the vertices in $V \setminus Y$ in 2^c types. Two vertices in $u, w \in V \setminus Y$ have the same type if $N_G(u) \cap Y = N_G(w) \cap Y$. Then, a cluster K can be described by a vector of length 2^c, where the ith entry contains the number of type-i vertices in K. Since a small cluster contains at most $3c$ vertices, a vector contains at most $3c$ nonzero entries. Clearly, two clusters have the same cluster type if these corresponding vectors are identical. This implies that there are at most

$$\sum_{i=1}^{3c} \binom{3c \cdot 2^c}{i} \leq 3c \cdot (3c \cdot 2^c)^{3c} = 2^{O(c^2)}$$

cluster types: for every position of the length-2^c vector describing a cluster type, one has up to $3c$ choices (resulting in a total of $3c \cdot 2^c$ choices). Thus, there are at most $\binom{3c \cdot 2^c}{i}$ cluster types for clusters with i, $1 \leq i \leq 3c$, vertices.

Finally, we show that for each cluster type we can delete all but c^2 clusters. To this end, consider a closest cluster graph G_c for G. First, note that there are at most c clusters in G_c that contain vertices from Y. Second, each cluster of G_c intersecting with Y contains vertices from at most c clusters of $G-Y$ since it is straightforward to verify that otherwise separating the vertices of one cluster results in a cluster graph with the same or smaller edit distance to G. As a consequence, vertices of at most c^2 clusters of $G-Y$ are contained in clusters of G_c intersecting with Y. Finally, observe that all other clusters of $G-Y$ are clusters in G_c, too. Hence, if there are c^2+i, $1 \leq i$, clusters of $G-Y$ of the same type, then at least i of these clusters are clusters in a closest cluster graph, and, hence, can be deleted (together with the edges between these clusters and Y). After deleting these clusters there are at most $2^{O(c^2)}c^2 + c = 2^{O(c^2)}$ vertices in G. Finally, compute the closest cluster graph by testing brute force all $(2^{O(c^2)})^{2^{O(c^2)}} = 2^{2^{O(c^2)}}$ partitions of the remaining graph. □

Running Time and Correctness. Using the results on the running times of `SolveLarge` and `SolveSmall`, we now can show the fixed-parameter tractability with respect to c.

Theorem 4.1. Cluster Editing *is fixed-parameter tractable with respect to the cluster vertex deletion number c of the input graph. It can be solved in $2^{2^{O(c^2)}} \cdot |V|^{O(1)}$ time.*

Proof. We use Alg. 1 to compute the size of an optimal solution for CE for an input graph $G = (V, E)$. First, we show that Alg. 1 is correct. To this end, let G' denote a cluster graph with closest edit distance to G. Since Alg. 1 enumerates all partitions of Y, the sets $Q_1, \ldots, Q_t$ (Line 6) once one-to-one correspond to the clusters in $G'[Y]$, implying the edge modifications in Lines 8 and 9.

By Lemma 4.2, all clusters of size at least $3c+1$ ("large cluster") in $G-Y$ either form a cluster in G' or are contained in a cluster of G' together with vertices from Y. Since the algorithm has already guessed the partition of Y, every such large cluster is contained in a cluster of G' with the vertices of at most one Q_i. By trying all

two-partitions of $\{1, \ldots, t\}$ (Line 11), Alg. 1 guesses the Q_i's that are clusters in G' or that are contained in a cluster together with one large cluster. Note that the corresponding subproblem exactly corresponds to Fixed Clique Cluster Editing since in G' each clique Q_i is in a cluster with at most one large cluster B_j and *vice versa*. By Lemma 4.3, Fixed Clique Cluster Editing can be solved in polynomial time. The remaining clusters in $G - Y$ all have size at most $3c$. Hence, by Lemma 4.4 the remaining instance can be solved in $2^{2^{O(c^2)}} \cdot |V|^{O(1)}$ time.

For the running time note that there are $O(c^c)$ partitions of Y. For each such partition we try all two-partitions. Hence, Alg. 1 enters the body of the inner for-loop at most $O(2^{c \log(c)+c})$ times. Since by Lemma 4.3 the subroutine `SolveLarge` can be applied in polynomial time and since the subroutine `SolveSmall` runs in $2^{2^{O(c^2)}} \cdot |V|^{O(1)}$ time (Lemma 4.4), Alg. 1 runs in $O(2^{c \log(c)+c}) \cdot 2^{2^{O(c^2)}} \cdot |V|^{O(1)} = 2^{2^{O(c^2)}} \cdot |V|^{O(1)}$ time. □

So far, due to the large exponential factor the presented algorithm is of purely theoretical interest. Recall that the factor $2^{2^{O(c^2)}}$ in the running time is due to the subproblem that deals with instances where all clusters in $G - Y$ have size at most $3c$, see Lemma 4.4. Improving the running time for this special case of Cluster Editing would directly lead to a significantly better worst-case running time.

4.2.2 Cluster Deletion

In this section, we devise a fixed-parameter algorithm for Cluster Deletion parameterized by the cluster vertex deletion number. First, note that it is not obvious how to adapt Alg. 1. The main problem is that Lemma 4.2 does not hold for Cluster Deletion. Recall that Lemma 4.2 states that in the case of CE large clusters in $G - Y$ (where Y is a cluster vertex deletion set for G) are not "split". The proof of Lemma 4.2, however, requires the insertion of edges, which is not allowed for CD. By employing some problem-specific properties of CD, we present a new solving strategy for CD, leading to a better running time than the algorithm for CE. The basic observation is that none of the clusters of $G - Y$ (where Y is a cluster vertex deletion set) can be merged with other clusters of $G - Y$ since only edge deletions are allowed.

The remaining part of this section is devoted to the proof of the following theorem.

Theorem 4.2. Cluster Deletion *can be solved in* $O(c^{2c}(c+1)^{5c} \cdot |V|^3)$ *time, where* c *is the cluster vertex deletion number of the input graph.*

We use the following notation. For a graph $G = (V, E)$ and a set $S \subseteq E$ let $G \setminus S := (V, E \setminus S)$. Moreover, for a graph G let $\mathrm{opt}_{CD}(G)$ denote the size of an optimal solution for CD for G, that is, the minimum cardinality over all sets $S \subseteq E$ such that $G \setminus S$ is a cluster graph.

The overall strategy of our algorithm for Cluster Deletion is to reduce Cluster Deletion to at most c^c computations of a maximum-weight matching in a bipartite auxiliary graph. A crucial step in this approach will be the computation of the edge weights in this bipartite graph. The edge weights will correspond to the size of optimal solutions for subinstances.

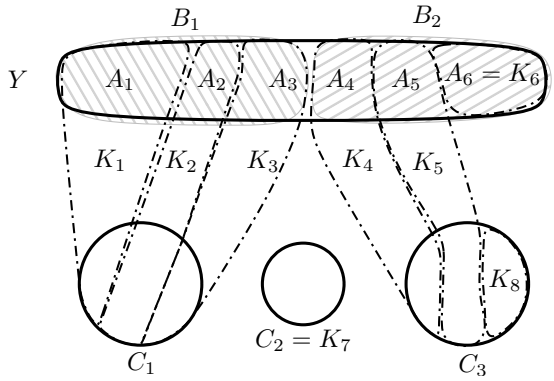


Figure 4.3: A graph over the vertex set $Y \uplus C_1 \uplus C_2 \uplus C_3$ with cluster vertex deletion set Y. C_1, C_2, and C_3 are the clusters of $G - Y$. The clusters resulting from a solution of Cluster Deletion are $K_1, \ldots, K_8$ (dashed lines). Moreover, the A_i's form a partition of Y according to the final clusters, that is, $A_i = K_i \cap Y$ $(1 \leq i \leq 6)$. Consider the partition B_1 and B_2 of Y with $B_1 = A_1 \uplus A_2 \uplus A_3$ and $B_2 = A_4 \uplus A_5 \uplus A_6$. For this partition, B_1 is matched with C_1 and B_2 is matched with C_3 while C_2 remains unmatched. Note that the algorithm first "guesses" the partition into the B_i's and then guesses an optimal partition of every B_i into A_j's.

Basic Idea of the Algorithm. Our algorithm employs some properties of the structure of a solution cluster graph for Cluster Deletion. Let $C_1, \ldots, C_z$ denote the clusters of $G - Y$ (where Y denotes a cluster vertex deletion set for G). Since only edge deletions are allowed, each cluster of a solution cluster graph can contain vertices from at most one C_i. Moreover, the vertices from C_i might be partitioned into further subsets and each of these subsets might form a cluster together with some vertices from Y. We try all possibilities to partition Y into $B_1, \ldots, B_x$ such that each B_i "matches" with at most one of the C_j's and *vice versa*. Basically, a set B_i "matches" a cluster C_j when there is a cluster in the solution graph that contains vertices from both B_i and C_j. As illustrated in Figure 4.3, a "matched pair" B_i and C_j might be further divided into several clusters in the cluster graph. Roughly speaking, the algorithm will build a bipartite graph with one side corresponding to the B_i's and the other side corresponding to the C_j's and the weight of an edge between B_i and C_j reflects the size of an optimal solution of the subgraph induced by $B_i \cup C_j$. To compute these weights, we will introduce a restricted version of Cluster Deletion which has given a further partition of B_i into subsets A_t's (see Figure 4.3) and makes use of the fact that C_j forms a clique.

Main Loop of the Algorithm. The overall algorithm `CDbyCVD` is provided in Alg. 2. Basically, `CDbyCVD` enumerates all possible partitions of Y (Line 3). The idea is that the sets of the considered partition will end up in different clusters. Hence, the edges between vertices of different B_i's are deleted (Line 4). For each such partition, the algorithm computes the cost of a maximum matching between the partition of Y and the clusters in $G - Y$ (Line 12). The corresponding auxiliary graph is built in Lines 5 to 11. The corresponding cost to match B_i with C_j is computed in Line 10.

Function `CDbyCVD(`G`,`Y`)`
Input: An CD-instance $(G = (V, E), k)$ and a cluster vertex deletion set Y for G.
Output: "true" if (G, k) is yes-instance; otherwise, "false".

1 Let $C_1, \ldots, C_z$ denote the clusters in $G - Y$;
2 Let $T := |E| + 1$;
3 **forall** *partitions* $B_1 \uplus \ldots \uplus B_x = Y$ *with* $x \leq \min(z, |Y|)$ **do**
4 Let G' denote the graph that results from G by deleting all edges between B_i and B_j, $1 \leq i < j \leq x$, and let m_1 denote the number of deleted edges;
5 Let $H = (U_B, U_C, F)$ with:
6 $U_B := \{u_{B_i} \mid 1 \leq i \leq x\}$,
7 $U_C := \{u_{C_i} \mid 1 \leq i \leq z\}$,
8 $F := \{\{u_{B_i}, u_{C_j}\} \mid 1 \leq i \leq x, 1 \leq j \leq z\}$.
9 **forall** $i, 1 \leq i \leq x$ and $j, 1 \leq j \leq z$
10 Compute $\kappa(B_i, C_j) := \mathrm{opt}_{CD}(G'[B_i \cup C_j])$;
11 Let $w(\{u_{B_i}, u_{C_j}\}) := T - \kappa(B_i, C_j) - |\{\{v, w\} \in E(G') \mid v \in B_i, w \in V \setminus (Y \cup C_j)\}|$;
12 Let W denote the weight of a maximum-weight matching in H with edge-weight function w;
13 **if** $x \cdot T - W \leq k - m_1$ **then**
14 **return** "true";
15 **end**
16 **end**
17 **return** "false";

Algorithm 2: Algorithm for Cluster Deletion parameterized by the cluster vertex deletion number.

These costs are used to define the edge-weights in the maximum matching instance H in Line 11.

Before presenting the correctness of procedure `CDbyCVD` (Alg. 2), we want to stress that the computation of $\kappa(B_i, C_j)$ (Line 10) requires to solve Cluster Deletion on the instance $G'[B_i \cup C_j]$. After presenting the correctness of procedure `CDbyCVD`, we focus on this aspect of the algorithm.

For the correctness of procedure `CDbyCVD` (Alg. 2), we show the following.

Lemma 4.5. *$(G = (V, E), k)$ is a yes-instance for CD if and only if procedure* `CDbyCVD((`G, k`),`Y`)` *(Alg. 2) returns "true", where Y is a cluster vertex deletion set for G.*

Proof. "⇐:" If the algorithm returns "true", then there is a partition $B_1, \ldots B_x$ of Y and a maximum-weight matching M for H with weight W such that $x \cdot T - W + m_1 \leq k$ (Line 13). We show how to obtain a cluster graph by at most $x \cdot T - W + m_1$ edge deletions.

First, observe that each vertex $u_{B_i} \in U_B$ is matched since H is a complete bipartite graph, $x \leq z$ (see Line 3), and all edge weights are positive. Hence, $|M| = x$. Consider the graph that results from G' by the following edge deletions. If a vertex $u_{B_i} \in U_B$ is matched to a vertex $u_{C_j} \in U_C$, then transform $G'[B_i \cup C_j]$ into a cluster graph and

delete all edges between B_i and $V \setminus (Y \cup C_j)$. This requires

$$\kappa(B_i, C_j) + |\{\{v, w\} \in E(G') \mid v \in B_i, w \in V \setminus (Y \cup C_j)\}| = T - w(\{u_B, u_c\})$$

edge deletions. Since all C_i's are clusters of $G - Y$ and there are no edges between B_i and B_j for $i \neq j$ in G', the constructed graph is a cluster graph. Finally, observe that we apply $\sum_{\{v,w\} \in M}(T - w(\{u, v\})) = x \cdot T - W$ edge deletions in this construction and m_1 edge deletions to separate the B_i's. Hence, we obtained a cluster graph by $x \cdot T - W + m_1 \leq k$ edge deletions.

"⇒": Let $S \subseteq E$ with $|S| \leq k$ denote an optimal solution for CD and let $G_S := G \setminus S$. We show that procedure `CDbyCVD((`$G = (V, E), k$`),`Y`)` returns "true".

Let x denote the number of clusters C_i of $G - Y$ containing vertices that have neighbors in Y in G_S. That is, $x = |\{C \in \{C_1, \ldots, C_z\} \mid N_{G_S}(C_i) \cap Y \neq \emptyset\}|$. Without loss of generality, let the corresponding clusters be denoted by $C_1, \ldots, C_x$. Then, for $1 \leq i < x$, let $B_i := \{v \in Y \mid N_{G_S}(v) \cap C_i \neq \emptyset\}$. Moreover, B_x consists of all remaining vertices of Y. We show that if procedure `CDbyCVD` encounters this partition in Line 3, then it returns "true" in Line 14.

First, observe that in G_S there are no edges between vertices of different B_i's. Thus, the m_1 applied edge deletions in Line 4 are contained in S. It remains to show that there is a matching in H with total weight $W \geq x \cdot T + m_1 - k$ (see Line 14).

Consider the matching M for H where $\{u_{B_i}, u_{C_i}\} \in M$ for $1 \leq i \leq x$. To analyze the total weight W of M, it is easy to verify that S can be partitioned into the sets $S_1, \ldots, S_x, S'$ where

$$S_i := (S \cap E(G[B_i \cup C_i])) \ \cup \ \{\{v, w\} \in E(G) \mid v \in B_i, w \in V \setminus (Y \cup C_i)\}$$

and S' are the m_1 edges deleted in Line 4. It directly follows by construction of the edge weights that $w(u_{B_i}, u_{C_i}) = T - |S_i|$. Hence, $W = \sum_{i=1}^{x}(T - |S_i|) = x \cdot T - k + m_1$. □

Computing the Edge Weights. For the computation of the edge weights in Line 11 of procedure `CDbyCVD` (Alg. 2) one needs to solve Cluster Deletion for subinstances of the input graph. More specifically, we need to determine $\text{opt}_{CD}(G'[B_i \cup C_j])$ (Line 10). This can be done by employing the following property of such a subinstance. First, $B_i \subseteq Y$ (where Y is the a cluster vertex deletion set of the input graph). Moreover, B_i is a cluster vertex deletion set of $G'[B_i \cup C_j]$ and its removal leaves a graph consisting of a single cluster, namely C_j. Hence, our goal is to find a fixed-parameter algorithm for such subinstances with respect to the parameter $|B_i|$. Formally, we end up with the following problem.

> **Input:** A graph $G = (V, E)$ with $B \uplus C = V$ such that $G[C]$ forms a clique.
> **Task:** Determine $\text{opt}_{CD}(G)$.

In Alg. 3, we present a solving strategy (`ComputeEdgeCost`) for this problem. In a first step, it tries all partitions of B (Line 2) in order to find the partition $A_1, \ldots, A_\ell$ of B generated by an optimal solution for CD. Clearly, the "correct choice" of $A_1, \ldots, A_\ell$ leads to the edge deletions between vertices of different A_i's in Line 4. Moreover, a vertex $v \in C$ can end up in a cluster with an A_i only if it is adjacent to all vertices of A_i. This observation justifies the edge deletions in Line 8. Clearly, if each vertex in C is adjacent either to all or no vertex in an A_i, then all vertices in A_i have the

Function `ComputeEdgeCost(`G,B,C`)`
Input: A graph $G = (V, E)$ and a two-partition B and C of V such that $G - B$ is the complete graph on C.
Output: Size of an optimal solution S for Cluster Deletion for G.

```
1  κ = ∞;
2  forall partitions A_1, ..., A_t of B (1 ≤ t ≤ |B|) do
3      if each A_i is a clique in G then
4          Delete all edges between A_i and A_j, 1 ≤ i < j ≤ t from E;
5          Let m_1 denote the number of these edge modifications;
6          forall v ∈ C and i ∈ {1, ..., t} do
7              if A_i \ N(v) ≠ ∅ then
8                  Delete all edges between v and A_i from E;
9              end
10         end
11         Let m_2 denote the number of edges deleted in Line 8;
12         x=SolveClusterSplit (G, A_1, ..., A_t, C);
13         κ = min(κ, x + m_1 + m_2);
14         undo all edge deletions in G;
15     end
16 end
17 return κ;
```

Algorithm 3: Algorithm to compute the edge cost $\kappa(B, C)$ used in Line 10 of Alg. 2. Herein, procedure "SolveClusterSplit" (Line 12) solves instances of Cluster Split using Lemma 4.7.

same closed neighborhood. Hence, each A_i is a critical clique in the resulting graph. Altogether, we arrive at the task to solve the following subproblem with a further restricted input structure (Line 12 of Alg. 3).

Cluster Split:
Input: A graph $G = (V, E)$ and the vertex sets C and $A_1, \ldots, A_\ell$ such that

- $C \uplus A_1 \uplus \ldots \uplus A_\ell = V$,
- C is a clique and each A_i is part of a critical clique, and
- $G[\bigcup_{i=1}^{\ell} A_i]$ is a cluster graph and $A_1, \ldots, A_\ell$ are the corresponding clusters.

Task: Determine $\text{opt}_{CD}(G)$.

Next, we show that Cluster Split can be solved in $(\ell+1)^{5d} \cdot \text{poly}(|V|)$ time where $d := \sum_{i=1}^{\ell} |A_i|$. The basic idea is as follows. If $|C| < 5d$, then we systematically try all partitions of C. Otherwise, based on the following lemma, we can reduce the instance to at most $\ell + 1$ instances with $|C| < 5d$. The underlying observation is that if $|C| \geq 5d$, then in the cluster graph generated by an optimal solution for Cluster Split, either C is a cluster or for some $1 \leq i \leq \ell$ there is a large cluster consisting of $N[A_i]$ and $N[A_i]$ contains all but at most $2d$ vertices of C.

Lemma 4.6. *For an instance of* Cluster Split *with* $|C| \geq 5d$ *(where* $d = \sum_{i=1}^{\ell} |A_i|$*), there is an optimal solution* S *such that one of the following two statements is true:*

- $G \setminus S$ *contains* C *as a cluster, or*
- $G \setminus S$ *contains a cluster* K *with* $K = N_G[A_i]$ *and* $|K \cap C| \geq |C| - 2d$*, for some* i*,* $1 \leq i \leq \ell$*.*

Proof. According to Lemma 3.7 there is an optimal solution for Cluster Deletion such that every critical clique ends up in a cluster of the corresponding cluster graph. Let S denote a solution such that each A_i is a subset of a cluster of $G \setminus S$. Since $N_G(C) \subseteq \bigcup_{i=1}^{\ell} A_i$ and $|C| \geq 5d > 2\sum |A_i|$, Lemma 4.1 implies that there is a cluster K in $G \setminus S$ with $|K \cap C| \geq |C| - 2d$. Since $|C| \geq 5d$, it holds that $|K \cap C| \geq 3d$. In the following, we show that K fulfills one of the two statements of the lemma.

If $K = C$, then the first statement is true. Hence, we assume that $K \neq C$ in what follows. We show that the assumption $K \neq C$ implies that $K = N[A_i]$ for some i, $1 \leq i \leq \ell$ and, hence, the second statement of the lemma follows.

To this end, we first we show that $K \cap (\bigcup_{i=1}^{\ell} A_i) \neq \emptyset$. Assume toward a contradiction that $K \subseteq C$. Note that $K \neq C$ implies that K is a proper subset of C. Let $x \in C \setminus K$ be arbitrarily chosen and let X denote the cluster of $G \setminus S$ containing x. Since $|K| = |C \cap K| \geq 3d$ one obtains a cluster graph that is closer to G than $G \setminus S$ by "adding" the vertices form X to K. More specifically, consider the cluster graph that results from $G \setminus S$ by deleting all edges between $X \cap C$ and $X \setminus C$ (at most $|X \cap C| \cdot d$) and undoing all edge deletions between $X \cap C$ and K (at least $|X \cap C| \cdot 3d$); a contradiction to the fact that S is optimal.

In summary, K contains at least one vertex from $\bigcup_{i=1}^{\ell} A_i$ and by the choice of S (according to Lemma 3.7) it holds that $A_i \subseteq K$ for some i, $1 \leq i \leq \ell$. Furthermore, $K \subseteq N_G[A_i]$ since only edge deletions are allowed.

It remains to show that $K = N_G[A_i]$. To this end, assume towards a contradiction that there is a vertex $x \in N_G(A_i) \setminus K$. Let X denote the cluster of $G \setminus S$ containing x. Next, we argue that $|X \setminus N_G[A_i]| < 3d$. First, $|X| \setminus |C| \leq \sum_{j \neq i} |A_j| < d$. Second, $|C| \geq 3d$ implies $|X \cap C| \leq 2d$. Finally, we show that one obtains a cluster graph that is closer to G than $G \setminus S$ by modifying $G \setminus S$ such that $N_G[A_i]$ becomes a cluster. More specifically, consider the cluster graph that results from $G \setminus S$ by first deleting all edges between $N_G(A_i) \cap X$ and $X \setminus N_G(A_i)$ (at most $|X \cap N_G(A_i)| \cdot (3d-1)$) and subsequently undoing the edge deletions between $X \cap N_G(A_i)$ and K (at least $|X \cap N_G(A_i)| \cdot 3d$); a contradiction to the optimality of S. □

Lemma 4.6 enables us to reduce instances with $|C| \geq 5d$ to at most $\ell + 1$ instances with $|C| < 5d$.

Lemma 4.7. Cluster Split *can be solved in* $(\ell+1)^{5d} \cdot (|V| + |E|)$ *time, where* $d = \sum_{i=1}^{\ell} |A_i|$*.*

Proof. We distinguish the cases $|C| < 5d$ and $|C| \geq 5d$.

Case: $|C| < 5d$**.** Systematically try all $(\ell+1)^{|C|}$ possibilities to assign every vertex from C a number from $\{1, \ldots, \ell + 1\}$; a number $i \leq \ell$ stands for the possibility that the vertex ends up in a cluster with A_i and the number $\ell + 1$ stands for the possibility that the vertex ends up in a cluster containing only vertices of C. For each choice, in $O(|E| + |V|)$ time, first, delete all edges between vertices with different numbers

and, second, check whether the resulting graph is cluster graph. Finally, report the minimum number of edge deletions over all possibilities. For the correctness, note that since C forms a clique, there is at most one cluster in an optimal solution cluster graph that contains only vertices from C. In summary, an optimal solution for Cluster Split can be found in $O((\ell+1)^{|C|}(|V|+|E|)) \subseteq O((\ell+1)^{5d}(|V|+|E|)$ time.

Case: $|C| \geq 5d$**.** We first branch into the case that C is an isolated clique. Then, for each i, $1 \leq i \leq \ell$ with $|N_G(A_i) \cap C| \geq |C| - 2d$ we branch into the case that $N_G[A_i]$ forms a cluster. In the first branch, the size of the solution is simply the number of edges between C and $V \setminus C$. In each of the other branches, we remove $N_G[A_i]$ and solve the remaining instance. Since in each branch all but at most $2d$ vertices of C are deleted the remaining instance can be solved in $(\ell)^{2d} \cdot (|V|+|E|)$ time analogously to the case that $|C| < 5d$. Finally, we report the minimum-cardinality solution over all branches. The correctness follows directly by Lemma 4.6. Hence, in this case, Cluster Split can be solved in $d \cdot (\ell+1)^{2d} \cdot (|V|+|E|)$ time.

Altogether, Cluster Split can be solved in $O((\ell+1)^{5d}(|V|+|E|))$ time. □

Next, we analyze the running time of `ComputeEdgeCost` (Alg. 3).

Lemma 4.8. *Given a graph $G = (V, E)$ with $B \uplus C = V$ such that $G[C]$ is a clique, procedure* `ComputeEdgeCost` *(Alg. 3) determines $\text{opt}_{CD}(G)$ in $O(|B|^{|B|}(|B|+1)^{5|B|}|V|^2)$ time.*

Proof. The correctness of procedure `ComputeEdgeCost` (Alg. 3) and the running time of the subroutine `SolveClusterSplit` can easily be derived from Lemma 4.7. It remains to analyze the overall running time. Clearly, the edge deletions in Line 4 can be applied in $O(|V|^2)$ time by one iteration over the edge set. To determine the edge deletions in Line 8 in $O(|V|^2)$ time, proceed as follows. First, in one iteration over the edge set, determine for every vertex $v \in V \setminus Y$ the numbers $a_v[i] := |N_G(v) \cap A_i|$ for all $1 \leq i \leq t$. Then, in a second iteration over the edge set delete each edge $\{v, w\} \in E$ with $v \in V \setminus Y$ and $w \in A_i$ if $a_v[i] \neq |A_i|$. Thus, since there are at most $|B|^{|B|}$ partitions of $|B|$, the running time follows directly by Lemma 4.7. □

Putting all together, we arrive at the proof of Theorem 4.2, which states that Cluster Deletion is fixed-parameter tractable with respect to the cluster vertex deletion number.

Proof. (Proof of Theorem 4.2) Let $(G = (V, E), k)$ denote a Cluster Deletion instance. To solve Cluster Deletion for (G, k), first compute a cluster vertex deletion set Y for G in $O(2^c c^9 + |V| \cdot |E|)$-time [109] and, second, apply procedure `CDbyCVD` (Alg. 2) with (G, k) and Y. The correctness follows directly from Lemma 4.5 and Lemma 4.8.

It remains to prove the running time bound. To this end, we analyze the running time of procedure `CDbyCVD` (Alg. 2). In a first step, procedure `CDbyCVD` tries all (at most $|Y|^{|Y|}$) partitions of $|Y|$. Clearly, for each partition the running time is "dominated" by the computations in Lines 9 to 10, where for every $i, 1 \leq i \leq x$ and $j, 1 \leq j \leq z$ the value $\text{opt}_{CD}\, G'[B_i, C_j]$ must be computed. Since $x \leq |Y|$, $z \leq |V|$, and $|B_i| + |C_j| \leq |V|$ according to Lemma 4.8 these computations are doable in $O(|Y| \cdot |V| \cdot |Y|^{|Y|} \cdot (|Y|+1)^{5|Y|} \cdot |V^2|)$ time. Thus, the overall running time of procedure `CDbyCVD` (Alg. 2) is upper-bounded by $O(c^{2c}(c+1)^{5c} c \cdot |V^3|)$, where $c = |Y|$ is the cluster vertex deletion number of G. □

4.3 Further Alternative Parameterizations

In this section, we briefly discuss some further parameterizations for Cluster Editing. To this end, we "deconstruct" known NP-hardness proofs for Cluster Editing. See Section 2.3 and Chapter 8 for details on the "deconstructive approach" for parameter identification.

To the best of our knowledge all known NP-hardness proofs for Cluster Editing require an unbounded vertex degree [125, 47, 150, 12]. Thus, in the sense of "deconstructing intractability", the maximum vertex degree is an interesting parameter. However, we can show that Cluster Editing is NP-hard even when restricted to graphs with maximum degree six. The NP-hardness is established by a reduction from 3-SAT using a similar strategy as Weller et al. [156]. We omit all details.

Theorem 4.3. *[122, Theorem 3]* Cluster Editing *and* Cluster Deletion *are NP-complete even when restricted to graphs with maximum vertex degree six.*

Analyzing Shamir et al.'s [150] NP-hardness proof for Cluster Editing shows the following property of the constructed instances. The NP-hardness proof requires that there are vertices in the constructed instances that are involved in an unbounded number of edge modifications of any optimal solution. This raises the question whether for instances admitting optimal solutions where each vertex is involved in a fixed number of edge modifications a corresponding solution can be found efficiently. However, as a consequence of the NP-hardness of Cluster Editing for graphs with maximum degree six, we can conclude that Cluster Editing is NP-hard restricted to graphs that allow solutions such that every vertex is involved in at most six edge modifications: For every optimal solution, the number of edge modification involving a vertex v is bounded by the degree of v, since otherwise (if v is involved in more than $\deg(v)$ edge modifications) one obtains a better solution by putting v into a new cluster. Hence, as an immediate consequence of Theorem 4.3, we arrive at the following.

Corollary 4.1. Cluster Editing *and* Cluster Deletion *are NP-hard even when restricted to instances for which each vertex is involved in at most six edge modifications of any optimal solution.*

Finally, we mention that although many NP-hardness proofs require an unbounded number of clusters, Cluster Editing remains NP-hard even when it is required that the cluster graph contains exactly two clusters [150].

These intractability results motivate a multivariate approach to Cluster Editing, in particular investigating the complexity of Cluster Editing for combined parameters. We initiate this research direction for Cluster Editing by showing that a constrained version of Cluster Editing is fixed-parameter tractable with respect to the combined parameter "number d of clusters in the target graph" and "maximum number t of modifications per vertex". More precisely, the problem under consideration is a generalization of Cluster Editing and is formalized as follows:

Definition 4.2. (d,t)-Constrained-Cluster Editing ((d,t)-CCE)
Input: An undirected graph $G = (V, E)$, a function $\tau : V \to \{0, \ldots, t\}$, and integers $d \geq 1$ and $k \geq 0$.
Question: Can G be transformed into a cluster graph G' by applying at most k edge modifications such that G' has at most d clusters and each vertex $v \in V$ is incident to at most $\tau(v)$ modified edges?

If only edge deletions are allowed, we refer to the corresponding problem as (d,t)-Constrained-Cluster Deletion. Clearly, Cluster Editing is exactly (d,t)-Constrained-Cluster Editing for $d = n$ and $t = \tau(v) = n$ for all $v \in V$. Next, we discuss several aspects of both the problem formulation and the parameterization by (d,t).

Concerning the problem formulation, in many application scenarios a reasonable upper bound for the number of clusters d is given in advance. Furthermore, constraining the maximum number t of modifications per vertex yields another measure of closeness of the cluster graph to the input graph. In comparison to Cluster Editing, (d,t)-Constrained-Cluster Editing allows to further constrain the solution by adjusting the values of d and t. In certain application scenarios this may help to obtain more reasonable clusterings.

Concerning the parameterization by (d,t), we consider the combined parameter (d,t) since (d,t)-Constrained-Cluster Editing is NP-hard for $t = 6$ (which follows from Theorem 4.3). Moreover, when comparing the parameterizations k and (d,t) one can observe that for some instances, k is not bounded by a function in d and t. Consider for example a graph $G = (V,E)$ that consists of two cliques K_1 and K_2, each of order $|V|/2$. Furthermore, let each $v \in K_1$ have exactly one neighbor in K_2 and vice versa (that is, the edges between K_1 and K_2 form a perfect matching). An optimal solution for CE for this graph is to delete all $|V|/2$ edges between K_1 and K_2. Hence, the parameter k is linear in $|V|$ for such an instance, whereas the number d of clusters is two and each vertex is involved in at most $t = 1$ edge modifications. In general, we can always assume $t \leq k$. The general relation between d and k is a bit more tricky. For example, in case G is connected, we can assume $d \leq k+1$ since we can create at most $k+1$ connected components by applying k edge modifications to G. Furthermore, in case G does not contain isolated cliques, we can assume $d \leq 2k$ since to each clique in the final cluster graph at least one edge modification is incident. In summary, the parameters d and t can be arbitrarily small compared to k and are bounded from above by a linear function of k when G does not contain isolated cliques.

We could show that (d,t)-Constrained-Cluster Editing is fixed-parameter tractable with respect to (d,t) by proving a problem kernel consisting of at most $4dt$ vertices [122, Theorem 4].

Theorem 4.4. (d,t)-Constrained-Cluster Editing *admits a* $4dt$*-vertex problem kernel which can be found in* $O(n^3)$ *time.*

The data reduction rules corresponding to Theorem 4.4 comprise two "high-degree" rules that identify edge modifications that have to be performed by every solution, since otherwise there would be vertices to which more than t edge modifications are incident. Furthermore, two further "clean-up"-rules are needed. We omit any details. Similar data reduction rules can be found in previous work [83] and Section 6.4.

4.4 Conclusion

In this chapter, we have initiated the investigation of Cluster Editing and Cluster Deletion with respect to parameters other than solution size k.

First, we have shown that Cluster Editing and Cluster Deletion are fixed-parameter tractable with respect to the cluster vertex deletion number of the input

graph. So far, due to the large exponential factors these results are only of theoretical interest.

Second, we have discussed further parameterizations. We have seen that Cluster Editing is NP-hard even for graphs with maximum vertex degree six and if every vertex is involved in at most six edge modifications. Motivated by these and further intractability results, we started a "multivariate approach" for Cluster Editing, showing that a generalization of Cluster Editing is fixed-parameter tractable with respect to the combined parameter "number d of clusters in the target graph" and "maximum number t of modifications per vertex".

There are numerous tasks for future research. Improving the running times for our fixed-parameter algorithms for parameter "cluster vertex deletion number" is desirable. We believe that significant running time improvements are achievable. We left open the existence of polynomial-size kernelizations for these two problems. Very recently, Jansen [112] showed that there is little hope for polynomial-size problem kernels for Cluster Deletion parameterized by the "cluster vertex deletion number". For Cluster Editing an analogous result seems plausible but is still open.

The fixed-parameter tractability of (d,t)-Constrained-Cluster Editing (see Definition 4.2) for the combined parameter (d,t) is based on a $4dt$-vertex problem kernel. It would be interesting to have a search tree algorithm complementing this kernelization result.

The NP-hardness of Cluster Editing for instances with maximum degree six directly raises the question whether Cluster Editing is polynomial-time solvable for instances with maximum degree at most five. For instances of maximum degree two, the polynomial-time solvability is easy to see. We conjecture that for maximum degree three Cluster Editing is solvable in polynomial time. Analogously, it is open whether (d,t)-Constrained-Cluster Editing is NP-hard for $t \leq 5$.

Recently, Marx and Razgon [132] established the fixed-parameter tractability of Correlation Clustering on general graphs. To the best of our knowledge the (non)existence of polynomial-size problem kernels for Correlation Clustering is still open.

Recall that a graph G is called (p,q)-cluster graph if its vertex set can be partitioned into subsets with each subset missing at most p edges from being a clique and having at most q edges going to other subsets [107]. Do (p,q)-cluster graphs have a characterization by forbidden induced subgraphs of sizes upper-bounded by a function of p and q? For constant values of p and q such a forbidden subgraph characterization would lead to simple algorithms for recognizing (p,q)-cluster graphs and would imply fixed-parameter tractability of (p,q,k)-Cluster Graph Editing parameterized by the solution size k.

Chapter 5

Clustering With Overlaps

5.1 Introduction

Concerning graph-based data clustering, parameterized complexity investigations focussed mainly on Cluster Editing. Intensive studies have led to impressive theoretical results and experimental studies demonstrating the practical usefulness of fixed-parameter algorithms to solve Cluster Editing (see Chapter 4 for an overview). However, Cluster Editing enforces a sometimes too strict notion of cluster graphs by disallowing any overlap, and has been explicitly criticized for this lack of overlaps [57].

In a nutshell, in this chapter we introduce generalizations of Cluster Editing to allow for overlapping clusterings. Moreover, we investigate both the classical and parameterized complexity of the introduced problems.

In Cluster Editing the task is to transform a graph into a cluster graph, that is, a disjoint union of cliques. Numerous recent publications build on this concept of cluster graphs (see Chapter 4 for more details). In some applications, however, a data item may belong to several clusters. This is obvious for members of social networks. Moreover, genes may be involved in several functional groups. To uncover the *overlapping* community structure of complex networks in nature and society [141], however, the concept of cluster graphs fails to model that clusters may overlap, which may lead to artificial and, hence, meaningless classifications. In this chapter, we introduce a graph-theoretic relaxation of the concept of cluster graphs by allowing, to a certain extent, overlaps between the clusters (which are cliques). We distinguish between "vertex overlaps" and "edge overlaps" and provide a thorough study of the corresponding cluster graph modification problems. In this introductory section, we formally define generalizations of Cluster Editing allowing for overlapping clusterings. To this end, we consider edge modification problems (see Section 3.3 and, in particular, Definition 3.1) where the desired graph properties ensure that each vertex or edge is contained in at most s maximal cliques, respectively. Formally, for a fixed positive integer $s \geq 1$, the desired graph properties are defined as follows.

Definition 5.1 (s-vertex-overlap property and s-edge-overlap property)**.** A graph $G = (V, E)$ has the *s-vertex-overlap property* (or *s-edge-overlap property*) if every vertex

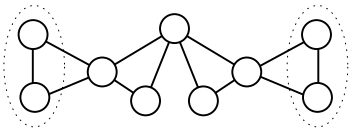


Figure 5.1: The shown example graph contains four maximal cliques, each of size three. Moreover, each vertex is contained in at most two maximal cliques and each edge is contained in one maximal clique. Hence, the shown graph has the 2-vertex-overlap property and the 1-edge-overlap property. The critical clique graph of the shown graph contains eight edges: the two critical cliques of size two are encircled by dotted lines and all other vertices form critical cliques of size one.

(or edge) of G is contained in at most s maximal cliques.

Furthermore, we call a graph with s-vertex-overlap property an *s-vertex-overlap cluster graph* and analogously, a graph with s-edge-overlap property an *s-edge-overlap cluster graph*.

Clearly, a 1-vertex-overlap cluster graph consists of a vertex-disjoint union of cliques. That is, 1-vertex-overlap cluster graphs are cluster graphs in the original sense (see Chapter 4) and there is no overlap between the clusters. For larger values of s, the overlap between the clusters increases. See Figure 5.1 for a graph fulfilling the 2-vertex-overlap property and the 1-edge-overlap property. Furthermore, note that, since if each vertex is contained in at most s maximal cliques then each edge is contained in at most s maximal cliques, the class of s-vertex-overlap cluster graphs is contained in the class of s-edge-overlap cluster graphs. Moreover, observe that a star (a tree of diameter two) on n vertices has the 1-edge-overlap property, however, its inner vertex is contained in $n-1$ maximal cliques. Thus, bounded vertex overlap imposes a higher restriction on the cluster graph than bounded edge overlap.

For the s-vertex-overlap property the respective edge modification problems, namely to transform a graph by at most k edge modifications into an s-vertex-overlap cluster graph, is called s-Vertex-Overlap Editing.

Definition 5.2. s-Vertex-Overlap Editing (s-VOE)
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Can G be transformed into an s-vertex-overlap cluster graph by applying at most k edge modifications?

Analogously, the deletion and addition versions for the s-vertex-overlap property are called s-Vertex-Overlap Deletion (s-VOD) and s-Vertex-Overlap Addition (s-VOA), respectively. Moreover, in the case of edge overlap, we refer to the respective problems as s-Edge-Overlap Editing (s-EOE), s-Edge-Overlap Deletion (s-EOD), and s-Edge-Overlap Addition (s-EOA). By the discussion above, 1-Vertex-Overlap Editing is the same as Cluster Editing.

Previous Work. To the best of our knowledge, relaxed versions of Cluster Editing and the cluster graph concept that allow for overlapping clusterings have been largely unexplored.[1] There are only two approaches studying overlapping cliques in

[1]See Chapter 4 for an overview of recent work dealing with relaxed versions of cluster graphs, as for example so-called s-plex cluster graphs [96] and (p, q)-cluster graphs [107]. Note that these

the context of Cluster Editing that we are aware of. The motivation of these works also stems from the observation that in some application scenarios the restriction to non-overlapping (that is, disjoint) clusters may lead to artificial and, hence, useless classifications.

One approach was proposed by Barthélemy and Brucker [13] under the name t-Zahn Clustering, where the aim is to obtain by a minimum number of edge modifications a graph in which each pair of maximal cliques has at most $t-1$ vertices in common. The base case $t = 1$ is thus equivalent to Cluster Editing. Moreover, for $t = 2$ any two maximal cliques are allowed to overlap in at most one vertex, implying that each edge is contained in at most one maximal clique. Hence, 2-Zahn Clustering is the same as 1-Edge-Overlap Editing. Among other things, Barthélemy and Brucker [13] showed that 2-Zahn Clustering is NP-hard. The model of Barthélemy and Brucker [13] allows, for constant $t \geq 3$, for vertices and edges to be in an unbounded number of maximal cliques. In contrast, our model limits the number of maximal cliques that a vertex or edge is contained in, but already for constant s there can be maximal cliques that intersect in an unbounded number of vertices.

The second approach was presented by Damaschke [54] who investigated the Twin Graph Editing problem. Here the goal is to obtain, by a minimum number k of edge modifications, a critical clique graph with at most t edges, where t is specified as part of the input. Recall that the critical clique graph is the representation of a graph obtained by keeping for each set of vertices with identical closed neighborhoods exactly one vertex as representative, see Section 2.6. Two vertices with identical closed neighborhoods are also known as *real twins* and the notion "twin graph" is a synonym for "critical clique graph". In Figure 5.1, a graph is shown whose corresponding critical clique graph has eight edges. Clearly, the twin graph of a cluster graph does not contain any edges. Hence, Twin Graph Editing with $t = 0$ is the same as Cluster Editing. Observe that in the model of Damaschke the overlap between the clusters is measured by the total number of edges in the critical clique graph. In our model, the overlap is measured by the number of maximal cliques involving a vertex or edge. Thus, our model expresses a more local property of the target graph. The main result of Damaschke [54] concerning Twin Graph Editing is fixed-parameter tractability with respect to the combined parameter (t, k). Already for $s = 2$ our s-vertex-overlap model includes graphs whose twin graphs can have an unbounded number t of edges. Hence, s is not a function of t, implying that our model is not subsumed by the model of Damaschke.

Our Results. We provide a thorough study of the computational complexity of clustering with vertex and edge overlaps, extending previous work on Cluster Editing and closely related problems. In particular, in terms of the overlap number s, we provide a complete complexity dichotomy (polynomial-time solvable versus NP-hard) of the corresponding edge modification problems, most of them turning out to be NP-hard (for an overview, see Table 5.1 in Section 5.3). For instance, whereas Cluster Editing restricted to only allowing edge additions (also known as Cluster Addition or 1-Vertex-Overlap Addition) is trivially solvable in polynomial time, 2-Vertex-Overlap Addition turns out to be NP-hard. We also study the parameterized complexity of clustering with overlaps. On the negative side, we show

relaxations do not allow for overlaps between the clusters.

W[1]-hardness results with respect to the parameter "number of edge modifications" in case of unbounded overlap number s. On the positive side, we prove that the problems become fixed-parameter tractable for the combined parameter (s, k). This result is based on forbidden subgraph characterizations of the underlying overlap cluster graphs, which may be of independent graph-theoretic interest. In particular, it turns out that the 1-edge-overlap cluster graphs are exactly the diamond-free graphs[2]. Finally, we develop polynomial-time data reduction rules for two special cases. More precisely, we show an $O(k^4)$-vertex problem kernel for 1-Edge-Overlap Deletion and an $O(k^3)$-vertex problem kernel for 2-Vertex-Overlap Deletion, where in both cases k denotes the number of allowed edge deletions. We conclude in Section 5.6 with a number of open problems.

See Section 2.6 for basic notation concerning graphs and Section 3.3.1 for basic notation concerning edge modification problems.

5.2 Recognition and Forbidden Subgraph Characterization

In this section, we show that, for each fixed s, it can be recognized in polynomial time whether a given graph has the respective overlap property. Moreover, we will show that the graph properties, for each fixed s, are characterized by a finite set of forbidden induced subgraphs. More specifically, we show that the forbidden graphs are all of order s^2 and that there is a polynomial-time algorithm that, given a graph, either determines that G fulfills the property or identifies an induced subgraph of G that is forbidden.

For a graph G and a non-negative integer s, we can decide in polynomial time whether G fulfills the s-vertex-overlap property using a clique enumeration algorithm with polynomial delay.

Theorem 5.1. *For a graph $G = (V, E)$ and a non-negative integer s, there is an algorithm that, in $O(s \cdot n^{3.376})$ (or $O(s \cdot m \cdot n^{2.376})$) time, either*

- *finds a vertex (or an edge) that is contained in more than s maximal cliques, or*
- *correctly concludes that G has the s-vertex-overlap (or s-edge-overlap) property.*

Proof. For each $v \in V$, we enumerate the maximal cliques in $G[N[v]]$. If we have found $s + 1$ maximal cliques in $G[N[v]]$ for some $v \in V$, then we abort the enumeration and report that v is in more than s maximal cliques. Otherwise, each $v \in V$ is contained in at most s maximal cliques, and the graph thus fulfills the s-vertex-overlap property. Using for example a polynomial-delay enumeration algorithm by Makino and Uno [131] that relies on matrix multiplication and enumerates cliques with delay $O(n^{2.376})$, the overall running time of this algorithm is $O(s \cdot n^{3.376})$.

For the edge case a similar approach applies; the only difference is that we consider the common neighborhood of the endpoints of every edge, that is, $N[u] \cap N[v]$ for an edge $\{u, v\}$. □

[2]A diamond is the graph on four vertices that contains all but one of the six possible edges. An illustration is given in Figure 5.6 on page 69.

The next lemma implies the existence of forbidden induced subgraph characterizations for graphs having the s-vertex-overlap or the s-edge-overlap property (see Definition 3.2 for the definition of hereditary graph properties).

Lemma 5.1. *The s-vertex-overlap property and the s-edge-overlap property are hereditary.*

Proof. To show that the s-vertex-overlap property is hereditary, it suffices to show the following.

> *Claim:* If $G = (V, E)$ has the s-vertex-overlap property, then so does $G - v$ for any $v \in V$.

Assume that G has the s-vertex-overlap property but there exists a vertex $v \in V$ such that $G - v$ does not have the s-vertex-overlap property. Then there exists a vertex $w \in N_G(v)$ contained in at least $s + 1$ distinct maximal cliques of $G - v$, say $C_1, \ldots, C_{s+1}$. For every $1 \leq i \leq s + 1$, there exists a maximal clique K_i of G with $C_i \subseteq K_i$. Moreover, since in G there are at most s maximal cliques containing w, there exist i and j, $1 \leq i < j \leq s + 1$, such that $K_i = K_j$. However, since C_i and C_j are two distinct maximal cliques of $G - v$, there exist vertices $u_i \in C_i$ and $u_j \in C_j$ such that $\{u_i, u_j\} \notin E$, a contradiction to the fact that K_i is a clique containing u_i and u_j.

In complete analogy one shows that the s-edge-overlap property is hereditary, replacing the vertex $w \in N_G(v)$ with an edge $\{u, w\} \subseteq N_G(v)$ in the argument above. □

Hereditary graph properties can be characterized by a finite or infinite set of forbidden induced subgraphs (see Section 3.3.1). Thus, by Lemma 5.1, such a set must exist for s-vertex-overlap cluster graphs as well as for s-edge-overlap cluster graphs. Here, we show that the minimal forbidden induced subgraphs contain $O(s^2)$ vertices. For fixed s, the number of minimal forbidden induced subgraphs is thus finite. Furthermore, we describe an algorithm for efficiently finding a forbidden induced subgraph.

Theorem 5.2. *For a graph G that violates the s-vertex-overlap (or s-edge-overlap) property, one can find in $O(s \cdot n^{3.376} + s^2 \cdot n)$ (or $O(s \cdot m \cdot n^{2.376} + s^2 \cdot n)$) time an $O(s^2)$-vertex forbidden induced subgraph.*

Proof. We first show the vertex case. Let G be a graph violating the s-vertex-overlap property and let v be a vertex that is contained in more than s maximal cliques. From Theorem 5.1 it follows that such a vertex can be found in $O(s \cdot n^{3.376})$ time. Given $s+1$ maximal cliques $K_1, \ldots, K_{s+1}$ containing v, we find a forbidden induced subgraph as follows. To "separate" two maximal cliques K_i and K_j, we need a vertex $v_1 \in K_i \setminus K_j$ and a vertex $v_2 \in K_j \setminus K_i$ with $\{v_1, v_2\} \notin E$. Clearly, such vertices exist since both K_i and K_j are maximal. To "separate"every pair of $s + 1$ maximal cliques we need at most $2\binom{s+1}{2}$ vertices. These vertices and v together induce a subgraph of order at most $(s + 1) \cdot s + 1$ and v is contained in at least $s + 1$ maximal cliques in this graph. For each pair of cliques, we can find the separating vertices in $O(n)$ time, scanning the vertex-lists of each clique, "marking" the vertices that are contained in both cliques, and then keeping one unmarked vertex of each list. Altogether, we thus need $O(s^2 \cdot n)$ time.

For the edge case, we can find in $O(s \cdot m \cdot n^{2.376})$ time an edge $\{u, v\}$ that is contained in at least $s+1$ maximal cliques. The vertices needed to "separate" the $s+1$ maximal cliques $K_1, \ldots, K_{s+1}$ in $G[N[v] \cap N[u]]$ can be found analogously to the vertex case. □

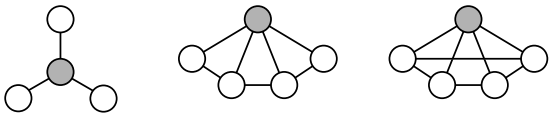


Figure 5.2: The forbidden induced subgraphs for the 2-vertex-overlap property. In every graph, the gray vertex is contained in at least three maximal cliques.

Table 5.1: Classical computational complexity of graph-based data clustering with overlaps. Herein, "NPh" means that the respective problem is NP-hard and "P" means that the problem can be solved in polynomial time.

	s-vertex-overlap	s-edge-overlap
Editing	NPh for $s \geq 1$	NPh for $s \geq 1$
Deletion	NPh for $s \geq 1$	NPh for $s \geq 1$
Addition	P for $s = 1$, NPh for $s \geq 2$	P for $s = 1$, NPh for $s \geq 2$

Figure 5.2 illustrates the minimal forbidden induced subgraphs for graphs with the 2-vertex-overlap property. Many important graph classes are contained in the class of graphs with some s-overlap property. For example, it is easy to see that diamond-free graphs are equivalent to graphs with the 1-edge-overlap property. A *diamond* is the graph that results from a four-vertex clique by deleting one edge. Diamond-free graphs, that is, graphs containing no diamond as an induced subgraph, are a natural graph class and have been already studied in earlier work [13, 153].

Proposition 5.1. *A graph has the 1-edge-overlap property if and only if it is diamond-free.*

Proof. Clearly, a diamond does not satisfy the 1-edge-overlap property. Thus, every 1-edge-overlap cluster graph is diamond-free. Moreover, if a graph does not have the 1-edge-overlap property, then there must be an edge contained in at least two maximal cliques. Hence, there is a pair of non-adjacent vertices that are both adjacent to the endpoints of this edge. Therefore, the graph contains at least one induced diamond. □

The property of being diamond-free can also be described as follows: every pair of maximal cliques has at most one vertex in common. Graphs with the 1-edge-overlap property are thus precisely the graphs with 2-Zahn property as defined by Barthélemy and Brucker [13].

5.3 A Complexity Dichotomy with Respect to the Overlap Number s

This section provides a complete picture of the classical computational complexity of the introduced problems. The results are summarized in Table 5.1. With the exception of the two basic addition problems for $s = 1$, all of the problems turn out to be NP-hard.

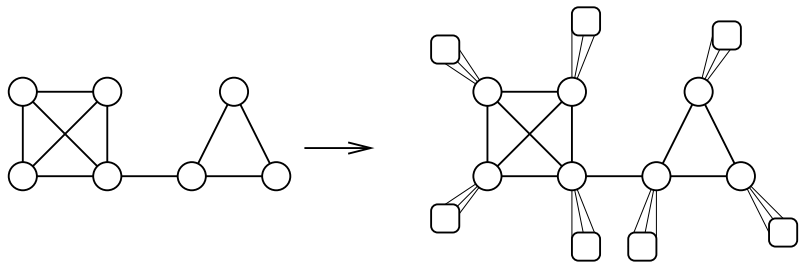


Figure 5.3: Illustration of the reduction from s-Vertex-Overlap Editing to $(s+1)$-Vertex-Overlap Editing. Herein, every rectangular vertex represents a clique on $2k+2$ vertices.

First, we show that if one of the problems is NP-hard for some $s \geq 1$, then it is NP-hard for every $s' \geq s$. The basic idea is that, given a problem instance for some value s, we can reduce to an instance for $s+1$ by adding for every vertex v for vertex overlap (respectively for every pair of distinct vertices u and v for edge overlap) one "large" clique that intersects with the original instance only in v (respectively u and v).

Lemma 5.2. *For $s \geq 1$, there is a polynomial-time many-one reduction from s-*Property Operation *to $(s+1)$-*Property Operation*, where* Property $\in$ {Vertex-Overlap, Edge-Overlap} *and* Operation $\in$ {Editing, Deletion, Addition}.

Proof. First, we focus on the case of vertex overlap. We describe the reduction from s-Vertex-Overlap Editing (s-VOE) to $(s+1)$-Vertex-Overlap Editing ($(s+1)$-VOE). Moreover, we will observe that the same construction yields a reduction for the deletion and addition variants as well. Second, we will show that the reduction can be adapted to the case of edge overlap.

The reduction from s-VOE to $(s+1)$-VOE works as follows. Given an s-VOE instance $G = (V, E)$ and an integer k, we construct an $(s+1)$-VOE instance consisting of a graph $H = (U, F)$ and an integer $k' := k$. For the construction of H, initially, we set $H := G$. Then, for every vertex $v \in V$, we add a set C_v of $2k+2$ new vertices to H and we make $\{v\} \cup C_v$ a clique. An illustration of this construction is given in Figure 5.3.

Next, we show the correctness of the reduction, that is, we show that G has a solution of size at most k for S-VOE if and only if H has a solution of size at most k for $(s+1)$-VOE. First, consider a solution S of size at most k for s-VOE with input graph G. In the graph that results from modifying G according to S, every vertex is contained in at most s maximal cliques. Hence, if we modify H according to S, we obtain a graph in which every vertex is contained in at most $s+1$ maximal cliques. Second, let S' denote a solution of size at most k for $(s+1)$-VOE for H. Moreover, let $H' = H \Delta S'$. Since $|S| \leq k$, there are at most $2k$ vertices that are affected by S. Hence, in H' every vertex $v \in V$ is adjacent to a non-empty set $N_v \subseteq C_v$ of non-affected vertices (since $|C_v| = 2k+2$). This implies that for every vertex $v \in V$ there exists one maximal clique C'_v containing v with $N_v \subseteq C'_v \subseteq C_v$. Consequently, every vertex $v \in V$ can be contained in at most s further maximal cliques, and, hence, v

can be contained in at most s maximal cliques in the induced subgraph $H'[V]$. That is, $H'[V]$ fulfills the s-vertex-overlap property and $S := S' \cap \mathcal{P}_2(V)$ is a solution for s-VOE for G (recall that for a set X of vertices $\mathcal{P}_2(X)$ denotes the set of all possible edges on X, see Section 2.6).

It is straightforward to verify that the given construction constitutes a reduction from s-Vertex-Overlap Deletion and s-Vertex-Overlap Addition to $(s+1)$-Vertex-Overlap Deletion and $(s+1)$-Vertex-Overlap Addition, respectively. Moreover, it is not hard to verify that adding for every pair of distinct vertices u and v (instead of every vertex) a clique $C_{u,v}$ with $2k+2$ vertices, which intersects with the original s-Edge-Overlap Operation instance only in u and v, yields a polynomial-time many-one reduction for the edge case. The correctness proof works in complete analogy. □

The following NP-hardness results can be obtained directly from combining known results with Lemma 5.2: Since Cluster Editing and Cluster Deletion (equivalent to 1-Vertex-Overlap Editing and 1-Vertex-Overlap Deletion, respectively) are NP-complete [125, 150], the NP-hardness of s-Vertex-Overlap Editing and s-Vertex-Overlap Deletion for all $s > 1$ directly follows. Furthermore, 1-Edge-Overlap Editing has also been shown to be NP-complete by a reduction from Cluster Editing [13] that can also be used to show the NP-hardness of 1-Edge-Overlap Deletion (simply by reducing from Cluster Deletion instead). The NP-hardness for both the editing and the deletion variant and $s > 1$ thus also follows for edge overlap. Overall, we arrive at the following theorem.

Theorem 5.3. *s-Vertex-Overlap Editing, s-Vertex-Overlap Deletion, s-Edge-Overlap Editing, and s-Edge-Overlap Deletion are NP-hard for $s \geq 1$.*

It thus remains to determine the classical computational complexity of s-Vertex-Overlap Addition and s-Edge-Overlap Addition. 1-Vertex-Overlap Addition is trivially polynomial-time solvable: one has to transform every connected component into a clique by adding the missing edges. The same observation can be made for 1-Edge-Overlap Addition, since there exists only one possibility to destroy a diamond by adding edges; by Proposition 5.1, diamonds are the only forbidden subgraph of graphs having the 1-edge-overlap property.

In contrast, for $s \geq 2$, both s-Vertex-Overlap Addition and s-Edge-Overlap Addition become NP-hard, as we will show in the following.

Theorem 5.4. *s-Vertex-Overlap Addition is NP-hard for $s \geq 2$.*

Proof. We present a polynomial-time many-one reduction from the NP-hard Maximum Edge Biclique problem [144] to 2-Vertex-Overlap Addition (2-VOA). Then, for $s \geq 2$, the NP-hardness follows directly from Lemma 5.2. The decision version of Maximum Edge Biclique is defined as follows: Given a bipartite graph $H = (U, W, F)$ and an integer $l \geq 0$, does H contain a biclique with at least l edges? A biclique is a bipartite graph with all possible edges.

The reduction from Maximum Edge Biclique to 2-VOA works as follows: For a bipartite graph $H = (U, W, F)$, we construct a graph $G = (V, E)$, where $V := U \cup W \cup \{r\}$ and $E := E_{\overline{F}} \cup E_r \cup E_U \cup E_W$. Herein,

- $E_{\overline{F}} := \{\{u, w\} \mid u \in U, w \in W\} \setminus F$,

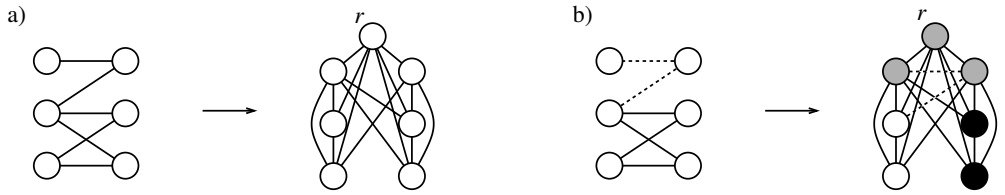


Figure 5.4: a) Example for the reduction from Maximum Edge Biclique (left graph) to 2-Vertex-Overlap Addition (right graph), b) The graph on the left contains a biclique with four edges (solid edges). Adding the edges not contained in this biclique (dashed edges) to the graph on the right results in a graph that contains two maximal cliques. The gray vertices are in both maximal cliques, the white and black vertices are in one maximal clique.

- $E_r := \{\{r, x\} \mid x \in U \cup W\}$, and
- $E_X := \{\{x, x'\} \mid x, x' \in X, x' \neq x\}$ for $X \in \{U, W\}$.

That is, the graph $(U, W, E_{\overline{F}})$ is the bipartite complement of H, in G both U and W are cliques, and r is adjacent to all vertices in G. See Figure 5.4 a) for an illustration of this construction.

For the correctness of the reduction, we show the following.

> *Claim:* In the graph H there is a biclique with at least l edges if and only if there exists a solution S with $|S| \leq |F| - l$ for 2-Vertex-Overlap Addition for G.

"⇒": Assume that H contains a biclique with at least l edges. Let $U' \subseteq U$ and $W' \subseteq W$ denote the vertices in such a biclique. Furthermore, let F' denote the edges not contained in this biclique. That is, the removal of F' from H results in a graph that consists of the disjoint union of isolated vertices and one complete bipartite graph with at least l edges. Moreover, $|F'| \leq |F| - l$. Let G' denote the graph that results from adding the edges in F' to G.

Now, we argue that G' fulfills the 2-vertex-overlap property, and, hence, $S := F'$ is a solution for 2-VOA for G. To this end, observe that in G' any two vertices $u, u' \in U'$ have the same closed neighborhood. The same is true for any two vertices in W', $U \setminus U'$, and $W \setminus W'$, respectively. With this observation, it follows that in G' there are two maximal cliques, namely the clique $U \cup (W \setminus W') \cup \{r\}$ and the clique $W \cup (U \setminus U') \cup \{r\}$. Hence, every vertex in G' is contained in at most two maximal cliques. See Figure 5.4 b) for an example.

"⇐": Assume that there exists a solution S with $|S| \leq |F| - l$ for 2-VOA for G. Moreover, let G' denote the graph that results from adding the edges in S to G. First, note that, since S contains only edges not contained in G, all edges in S are between U and W, and, hence, $S \subseteq F$. We show that the graph H' that results from deleting the edges in S from H consists of isolated vertices and a complete bipartite graph with at least l edges. Assume towards a contradiction that H' is not of the claimed form. Then, either H' contains a connected component with more than one vertex that is not a biclique, or H' contains at least two connected components with more than one vertex. We distinguish both cases, and, in each case, derive a contradiction.

First, assume that H' contains a connected component with more than one vertex that is not a biclique. In this case, H' contains an induced P_4, an induced path on four vertices. Without loss of generality, we can assume that the first and the third vertex, say u and u', are from U and the second and fourth vertex, say w and w', are from W. Since $G'[U \cup W]$ is the (non-bipartite) complement graph of H', in G' we have an induced P_4 containing the edges $\{u', u\}, \{u, w'\}$, and $\{w', w\}$. Since r is adjacent to all vertices, this implies that $G'[\{u, u', w, w', r\}]$ is isomorphic to the second graph shown in Figure 5.2, a contradiction to the fact that G' fulfills the 2-vertex-overlap property.

Second, assume that H' contains at least two connected components with more than one vertex. Let $e = \{u, w\}$ and $e' = \{u', w'\}$ with $u, u' \in U$ and $w, w' \in W$ be two edges from different connected components of H'. This implies that $G'[\{u, u', w, w'\}]$ is an induced cycle of length four. Furthermore, since r is adjacent to all vertices, $G'[\{u, u', w, w', r\}]$ is isomorphic to the third graph shown in Figure 5.2, a contradiction to the fact that G' fulfills the 2-vertex-overlap property. □

Finally, we consider s-Edge-Overlap Addition. The reduction given in the proof of Theorem 5.4 can be easily modified to show the NP-hardness of 2-Edge-Overlap Addition: Simply replace the introduced vertex r by an edge e and connect both endpoints of e to all vertices in the given bipartite graph of the Maximum Edge Biclique instance. The correspondence between the solutions of both instances can be shown in complete analogy with the vertex overlap case.

Theorem 5.5. *s-Edge-Overlap Addition is NP-hard for $s \geq 2$.*

5.4 Parameterized Complexity

Here, we consider the parameterized complexity of overlap clustering. First, due to Theorem 5.2, we have a set of forbidden subgraphs for both properties whose size only depends on s. Cai [36] showed that edge modification problems for properties that can be described by forbidden subgraphs of fixed size are fixed-parameter tractable with respect to the parameter "solution size". Hence, we can conclude that for both overlap properties all three problems are fixed-parameter tractable with respect to the combined parameter (s, k).

Theorem 5.6. *For $\Pi \in \{s$-Vertex-Overlap, s-Edge-Overlap$\}$, Π-Editing, Π-Addition, and Π-Deletion are fixed-parameter tractable with respect to the combined parameter (s, k).*

A naive estimation gives a running time bound of $s^{4k} \cdot n^{O(1)}$ for the corresponding search tree algorithms that branch into all cases (at most s^4) to destroy an induced forbidden subgraph with $O(s^2)$ vertices by deleting or inserting edges and then recursively solve the created subinstances. Better branching strategies are conceivable, in particular for small constant values for s for which all forbidden subgraphs can be easily listed.

Next, we consider the parameterization with only k as the parameter. This means that s can have an unbounded value. For this parameterization, we show that for both overlap properties the deletion and editing problems are W[1]-hard by developing a parameterized reduction from the W[1]-complete Set Packing problem [63]. We

leave open the parameterized complexity of the two addition problems with only k as parameter.

Theorem 5.7. *For* $\Pi \in \{s\text{-Vertex-Overlap}, s\text{-Edge-Overlap}\}$, Π-Editing *and* Π-Deletion *are W[1]-hard with respect to the parameter* k.

Proof. We give the proof details only for s-Vertex-Overlap Deletion (s-VOD), and then discuss how the reduction can be modified to work for s-Vertex-Overlap Editing and edge overlap. We show the W[1]-hardness of s-VOD by presenting a parameterized reduction from the W[1]-complete Set Packing problem [63], which is defined as follows:

> Input: A family of sets $\mathcal{S} = \{S_1, \ldots, S_n\}$ over a universe $U = \{1, \ldots, m\}$ and a nonnegative integer $k \leq n$.
> Question: Is there a set $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \geq k$ and $\forall S_i, S_j \in \mathcal{S}' : S_i \cap S_j = \emptyset$?

Consider an instance $I = (\mathcal{S}, k)$ of Set Packing. Without loss of generality we can assume that $k < m$ and $k < n$. We construct an s-VOD instance $(G = (V, E), k)$ as follows. The vertex set V comprises six subsets V_U, V_S, V_X, V_Y, V_C, and V_P:

- $V_U := \{u_1, \ldots, u_m\}$ contains one vertex for each element $i \in U$.
- $V_S := \{s_1, \ldots, s_n\}$ contains one vertex for each $S_i \in \mathcal{S}$.
- $V_X := \{x_1, \ldots, x_k\}$ contains k vertices and $V_Y := \{y_1, \ldots, y_{2k+1}\}$ contains $2k + 1$ vertices; together they serve as a "selection" gadget.
- V_C contains vertices that are part of some "shielding" cliques. With these cliques, we can enforce that some edges will never be edited.
- V_P contains "padding" cliques that are used to increase the number of maximal cliques for certain vertices.

First, we describe the construction of the graph $G[V_U \cup V_S \cup V_X \cup V_Y]$, then we describe how the additional cliques are added to this graph. For a vertex $s_i \in V_S$ corresponding to set S_i and a vertex $u_j \in V_U$ corresponding to an element $j \in U$, we add the edge $\{u_j, s_i\}$ if $j \in S_i$. Furthermore, we connect each $x_i \in V_X$ by edges to all vertices in $V_U \cup V_S$. Finally, we make V_Y a clique and connect each $y \in V_Y$ to all vertices in $V_X \cup V_S$. This concludes the construction of $G[V_U \cup V_S \cup V_X \cup V_Y]$. An example is shown in Figure 5.5a.

Next, the shielding cliques are added. For each $x_i \in V_X$ we add the vertex set $C^{x_i} := \{c_1^{x_i}, \ldots, c_{k+1}^{x_i}\}$ to V_C. Furthermore, we make $C^{x_i} \cup \{x_i\} \cup V_Y$ a clique. This construction ensures that deleting the edge $\{x_i, s_j\}$ for a vertex $x_i \in V_X$ and $s_j \in V_S$ decreases the number of maximal cliques that contain x_i by one: the maximal clique $K = \{x_i, s_j\} \cup V_Y$ is destroyed and the clique $K' = \{x_i\} \cup V_Y$ (which after deleting $\{x_i, s_j\}$ is the maximal subset of K that is a clique) is a subset of the clique $\{x_i\} \cup V_Y \cup C^{x_i}$. Furthermore, no additional maximal cliques are created by the deletion of $\{x_i, s_j\}$.

Then, for each edge $\{u_i, s_j\}$, and for each $x_l \in V_x$, we add the vertex set $C^{u_i,s_j,x_l} := \{c_1^{u_i,s_j,x_l}, \ldots, c_{k+1}^{u_i,s_j,x_l}\}$ to V_C and we make $\{u_i, s_j, x_l\} \cup C^{u_i,s_j,x_l}$ a clique. This clique

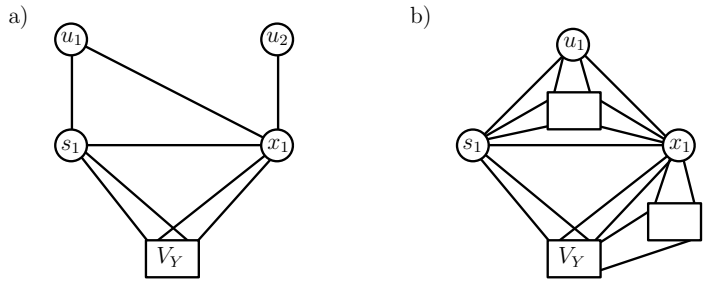


Figure 5.5: Parts of the graph constructed in the reduction from Set Packing to s-Vertex-Overlap Deletion. Rectangles depict cliques of size at least $k+1$. a) A subgraph containing $u_1, u_2 \in V_U$, $s_1 \in V_S$, $x_1 \in V_X$, and V_Y. Edges are drawn between $s_i \in V_S$ and $u_j \in V_U$ if $j \in S_i$. Here, $1 \in S_1$ but $2 \notin S_1$. b) Shielding cliques are added for each triangle in $G[V_U \cup V_X \cup V_S]$ and between x_i and V_Y for all $x_i \in V_X$.

has the following purpose: if we delete an edge $\{s_j, x_l\}$, then we increase the number of maximal cliques that contain u_i. Altogether, the shielding cliques ensure that in order to decrease the number of maximal cliques for a vertex $x_i \in V_X$ with at most k edge deletions, one can only delete edges between x_i and V_S. An example of these shielding cliques is shown in Figure 5.5b.

Before we describe how the padding cliques are added, we compute the number of maximal cliques in $G[V_U \cup V_S \cup V_X \cup V_Y \cup V_C]$ that each vertex $v \in V_U \cup V_X$ is contained in. We denote this number for some vertex v by $\#(v)$.

- Each vertex $u_i \in V_U$ is contained in $\#(u_i) = |N(u_i) \cap V_S| \cdot k \leq n \cdot k$ maximal cliques: For each $s_j \in N(u_i) \cap V_S$ and each $x_l \in V_X$ the set $\{u_i, s_j, x_l\} \cup C^{u_i, s_j, x_l}$ is a maximal clique since V_X and V_S are independent sets and by the definition of V_C; no other maximal cliques contain u_i.

- Each vertex $x_i \in V_X$ is contained in

$$\#(x_i) = \sum_{s_j \in V_S} (|N(s_j) \cap V_U| + 1) + 1 \leq n \cdot (m+1) + 1$$

  maximal cliques: For each $s_j \in V_S$ and for each $u_l \in N(s_j) \cap V_U$ the set $\{x_i, s_j, u_l\} \cup C^{u_l, s_j, x_i}$ is a maximal clique, for each $s_j \in V_S$ the set $\{x_i, s_j\} \cup V_Y$ is a maximal clique, and $\{x_i\} \cup V_Y \cup C^{x_i}$ is a maximal clique; no other maximal cliques contain x_i.

We add padding cliques of size $k+1$ that are "attached" to the vertices in V_U and V_X as follows. For each $u_i \in V_U$, we add $n \cdot (m+1) - 1 - \#(u_i)$ size-$(k+1)$ vertex sets $C_l^{u_i}$ to V_P, where $1 \leq l \leq n \cdot (m+1) - 1 - \#(u_i)$. For each $C_l^{u_i}$, we make $\{u_i\} \cup C_l^{u_i}$ a clique. Then, for each $x_i \in V_X$, we add $n \cdot (m+1) + 1 - \#(x_i)$ size-$(k+1)$ vertex sets $C_l^{x_i}$ to V_P, where $1 \leq l \leq n \cdot (m+1) + 1 - \#(x_i)$. Again, for each $C_l^{x_i}$, we make $\{x_i\} \cup C_l^{x_i}$ a clique. Note that since $k < m$ and $k < n$, the number of added cliques is nonnegative.

This concludes the construction of G. Note that in G

- each vertex $u_i \in V_U$ is contained in exactly $n \cdot (m+1) - 1$ maximal cliques (by the definition of $\#(u_i)$ and the number of added padding cliques),

- each vertex $x_i \in V_X$ is contained in exactly $n \cdot (m+1)+1$ maximal cliques (by the definition of $\#(x_i)$ and the number of added padding cliques),
- each vertex $y_i \in V_Y$ is contained in exactly $n \cdot k + k < n \cdot (m+1)$ maximal cliques (one maximal clique for each pair of $x_j \in V_X$ and $s_l \in V_S$, and one maximal clique for each $\{x_j\} \cup C^{x_j}$, $x_j \in V_X$),
- each vertex $s_i \in V_S$ is contained in exactly $k \cdot (|N(s_i) \cap V_U| + 1) < n \cdot (m+1)$ maximal cliques (one maximal clique for each pair of $x_j \in V_X$ and $u_l \in N(s_i) \cap V_U$ and, furthermore, for each $x_j \in V_X$ the maximal clique $\{s_i, x_j\} \cup V_Y$), and
- each vertex $v \in V_P \cup V_C$ is contained in exactly one maximal clique.

Finally, we set $s := n \cdot (m+1)$. Clearly, the construction can be performed in polynomial time. The main idea of the reduction can be described as follows. For each vertex in V_X we have to reduce the number of maximal cliques it is contained in. This can only be done by deleting edges between V_X and V_S. This corresponds to selecting a set in the Set Packing instance. However, we also force that for each vertex in V_U the number of maximal cliques it is contained in increases at most by one. Hence, at most one of its neighbors in V_S can be "selected". This corresponds to the disjointness of the sets of the Set Packing solution.

To show the W[1]-hardness of s-VOD parameterized by k, we prove the following.

> *Claim:* (I, k) is a yes-instance for Set Packing if and only if (G, k) is a yes-instance for $(n \cdot (m+1))$-VOD.

"⇒": Let $\mathcal{S}'$ be a size-k solution of Set Packing, and assume without loss of generality that $\mathcal{S}' = \{S_1, \ldots, S_k\}$. We obtain a solution S' of $(n \cdot (m+1))$-VOD by setting $S' := \{\{x_i, s_i\} \mid 1 \leq i \leq k\}$. Let $G' := G \Delta S'$. To see that G' fulfills the $(n \cdot (m+1))$-vertex-overlap property, we only need to consider vertices $v \in V$ such that there is at least one edge that has been removed from $G[N[v]]$, since for the other vertices the number of maximal cliques containing them has not changed.

First, since $\mathcal{S}'$ is a solution for Set Packing, for each $u_i \in V_U$, there is at most one $s_j \in N[u_i]$ with $1 \leq j \leq k$. Hence, at most one edge in $G[N[u_i]]$ has been removed. Let $\{s_j, x_j\}$ denote such an edge. There is one maximal clique in G that contains u_i, s_j, and x_j, namely, $\{u_i, s_j, x_j\} \cup C^{u_i, s_j, x_j}$. After the deletion of $\{s_j, x_j\}$, we have two maximal cliques that contain the vertices from C^{u_i, s_j, x_j}, namely $C^{u_i, s_j, x_j} \cup \{u_i, s_j\}$ and $C^{u_i, s_j, x_j} \cup \{u_i, x_j\}$. Hence, for each $u_i \in V_U$ the number of maximal cliques has increased by at most one. Therefore, each $u_i \in V_U$ is in at most $n \cdot (m+1)$ maximal cliques.

Next, we show that for each vertex $x_i \in V_x$ the number of maximal cliques has decreased by one. This can be seen as follows. For each $x_i \in V_X$, we have removed only the edge $\{s_i, x_i\}$ in $G[N[x_i]]$. This means that the number of maximal cliques that contain x_i cannot increase. Furthermore, by removing $\{s_i, x_i\}$ we destroy the maximal clique $\{s_i, x_i\} \cup V_Y$, since the clique $\{x_i\} \cup V_Y$ is a subset of the existing shielding clique $\{x_i\} \cup V_Y \cup C^{x_i}$. The number of maximal cliques that contain x_i has thus decreased by one. Hence, each $x_i \in V_x$ is now in exactly $n \cdot (m+1)$ maximal cliques. For each vertex in $v \in V_Y$ the number of maximal cliques that it is contained in has not increased, since for each $\{x_i, s_i\}$ that was deleted, the maximal clique $\{s_i, x_i\} \cup V_Y$ is

destroyed, the clique $V_Y \cup \{s_i\}$ becomes a new maximal clique, and the clique $V_Y \cup \{x_i\}$ is a subset of the clique $V_Y \cup \{x_i\} \cup C^{x_i}$.

For each vertex $s_i \in V_S$, the number of maximal cliques that contain s_i has not increased, since if an edge in $G[N[s_i]]$ has been deleted, then it is the edge $\{s_i, x_i\}$. Since this edge is incident to s_i, its deletion does not increase the number of maximal cliques that contain s_i. Hence, each $s_i \in V_S$ is still in at most $n \cdot (m+1)$ maximal cliques.

Finally, each $v \in V_P \cup V_C$ is contained in at most two maximal cliques in G', since each $v \in V_P \cup V_C$ is contained in at most one maximal clique in G, and at most one edge in $G[N[v]]$ has been deleted.

Altogether, each vertex in G' is contained in at most $n \cdot (m+1)$ maximal cliques and S' is thus a size-k solution for $n \cdot (m+1)$-VOD.

"$\Leftarrow$": Let S' be a size-k solution for (G, k) and $G' := G \Delta S'$.

First, we show that for each $x_i \in V_X$, at least one edge between x_i and V_S must be deleted. To see this, consider the following. There are $n \cdot (m+1)+1$ maximal cliques that contain x_i. Hence, the number of maximal cliques containing x_i must be reduced by at least one. The vertex x_i is contained in two types of maximal cliques: those that contain a shielding or a padding clique and those that contain x_i, V_Y, and one vertex $s_j \in V_S$. Note that with k edge deletions, we cannot decrease the number of maximal cliques that contain both x_i and some shielding (or padding) clique. This can be seen as follows. The shielding and padding cliques are pairwise vertex-disjoint in G and with k edge deletions, for each shielding or padding clique there remains at least one vertex that is adjacent to x_i in G'. Next, consider the cliques that contain x_i, V_Y, and one vertex from V_S. In G, there are exactly $|V_S|$ cliques of this type. Suppose S' does not delete any edges between x_i and V_S. We show that in this case G' contains at least $|V_S|$ cliques of this type. Consider some $s_j \in V_S$. Since V_Y has size $2k+1$, there is in G' at least one vertex $y \in V_Y$ that is adjacent to x_i and s_j. Hence, for each s_j there is at least one maximal clique that contains x_i, s_j, and y. Since V_S is an independent set, this means that there are at least $|V_S|$ maximal cliques of this type in G'. Hence, the number of cliques that contain x_i has not decreased in the case that we do not delete any edge between x_i and V_S. Therefore, a size-k solution S' contains for each $x_i \in V_X$ an edge from x_i to a vertex from V_S.

Second, we show that for each $s_i \in V_S$ there is at most one edge incident to s_i that is deleted by S', and thus that S' corresponds to a size-k subset of $\mathcal{S}$. Suppose, otherwise, that for some $s_i \in V_S$, at least two incident edges, say $\{s_i, x_i\}$ and $\{s_i, x_j\}$ have been deleted. Then, for each $u_l \in V_U \cap N(s_i)$ the number of maximal cliques that contain u_l has increased by at least two, since, instead of the two maximal cliques $\{s_i, x_i, u_l\} \cup C^{u_l, s_i, x_i}$ and $\{s_i, x_j, u_l\} \cup C^{u_l, s_i, x_j}$ that are destroyed, there are now four maximal cliques (two for each deleted edge, one that contains s_i and one that contains x_i or x_j, respectively). Then, however, u_j is in more than $n \cdot (m+1)$ maximal cliques, which contradicts that S' is a solution. Hence, we can assume without loss of generality that $S' := \{\{s_i, x_i\} \mid 1 \leq i \leq k\}$.

Finally, we show that the set $\mathcal{S}' := \{S_i \mid 1 \leq i \leq k\}$ is a solution of the Set-Packing instance (I, k). To this end we show that each $u \in V_U$ can have at most one neighbor in $\{s_i \mid 1 \leq i \leq k\}$. Otherwise, the number of maximal cliques that contain u has increased by at least two, a contradiction to the fact that S' is a solution. Hence, $\mathcal{S}'$ is a size-k subset of $\mathcal{S}$ such that every $u \in U$ is contained in at most one $S_i \in \mathcal{S}'$.

Altogether, we have shown the equivalence between the solutions of s-Vertex-Overlap Deletion and Set Packing. This implies that s-Vertex-Overlap Deletion is W[1]-hard when parameterized only by k.

For s-Vertex-Overlap Editing, the construction has to be modified as follows. Instead of adding only one clique V_Y, we add a clique C_j^i for each pair of vertices x_i and s_j. This ensures that adding edges between distinct $s_j, s_l \in V_S$ does not reduce the number of cliques that each x_i is contained in. Note also that edge additions between distinct $u_i, u_j \in V_U$ and between V_U and V_S do not decrease the number of cliques that a vertex from x_i is contained in because of the large shielding cliques for each triangle in $G[V_U \cup V_S \cup V_X]$. Hence, the only choice to decrease the number of cliques that each $x_i \in V_X$ is contained in is again the deletion of an edge between x_i and V_S. The correctness proof then works in complete analogy with the deletion case.

For s-Edge-Overlap Deletion and Editing, we replace each vertex of $v \in V_U \cup V_X$ with two adjacent vertices, and add further "shielding cliques" that ensure that the edge between these two vertices is not deleted. The correctness proofs work analogously; we omit the details. □

5.5 Two Kernelization Results for Edge Deletion

Nontrivial overlap clustering problems seem to be algorithmically more demanding than clustering without overlaps. In particular, finding polynomial-size problem kernels turned out to be challenging even for small constant values of s. We present polynomial-time kernelization algorithms for the two most basic NP-hard clustering problems with nontrivial overlap. More precisely, we present an $O(k^4)$-vertex problem kernel for 1-Edge-Overlap Deletion and an $O(k^3)$-vertex kernel for 2-Vertex-Overlap Deletion.

Both kernelization algorithms make use of the universal data reduction rules presented in Section 3.4. Recall that a clique K is called critical clique if all its vertices have an identical closed neighborhood and K is maximal under this property (see Definition 2.4). In Section 3.4, we presented a data reduction rule that applies for hereditary graph properties and shrinks large critical cliques. More precisely, for r-clip graph properties Reduction Rule 3.2 deletes all but $k + r$ vertices of a critical clique. Recall that a graph property Π is called r-critical clique preserving (r-clip) if the sizes of the critical cliques of the forbidden induced subgraphs characterizing Π are bounded from above by r (Definition 3.3).

5.5.1 An $O(k^4)$-Vertex Kernel for 1-Edge-Overlap Deletion

We present a kernelization for 1-edge-overlap Deletion, which, by Proposition 5.1, is equivalent to the problem of destroying *diamonds* by at most k edge deletions. We introduce four data reduction rules for this problem and show that a yes–instance reduced with respect to these rules has $O(k^4)$ vertices. Reduction Rules 5.1, 5.2, and 5.4 find parts of the graph that need not be modified by optimal solutions, whereas Reduction Rule 5.3 identifies edges that must be in any solution of size at most k.

Reduction Rule 5.1. If there is a maximal clique K containing only edges which are not in any other maximal clique, then remove all edges of K.

Lemma 5.3. *Reduction Rule 5.1 is correct and can be carried out in $O(m^2)$ time.*

Proof. Let G denote the input instance and G' be the graph resulting from applying Reduction Rule 5.1 to a maximal clique K in G. To show the correctness of Reduction Rule 5.1, we prove the following.

Claim: (G, k) is a yes-instance if and only if (G', k) is a yes-instance.

"$\Rightarrow$": Let S denote an optimal solution for G. Then S contains no edge from K. To see this, observe that the only possible way to create a diamond containing some edge from K is to delete edges from K. However, since all edges of K are not in a diamond in G, an optimal solution will never delete them. This means that K remains a maximal clique in $G\Delta S$ and no two vertices of K have common neighbors outside of K. Thus, removing the edges of K from $G\Delta S$ does not create any diamond and S is also a solution for G'.

"$\Leftarrow$": Observe that after applying Reduction Rule 5.1 to K, no two vertices of K have common neighbors in G', since otherwise the edge connecting these two vertices would be contained in a diamond in G. Therefore, we can add the edges of K to the graph $G'' := G'\Delta S$, where S is an optimal solution for G', without destroying the 1-edge-overlap property of G''.

To check the applicability of Reduction Rule 5.1, we compute for each edge whether it is in only one maximal clique K. If so, we check further for all edges of K whether K is the only maximal clique in which these edges are contained. Clearly, this is doable in $O(m^2)$ time. □

Reduction Rule 5.2. Remove all isolated vertices.

Reduction Rule 5.2 is clearly correct and can be performed in linear time. After the exhaustive application of Reduction Rule 5.1, Reduction Rule 5.2 is sufficient to remove all vertices from G that are not in a diamond, as we show in the following.

Proposition 5.2. *Let G be a graph that is reduced with respect to Reduction Rules 5.1 and 5.2. Then every vertex in G is contained in a diamond.*

Proof. Assume towards a contradiction that G contains a vertex v that is not contained in any diamond. Since G is reduced with respect to Reduction Rule 5.2, v has at least one neighbor. Furthermore, since v is not contained in any diamond, $G[N(v)]$ is a cluster graph, that is, a disjoint union of cliques. Let K be one of the cliques of $G[N(v)]$. Clearly, $K \cup \{v\}$ is a maximal clique in G. Furthermore, since v is not contained in any diamond, there is no vertex $u \in V \setminus N[v]$ that is adjacent to more than one vertex in K. Hence, none of the edges of $G[K \cup \{v\}]$ is contained in any other maximal clique $K' \neq K \cup \{v\}$. This contradicts G being reduced with respect to Reduction Rule 5.1. □

Reduction Rule 5.3. If there is an edge $e = \{u, v\}$ such that the complement graph of $G[N(u) \cap N(v)]$ contains a matching of size greater than k, then remove e, add e to the solution, and decrease the parameter k by one.

Lemma 5.4. *Reduction Rule 5.3 is correct and can be carried out in $O(m^2\sqrt{n})$ time.*

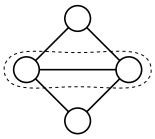


Figure 5.6: A diamond. The only critical clique of size two is encircled by dotted lines.

Proof. For $e = \{u, v\}$, let G_e denote $G[N(u) \cap N(v)]$. Every edge e' in the complement graph $\overline{G_e}$ of G_e implies a diamond in G consisting of the endpoints of e' and u and v. Therefore, every matching of $\overline{G_e}$ corresponds to a set of diamonds in G, whose edge sets pairwise only have e in common. Hence, to destroy all these diamonds, we either delete e or delete one edge for every diamond. A matching of size greater than k thus forces the deletion of e. Since a maximum matching can be computed in $O(m\sqrt{n})$ time [134], the applicability of Reduction Rule 5.3 can be checked in $O(m^2\sqrt{n})$ time by iterating over all edges of G. □

The final data reduction rule shrinks large cliques whose vertices have identical neighborhoods, so-called critical cliques (see Definition 2.4 for a formal definition of critical cliques). Recall that 1-Edge-Overlap Deletion is equivalent to the problem of destroying all induced *diamonds* by at most k edge deletions (see Proposition 5.1). As depicted in Figure 5.6, the largest critical clique in a diamond has size two. Hence, the property of being diamond-free is a 2-critical clique preserving graph property. Thus, Reduction Rule 3.2 ensures that we can savely remove all but $k + 2$ arbitrary vertices of each critical clique. That is, for 1-Edge-Overlap Deletion Reduction Rule 3.2 reads as follows.

Reduction Rule 5.4. If there is a critical clique K with more than $k + 2$ vertices, then remove arbitrary vertices from K until $|K| = k + 2$.

See Lemma 3.3 for the correctness and running time of Reduction Rule 5.4. Making combined use of Reduction Rules 5.1-5.4, we obtain a polynomial-size problem kernel for 1-Edge-Overlap Deletion.

Theorem 5.8. 1-Edge-Overlap Deletion *admits a problem kernel with $O(k^4)$ vertices which can be found in $O(m^3\sqrt{n})$ time.*

Proof. Let G denote an input graph reduced with respect to the above four data reduction rules, and let S be a solution of size at most k. Partition the vertices of the graph $G' := G\Delta S$ into two subsets, one set X containing the vertices that are endpoints of edges deleted by S, and $Y := V \setminus X$. Clearly, $|X| \leq 2k$. It thus remains to show that $|Y| = O(k^4)$. Define for each edge $e \in S$ the set Y_e containing the vertices in Y that, in G, occur together with e in at least one diamond. By Proposition 5.2, $Y = \bigcup_{e \in S} Y_e$. First, we show that every maximal clique K in $G'[Y]$ is contained in Y_e for some $e \in S$. Second, we show that for each $e \in S$ at most $4k$ maximal cliques of $G'[Y]$ are contained in Y_e, which means that there can be at most $4k^2$ maximal cliques in $G'[Y]$. Finally, we show that each of these cliques contains $O(k^2)$ vertices, yielding the claimed overall bound on the number of vertices.

First, we show that for every maximal clique K in $G'[Y]$ there is an edge $e \in S$ with $K \subseteq Y_e$. In G, there is a maximal clique K' containing K and, by Reduction

Rule 5.1, K' has an edge $\{u, v\}$ which is in two maximal cliques, and thus there is a vertex $x \in K$ and a vertex $w \in V \setminus K'$ such that $G[\{u, v, w, x\}]$ is a diamond. Note that if $|K \cap \{u, v\}| = 2$, then no edge of $G[\{u, v, w, x\}]$ is contained in $G[X]$, contradicting the fact that S is a solution. We distinguish the cases that $|K \cap \{u, v\}|$ is either 1 or 0. First, consider the case that $|K \cap \{u, v\}| = 1$. Without loss of generality, let $u \in K$ and $v \in K \setminus K'$. Note that $\{v, w\}$ is the only edge of $G[\{u, v, w, x\}]$ with both endpoints in X, and hence $\{v, w\} \in S$. We show that for every $x' \in K$ it holds that $G[\{u, v, w, x'\}]$ is a diamond, and, hence, $K \subseteq Y_{v,w}$. Assume towards a contradiction that there is a vertex $x' \in K$ such that $G[\{u, v, w, x'\}]$ is not a diamond. Observe that $G[\{u, v, w, x'\}]$ is a clique and, hence, $G'[\{u, v, w, x'\}]$ is a diamond, contradicting the fact that S is a solution. Second, consider the case that $|K \cap \{u, v\}| = 0$, that is, $u, v \in K' \setminus K$. If for every vertex x' it holds that $G[\{u, v, w, x'\}]$ is a diamond, then $K \subseteq Y'_e$ for at least one $e' \in \{\{u, v\}, \{v, w\}, \{u, w\}\}$. Otherwise, there is a vertex $x' \in K$ such that $G[\{u, v, w, x'\}]$ is a clique. Then, however $G[\{v, w, x, x'\}]$ is a diamond and the first case applies since $\{x', v\}$ is contained in two maximal cliques and $x' \in K$ and $v \in K' \setminus K$.

Second, we show that, for every edge $e = \{u, v\} \in S$, at most $4k$ maximal cliques of $G'[Y]$ are subsets of Y_e. Clearly, all vertices in Y_e must be adjacent to at least one of u and v. Let $N_{u,v}$ denote the common neighbors of u and v in Y_e. Since G' is diamond-free, $N_{u,v}$ is an independent set and, by Reduction Rule 5.3, $|N_{u,v}| \leq 2k$. Let $N_u := (N(u) \setminus N(v)) \cap Y_e$ and $N_v := (N(v) \setminus N(u)) \cap Y_e$. Since $N_{u,v}$ is an independent set, no vertex from $N_u \cup N_v$ can be adjacent to two vertices in $N_{u,v}$. Then, we can partition the vertices in $N_u \cup N_v$ into at most $4k$ subsets according to their adjacency to the vertices from $N_{u,v} = \{x_1, \ldots, x_l\}$ with $l \leq 2k$, every subset N_{u,x_i} (or N_{v,x_i}) containing the vertices in $N(u) \cap N(x_i)$ (or $N(v) \cap N(x_i)$). Each subset N_{u,x_i} is a clique, since otherwise two non-adjacent vertices from N_{u,x_i} would form a diamond with x_i and u. The same holds for each N_{v,x_i}. Furthermore, there cannot be an edge between N_{u,x_i} and N_{u,x_j} with $i \neq j$, since otherwise two adjacent vertices $w \in N_{u,x_i}$ and $y \in N_{u,x_j}$ would form a diamond with u and x_i. Moreover, there is no maximal clique K of $G'[Y]$ completely contained in Y_e and containing an edge $\{a, b\}$ such that $a \in N_{u,x_i}$ and $b \in N_{v,x_j}$ for $i, j \in \{1, \ldots, \ell\}$. Suppose that such a clique K exists. Note that a and b must have a common neighbor in G—otherwise, the edge $\{a, b\}$ is a maximal clique to which Reduction Rule 5.1 applies, and thus it would have been removed. Hence, any maximal clique containing a and b also contains at least one further vertex w. In case $K \subseteq Y_e$, this further vertex is in $N(u) \cup N(v)$. Suppose without loss of generality that $w \in N(u)$. Then $G'[\{u, a, b, w\}]$ is a diamond, contradicting the diamond-freeness of G'.

In summary, we have at most $4k$ maximal cliques in $G'[Y]$ which are entirely contained in Y_e. Since there are at most k different Y_e's, and since every maximal clique in $G'[Y]$ is completely contained in at least one Y_e, there can be at most $4k^2$ maximal cliques in $G'[Y]$.

Finally, we show that every maximal clique K in $G'[Y]$ contains $O(k^2)$ vertices. This can be seen as follows. From the vertices of K, only $4k^2$ many can be in more than one maximal clique in $G'[Y]$, since every two cliques in $G'[Y]$ overlap in at most one vertex. Moreover, as argued above, $K \subseteq Y_e$ for some $e = \{u, v\} \in S$ and there is exactly one vertex in K which is adjacent to both u and v. Let K' denote the remaining vertices of K, that is, each vertex of K' has no neighbors in $Y \setminus K$ and is adjacent to at most one of u and v. We show that $|K'| \leq 2k + k + 2$. Clearly, we can assume

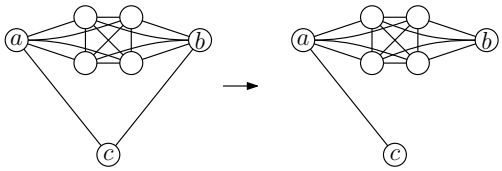


Figure 5.7: Counterexample for the correctness of Reduction Rule 5.1 for 1-Edge-Overlap Editing. The vertices a and b form a diamond with every pair of vertices from the 4-clique in the middle. Since b and c do not have any common neighbor, $\{b, c\}$ is a maximal clique and $\{b, c\}$ is not contained in any other maximal clique. Hence, for 1-Edge-Overlap Deletion, by Reduction Rule 5.1 one can safely remove edge $\{b, c\}$. In contrast, for 1-Edge-Overlap Editing, the resulting instance (on the right) is not equivalent to the original instance (on the left): By adding the edge $\{a, b\}$ the new instance becomes diamond-free. However, it is easy to observe that the original instance can not be transformed into a diamond-free graph by applying only one edge modification.

that $|K'| > 2$, since otherwise the claim is trivially fulfilled. Note that $K' \subseteq N(u)$ or $K' \subseteq N(v)$, since otherwise there would be a vertex $a \in K'$ that is adjacent to u but not to v and a vertex $b \in K'$ that is adjacent to v but not to u. Moreover, since $|K'| > 2$ there is a vertex $x' \in K'$ that is either adjacent to u or v. Assume without loss of generality that $\{x', u\} \in E$. Then, however $G'[\{a, b, u, x'\}]$ is a diamond. We now claim that for every vertex $w \in X \setminus \{u, v\}$, either $K' \subseteq N(w)$ or $|N(w) \cap K'| \leq 1$. Assume the claim is not true. Then we have two vertices $a, b \in K' \cap N(w)$ and one vertex $c \in K' \setminus N(w)$. This implies that there is a diamond consisting of a, b, c, w in G', contradicting that G' is diamond-free. This claim implies that all except for at most $2k$ vertices in K' have the same neighborhood in X. This means that they have the same neighborhood in G and thus they form a critical clique. By Reduction Rule 5.4, there can be at most $k + 2$ of such vertices. Hence, K contains altogether at most $4k^2 + 1 + 2k + k + 2$ vertices.

Summarizing, we have at most $4k^2$ maximal cliques in $G'[Y]$, and each clique contains at most $4k^2 + 3k + 3$ vertices. Hence, $|Y| = O(k^4)$. Since each of the four data reduction rules is performed at most $O(m)$ times, the running time follows from Lemmas 5.3–5.4 and Lemma 3.3. □

Finally, we note that all presented data reduction rules for 1-Edge-Overlap Deletion with exception of Reduction Rule 5.1 are correct for 1-Edge-Overlap Editing, as well. Indeed, in Figure 5.7 we present a simple counterexample for the correctness of Reduction Rule 5.1 for 1-Edge-Overlap Editing. However, for the analysis of the kernel size in the proof of Theorem 5.8 it is decisive that every vertex of a reduced instance is contained in an induced diamond (see Proposition 5.2) which, in turn, is a consequence of the assumption that the instance is reduced with Reduction Rule 5.1.

5.5.2 An $O(k^3)$-Vertex Kernel for 2-Vertex-Overlap Deletion

We present four polynomial-time data reduction rules for 2-Vertex-Overlap Deletion and show that a yes–instance reduced with respect to these rules has $O(k^3)$ ver-

tices. In the following, we say that a vertex is *satisfied* if it is contained in at most two maximal cliques and a clique is satisfied if all its vertices are satisfied. Moreover, a maximal clique all whose vertices are satisfied is called a *satisfied maximal clique*.

Next, we argue that the 2-vertex-overlap property is a 1-critical clique preserving (1-clip) graph property, and, hence, we can make use of the universal data reduction rules and structural observations concerning edge modification problems for 1-clip graph properties presented in Section 3.4. Figure 5.2 (see Section 5.2) shows the forbidden induced subgraphs for graphs with the 2-vertex-overlap property. It is straightforward to verify that for each forbidden induced subgraph no two adjacent vertices have the same neighborhood. This implies that the 1-vertex-overlap property is a 1-clip graph property (see Definition 3.3).

Thus, by Reduction Rule 3.2 it suffices to keep $k+1$ vertices for critical cliques of size at least $k+2$. More specifically, for 2-Vertex-Overlap Deletion Reduction Rule 3.2 reads as follows.

Reduction Rule 5.5. If there is a critical clique K with more than $k+1$ vertices, then remove arbitrary vertices from K until $|K| = k+1$.

Moreover, since the 2-vertex-overlap property is an 1-critical clique preserving graph property, Lemma 3.7 applies for 2-Vertex-Overlap Deletion. Recall that Lemma 3.7 says that there always exists an optimal solution S such that all vertices having the same closed neighborhood in the input graph G also have the same closed neighborhood in $G\Delta S$ and, hence, are contained in the same maximal cliques in $G\Delta S$. Lemma 3.7 is central for the correctness of the following data reduction rules. Recall Lemma 3.7.

Lemma 5.5. *There is an optimal edge modification set $S \subseteq E$ such that every critical clique K of G is part of a critical clique in $G\Delta S$.*

The next rule deals with maximal cliques all whose vertices and neighboring vertices are contained in at most two maximal cliques. The rule removes all edges of such a clique whose endpoints are contained in different maximal cliques, splitting the clique into smaller parts.

Reduction Rule 5.6. If there exists a satisfied maximal clique K such that all vertices in $N(K)$ are satisfied, then remove every edge e for which K is the only maximal clique containing e.

To prove the correctness of Reduction Rule 5.6, we use the following two lemmas.

Lemma 5.6. *Let K and K' be two maximal cliques with $K \cap K' \neq \emptyset$. If all vertices in $K \cap K'$ are satisfied, then*

1. *there is no edge between $K \setminus K'$ and $K' \setminus K$, and*
2. *K is vertex-disjoint to all other maximal cliques intersecting with K'.*

Proof. First, we prove part 1. Assume towards a contradiction that there exist two vertices $v \in K \setminus K'$ and $u \in K' \setminus K$ such that v and u are adjacent. Let $x \in K \cap K'$. Clearly, $\{u, v, x\}$ forms a clique. Let X denote an arbitrary maximal clique containing $\{u, v, x\}$. Note that X is neither K (since $u \notin K$) nor K' (since $v \notin K'$).

Hence, x is contained in at least three maximal cliques, a contradiction to the fact that all vertices in $K \cap K'$ are satisfied.

Next, we prove part 2. Assume towards a contradiction that there exists a maximal clique K'' with $K \cap K'' \neq \emptyset$ and $K' \cap K'' \neq \emptyset$. Since the vertices in $K \cap K'$ are satisfied, K'' intersects with K and K' only in $K \setminus K'$ and $K' \setminus K$, respectively. Hence, there exists a vertex in $K \setminus K'$ and a vertex in $K' \setminus K$ both contained in K'', a contradiction to part 1 of the lemma. □

Lemma 5.7. *Let K be a satisfied maximal clique in G. If there exists a vertex $v \in K$ such that $N[v] = K$, then there exists an optimal solution S such that v is contained in exactly one maximal clique in $G\Delta S$.*

Proof. Let S be an optimal solution for G such that every critical clique of G is part of a critical clique in $G_S := G\Delta S$. By Lemma 5.5, such a solution must exist. Assume towards a contradiction that v is contained in two maximal cliques K_1 and K_2 in G_S. Let W denote the set of vertices in the connected component of $G_S[K]$ containing v and let $X \subseteq S$ denote the edge deletions between vertices of W. Note that $X \neq \emptyset$ since $K_1 \cup K_2 \subseteq W$. We show that $S' := S \setminus X$ is a solution, which contradicts the optimality of S. More precisely, we show that in $G_{S'} := G\Delta S'$ every vertex is satisfied.

Since in $G_{S'}$ there is no edge $\{x, y\}$ with $x \in W$ and $y \in K \setminus W$ (otherwise, y would be in a connected component with v in G_S) it holds that $N_{G_{S'}}(W) \subseteq V \setminus K$. Moreover, since we only undo edge deletions between vertices of W it suffices to show that the vertices in $N_{G_{S'}}(W) \cup W$ are satisfied (for all other vertices the graph induced by their closed neighborhood does not change).

First, consider a vertex $u \in N_{G_{S'}}(W)$. Recall that $u \in V \setminus K$. Let $B := K \cap N_G(u)$. Observe that, since K is satisfied, B is a critical clique in G. Hence, by Lemma 5.5, B is part of a critical clique in G_S and u is adjacent to all vertices of B (that is, $B = N_{G_S}(u) \cap W$). Clearly, this implies that $B = N_{G_{S'}}(u) \cap W$. Thus, the graphs induced by the closed neighborhoods of u in G_S and $G_{S'}$ are identical. Hence, u is satisfied.

Second, consider a vertex $w \in W$. We argue that w is contained in a maximal clique completely contained in W. Let B denote the critical clique of G containing w. Note that $B \subseteq K$. By Lemma 5.5 it follows that $B \subseteq W$ and that all vertices in B have an identical closed neighborhood in G_S. Since W contains more than one critical clique in G_S (note that v is contained in two maximal cliques in G_S), by definition of W there exists a vertex $x \in W \setminus B$ adjacent to w in G_S. Let Q denote a maximal clique of G_S with $\{x, w\} \subseteq Q$. We show that $Q \subseteq W$. Assume towards a contradiction that $Q \setminus W \neq \emptyset$ and let $z \in Q \setminus W$. Let $B' := N_G(z) \cap K$. Since K is satisfied, B' forms a critical clique in G. Moreover, since $w \in B'$ we have $B' = B$, contradicting the fact that $x \in W \setminus B$. Hence, there exists a maximal clique Q in G_S contained in W. This means that there exists at most one further maximal clique K' in G_S containing w and vertices from $V \setminus K$. Hence, w is in $G_{S'}$ contained in at most two maximal cliques, namely W and K' (if it exists). □

Lemma 5.8. *Reduction Rule 5.6 is correct and can be carried out in $O(m \cdot n)$ time.*

Proof. Let $G = (V, E)$ be a graph containing a satisfied maximal clique K such that all vertices in $N_G(K)$ are satisfied. Moreover, let $G' = (V', E')$ denote the graph that results from removing all edges contained only in K.

To show the correctness of Reduction Rule 5.6, we use the following. Let $\mathcal{B} := \{B_1, \ldots, B_\ell\}$ denote the critical cliques of G contained in K. Note that, since K is satisfied, for every B_i there exists at most one further maximal clique K_i in G with $B_i \subseteq K_i$. Furthermore, by Lemma 5.6, it follows that the K_i's are pairwise vertex-disjoint.

Claim: (G, k) is a yes-instance if and only if (G', k) is a yes-instance.

"⇒": Let S be an optimal solution of size at most k for G and let $G_S := G\Delta S$. We show that $S' := S \setminus E_K$ is a solution for G', where E_K denotes the set of all possible edges between two vertices of K. Let $G'_{S'} := G'\Delta S'$. According to Lemma 5.5, we can assume that every B_i is completely contained in at most two maximal cliques in G_S. In particular, this means that S does not contain edge deletions between two vertices of the same B_i. Hence, $G'_{S'}$ differs from G_S in that all edges between different B_i's are deleted. Moreover, every vertex $x \in V \setminus K$ being adjacent in G_S to a vertex of B_i is adjacent in G_S to every vertex in B_i but not to any other vertex in K. Since $G'_{S'}$ differs from G_S in that all edges between different B_i's are deleted, the graphs induced by the closed neighborhood of every vertex in $x \in V \setminus K$ in G_S and $G'_{S'}$ are identical. Hence, these vertices are satisfied and it remains to show that all vertices in the B_i's are satisfied.

To this end, we argue that every B_i is contained in at most two maximal cliques in $G'_{S'}$. First, consider the case that B_i is contained in two maximal cliques C_i^1 and C_i^2 in G_S. If $C_i^1 \subseteq K_i$ and $C_i^2 \subseteq K_i$, then there cannot be any edge between B_i and $K \setminus B_i$ in G_S since otherwise B_i would be contained in three maximal cliques (note that by Lemma 5.6 there is no edge between $C_i^j \setminus K$ and $K \setminus C_i^j$, $j \in \{1, 2\}$). Hence, the graphs induced by the closed neighborhood of a vertex in B_i in G_S and $G'_{S'}$ are identical. If $C_i^1 \subseteq K$ and $C_i^2 \subseteq K_i$, then $C_i^1 \cap C_i^2 = B_i$, since a vertex in $C_i^2 \setminus B_i$ is adjacent in G_S to all vertices in B_i but not to any other vertex in K. Hence, after deleting the edges between B_i and $K \setminus B_i$, the vertices of B_i are contained in exactly one maximal clique, namely $C_i^2 \subseteq K_i$. Second, for the case that B_i is contained in exactly one maximal clique in G_S, the argumentation works in analogy. In summary, $G'_{S'}$ fulfills the 2-vertex-overlap property.

"⇐": Let S denote an optimal solution of size at most k for G', and let $G'_S := G'\Delta S$. We show that S is solution for G as well, that is, we show that in $G_S := G\Delta S$ every vertex is satisfied. Note that for every B_i every vertex of B_i is contained in exactly one satisfied maximal clique in G', namely K_i. Thus, by Lemma 5.7, we can assume that every B_i is completely contained in exactly one satisfied maximal clique $K'_i \subseteq K_i$ in G'_S. Furthermore, recall that all K'_i's are pairwise vertex-disjoint and every vertex in $K'_i \setminus B_i$ is adjacent in G'_S to all vertices in B_i but not to any other vertex in K. Hence, if we add all missing edges between the vertices of K in G'_S (resulting in G_S), then none of the added edges is between two neighbors of vertices in $K'_i \setminus K$. Hence, these vertices are satisfied in G_S. Moreover, since these are the only vertices in $V \setminus K$ having in G'_S at least two neighbors in K, all vertices in $V \setminus K$ are satisfied in G_S. Finally, all vertices in K are clearly contained in at most two maximal cliques in G_S, namely in K and in at most one further clique $K'_i \subseteq K_i$ (note that, in G_S, the common neighborhood for two vertices from different B_i's is K).

For the running time, note that one can compute the set U of all satisfied vertices in $O(m \cdot n)$ time as follows. For each $v \in V$, build $G[N[v]]$ and then check in $O(|N[v]|^2)$

time whether $G[N[v]]$ contains at most two maximal cliques. The running time for computing U hence sums up to

$$O(\sum_{v \in V} \deg(v)^2) = O(n \cdot \sum_{v \in V} \deg(v)) = O(n \cdot m).$$

After that, consider the vertices in U one by one. Every vertex $u \in U$ is contained in at most two maximal cliques K_1 and K_2. These two cliques can be computed in $O(|N[v]|^2)$ time for every $u \in U$. Finally, check in $O(m)$ time whether K_1 or K_2 fulfills the precondition of Reduction Rule 5.6. Hence, the overall running time for one application of Reduction Rule 5.6 is bounded by $O(n \cdot m + \sum_{u \in U}(\deg(v)^2 + m)) = O(m \cdot n)$. □

Reduction Rule 5.7. Let G be a graph reduced with respect to Reduction Rule 5.5. Let K be a maximal clique of G. If there are maximal cliques $K_1, \ldots, K_\ell$ fulfilling the following three conditions:
1.) $K \cap K_i \neq \emptyset$, $1 \leq i \leq \ell$,
2.) all vertices in K_i, $1 \leq i \leq \ell$ are satisfied, and
3.) $\sum_{i=1}^{\ell} |K_i \cap K| \geq 3k + 4$,
then remove all edges between $K_1 \cap K$ and $K \setminus K_1$.

To prove the correctness of Reduction Rule 5.7, we need the following lemma.

Lemma 5.9. *Let $G = (V, E)$ denote a graph reduced with respect to Reduction Rule 5.5. Let K and $K_1, \ldots, K_\ell$ be maximal cliques in G fulfilling Conditions* 1 *and* 2 *of Reduction Rule 5.7 and suppose that $\sum_{i=1}^{\ell} |K_i \cap K| \geq 2k + 2$. If (G, k) is a yes–instance, then there exists an optimal solution of size at most k not deleting any edge between vertices of K.*

Proof. Suppose that there exists an optimal solution S of size at most k for G and let $G_S := G \Delta S$. Assume towards a contradiction that S contains an edge $\{v, w\}$ with $v, w \in K$. In the following, we refer by $\{u_1, \ldots, u_t\}$ to the vertices in $\bigcup_{i=1}^{\ell}(K_i \cap K)$. Since all K_i's are satisfied (Condition 2), according to Lemma 5.6 the K_i's are pairwise vertex-disjoint. Because $t \geq 2k + 2$, one of the u_i's is non-affected by S. Without loss of generality, assume that u_1 is one of these non-affected vertices and $u_1 \in K_1$. Let $B_1 := K \cap K_1$. Clearly, B_1 is a critical clique in G. By Lemma 5.5, we have that B_1 is (part of) a critical clique in G_S, and, hence, all vertices in B_1 are non-affected. This implies that neither v nor w is contained in B_1. Let $z \in K_1 \setminus K$. Since u_1 is non-affected by S, this implies $\{z, u_1\} \in E(G_S)$. Moreover, by Lemma 5.6 (and Condition 2 of Reduction Rule 5.7), it follows that $\{z, v\}$ and $\{z, w\}$ are not contained in E and hence not in $E(G_S)$. This implies that u_1 is contained in at least three maximal cliques in G_S: the vertices u_1, v, w, and z induce a star with center vertex u_1 and three leaves (see first graph in Figure 5.2). This is a contradiction to the fact that S is a solution. □

Lemma 5.10. *Reduction Rule 5.7 is correct and can be carried out in $O(m \cdot n)$ time.*

Proof. Let $G = (V, E)$, K, and $K_1, \ldots, K_\ell$ be as described in Reduction Rule 5.7. Furthermore, let $B_i := K_i \cap K$ for every $1 \leq i \leq \ell$. Again, since all K_i's are satisfied, the B_i's are critical cliques and according to Lemma 5.6 the K_i's are pairwise vertex-disjoint. Let $G' = (V, E')$ be the graph resulting from one application of Reduction Rule 5.7. We show the following.

Claim: (G, k) is a yes–instance if and only if (G', k) is a yes–instance.

"$\Rightarrow$": Let S denote an optimal solution of size at most k for G and let $G_S := G\Delta S$. We show that S is a solution for G'. Let $G'_S := G'\Delta S$. By Lemma 5.9, S does not delete any edge within K. Together with Lemma 5.5, this implies that, in G_S, B_1 is contained in K and in at most one further maximal clique $K'_1 \subseteq K_1$. Note that G'_S results from G_S by deleting all edges between B_1 and $K \setminus B_1$. Since by Lemma 5.6, there is no edge between $K'_1 \setminus K$ and $K \setminus K'_1$, this does not create any unsatisfied vertices.
"$\Leftarrow$": Let S' denote an optimal solution of size at most k for G' and let $G'_{S'} := G'\Delta S'$. We show that in $G_{S'} := G\Delta S'$ all vertices are satisfied. Note that in G', K_1 forms a clique whose vertices are all satisfied and that the vertices in B_1 are contained in exactly one maximal clique, namely K_1. Hence, according to Lemma 5.7, we can assume that B_1 is contained in exactly one maximal clique $K'_1 \subseteq K_1$ in $G'_{S'}$. Moreover, note that for every $1 \leq i \leq \ell$, since G is reduced with respect to Reduction Rule 5.5 and since B_i is a critical clique in G, it holds that $|B_i| \leq k+1$. In particular, $|B_1| \leq k+1$ and since $\sum_{i=1}^{\ell} |B_i| \geq 3k+4$, we have $\sum_{i=2}^{\ell} |B_i| \geq 2k+2$ and $\ell \geq 3$. Hence, $\sum_{i=2}^{\ell} |(K \setminus B_1) \cap K_i| \geq 2k+2$. Moreover, it is not hard to verify that $K \setminus B_1$ forms a maximal clique in G'. Thus, by Lemma 5.9, S' does not delete any edge between two vertices from $K \setminus B_1$. Hence, $K \setminus B_1$ is a maximal clique in $G'_{S'}$. Note that $G_{S'}$ results from $G'_{S'}$ by inserting all edges between a vertex in B_1 and the vertices in $K \setminus B_1$. Clearly, this does not change the number of maximal cliques for a vertex in $V \setminus K$, since, by Lemma 5.6, none of these has neighbors in both B_1 and $K \setminus B_1$. Finally, all vertices in K clearly are satisfied.

For the running time note the following. First, as argued in the proof of Lemma 5.8, we can compute the set U of all satisfied vertices in $O(n \cdot m)$ time. Hence, in the following we assume that for each vertex in the graph, we can determine in $O(1)$ time whether it is satisfied or not. Then, for every vertex $u \in U$ we proceed as follows. Vertex u is contained in at most two maximal cliques K' and K''. These two cliques can be computed in $O(\deg(u)^2)$ time. Next, we check whether K' and K'' can play the role of K and K_1 in Reduction Rule 5.7. Consider the case that $K = K'$ and $K_1 = K''$. Clearly, we can check in $O(\deg(u))$ time whether all vertices in K'' are satisfied. It remains to verify that there are at least $3k+4$ vertices in the intersections of satisfied maximal cliques with K'. We argue that this is possible in $O(m)$ time. We first label all vertices in K' that are contained in exactly two maximal cliques by '+'. All other vertices in K' are labeled by '−'. Next, we iterate over the edge set. For an edge $\{x, y\} \in E$ if $y \notin K'$ and not satisfied and x is labeled '+' then mark x with '−'. After that, if a satisfied vertex $v \in K'$ is contained in a second maximal clique containing non-satisfied vertices, then this vertex clearly is labeled '−'. Hence, all vertices labeled by '+' are contained in the intersections of satisfied maximal cliques with K. Thus, to check whether Reduction Rule 5.7 can be applied we just need to count the number of '+'-vertices in K. In summary, the overall running time is $O(m \cdot n + \sum_{u \in U}(\deg(u)^2 + m)) = O(m \cdot n)$. □

Reduction Rule 5.8. Remove connected components fulfilling the 2-vertex-overlap property.

Theorem 5.9. 2-Vertex-Overlap Deletion *admits a problem kernel with* $O(k^3)$ *vertices.*

Proof. Let $G = (V, E)$ be a graph reduced with respect to Reduction Rule 5.5–Reduction Rule 5.8. We show that if G has a solution of size at most k, then the number of vertices of G is $O(k^3)$.

Assume that G has a solution S of size at most k and let $G_S := G\Delta S$. Furthermore, let X denote the vertices affected by S and let $Y := V \setminus X$. First, note that $|X| \leq 2k$. Hence, it remains to show $|Y| = O(k^3)$.

Let $K_1, \ldots, K_t$ denote the maximal cliques of G_S containing at least one vertex of X. Note that $t \leq 4k$ since a vertex $x \in X$ is contained in at most two maximal cliques in G_S. Furthermore, define $K'_i := K_i \cap Y$, $1 \leq i \leq t$ and let $Z := \{Z_1, \ldots, Z_q\}$ denote the set of all other maximal cliques of G_S. For every $1 \leq i < j \leq t$ let $K'_{i,j} := K'_i \cap K'_j$. Note that every $K'_{i,j}$ is part of a critical clique in G_S, since it belongs to two maximal cliques. Furthermore, since the vertices in $K'_{i,j}$ are non-affected, they are also part of a critical clique in G. As a consequence, we have $|K'_{i,j}| \leq k+1$ since G is reduced with respect to Reduction Rule 5.5. Let $K'_{i,cc}$ denote the vertices of K'_i that are contained only in the maximal clique K_i in G_S. By the same argument as above, $|K'_{i,cc}| \leq k + 1$. Finally, let $A_i := K'_i \setminus (K'_{i,cc} \cup \bigcup_{j \neq i} K'_{i,j})$ denote the other vertices of K'_i. Note that $A_i \subseteq \bigcup_{i=1}^{q} Z_i$.

Next, we show that

a) every vertex in A_i is contained in at most two maximal cliques in G,

b) for $1 \leq j \leq q$, every vertex in Z_j is contained in at most two maximal cliques in G,

c) for $1 \leq j \leq q$, every Z_j has a nonempty intersection with at least one A_i, $1 \leq i \leq t$,

d) for $1 \leq\ i \leq t$, $|A_i| \leq 3k + 3$, and

e) $q \leq (3k + 4) \cdot 4k$ and, for $1 \leq j \leq q$, $|I_j| \leq 4k + 4$, with $I_j := Z_j \setminus (\bigcup_{i=1}^{t} A_i)$.

a) Consider an arbitrary vertex $y \in A_i$. Note that y is adjacent in G_S only to the vertices $K_i \setminus K'_i$ of X. Since $K_i \setminus K'_i$ is a clique in G_S, no edge between the vertices in $K_i \setminus K'_i$ is deleted. Hence, no edge between any two neighbors of y is deleted and, therefore, y is contained in the same number of maximal cliques in G as in G_S.

b) A vertex $y \in Z_j$ is either contained in A_i, $1 \leq i \leq t$, or all its neighbors are non-affected. In the first case, y is satisfied according to a). In the second case, y is clearly contained in at most two maximal cliques in G.

c) Assume that there exists a Z_j that does not intersect with any A_i for some i, $1 \leq i \leq t$. Then, Z_j intersects only with other elements from Z. Hence, Z_j and all of Z_j's neighbors are satisfied and, as a consequence, Reduction Rule 5.6 applies, contradicting the fact that G is reduced.

d) Assume that there exists an i with $|A_i| \geq 3k + 4$. Without loss of generality, let $Z_1, \ldots, Z_p$ be the sets in Z intersecting with A_i. Hence, $A_i \subseteq \bigcup_{j=1}^{p} Z_j$ (recall that $A_i \subseteq \bigcup_{j=1}^{q} Z_j$) and as a consequence $|A_i| = \sum_{j=1}^{p} |Z_j \cap A_i| \geq 3k + 4$. Moreover, according to b), all Z_j's are satisfied. Thus, Reduction Rule 5.7 applies to a maximal clique K with $A_i \subseteq K$ in G, contradicting the fact that G is reduced.

e) First, since every Z_j has nonempty intersection with some A_i and since any other Z_h, $h \neq j$, cannot intersect with A_i in the same vertices as Z_j, it follows that $|Z| \leq$

$(3k+3) \cdot 4k$. Second, assume that there exists an I_j with $|I_j| > 4k+4$. Since G is reduced with respect to Reduction Rule 5.5, there are at most $k+1$ vertices in I_j that are contained in the single maximal clique Z_j (these vertices form a critical clique in G). All other vertices of I_j are contained in some Z_h with $h \neq j$. Let $Z'_1, \ldots, Z'_p$ denote the sets in Z having nonempty intersection with Z_j. Since $|I_j| > 4k+4$, it holds that $\sum_{r=1}^{p} |Z_j \cap Z'_r| > 3k+3$, and, as a consequence, Reduction Rule 5.7 applies, contradicting the fact that G is reduced.

Putting everything together, one obtains

$$\begin{aligned}
|Y| &\leq \sum_{i=1}^{t} |K'_i| + \sum_{j=1}^{q} |I_j| \\
&\leq \sum_{i=1}^{t} (|K'_{i,cc}| + |A_i| + \sum_{j=1}^{t} |K'_{i,j}|) + |Z| \cdot (4k+4) \\
&\leq 4k \cdot (k+1+(3k+3)+4k \cdot (k+1)) + (3k+3) \cdot 4k \cdot (4k+4).
\end{aligned}$$

□

Finally, we discuss some differences of the kernelizations for 1-Edge-Overlap Deletion and 2-Vertex-Overlap Deletion. Reduction Rule 5.1 of the kernelization for 1-Edge-Overlap Deletion deletes all edges of a maximal clique K if all edges of K are satisfied. This ensures that all vertices of a reduced instance are contained in some forbidden induced subgraph. For 2-Vertex-Overlap Deletion it is not correct to delete satisfied vertices or the vertices/edges of a satisfied maximal clique (see Figure 5.8). Hence, for 2-Vertex-Overlap Deletion we need that all vertices in the neighborhood of a satisfied maximal clique K are satisfied for showing that it is correct to remove the edges in K (see Reduction Rule 5.6). Hence, we cannot ensure that all vertices of a reduced instance are contained in some forbidden induced subgraph, making the kernelization for 2-Vertex-Overlap Deletion more involved.

5.6 Conclusion

This chapter provided a first theoretical study of a set of new cluster graph modification problems motivated by the practical relevance of clustering with overlaps [57, 141]. Naturally, studying a set of problems that is so far barely explored, there remain many challenges for future work.

First, it is conceivable that the forbidden subgraph characterizations we developed for cluster graphs with overlaps can be further refined. Observe that we presented an upper bound for the size of the forbidden induced subgraphs but we could not show that this upper bound is tight. In addition, a more precise graph-theoretic characterization of the forbidden subgraphs is desirable. Improvements in this direction would immediately imply better search tree algorithms.

Second, extending the list of kernelization results for our problems other than 1-Edge-Overlap Deletion and 2-Vertex-Overlap Deletion is a natural next step. In particular, we left open whether 1-Edge-Overlap Editing and 2-Vertex-Overlap Editing admit polynomial-size problem kernels. Moreover, in the light of

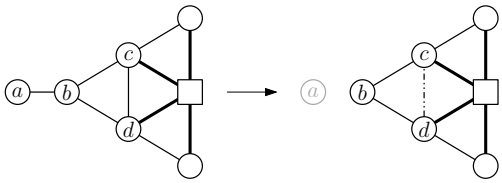


Figure 5.8: Example showing that the deletion of satisfied vertices (or the vertices/edges of satisfied maximal cliques) does not lead to an equivalent instance for 2-Vertex-Overlap Deletion. A rectangular vertex represents a critical clique of size two. All other vertices represent single vertices. Moreover, a bold edge represent two edges between the critical clique and the respective single vertex. Note that $\{a, b\}$ forms a satisfied maximal clique in the graph drawn on the left. Deleting either a or the edge $\{a, b\}$ (as shown on the right) leaves a graph that can be transformed into a graph with 2-vertex-overlap property by deleting one edge (the dashed edge). In contrast, at least two edge deletions are necessary to transform the graph on the left into a graph with 2-vertex-overlap property.

recent nonexistence results concerning polynomial-size problem kernels for specific edge modification problems [124, 87], one may wonder whether s-Vertex/Edge-Overlap Editing/Deletion/Addition admit polynomial-size problem kernels when parameterized by (s, k), or, what seems more plausible, whether even for constant values of s it is very unlikely that the corresponding edge modification problems admit polynomial-size problem kernels. It seems challenging to provide a dichotomy regarding the "polynomial-size kernelizability" of the considered problems.

Third, in addition to kernelization algorithms it seems worth to investigate other algorithm design techniques, such as refined search tree algorithms. In ongoing work, we already obtained an improved search tree algorithm for 1-Edge-Overlap Deletion [118].

Fourth, corresponding experimental studies (like those undertaken for Cluster Editing, see [25, 57]) are a natural next step. In particular, speed-up tricks for our fixed-parameter algorithms and heuristics for finding good approximative solutions are desirable in this context.

Fifth, the polynomial-time approximability of our problems remains unexplored.

Sixth, the vertex deletion versions of our problems deserve attention. For for the vertex deletion version without overlap, that is, Cluster Vertex Deletion, there exist iterative compression algorithms [109]. Moreover, parameterized complexity studies have recently been undertaken for the problem to transform a graph into an s-plex cluster graph by a minimum number of vertex deletions [22].

Finally, we discuss some extensions of our overlap models.

As argued in Section 5.1 the "t-Zahn property" enforces that two maximal cliques intersect in at most $t-1$ vertices. Our models enforce that a vertex/edge is contained in at most s maximal cliques. Clearly, it makes sense to consider a combination of both properties. Analogously, the combination of our models with the model of Damaschke allows to further constrain the solution. Moreover, note that combinations of the different overlap models lead to more restricted and, hence, simpler, cluster graph models. It seems plausible that simpler cluster graph models allow for more efficient algorithms. However, there is always a trade-off between having a simple model (that

is easier to handle algorithmically) and the ability of the model to find meaningful clusterings: an overly restrictive cluster graph model leads to artificial and, hence, meaningless clusterings.

In several clustering scenarios, the given input data can be modeled as a bipartite graph. For example, in clustering gene expression data, one partition represents the genes and the other partition represents the investigated conditions [130]. Another example is document clustering, where one partition represents the documents and the other partition represents the terms. Bicluster Editing (also known as Bipartite Correlation Clustering) is the "bipartite sister problem" of Cluster Editing [145, 93, 4]. Similar to Cluster Editing, Bicluster Editing fails to model that the (bi)clusters may overlap, although overlapping clusters are interesting in document clustering and clustering of gene expression data [165, 130, 154]. Hence, extending the notion of "bicluster graph" (which is defined as a disjoint union of bicliques) to allow for some degree of overlap between the biclusters is desirable.

Chapter 6

Hierarchical Tree Clustering

6.1 Introduction

Hierarchical representations of data play an important role in biology, the social sciences, and statistics [5, 55, 106, 125]. The basic idea behind hierarchical clustering is to obtain a recursive partitioning of the input data in a tree-like fashion such that the leaves one-to-one represent the single data items and all inner points represent clusters of various granularity degrees. Hierarchical clusterings do not require a prior specification of the number of clusters and they allow to understand the data at many levels of fine-grainedness (the root of the tree representing the whole data set).

In a nutshell, we contribute new algorithms and an experimental study for a well-studied NP-hard problem in this context, called M-Hierarchical Tree Clustering. In particular, we present two polynomial-size problem kernels for M-Hierarchical Tree Clustering.

Problem Statement. Let $X = \{1, \ldots, n\}$ be the input set of elements (representing data items) to be clustered. The dissimilarity of the elements is expressed by a symmetric function $D : X \times X \to \{0, \ldots, M+1\}$ with $D(i, j) > 0$ if and only if $i \neq j$, called *distance function*. Herein, the integer $M \in \mathbb{N}$ specifies the depth of the clustering tree to be computed. Roughly speaking, the task is then to find an "ultrametric tree" that fits the dissimilarity data in the best possible way. Herein, an *ultrametric tree* is a rooted tree whose leaves are bijectively labeled with the elements in X and in which all leaves are at the same distance to the root. Then, the "tree distance" of a pair of leaves is the height of their least common ancestor. An ultrametric tree represents a distance function D, when the pairwise tree distances between all pairs of leaves are the same as given by D. Clearly, not every distance function can be represented by an ultrametric tree. For such instances, the goal is to find a "closest" ultrametric tree. In the following, we describe some properties and provide definitions to formalize this task.

For an ultrametric tree, the distance $U(i, j)$ between two elements $i \in X$ and $j \in X$ is the height of the least common ancestor of the leaves corresponding to i and j. It is easy to see that U fulfills the strong triangle inequality, that is, it holds that

	1	2	3	4	5
0	1	2	2	1	3
1	-	2	2	3	3
2	-	-	1	3	3
3	-	-	-	3	3
4	-	-	-	-	2

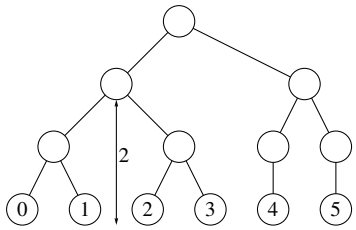


Figure 6.1: An ultrametric (distance function) and a corresponding ultrametric tree. The distances in the distance function are the same as the distances between the leaves in the tree. For example, the leaves corresponding to "1" and "2" have distance two as given by the height of their least common ancestor. Note that the nodes of each level of the tree define a partition of the leaves into clusters: for each node a cluster contains the leaves in the subtree rooted at this node. For example, the first level defines the clusters $\{0,1,2,3\}$ and $\{4,5\}$, and the second level further partitions the cluster $\{0,1,2,3\}$ into $\{0,1\}$ and $\{2,3\}$.

$U(i,j) \leq \max\{U(i,l), U(j,l)\}$ for all $i,j,l \in X$. Indeed, a distance function fulfills the strong triangle inequality if and only if it can be represented by an ultrametric tree [105, 114, 116].

Definition 6.1. A distance function $D : X \times X \to \{0,\ldots,M+1\}$ is called *ultrametric* if the *strong triangle inequality* holds, that is, for all $i,j,l \in X$

$$D(i,j) \leq \max\{D(i,l), D(j,l)\}. \tag{6.1}$$

See Figure 6.1 for an example of an ultrametric and a corresponding ultrametric tree.

In this chapter, we study the problem of finding a *closest ultrametric* U for a given dissimilarity function D where the distance between U and D is measured by the ℓ_1 norm, that is, we want to minimize

$$||D-U||_1 := \sum_{\{i,j\}\subseteq X} |U(i,j) - D(i,j)|.$$

The problem under consideration can thus be formulated as follows:

Definition 6.2. M-Hierarchical Tree Clustering (M-HTC)
Input: A set X of elements, a distance function $D : X \times X \to \{0,\ldots,M+1\}$, and an integer $k \geq 0$.
Question: Is there an ultrametric $U : X \times X \to \{0,\ldots,M+1\}$ with $||D-U||_1 \leq k$?

In other words, given any distance function D, the goal in M-HTC is to modify D as little as possible to obtain an ultrametric U.

Next, we shed light on the relationship between M-HTC and Cluster Editing and point to related problems in computational phylogenetics. Then, we given an overview over known results for M-HTC.

Cluster Editing is 1-Hierarchical Tree Clustering. As observed previously [125, 5], 1-Hierarchical Tree Clustering is the same as Cluster Editing,

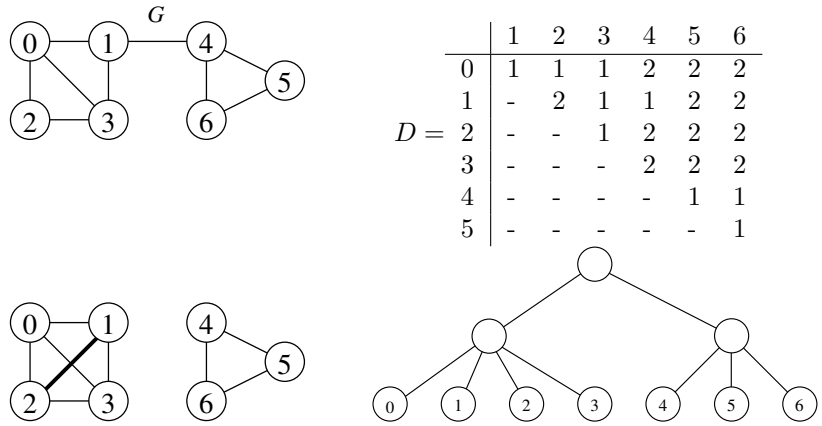


Figure 6.2: The first row shows the graph $G = (V, E)$ of a Cluster Editing instance and the distance function D of an equivalent 1-Hierarchical Tree Clustering instance. Observe that $\{i, j\} \in E$ if and only if $D(i, j) = 1$. The second row shows a cluster graph and a corresponding ultrametric tree. Observe that two edge modifications are required to obtain the cluster graph and that the distance between the distance function and the ultrametric tree is two, as well.

which asks for a minimum number of edge modifications to obtain a cluster graph (see Chapter 4). As illustrated in Figure 6.2 (second row), a cluster graph can equivalently be represented by an ultrametric tree of height two, and vice versa. More precisely, two distinct vertices $x \in X$ and $y \in X$ are grouped together into a cluster of the cluster graph $G_c = (X, E_c)$ if and only if x and y are siblings (that is, have a common parent) in the ultrametric tree. That is, in the ultrametric U corresponding to this tree we have $U(x, y) = 1$ if and only if $\{x, y\} \in E_c$. Now, the equivalence between Cluster Editing and 1-Hierarchical Tree Clustering can be seen as follows (an illustration is given in Figure 6.2).

Given a Cluster Editing instance $(G = (V, E), k)$ one obtains an equivalent 1-Hierarchical Tree Clustering instance (D, X, k) with $X = V$ by setting, for any two distinct vertices $v, w \in V$, $D(v, w) = 1$ if $\{v, w\} \in E$ and $D(v, w) = 2$, otherwise. Consider a cluster graph G_c on V and a corresponding ultrametric U (defined as described above). If two vertices x and y that are adjacent in G occur in different clusters of G_c (that is, the edge $\{x, y\}$ has been deleted), then $D(x, y) = 1$ and $U(x, y) = 2$, and, hence, the contribution of this pair to $||D-U||_1$ is one. Similarly, if two vertices that are nonadjacent in G occur in the same cluster of G_c (that is, the edge between them is added), then these two vertices contribute one to $||D - U||_1$. Thus, the edit distance between a cluster graph G_c on V and G equals the distance between the corresponding ultrametric U and the distance function D and vice versa, giving the equivalence between 1-Hierarchical Tree Clustering and Cluster Editing.

M-Hierarchical Tree Clustering and Phylogenetic Tree Reconstruction. In addition to be relevant for clustering, M-HTC is also closely related to the reconstruction of phylogenetic trees [67, 5]. In phylogenetics, the evolutionary relationship

between species is usually depicted by arranging the species in a *phylogenetic tree*. Phylogenetic trees are usually inferred based on dissimilarities in the physical or genetic characteristics, reflecting their evolutionary distances. In an idealized model, evolutionary data is ultrametric. Indeed, Gusfield pointed out [101, Section 17.2]: "Ultrametric data are the Holy Grail of phylogenetic reconstruction—when time-since-divergence data are ultrametric, the belief is that the true evolutionary history can be reconstructed. But this is mostly an idealized abstraction, and real data are rarely ultrametric". In this context, M-Hierarchical Tree Clustering can be seen as the problem to correct the input data as little as possible to obtain an ultrametric tree. More formally, M-HTC is as a special case of the *numerical taxonomy problem*. Here the task is to find a tree T that spans the set X of elements and fits the given distance function D on X [3]. In this context, "fitting" means to minimize $||T - D||_p := \sum_{\{i,j\} \subseteq X} |T(i,j) - D(i,j)|^p$ for a specified $p \geq 1$, where $T(i,j)$ denotes the length of the path between i and j. Thus, M-HTC is the restriction of the numerical taxonomy problem to ultrametric trees[1] using the ℓ_1 norm. In the area of phylogeny reconstruction, M-HTC is known as "Fitting Ultrametrics under the ℓ_1 norm". Fitting Ultrametrics is NP-hard under the ℓ_1 norm [125] as well as under the ℓ_2 norm [56]. However, it is linear-time solvable under the ℓ_∞ norm [67].

Previous Work. M-HTC is NP-complete [125] and APX-hard [3], excluding any hope for polynomial-time approximation schemes. Ailon and Charikar [5] presented an elegant randomized polynomial-time algorithm for M-HTC that achieves an approximation ratio of $M + 2$. Subsequently, a deterministic algorithm achieving the same approximation guarantee was presented [166].

As argued above 1-Hierarchical Tree Clustering is the same as Cluster Editing. Cluster Editing has applications in document clustering and agnostic learning [12] as well as clustering of gene expression data [15] or protein similarity data [24]. Numerous papers deal with Cluster Editing and its polynomial-time approximability [150, 6, 166] and parameterized complexity [83, 73, 145, 89, 24, 46, 28]. In particular, there have been encouraging experimental studies on solving Cluster Editing based on fixed-parameter algorithms [57, 146, 24, 25] and integer linear programming formulations [25]. See Section 4.1 for an overview.

In our work, we concentrate on the problem where the dissimilarity data are specified as integers from $\{0, \ldots, M + 1\}$. Notably, a result of Harb et al. [104, Lemma 1] implies that whenever the input data are integers, then there exists a closest ultrametric (under the ℓ_1-norm) that only takes integer values.

Our Results. We investigate the parameterized complexity of M-HTC with respect to the following two parameters: first, the distance k between the input distance function and a closest ultrametric, and, second, the depth M of the ultrametric tree.

The contributions of this chapter are both of theoretical as well as of practical nature. On the theoretical side, we provide two kernelization results for M-HTC. More precisely, we develop polynomial-time data reduction rules that provably transform an original input instance of M-HTC into an equivalent instance consisting of only $O(k^2)$ elements or $O(M \cdot k)$ elements, respectively. Moreover, an $O(2.562^k)$-size search tree is presented.

[1] Note that, in case of ultrametric trees, usually half the length of the path is used.

On the practical side, we contribute implementations and experiments for our new data reduction rules (combined with the search tree strategy), the approximation algorithm of Ailon and Charikar [5], and an (I)LP formulation also due to Ailon and Charikar [5]. Our main finding is that for parameter values $k < |X|$ our data reduction based algorithms are the method of choice whereas otherwise the other approaches are mostly superior.

6.2 Preliminaries

In the following, we define some of the notations used in this chapter.

Let D denote a distance function over a set X. Throughout this chapter let $n := |X|$. A *closest ultrametric* U for D is an ultrametric on X such that $||D - U||_1$ is minimum among all ultrametrics on X. A *conflict* is a triple $\{i, j, l\}$ of elements from X that does not fulfill Condition 6.1 of Definition 6.1, that is, one of the three distances $D(i, j)$, $D(i, l)$, and $D(j, l)$ is larger than each of the other two. A pair $\{i, j\}$ is the *max-distance pair* of a conflict $\{i, j, l\}$ if $D(i, j) > D(i, l)$ and $D(i, j) > D(j, l)$. The pairs $\{i, l\}$ and $\{j, l\}$ are called *non-max-distance pairs* of $\{i, j, l\}$. We say that an element $i \in X$ is *satisfied* if it is not part of any conflict. For $Y \subseteq X$ the restriction of D to Y is denoted by $D[Y]$ and is called the distance function *induced* by Y. *Removing* an element $i \in X$ from (X, D, k) is defined by replacing (X, D, k) with a new instance (X', D', k) where $X' := X \setminus \{i\}$ and $D' := D[X \setminus \{i\}]$.

For some of our data reduction rules we use notation from graph theory. In this chapter, we only consider undirected, simple graphs $G = (V, E)$. See Section 2.6 for basic graph notation. For a distance function D over a set X, let $G_{\perp}(D, X) := (X, E)$, where $E := \{\{i, j\} \mid \exists_{l \in X}\{i, j, l\} \text{ is a conflict}\}$, that is, there is an edge between two elements if and only if they appear together in some conflict.

Regarding kernelizations and depth-bounded search trees, we use the notation introduced in Section 2.4 and Section 2.5, respectively.

6.3 A Decomposition Property and Two Search Tree Strategies

This section is organized as follows. First, we describe a simple $O(3^k \cdot n^3)$-time search tree algorithm for M-HTC that is the basis of our implementation (see Section 6.5.1). Then, we establish a simple decomposition property that, roughly speaking, implies that a set of conflicts not intersecting with any other conflicts can be "resolved" independently from the rest of the instance. Finally, using this decomposition property, we describe a refined search tree algorithm with running time $O(2.562^k \cdot n^3)$. Note that the decomposition property is also important for our kernelization algorithms presented in Section 6.4.

We describe the simple search tree strategy by a branching rule. Recall that, given an instance (X, D, k), a branching rule creates $\ell \geq 1$ subinstances (X_i, D_i, k_i), $1 \leq i \leq \ell$. A branching rule is called correct if (X, D, k) is a yes-instance if and only if (X_i, D_i, k_i) is a yes-instance for some i, $1 \leq i \leq \ell$. See Section 2.5 for more details.

Branching Rule 1. Let i, j, l denote three distinct elements such that $\{i, j, l\}$ is a conflict where $\{i, j\}$ is the max-distance pair. Proceed as follows.

- Branch into the case to decrease the distance of the max-distance pair $\{i, j\}$ by one and set $k := k - 1$.
- Branch into the case to increase the distance between $\{i, l\}$ by one and set $k := k - 1$.
- Branch into the case to increase the distance between $\{j, l\}$ by one and set $k := k - 1$.

For the correctness of Branching Rule 1 note the following. If the distance between the max-distance pair is not decreased and none of the other two distances is increased, then $\{i, j, l\}$ remains a conflict. Hence, at least one of the above changes must be applied.

The running time of the above algorithm can be seen as follows: The depth of the search tree is bounded by k, because the algorithm decreases the parameter in each branching step. Furthermore, we branch into three cases in every node of the search tree. Hence, the number of search tree nodes is $O(3^k)$. The steps that have to be performed at each search tree node (for example, finding a conflict to branch on) can be clearly performed in $O(n^3)$ time. Hence, the overall running time of this simple branching strategy is bounded by $O(3^k \cdot n^3)$.

Proposition 6.1. *M-Hierarchical Tree Clustering can be solved in $O(3^k \cdot n^3)$ time.*

From the above branching strategy it is obvious that the maximum distance $M+1$ is not increased in the course of the algorithm. Since the search tree algorithm finds all closest ultrametrics with distance at most k to D, we can observe the following (see Harb et al. [104, Lemma 1] for a similar statement).

Observation 6.1. *Let D be a distance function over a set X with maximum distance m. Then, the maximum distance in every closest ultrametric for D is at most m.*

Recall that $G_\perp(D, X)$ denotes the graph on X that contains an edge $\{i, j\}$ if and only if i and j are contained together in a conflict (see Section 6.2). Next, we show that for each connected component C of $G_\perp(D, X)$ the conflicts of $D[C]$ can be resolved independently from the conflicts of $D[X \setminus C]$. As a consequence, if all conflicts are disjoint, then M-HTC can be solved in polynomial time. This is exploited by our refined search tree strategy. Moreover, based on this fact, some of the data reduction rules to be presented in Section 6.4 work by partitioning the input instance into small subinstances that can be handled independently.

Lemma 6.1. *Let D be a distance function over a set X and let $C_1, \ldots, C_\ell$ denote the connected components of $G_\perp(D, X)$. Let D_s be a closest ultrametric for $D[C_s]$, $1 \leq s \leq \ell$. Then, D' with*

$$D'(i,j) := \begin{cases} D_s(i,j), & \text{if there is an } s,\ 1 \leq s \leq \ell, \text{ with } i,j \in C_s, \\ D(i,j), & \text{otherwise,} \end{cases}$$

is a closest ultrametric for D.

Proof. First, observe that $||D' - D||_1$ is a lower bound for the distance between D and any ultrametric since the C_s's are disjoint and each D_s is a closest ultrametric for $D[C_s]$.

Second, we show that D' is an ultrametric. Assume toward a contradiction that there is a conflict $\{i, j, l\}$ in D'. Since there are no conflicts within each connected component, there is a connected component C_s with $|C_s \cap \{i, j, l\}| = 1$. Without loss of generality, assume that $C_1 \cap \{i, j, l\} = \{l\}$. Moreover, note that i and j are in the same connected component since the distances between elements from different connected components in D and D' are equal and in D there are no conflicts containing elements from different components. Hence, without loss of generality, assume that $i, j \in C_2$, $C_1 \neq C_2$.

We use the following observation on the structure of D and $G_\perp(D, X)$. We argue that there is a positive integer m such that $D(l, i') = m$ for all $i' \in C_2$. To this end, consider the following partition of C_2. Let $X_r := \{i' \in C_2 \mid D(l, i') = r\}$ for $1 \leq r \leq M + 1$. Assume toward a contradiction that at least two different X_r's are nonempty. First, observe that for any $1 \leq r \leq M + 1$ and for any two elements $i', j' \in X_r$, we have that $D(i', j') \leq r$ since otherwise $\{i', j', l\}$ forms a conflict in D containing elements from two different connected components of $G_\perp(D, X)$. Furthermore, note that for $1 \leq r' < r \leq M+1$ and for every $i' \in X_r$ and $j' \in X_{r'}$ it holds that $D(i', j') = r$ since otherwise $\{i', j', l\}$ forms a conflict because $D(l, i') = r > r' = D(l, j')$. Hence, if one chooses three elements from at least two different X_r's, then two of the three distances are equal and are at least the third distance. As a consequence, within C_2 there are no conflicts with elements from different X_r's; a contradiction to the fact that C_2 is connected.

In summary, l has the same distance m to all elements in C_2. Moreover, since l does not form a conflict with any two vertices in C_2, it holds that $D(i', j') \leq m$ for all $i', j' \in C_2$. That is, m is the maximum distance in $D[C_2]$. Thus, according to Observation 6.1 this maximum distance m will not be increased by a closest ultrametric, and, as a consequence, $D'(i, j) \leq m$. In summary, $D(l, i) = D'(l, i) = m$, $D(l, j) = D'(l, j) = m$, and $D'(i, j) \leq m$; a contradiction to the assumption that $\{i, j, l\}$ is a conflict in D'. □

Lemma 6.1 implies that for every closest ultrametric U the distance between two elements from different connected components of $G_\perp(D, X)$ is not changed and that, for every connected component C of $G_\perp(D, X)$, $U[C]$ is a closest ultrametric for $D[C]$.

Corollary 6.1. *Let D be a distance function over a set X and let $C_1, \ldots, C_\ell$ denote the connected components of $G_\perp(D, X)$. Moreover, let U denote a closest ultrametric for D. Then, $U[C_r]$ is a closest ultrametric for $D[C_r]$ and $U(i, j) = D(i, j)$ for all $1 \leq r < r' \leq \ell$ and for all $i \in C_r$ and $j \in C_{r'}$.*

In the following, we provide a refined branching strategy. The basic idea is as follows. If there are two conflicts that are intersecting in two elements, then we can give a better branching strategy. Moreover, we will show that otherwise all conflicts are element-disjoint and, hence, according to Lemma 6.1, can be resolved independently from each other. We describe our refined search tree algorithm by two branching rules.

The first branching rule branches on conflicts that have the same max-distance pair.

Branching Rule 2. Let $i, j, l, h \in X$ denote four distinct elements such that $\{i, j, l\}$ and $\{i, j, h\}$ form conflicts where $\{i, j\}$ is the max-distance pair of both conflicts. Proceed as follows.

- Branch into the case to decrease the distance of $\{i, j\}$ by one and set $k := k - 1$.
- Branch into the four cases to increase the distance of $\{l, x\}$ and $\{h, y\}$, for all combinations of $x, y \in \{i, j\}$ and, in each branch, set $k := k-2$.

For the correctness of this branching strategy note the following. In the first branch, the branching rule covers the case that the distance of the max-distance pair is decreased (by at least one). Hence, in the following cases, one can assume that the distance of the max-distance pair is not decreased. As a consequence, at least one of the distances of $\{l, i\}$ and $\{l, j\}$ and at least one of the distances of $\{h, i\}$ and $\{h, j\}$ must be increased. Clearly, the rule covers all four possibilities.

The next branching rule branches on a pair of conflicts sharing two elements that are not the max-distance pairs of both conflicts.

Branching Rule 3. Let $i, j, l, h \in X$ denote four distinct elements such that $\{i, j, l\}$ and $\{i, j, h\}$ form conflicts where $\{i, j\}$ is neither the max-distance pair of $\{i, j, l\}$ nor of $\{i, j, h\}$. Proceed as follows.

- Branch into the case to increase the distance of $\{i, j\}$ by one and set $k := k - 1$.
- Branch into the four cases to
  - decrease the distance of both max-distance pairs by one and set $k := k - 2$, or
  - decrease the distance of the max-distance pair of $\{i, j, l\}$ and increase the distance of the non-max-distance pair of $\{i, j, h\}$ containing h and set $k := k - 2$, or
  - decrease the distance of the max-distance pair of $\{i, j, h\}$ and increase the distance of the non-max-distance pair of $\{i, j, l\}$ containing l and set $k := k - 2$, or
  - increase the distance of both non-max-distance pairs containing l or h and set $k := k - 2$.

Observe that, in the first branch, the rule covers the case that the distance of the pair contained in both conflicts is increased. Hence, for the other cases one can assume that the distance of the common pair is not increased. Hence, for each of the two conflicts either the max-distance pair must be decreased or the non-max-distance pair different from $\{i, j\}$ must be increased. Clearly, all four possibilities are covered by the branching rule.

For a branching rule creating $\ell \geq 2$ subinstances, a branching vector is an ℓ-tuple that describes how the parameter is decreased in each subinstance. For example, the branching vector of Branching Rules 2 and 3 is $(1, 2, 2, 2, 2)$. Moreover, the corresponding branching number describing the base of the exponential search-tree size is 2.562. See Section 2.5 and Niedermeier [137, Chapter 8] for more details. In the remainder of this section, we show that if none of the two branching rules can be applied, then

M-HTC can be solved in polynomial time, leading to a search tree algorithm with running time $O(2.562^k \cdot n^3)$. First, we show that if there are two conflicts intersecting in at least one element, then we can apply either Branching Rule 2 or 3. Conversely, if none of Branching Rule 2 or 3 applies, then all conflicts are element-disjoint.

Lemma 6.2. *If there are two distinct conflicts C and C' with $C \cap C' \neq \emptyset$, then either Branching Rule 2 or Branching Rule 3 can be applied.*

Proof. First, we show that if there are two conflicts C and C' with $|C \cap C'| = 1$, then there is a conflict C'' with $C'' \subseteq C \cup C'$ such that $|C \cap C''| = 2$ or $|C' \cap C''| = 2$. To this end, let $C = \{i, j, l\}$ and $C' = \{l, p, q\}$ (that is, $|C \cap C'| = 1$). Furthermore, assume without loss of generality that $\{i, l\}$ is not the max-distance pair of C and $\{l, p\}$ is not the max-distance pair of C'. Moreover, let m and m' denote the distances of the max-distance pairs of C and C', respectively, and assume without loss of generality that $m' \geq m$. First, note that $D(i, l) < m'$ and $D(l, p) < m'$. Thus, if $D(i, p) \geq m'$, then $C'' := \{i, l, p\}$ forms a conflict such that $C' \cap C'' = \{l, p\}$. Hence, in the following, we assume that $D(i, p) < m'$. We distinguish the cases that either $\{p, q\}$ or $\{l, q\}$ is the max-distance pair in C'.

Case 1: $D(p, q) = m'$. Since $D(p, q)$ is the max-distance pair in the conflict $C' = \{l, p, q\}$ it holds that $D(l, p) < m'$ and $D(l, q) < m'$. First, if $D(i, q) \geq m'$, then $\{i, l, q\}$ forms a conflict since $D(l, q) < m'$ and $D(i, l) < m'$ (recall that we assume that $D(i, l)$ is not the max-distance pair in $\{i, j, l\}$ and, thus, $D(i, l) < m \leq m'$). Clearly, $C' \cap \{i, l, q\} = \{l, q\}$. Second, if $D(i, q) < m'$, then $\{i, p, q\}$ forms a conflict, since $D(i, p) < m'$ and $D(i, q) < m'$ but $D(p, q) = m'$. Note that $C' \cap \{i, p, q\} = \{p, q\}$.

Case 2: $D(l, q) = m'$. First, if $D(i, q) \geq m'$, then $C'' := \{i, p, q\}$ forms a conflict with $C' \cap C'' = \{p, q\}$ since $D(i, p) < m'$ and $D(p, q) < m'$. Second, if $D(i, q) < m'$, then $C'' := \{i, l, q\}$ forms a conflict with $C' \cap C'' = \{l, q\}$ since $D(i, l) < m \leq m'$ and $D(i, q) < m'$ but $D(l, q) = m'$.

In summary, we have shown that if there are two conflicts with nonempty intersection, then there are two conflicts intersecting in *two* elements.

Next, we show that if there are two conflicts intersecting in two elements, then there are two conflicts $\{i, j, p\}$ and $\{i, j, l\}$ such that $\{i, j\}$ is either the max-distance pair in both conflicts or not the max-distance pair in both $\{i, j, p\}$ and $\{i, j, l\}$ and, hence, either Branching Rule 2 or Branching Rule 3 applies.

Assume that this is not the case. That is, without loss of generality, $\{i, j\}$ is the max-distance pair of $\{i, j, p\}$ but not the max-distance pair of $\{i, j, l\}$. Moreover, without loss of generality assume that $\{i, l\}$ is the max-distance pair of $\{i, j, l\}$. First, observe that $D(i, j) < D(i, l)$ since $\{i, l\}$ is the max-distance pair in $\{i, j, l\}$. Moreover, since $D(i, j)$ is the max-distance pair in $\{i, j, p\}$ it holds that $\max\{D(i, p), D(j, p)\} < D(i, j) < D(i, l)$. Hence, if $D(p, l) < D(i, l)$, then $\{l, p, i\}$ forms a conflict with max-distance pair $\{i, l\}$. Otherwise, if $D(p, l) \geq D(i, l)$, then $\{p, j, l\}$ forms a conflict since $D(j, l) < D(i, l)$ and $D(j, p) < D(i, l)$ but $D(p, l) \geq D(i, l)$. Moreover, the max-distance pair of $\{p, j, l\}$ is $\{p, l\}$. Thus, $\{j, l\}$ is neither the max-distance pair of $\{p, j, l\}$ nor the max-distance pair of $\{i, j, l\}$. □

By Lemma 6.2, if neither Branching Rule 2 nor Branching Rule 3 applies, then all conflicts are element-disjoint. A conflict that does not intersect with any other conflict is called isolated. If all conflicts are isolated, then, according to Lemma 6.1, all conflicts can be solved independently from each other in polynomial time. Note

that for an isolated conflict it is optimal to set the distance between the elements of the max-distance pair to the maximum of the other two distances.

Observation 6.2. *If all conflicts are isolated, then M-HTC can be solved in polynomial time.*

Clearly, in $O(n^3)$ time one can find two conflicts for which either Branching Rule 2 or Branching Rule 3 applies, or decide that such two conflicts do not exist. Thus, altogether, we arrive at the following.

Theorem 6.1. *M-Hierarchical Tree Clustering can be solved in $O(2.562^k \cdot n^3)$ time.*

6.4 Two Kernelization Results

In this section, we present our main theoretical results, two kernelization algorithms for M-HTC which achieve problem kernels with $O(k^2)$ and $O(M \cdot k)$ elements, respectively. Both algorithms partition the input instance into small subinstances and handle these subinstances independently. This partitioning is based on Lemma 6.1.

We present our kernelization algorithms by describing a set of *data reduction rules*. A data reduction rule is called *correct* if the new instance after an application of this rule is a yes-instance if and only if the original instance is a yes-instance. An instance is called *reduced* with respect to a set of data reduction rules if each of the data reduction rules has been exhaustively applied. See Section 2.4 for more details.

6.4.1 An $O(k^2)$-Element Problem Kernel

Our first and simpler kernelization algorithm uses two data reduction rules which handle two extremal cases concerning the elements; the first rule corrects the distance between two elements which together appear in many conflicts, while the second rule safely removes elements which are not in any conflict. Basically, these two rules are adaptions of reduction rules that lead to an $O(k^2)$-vertex problem kernel for Cluster Editing [83]. The main difference is that removing vertices that are not in conflict is trivial for Cluster Editing but for the more general M-HTC the correctness of this rule is not that obvious. We show the correctness of the second rule using the decomposition property from Lemma 6.1.

Let $I = (X, D, k)$ denote an instance of M-HTC. The first data reduction rule is based on the observation that the distance between two elements $i, j \in X$ that either occur as max-distance pair in more than k conflicts or occur as non-max-distance pair in more than k conflicts must be adjusted.

Reduction Rule 6.1. If there is a pair $\{i, j\} \subseteq X$ which is the max-distance pair (or not the max-distance pair) in at least $k + 1$ conflicts, then decrease (or increase) the distance $D(i, j)$ by one and decrease the parameter k by one.

For Reduction Rule 6.1, we assume that $k \geq 0$ before its application since otherwise we can reject the instance immediately.

Lemma 6.3. *Reduction Rule 6.1 is correct.*

Proof. Let I be an instance to which Reduction Rule 6.1 is applied and let I' be the resulting instance. We show that I is a yes-instance if and only if I' is a yes-instance.

We only show the correctness of the first part of the rule, that is, $\{i, j\}$ is the max-distance pair in at least $k + 1$ conflicts. The second part can be shown analogously. To this end, we show that every solution conducting at most k changes on D must decrease $D(i, j)$. Suppose toward a contradiction that this is not the case. Then, in each of the $k + 1$ conflicts that contain $\{i, j\}$ the distance of one of the other two pairs has to be changed. Since the conflicts that contain $\{i, j\}$ have no other pair in common, this means that we have to modify at least $k + 1$ distances. □

As mentioned above, the second rule is based on Lemma 6.1. Consider a satisfied element x (that is, x is not part of any conflict). Observe that in the conflict graph $G_\perp(D, X)$ (see Section 6.2) the element x is an isolated vertex. Moreover, according to Lemma 6.1, one obtains a closest ultrametric by choosing a closest ultrametric for every connected component for $G_\perp(D, X)$. Thus, since the connected component $\{x\}$ induces an ultrametric, x can be safely removed.

Reduction Rule 6.2. Remove all satisfied elements.

Lemma 6.4. *Reduction Rule 6.2 is correct.*

Proof. Let D be a distance function over a set X, and let $x \in X$ be a satisfied element. Moreover, let (X', D', k) with $X' := X \setminus \{x\}$ and $D' := D[X']$ denote the instance that results by removing x from (X, D, k).

Observe that every conflict in D is also a conflict in D'. As a consequence, $G_\perp(D, X)$ and $G_\perp(D', X')$ contain the same set of connected components with the exception of $\{x\}$ (note that $\{x\}$ is a connected component in $G_\perp(D, X)$ but does not exist in $G_\perp(D', X')$). Moreover, the two distance functions induced by a connected component different from $\{x\}$ are equal. Thus, since $\{x\}$ induces an ultrametric, it follows directly by Lemma 6.1 that (X, D, k) is a yes-instance if and only if (X', D', k) is a yes-instance. □

Finally, we show that a yes-instance reduced with respect to the two data reduction rules presented in this section contains at most $k \cdot (k + 2)$ elements. Consequently, for any reduced instanced with more than $k \cdot (k + 2)$ elements we can output "no" and only instances with at most $k \cdot (k + 2)$ elements remain.

Theorem 6.2. *M-Hierarchical Tree Clustering admits a problem kernel with $k \cdot (k + 2)$ elements. The running time for the kernelization is $O(M \cdot n^3)$.*

Proof. Let (X, D, k) be a yes-instance that is reduced with respect to Reduction Rules 6.1 and 6.2. That is, there exists an ultrametric D' on X with $||D - D'|| \leq k$. We show that $|X| \leq k \cdot (k+2)$. For the analysis of the kernel size we partition the elements of X into two subsets A and B, where $A := \{i \in X \mid \exists j \in X : D'(i, j) \neq D(i, j)\}$ and $B := X \setminus A$. Note that $|A| \leq 2k$ since D' has distance at most k to D. Hence, it remains to show that $|B| \leq k^2$. Let $S := \{\{i, j\} \subseteq X \mid D'(i, j) \neq D(i, j)\}$ denote the set of pairs whose distances have been modified, and for each $\{i, j\} \in S$ let $B_{\{i,j\}}$ denote the elements of B that form a conflict together with i and j. Since the input instance is reduced with respect to Reduction Rule 6.2, we have $B = \bigcup_{\{i,j\} \in S} B_{\{i,j\}}$. Observe that $\{i, j\}$ is either the max-distance pair in all conflicts of $B_{\{i,j\}}$ or a non-max-distance pair in all conflicts of $B_{\{i,j\}}$. Thus, since the input instance is reduced

with respect to Reduction Rule 6.1, we have $|B_{\{i,j\}}| \leq k$ for all $\{i,j\} \in S$. The size bound $|B| \leq k^2$ then immediately follows from $|S| \leq k$.

The running time can be achieved as follows. First, we calculate for each pair of elements the number of conflicts in which it is the max-distance pair and the number of conflicts in which it is not the max-distance pair in $O(n^3)$ time. Then we check whether Reduction Rule 6.1 can be applied. If this is the case, we update in $O(n)$ time the number of conflicts for all pairs that contain at least one of the elements whose distance has been modified. This is repeated as long as $k \geq 0$ and a pair to which Reduction Rule 6.1 can be applied has been found. Since in each application of Reduction Rule 6.1 parameter k is decreased, it is applied at most $O(k)$ times. Hence, the overall running time of exhaustively applying Reduction Rule 6.1 is $O(n^3 + k \cdot n)$. Afterwards, we exhaustively apply Reduction Rule 6.2 in $O(n^3)$ total time. Finally, note that every triple of elements that forms a conflict prior to the application of Reduction Rule 6.2 forms a conflict after its application. Hence, the application of Reduction Rule 6.2 does not leave an instance to which Reduction Rule 6.1 can be applied again. Finally, note that we can assume that $k < M \cdot n^2$. Hence, the running time of the kernelization is $O(n^3 + k \cdot n) = O(M \cdot n^3)$. □

Using the standard technique of interleaving search trees with kernelization (see Section 2.5), one can improve the worst-case running time of the search tree algorithm from Section 6.3. As our experiments show (see Section 4), there is also a speed-up in practice.

Corollary 6.2. *M-HTC can be solved in $O(2.562^k + M \cdot n^3)$ time.*

6.4.2 An $O(M \cdot k)$-Element Problem Kernel

Our second kernelization algorithm extends the basic idea of an $O(k)$-element problem kernel for Cluster Editing [89]. However, since the clusterings required by Cluster Editing have no hierarchical structure, this extension is nontrivial and needs more technical effort, in particular, the correctness proof of the data reduction rule is far more involved than in the Cluster Editing case.

We use the following notation. For a distance function D over a set X and an integer t with $1 \leq t \leq M$, the *t-threshold graph* $G_{D,X}(t)$ is defined as (X, E_t) with $\{i,j\} \in E_t$ if and only if $D(i,j) \leq t$. If D is an ultrametric, then, for each $1 \leq t \leq M$, the corresponding graph $G_{D,X}(t)$ is a *cluster graph*, that is, a disjoint union of *cliques*. These cliques of $G_{D,X}(t)$ are called *t-clusters*. Moreover, for a distance function D over a set X define $\alpha_D := \max\{D(i,j) \mid i,j \in X\} - 1$. That is, $\alpha_D + 1$ is the maximum distance in D. A clique K is a *critical clique* if all its vertices have an identical closed neighborhood and K is maximal under this property (see Definition 2.4).

The kernelization algorithm employs only one data reduction rule. This rule works on the t-threshold graphs, beginning with $t = \alpha_D$, and is then applied recursively for smaller values of t. It applies a procedure (*Critical-Clique*, see Figure 6.3) which deals with large critical cliques in t-threshold graphs.

Reduction Rule 6.3. Let $I = (D, X, k)$ denote an M-HTC-instance. Call procedure *RR3* (Figure 6.3) with parameters X and α_D. Herein, let I be the respective global variable I^*.

Global variable: An instance $I^* = (X, D^*, k^*)$ of M-HTC.

Procedure: *RR3*
Input: A set $X' \subseteq X$ and an integer $t \geq 1$.

1. *Critical-Clique*(X', t).
2. Let $G_t := G_{D^*[X'],X'}(t)$.
3. **for** each isolated clique K in G_t **do**
   (a) **if** $D^*[K]$ is an ultrametric
   (b) **then** remove K from I^* and G_t.
   (c) **else** *RR3*$(K, t-1)$

Procedure: *Critical-Clique*
Input: A set $X' \subseteq X$ and an integer $t \geq 1$.

1. Construct $G_t := G_{D^*[X'],X'}(t)$.
2. **while** G_t contains a nonisolated critical clique K with $K \subseteq X'$ such that
   - $|K| \geq t \cdot |N_{G_t}(K)|$ and
   - $|K| \geq |N_{G_t}(K)| + t \cdot |N^2_{G_t}(K)|$
3. **do**
   (a) For all $x \in N_{G_t}[K]$ and $y \in X' \setminus (N_{G_t}[K])$, set $D^*(x, y) := t + 1$.
   (b) For all $x, y \in N_{G_t}(K)$ with $D^*(x, y) = t + 1$, set $D^*(x, y) := t$.
   (c) Decrease the parameter k^* correspondingly, that is, by the distance between the original and the new instance.
   (d) Update $G_t := G_{D^*[X'],X'}(t)$.
   (e) **if** $k^* < 0$ **then** return "no".
4. **end while**

Figure 6.3: The *Critical-Clique* procedure deals with large critical cliques in t-threshold graphs. It is recursively applied by procedure *RR3*. Both procedures use the global variable $I^* = (X, D^*, k^*)$.

In the following, we first show that the *Critical-Clique* procedure is correct when it is applied to $X' = X$ and $t = \alpha_D$. Then, based on the correctness for $X' = X$ and $t = \alpha_D$, we will show the correctness of Reduction Rule 6.3.

The following two lemmas are essential for our proof. In the case of Cluster Editing (and, hence, for 1-HTC) it holds that a critical clique in the input graph is entirely contained in a clique of the final cluster graph (see [89, Lemma 1] and Lemma 3.7). In contrast, it is not hard to see that an arbitrary critical clique in $G_{D,X}(\alpha_D)$ is not necessarily subset of an α_D-cluster of a closest ultrametric. However, we can show that a critical clique that contains α_D times more vertices than it has neighbors is not split, that is, it is entirely contained in an α_D-cluster of a closest ultrametric.

Lemma 6.5. *Let D be a distance function over a set X and let $\alpha := \alpha_D$. Moreover, let K be a critical clique in $G_\alpha := G_{D,X}(\alpha)$ with $|K| \geq \alpha \cdot |N_{G_\alpha}(K)|$. Then, there exists a closest ultrametric U that contains an α-cluster C such that*

1. *$K \subseteq C$ and*
2. *$C \subseteq N_{G_\alpha}[K]$.*

Proof. To prove part 1, we show that for every closest ultrametric U there exists an α-cluster C of U with $K \subseteq C$. Assume toward a contradiction that K is not contained in an α-cluster of U. Then, for an $\ell \geq 2$, there exist α-clusters $C_1, \ldots, C_\ell$ with $C_i \cap K \neq \emptyset$. Define

- $K_i := K \cap C_i$,
- $A_i := N_{G_\alpha}(K) \cap C_i$, and
- $R_i := C_i \setminus (K_i \cup A_i)$.

Furthermore, let $A := \bigcup_{i=1}^{\ell} A_i$ and note that $A \subseteq N_{G_\alpha}(K)$.

First, we show that there exist two integers i, j, with $1 \leq i, j \leq \ell$, $i \neq j$ such that $|R_i|+|K_j|+|A_j| > \alpha \cdot |A_i|+|R_j|$. Assume toward a contradiction that $|R_i|+|K_j|+|A_j| \leq \alpha \cdot |A_i| + |R_j|$ for all $1 \leq i, j \leq \ell$. This clearly implies $\sum_{i=1}^{\ell}(|R_i| + |K_{i+1}| + |A_{i+1}|) \leq \sum_{i=1}^{\ell}(\alpha \cdot |A_i| + |R_{i+1}|)$ (herein, $X_{\ell+1} = X_1$ for $X \in \{K, A, R\}$). Note that the above inequality is equivalent to $|K| + |A| \leq \alpha \cdot |A|$; a contradiction to the fact that $|K| \geq \alpha \cdot |N_{G_\alpha}(K)|$.

Let i and j with $1 \leq i, j \leq \ell$, $i \neq j$, be two integers such that $|R_i| + |K_j| + |A_j| > \alpha \cdot |A_i| + |R_j|$. We show that "adding" K_i to the α-cluster C_j yields an ultrametric U' with smaller distance to D than U has to D, contradicting the assumption that U is a closest ultrametric. More specifically, building U' by "adding" K_i to C_j means setting

$$U'(x, y) := \alpha + 1 \text{ if } x \in K_i \text{ and } y \in A_i \cup R_i;$$
$$U'(x, y) := \alpha \text{ if } x \in K_i \text{ and } y \in C_j;$$
$$U'(x, y) := U(x, y), \text{ otherwise.}$$

Next, we show that U' is an ultrametric. Assume toward a contradiction that U' contains a conflict $F := \{p, q, r\}$. In the construction of U' from U only distances from elements in K_i to elements in $X \setminus K_i$ are changed. Hence, F contains one or two elements from K_i. Let $R := X \setminus (K_i \cup C_j)$. Observe that $U'(x, y) = \alpha + 1$ for all $x \in K_i$ and $y \in R$, and $U'(x, y) = \alpha + 1$ for all $x \in K_i \cup C_j$ and $y \in R$. Hence, $F \cap R = \emptyset$ since, otherwise, two of the three distances between the elements of F are $\alpha + 1$ (and $\alpha + 1$ is the maximum distance). Thus, $F \subseteq K_i \cup C_j$ and $F \cap C_j \neq \emptyset$. Since C_i and C_j are α-clusters of U and by the construction of U' from U, it holds that $U'(x, y) = \alpha$ for all $x \in K_i$ and $y \in C_j$ and $U'(x, y) \leq \alpha$ for all $x, y \in K_i \cup C_j$. This, however, implies that two of the three distances of U' between the elements in F are α and the third distance is at most α; a contradiction to the assumption that F is a conflict in U'.

For part 1 of Lemma 6.5, it remains to show that $||U' - D||_1 < ||U - D||_1$, contradicting the assumption that U is a closest ultrametric. Note that by construction, we have:

- for all $x \in K_i$ and $y \in A_i$ it holds that $U'(x, y) = \alpha + 1$, $D(x, y) \leq \alpha$, and $U(x, y) \leq \alpha$, and
- for all $x \in K_i$ and $y \in R_j$ it holds that $U'(x, y) = \alpha < \alpha + 1 = U(x, y) = D(x, y)$.

Hence, in comparison with transforming D to U the extra costs of transforming D to U' are at most $|K_i| \cdot (\alpha \cdot |A_i| + |R_j|)$. However, we save at least $|K_i| \cdot (|R_i| + |A_j| + |K_j|)$ since

- for all $x \in K_i$ and $y \in R_i$ it holds $U(x,y) \leq \alpha < \alpha+1 = U'(x,y) = D(x,y)$, and
- for all $x \in K_i$ and $y \in A_j \cup K_j$ it holds $D(x,y) \leq \alpha = U'(x,y) < \alpha+1 = U(x,y)$.

Finally, $\alpha \cdot |A_i| + |R_j| < |R_i| + |A_j| + |K_j|$ implies $||U'-D||_1 < ||U-D||_1$; a contradiction to the assumption that U is a closest ultrametric.

Next, we prove part 2 of Lemma 6.5. From the above argument, every closest ultrametric U for D contains an α-cluster C with $K \subseteq C$. Now, assume that $R := C \setminus N_{G_\alpha}[K]$ is not empty. Let $A := N_{G_\alpha}(K) \cap C$. We build a new ultrametric U' fulfilling the conditions of the lemma as follows: in comparison with U, ultrametric U' contains an additional α-cluster R, that is, for all $i \in K \cup A$ and $j \in R$, set $U'(i,j) := \alpha+1$. For each other pair of elements the distance in U' is equal to the distance in U. Thus, U' is clearly an ultrametric. Moreover, note that in comparison of transforming D into U the extra costs for transforming D into U' are at most $|R| \cdot \alpha \cdot |A|$. However, we save $|R| \cdot |K|$ by setting the distance between an element in R and an element in K to $\alpha+1$ (note that $D(x,y) = \alpha+1$ for all $x \in R$ and $y \in K$). Since $|K| \geq \alpha \cdot |N_{G_\alpha}(K)|$, it holds that $||U'-D||_1 \leq ||U-D||_1$. This completes the proof of part 2. □

The next lemma shows that critical cliques that are large compared to their neighborhood *and* second neighborhood form, together with their neighborhood, an α_D-cluster of a closest ultrametric. These are precisely the cliques fulfilling the while-condition in Line 2 of the *Critical-Clique* procedure (see Figure 6.3).

Lemma 6.6. *Let D be a distance function over a set X and let $\alpha := \alpha_D$. Moreover, let K be a critical clique in $G_\alpha := G_{D,X}(\alpha)$ with*

- $|K| \geq \alpha \cdot |N_{G_\alpha}(K)|$ *and*
- $|K| \geq |N_{G_\alpha}(K)| + \alpha \cdot |N^2_{G_\alpha}(K)|$.

Then, there exists a closest ultrametric U for D such that $N_{G_\alpha}[K]$ is an α-cluster in U.

Proof. Let U' denote a closest ultrametric for D. By Lemma 6.5, we can assume that U' has an α-cluster C such that $K \subseteq C \subseteq N_{G_\alpha}[K]$. If $C = N_{G_\alpha}[K]$, then we are done. Hence, in the following we consider the case $K \subseteq C \subset N_{G_\alpha}[K]$. We show that, given U', one can build a closest ultrametric U such that $N_{G_\alpha}[K]$ is an α-cluster of U. Let $B := N_{G_\alpha}[K] \setminus C$. Observe that $K \subseteq C$ implies that $B \subseteq N_{G_\alpha}(K)$. We show that "adding" B to C yields another closest ultrametric. More specifically, given U', build U as follows. Start with $U := U'$. Then, change the distances between B and $X \setminus B$ as follows.

1. "Separate" the elements in B from their respective α-clusters by setting $U(i,j) := \alpha+1$ for all $i \in B$ and $j \in X \setminus (B \cup C)$.
2. Transform B into an α-cluster by setting $U(i,j) := \min(\alpha, U'(i,j))$ for all $i \in B$ and $j \in B$.
3. "Merge" the α-clusters C and B by setting $U(i,j) := \alpha$ for $i \in B$ and $j \in C$.

It is not hard to verify that U is an ultrametric after each of the three steps above. Hence, it remains to show that $||U - D||_1 \leq ||U' - D||_1$. Since by definition $D(x, y) = U(x, y) = \alpha + 1$ for all $x \in B$ and $y \in X \setminus (N_{G_\alpha}[K] \cup N^2_{G_\alpha}(K))$, we consider only the distances between elements in B and $N_{G_\alpha}[K] \cup N^2_{G_\alpha}(K)$ (cases where for $x \in B$ and $y \in X \setminus (N_{G_\alpha}[K] \cup N^2_{G_\alpha}(K))$ it holds that $U'(x, y) < D(x, y) = U(x, y) = \alpha + 1$ decrease the distance between U and D but do not occur in the worst case). We analyze the three steps above separately.

First, observe that $D(i, j) \leq \alpha + 1$, $U'(i, j) \leq \alpha + 1$, and $U(i, j) = \alpha + 1$ for all $i \in B$ and $j \in N^2_{G_\alpha}(K)$. Hence, in the first step, the distance of U to D increases by at most $|B| \cdot \alpha \cdot |N^2_{G_\alpha}(K)|$ (the worst case occurs when $D(i, j) = U'(i, j) = 1$ for all $i \in B$ and $j \in N^2_{G_\alpha}(K)$).

Second, observe that $D(i, j) \leq \alpha+1$, $U'(i, j) \leq \alpha+1$ and $U(i, j) = \min(\alpha, U'(i, j))$ for all $i, j \in B$. Hence, in the second step, the distance of U to D increases by at most $|B| \cdot |B|$ (the worst case occurs when $U'(i, j) = D(i, j) = \alpha + 1$ for all $i, j \in B$).

For the third step, let $C' := C \setminus K$. Note that $C' \subseteq N_{G_\alpha}(K)$. We separately analyze the distances between B and C' and the distances between B and K. Observe that $D(i, j) \leq \alpha + 1 = U'(i, j)$ and $U(i, j) = \alpha$ for all $i \in B$ and $j \in C'$. Hence, the distance of U to D increases by at most $|B| \cdot |C'|$ by changing the distances between B and C'. Finally, note that $D(i, j) \leq \alpha = U(i, j)$ and $U'(i, j) = \alpha + 1$ for all $i \in B$ and $j \in K$. Hence, the distance of U to D *decreases* by at least $|B| \cdot |K|$ by changing the distances between B and K.

In summary, the total increase of the distance of U to D is at most $|B| \cdot (\alpha |N^2_{G_\alpha}(K)| + |B| + |C'|)$ and the total decrease of the distance of U to D is at least $|B| \cdot |K|$. Since $B \cup C' \subseteq N_{G_\alpha}(K)$ and $|K| \geq |N_{G_\alpha}(K)| + \alpha \cdot |N^2_{G_\alpha}(K)|$, the decrease of the distance of U to D in the third step compensates the total increase of this distance caused by all other changes. □

Now, we are able to prove the correctness of the *Critical-Clique* procedure when called with X and α_D. More specifically, let D be a distance function over a set X and let $\alpha := \alpha_D$. Moreover, let K denote a critical clique of $G_\alpha := G_{D,X}(\alpha)$ fulfilling the while-condition in the *Critical-Clique* procedure (line 2 of *Critical-Clique* in Figure 6.3). Furthermore, let D' denote the distance function that results by executing lines (a) and (b) of *Critical-Clique* on K and let $d := ||D - D'||_1$. For the correctness of *Critical-Clique* we show the following.

Lemma 6.7. *(X, D, k) is a yes-instance if and only if $(X, D', k - d)$ is a yes-instance.*

Proof. "⇒": If (X, D, k) is a yes-instance, then, by Lemma 6.6, there exists an ultrametric U of distance at most k to D such that $N_{G_\alpha}[K]$ is an α-cluster of U. Hence, it must hold that $U(i, j) = \alpha+1$ for all $i \in N_{G_\alpha}[K]$ and $j \in X \setminus N_{G_\alpha}[K]$ and $U(i, j) \leq \alpha$ for all $i, j \in N_{G_\alpha}[K]$. Hence, the changes performed by *Critical-Clique* are necessary to obtain U.
"⇐": Let U denote an ultrametric with $||D' - U|| \leq k - d$. Thus, $||D - D'||_1 \leq d$ implies $||D - U||_1 \leq k$ by the triangle inequality. □

The correctness of Reduction Rule 6.3 is based on the correctness of *Critical-Clique* and the observation that all vertices of an isolated clique K in $G_{D,X}(\alpha_D)$ are not contained in any conflict with vertices from $X \setminus K$, and, hence, the subinstance induced by K can be solved independently according to Lemma 6.1. The details follow.

Lemma 6.8. *Reduction Rule 6.3 is correct.*

Proof. Let D be a distance function over a set X. The correctness of Reduction Rule 6.3 follows by induction on the maximum distance α_D+1 of the distance function.

For the induction base (that is, $\alpha_D = 1$) observe that for each isolated clique K in $G_{D,X}(1)$ it holds that $D(i,j) = 1$ for all $i,j \in K$ and $D(i,j) = 2$ for all $i \in K$ and $j \in X \setminus K$. As a consequence, K induces an ultrametric and each element in K is satisfied. Hence, the elements in K can be removed (see Reduction Rule 6.2 and Lemma 6.4). Thus, the recursion terminates, and the correctness of Reduction Rule 6.3 follows directly from the correctness of *Critical-Clique* (Lemma 6.7).

Assume that Reduction Rule 6.3 is correct for any instance (X'', D'', k'') with $\alpha_{D''} = \alpha - 1$. We show that then Reduction Rule 6.3 is correct for an instance (X, D, k) where $\alpha_D = \alpha$.

First, note that by Lemma 6.7 it is correct to apply the *Critical-Clique* procedure for $X' = X$ and $t = \alpha$. Moreover, observe that each vertex of an isolated clique K of $G_\alpha = G_{D,X}(\alpha)$ such that $D[K]$ is an ultrametric is satisfied: First, there is no conflict contained in K, and, second, since each element in K has distance $\alpha+1$ to all elements from $X \setminus K$, there is no conflict that contains vertices from both K and $X \setminus K$. Hence, these vertices can be removed (see Reduction Rule 6.2 and Lemma 6.4). Thus, it remains to show the correctness of line (c) of *RR3* (see Figure 6.3), that is, the correctness of recursively applying procedure *RR3* to the isolated cliques of G_α.

To this end, let K denote an isolated clique of G_α that does not induce an ultrametric. Moreover, let (X', D', k') denote the instance that results by calling $RR3(K, \alpha-1)$. For the correctness of the recursion, we show the following.

Claim. (X, D, k) is a yes-instance if and only if (X', D', k') is a yes-instance.

Let $Y := X \setminus K$ and $K' := X' \setminus Y$. By calling $RR3(K, \alpha - 1)$ only elements from K are removed. Hence, $K' \subseteq K$. Observe that $D'[K']$ equals the distance function that results from calling $RR3(K, \alpha-1)$ and setting the global distance function D^* to $D[K]$ and the global parameter k^* to k.

Next, we show that $D'(i,j) = D(i,j)$ for all $i \in X'$ and $j \in Y$, and $K' \subseteq K$ is an isolated clique in $G_{D',X'}(\alpha)$. By calling $RR3(K, \alpha - 1)$, only distances between elements of K are changed. Moreover, the maximum distance in $D'[K']$ is at most α since, first, the maximum distance in $D[K]$ is at most α, second, $t + 1 \leq \alpha$ in each recursive call of *RR3*, and, third, procedure *Critical-Clique* changes a distance to at most $t + 1$. Hence, $D'(i,j) \leq \alpha$ for all $i,j \in K'$ and $D'(i,j) = \alpha + 1$ for all $i \in K'$ and $j \in Y$, implying that K' is an isolated clique in $G_{D',X'}(\alpha)$.

Furthermore, note that there is no conflict $C = \{i,j,l\} \subseteq X$ with $K \cap C \neq \emptyset$ and $Y \cap C \neq \emptyset$: for any $i,j,l \in X$ with $i \in K$, $j \in Y$ and $l \in X$ two of the three distances $D(i,j)$, $D(i,l)$, and $D(j,l)$ are $\alpha + 1$ which is the maximum distance in D and, hence, $\{i,j,l\}$ does not form a conflict. The same argument implies that there is no conflict $C' = \{i,j,l\} \subseteq X'$ with $K' \cap C' \neq \emptyset$ and $Y \cap C' \neq \emptyset$. Hence there is no connected component in $G_\perp(D, X)$ ($G_\perp(D', X')$) containing vertices from both Y and K (K'). Recall that $G_\perp(D, X)$ denotes the graph with vertex set X that contains an edge $\{i,j\}$ if and only if there is a conflict containing both i and j (see Section 6.2).

For a distance function D'' over a set X'', define $\mathrm{opt}(X'', D'') := ||D'' - U''||_1$ for a closest ultrametric U'' to D''. Since there is no conflict $C \subseteq X$ in D such that $K \cap C \neq \emptyset$ and $Y \cap C \neq \emptyset$, and no conflict $C' \subseteq X'$ in D' such that $K' \cap C' \neq \emptyset$

and $Y \cap C' \neq \emptyset$, Lemma 6.1 implies that $\mathrm{opt}(X, D) = \mathrm{opt}(K, D[K]) + \mathrm{opt}(Y, D[Y])$ and $\mathrm{opt}(X', D') = \mathrm{opt}(K', D'[K']) + \mathrm{opt}(Y, D'[Y])$. Let $\mathrm{opt}_y := \mathrm{opt}(Y, D[Y])$. Clearly, $\mathrm{opt}_y = \mathrm{opt}(Y, D'[Y])$. By the correctness for the case that the maximum distance of the distance function is α (which is the induction hypothesis), it follows that $(K, D[K], k - \mathrm{opt}_y)$ is a yes-instance if and only if $(K', D'[K'], k' - \mathrm{opt}_y)$ is a yes-instance. As a consequence, $\mathrm{opt}(X, D) \leq k$ if and only if $\mathrm{opt}(X', D') \leq k'$. □

Theorem 6.3. *M-Hierarchical Tree Clustering admits a problem kernel with $2k \cdot (M + 2)$ elements. The running time for the kernelization is $O(M \cdot n^3)$.*

Proof. Let $I = (X, D, k)$ be an instance reduced with respect to Reduction Rule 6.3. Assume that I is a yes-instance, that is, there exists an closest ultrametric U on X with $||U - D||_1 \leq k$. We show that $|X| \leq 2k \cdot (\alpha_D + 2) \leq 2k \cdot (M + 2)$. As in the proof of Theorem 6.2, we partition the elements of X into two subsets A and B, where $A := \{i \in X \mid \exists j \in X : U(i, j) \neq D(i, j)\}$ contains the affected elements and $B := X \setminus A$ contains the unaffected elements. Clearly, $|A| \leq 2k$. Hence, in the following our goal is to bound the unaffected elements.

Consider an isolated clique K of $G_{\alpha_D} := G_{D,X}(\alpha_D)$. Since (X, D, k) is reduced with respect to Reduction Rule 6.3, every α_D-cluster of U contains at least one affected element (otherwise all elements from K are removed, see line 3(b) of procedure *RR3* in Figure 6.3).

Consider an isolated clique Q in G_{α_D}. Clearly, there is no conflict containing elements from both Q and $X \setminus Q$. Hence, by Lemma 6.1, we can assume that every isolated clique in G_{α_D} is an α_D-cluster of U. Let $Q_1, \ldots, Q_q$ denote the α_D-clusters of U that are isolated cliques in G_{α_D} and let $C_1, \ldots, C_\ell$ denote the other α_D-clusters of U. Furthermore, let $k_Q := ||U[\bigcup Q_i] - D[\bigcup Q_i]||_1$ and $k_C := ||U[\bigcup C_i] - D[\bigcup C_i]||_1$. By Lemma 6.1 we can conclude that the distance between U and D is $k_Q + k_C$. Since the Q_i's are isolated cliques in G_{α_D}, the analysis of the sizes of Q_i's can be done independently from the C_i's. Moreover, since, further calls of procedure *RR3* are made with Q_i's and $\alpha_D - 1$ as parameters, it suffices to bound $|B \cap \bigcup C_i|$ by $2k_C(\alpha_D + 1)$ and the same argument will then inductively imply the size bound of $\bigcup Q_i$, namely, $2k_Q(\alpha_D + 1)$. The details follow.

To bound the elements of B, we first prove the following.

$$|B \cap \bigcup C_i| \leq 2k_C(\alpha_D + 1) \tag{6.2}$$

Let $\alpha := \alpha_D$. For all $1 \leq i \leq \ell$, let $A_i := C_i \cap A$ and $B_i := C_i \setminus A_i$. First, note that every B_i is contained in a critical clique K_i of G_α: for each element $x \in B_i$ and for each element $y \in C_i$ it holds that $D(x, y) = U(x, y) \leq \alpha$ and, hence, x and y are adjacent in G_α. Moreover, for each element $x \in B_i$ and for each element $y \in X \setminus C_i$, we have $D(i, j) = U(i, j) = \alpha + 1$, and, hence, x and y are nonadjacent.

Since the *Critical-Clique* procedure has been applied to X and α and C_i is not an isolated clique in G_α, we have $|K_i| \leq \max(\alpha \cdot |N_{G_\alpha}(K_i)|, |N_{G_\alpha}(K_i)| + \alpha \cdot |N^2_{G_\alpha}(K_i)|)$. In order to bound $\sum |K_i|$ we partition $N_{G_\alpha}(K_i)$ into two sets Y_i and Z_i. An element $x \in N_{G_\alpha}(K_i)$ is contained in Z_i if there is an element $y \in X$ with $D(x, y) \leq \alpha$ and $U(x, y) = \alpha + 1$; otherwise it is contained in Y_i. By this definition, for every $x \in Z_i$ there is at least one $y \in Z_j$ ($j \neq i$) such that x and y are adjacent. For $i \neq j$, let $k_{i,j}$ denote the edges between Z_i and Z_j in G_α. Note that each of these edges contributes at least one to the distance between U and D and that $|Z_i| \leq \sum_{j \neq i} k_{i,j}$ and $|N^2_{G_\alpha}(K_i)| \leq \sum_{j \neq i} k_{i,j}$.

Second, let k_i denote the number of nonadjacent pairs of vertices in $N_{G_\alpha}(K_i)$. Observe that each of these pairs contributes at least one to the distance between U and D and that $|Y_i| \leq k_i$. Hence, $\max(\alpha|N_{G_\alpha}(K_i)|, |N_{G_\alpha}(K_i)| + \alpha|N^2_{G_\alpha}(K_i)|)$ can be bounded from above by

$$\begin{aligned}
&\max(\alpha(|Y_i| + |Z_i|)\,,\, |Y_i| + |Z_i| + \alpha \cdot |N^2_{G_\alpha}(K_i)|) \\
\leq &\max(\alpha(k_i + \sum_{i \neq j} k_{i,j})\,,\, k_i + (\alpha+1) \cdot \sum_{i \neq j} k_{i,j}) \\
\leq &(\alpha+1)(k_i + \sum_{i \neq j} k_{i,j}).
\end{aligned}$$

Thus, $\sum_{i=1}^{\ell} |K_i| \leq (\alpha+1)(\sum_{i=1}^{\ell} k_i + \sum_{i=1}^{\ell} \sum_{j \neq i} k_{i,j})$. Finally, note that $\sum_{i=1}^{\ell} k_i + \sum_{i=1}^{\ell} \sum_{j \neq i} k_{i,j} \leq 2k_C$. As a consequence, $|B \cap \bigcup C_i| \leq \sum_{i=1}^{\ell} |K_i| \leq 2k_C(\alpha+1)$. This concludes the proof of (6.2).

With the help of (6.2), we show the kernel size bound by induction on $\alpha_D + 1$, the maximum distance in D.

For the case $\alpha_D = 1$ (induction base) note that each isolated clique in G_1 induces an ultrametric. Hence, there are no isolated cliques in G_1, that is, $X = \bigcup_{i=1}^{\ell} C_i$. By (6.2) there are at most $2k(\alpha_D + 1)$ unaffected elements and, hence, $|X| = |A| + |B| \leq 2k + 2k(\alpha_D + 1) = 2k(\alpha_D + 2)$.

Next, consider the case that $\alpha_D > 1$ and assume that the kernel size bound holds for any instance (D'', X'', k'') with $\alpha_{D''} < \alpha_D$. Note that for each Q_i it holds that $\alpha_{D[Q_i]} < \alpha_D$. Moreover, since (D, X, k) is reduced with respect to Reduction Rule 6.3, procedure *RR3* has been applied with parameters $X' = Q_i$ and $t = \alpha_D - 1$. Hence, by the induction hypothesis we have that $|Q_i| \leq 2\,\mathrm{opt}(D[Q_i], Q_i)((\alpha_D - 1)+2)$. Hence, $\sum_{i=1}^{q} |Q_i| \leq 2k_Q(\alpha_D + 1)$. Moreover, according to (6.2), we have $\sum_{i=1}^{\ell} |C_i| \leq 2k_C(\alpha_D + 1)$. Hence, the total number of elements in X is at most $|A| + 2k_Q(\alpha_D + 1) + 2k_C(\alpha_D + 1) = |A| + (\alpha_D + 1) \cdot 2(k_C + k_Q) \leq 2k + (\alpha_D + 1) \cdot 2k \leq 2k \cdot (M + 2)$.

For the running time, note that for all $1 \leq t \leq M$, the G_t's can be built in $O(M \cdot n^2)$ time. Since $N_{G_t}(K)$ and $N^2_{G_t}(K)$ for a critical clique K are computable in $O(n^2)$ time, the *Critical-Clique* procedure needs $O(n^3)$ time. Then, Reduction Rule 6.3 first finds all isolated cliques $K_1, \ldots, K_\ell$ in G_α. This is doable in $O(n^2)$ time. Then, for every isolated clique K_i, it calls *Critical-Clique* on it. Hence, the total running time is $T(n, M) = O(n^3) + \sum_{i=1}^{\ell} T(|K_i|, M-1)$, where $T(|K_i|, M-1)$ denotes the running time of procedure *RR3* with the parameters K_i and $M - 1$. Since $\sum |K_i| \leq n$ and the recursion stops at level M, the running time of Reduction Rule 6.3 is $O(M \cdot n^3)$. Therefore, the total running time is $O(M \cdot n^3)$. □

The $k \cdot (k+2)$-element problem kernel and the $2k \cdot (M+2)$-element problem kernel complement each other in the sense that the $k \cdot (k+2)$-element problem kernel is independent of M but quadratic in k, whereas the $2k \cdot (M+2)$-element problem kernel is only linear in k but depends on M.

The applicability of Reduction Rule 6.1 needed for the $O(k^2)$-element kernel depends on the parameter value k, that is, Reduction Rule 6.1 is *parameter-dependent*. In contrast, for the applicability of Reduction Rule 6.3, the parameter value is irrelevant, that is, this rule is *parameter-independent*. Although both rules have a comparable

worst-case running time, the data reduction rules for the $O(k^2)$-element problem kernel are conceptually simpler and easier to implement. In particular, in combination with the search tree algorithm for which we keep a list of all conflicts there is only little overhead for applying Reduction Rule 6.1. This partially explains why in our experiments, presented in the next section, Reduction Rule 6.1 turns out to be more effective than the other rules.

6.5 Experimental Results

Based on our simple search tree algorithm (see Section 6.3) and data reduction rules (see Section 6.4), we implemented an exact algorithm for M-HTC. To demonstrate the competitiveness of this algorithm, we also implemented the randomized $(M+2)$-factor approximation algorithm due to Ailon and Charikar [5]. In our experiments, denoting the number of elements by n, we repeated their algorithm at least 100 and at most n^2 times and took the closest ultrametric during these repetitions as solution. In order to compare the time performance of our exact algorithm to another one, we also implemented the ILP (integer linear program) proposed by Ailon and Charikar [5]. We also used the corresponding relaxed linear program (LP), where all variables are allowed to be continuous, as a heuristic improvement in our algorithm.

We implemented all algorithms in the Java programming language. The program is free software and publicly available along with all our instances.[2]

All experiments were run on an Intel Core i3 550 machine with 3.2 GHz and 4 GB main memory running under the Ubuntu 10.10 64bit operating system with Java version 1.6.0_20 (Java runtime options: -Xms256M -Xmx256M). The ILP was solved on the same machine by the Gurobi solver[3] in version 4.0.0 64bit with standard options.

6.5.1 Implementation Aspects

Our implementation is based on the simple 3^k search tree algorithm described in Section 6.3. First, for a given M-HTC instance, the algorithm solves the corresponding LP. Denoting the resulting solution value by $k_{\mathrm{LP}}^{\mathrm{opt}}$, the value $\lceil k_{\mathrm{LP}}^{\mathrm{opt}} \rceil$ provides a lower bound on the optimal solution value of the M-HTC instance. We hence call the search tree algorithm (see Section 6.3) with increasing k, starting with $k = \lceil k_{\mathrm{LP}}^{\mathrm{opt}} \rceil$ and aborting when an optimal solution has been found. Next, we describe the heuristics that we have implemented in order to improve its running time.

Modification flags: We use flags to mark distances that may not be decreased (or increased). There are three reasons for setting such a mark: the distance has been already increased (or decreased); decreasing (or increasing) it leads to a solution with distance more than k; decreasing (or increasing) it leads to a conflict that cannot be repaired without violating previous flags.

Data reduction rules: We implemented all of the presented data reduction rules. However, in the experiments, our implementation of Reduction Rule 6.3 showed to be relatively slow and was thus deactivated.

Interleaving: In the search tree we interleave branching with the application of the data reduction rules, that is, after a suitable number of branching steps the data

[2] `http://fpt.akt.tu-berlin.de/tree-cluster`
[3] `http://www.gurobi.com/`

reduction rules are invoked. In the experiments described below we performed data reduction in every second step since this value yielded the largest speed-up.

Branch and bound: As mentioned above, the rounded LP solution $\lceil k_{\mathrm{LP}}^{\mathrm{opt}} \rceil$ provides a lower bound on the number of modifications that have to be performed to transform the underlying distance function into an ultrametric. Clearly, the deeper we get into the search tree, the more modification flags we have. Hence, the corresponding LP imposes more and more constraints whereas the parameter decreases, which together increases the likelihood that we can prune the search tree. This means that, after a number of branching steps, we solve the corresponding LP program and check whether the parameter k is at least as large as $\lceil k_{\mathrm{LP}}^{\mathrm{opt}} \rceil$. If not, we can abort branching into further subcases, because none of them can provide a solution with at most k modifications.

Choice of conflicts for branching: We choose the conflict to branch on in the following order of preference: The first and most preferable type of conflicts are conflicts where either both non-max-distance pairs cannot be increased or the max-distance pair cannot be decreased and one non-max-distance pair cannot be increased. In this case, no actual branching takes place since only one option to destroy the conflict remains. Second, if no such conflict exists we choose conflicts where the max-distance pair cannot be decreased or one of the non-max-distance pairs cannot be increased. If such conflicts are also not present, we choose the smallest conflict with respect to a predetermined lexicographic order. This often creates a conflict of the first two types.

In the following sections, we refer to the above algorithm as "combination of FPT and LP" since it uses the LP for pruning the search tree. The algorithm without branch and bound is referred to as "simple FPT algorithm".

6.5.2 Experiments with Synthetic Data

We generated random instances to chart the border of tractability with respect to different values of n and k. All experiments were performed for $M = 2$. We considered three different values for n: 50, 75, and 100. For each fixed value of n we generated four ultrametrics and perturbed each of these ultrametrics, increasing step by step the number k of perturbations. For each pair of n and k we generated four distance functions. We thus created 16 instances for each pair of n and k. Next, we describe in detail how we generated and perturbed the ultrametrics.

Generation of Ultrametrics. We generated the instances by creating a random ultrametric tree of depth $M + 1$. We started at the root and randomly drew the number of its children under uniform distribution from $\{2, \ldots, \lceil \ln n \rceil\}$. Then, the elements were randomly (again under uniform distribution) assigned to the subtrees rooted at these newly created nodes. For each child we recursively created ultrametric trees of depth M. The only difference for a node at a lower level is that we randomly drew the number of its children under uniform distribution from $\{1, \ldots, \lceil \ln n \rceil\}$. That is, in contrast to the root node, we allowed that an inner node has only one child.

Perturbation of Generated Ultrametrics. We randomly chose a pair $\{i, j\}$ of elements under uniform distribution and changed the distance value $D(i, j)$. This step was repeated until k distance values have been changed. For each chosen pair, we randomly decided whether $D(i, j)$ will be increased or decreased (each with probability 1/2). We did not increase $D(i, j)$ if it has been previously decreased or if $D(i, j) = M + 1$; we did not decrease $D(i, j)$ if it has been previously increased or if $D(i, j) = 1$.

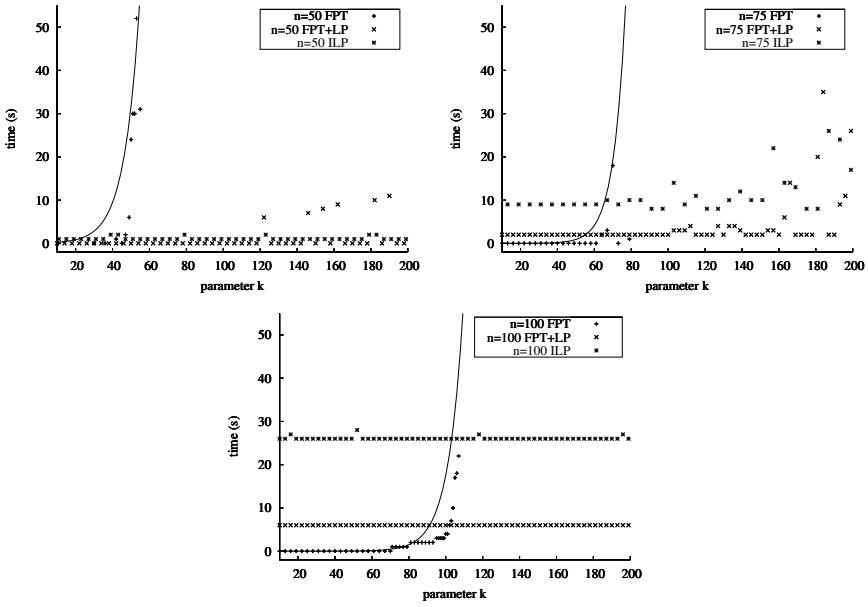


Figure 6.4: Running times for fixed n and varying k.

Experimental Results. The running times for all three sets of generated instances are shown in Figure 6.4. We tested three algorithms, the first is the FPT algorithm that only uses data reduction and branching. The second algorithm works as follows. First, an LP-solution is computed. If this solution is integral, then we are done. Otherwise, the solution of the approximation algorithm is computed and will be compared to the LP-solution. If the lower bound provided by the LP-solution meets the upper bound solution provided by the approximation algorithm, then the approximative solution must be optimal and hence we are done in this case. In all other cases, we afterwards perform the algorithm based on the combination of FPT and LP.

Our observations are as follows. First, as expected from the theoretical running time analysis for our algorithms, they are very fast when k is small. In particular, for $k < n$ the simple FPT algorithm that is based on the search tree and data reduction outperforms the ILP approach, and adding the LP to the FPT algorithm does not result in a speed-up. This is due to the fact that most instances with $k < n$ could be solved without branching, just by applying the data reduction rules. Second, for the simple FPT algorithm, the combinatorial explosion sets in at $k \approx n$, and its running times quickly become infeasible for increasing k. In contrast, the ILP appears to show parameter-independent running times,[4] and for the combination of FPT and LP algorithm, the combinatorial explosion apparently sets in at $k \approx 2n$. Finally, using $\exp(a \cdot (x - b))$ as regression function, the corresponding regression analysis shows that the running time of the FPT algorithm is best described by exponential functions of the type α^k with $\alpha \leq 1.4$. This is due to the data reduction: switching it off leads to running times with $\alpha \approx 2.4$. Note that, by Proposition 6.1, in our theoretical analysis

[4]The running times for the ILP depend heavily on n: already for $n = 125$ and very low values of k, the average running time of the ILP was roughly 2 minutes.

the algorithm has a running time of $O(3^k \cdot n^3)$.

6.5.3 Experiments with Protein Similarity Data

We performed experiments on protein similarity data which have been previously used in an experimental evaluation of fixed-parameter algorithms for Cluster Editing [146]. The data set contains 3964 files with pairwise similarity data of sets of proteins. The number of proteins n for each file ranges from 3 to 8836. We consider a subset of these files, where $n \leq 70$, covering about 92% of the files.

From each file, we created four discrete distance matrices for $M+1 = 3$ as follows. We set the distance of the c% of the pairs with lowest similarity to 3, where c is a predetermined constant. From the remaining pairs the c% of the pairs with lowest similarity were set to two, and all others to one. In our experiments, we set c to 75, 66, 50, and 33, respectively. This approach is motivated by the following considerations. In a distance function represented by a balanced ultrametric tree of depth $M+1$ at least half of all distances are $M+1$ and with increasing degree of the root of the clustering tree the number of pairs with distance $M+1$ increases. If we assume that the ultrametric tree is more or less balanced we thus expect a large portion of pairwise distances to have maximum value making the choices of $c = 75$ and $c = 66$ the most realistic.

Experimental Results. Table 6.1 (on page 104) contains the results of our experiments. We summarize our observations as follows. First, we used the LP-solver followed by the approximation algorithm. In cases where the LP-solver already has found an integral solution or $\lceil k_{\text{LP}}^{\text{opt}} \rceil$ is equal to the distance value found by the approximation algorithm, we can conclude that the instance has been solved to optimality. The LP-solver in combination with the approximation algorithm solves a large range of instances. We refer to the instances that could not be solved by this approach as "hard instances".

Second, we investigated how many of the hard instances could be solved within 2, 10, 60, and 300 seconds, respectively, by our search tree based algorithm (interleaved with data reduction rules and LP-solver). Observe that a significant fraction of the hard instances of size at most 50 could be solved within 300 seconds. For larger instances, the performance of the algorithm becomes worse.

Third, we compared our search tree based algorithm with the ILP. The ILP outperforms our algorithm in most cases. There are, however, some instances where our approach finds optimal solutions that are not found by the ILP based approach.

Fourth, for the evaluation of the use of the data reduction rules, we switched off the data reduction rules and repeated the experiments for $c = 66$. These experiments show that the performance of the search tree based approach gets significantly worse without the interleaving with the data reduction rules.

Fifth, for the evaluation of the approximation algorithm, we counted the number of instances for which it yields optimal solutions. For all considered instance ranges, some instances could be solved to optimality and the average distance found by the approximation algorithm is close to the optimal average distance, implying that the approximation solutions are close to the optimum for a large fraction of the instances. In addition, note that the approximation algorithm is the fastest of all algorithms. However, there is a tendency that for larger instances the approximation factor gets worse.

Table 6.1: Summary of our experiments for the protein similarity data. The instances are created with $M+1 = 3$ and $c = 75, 66, 50$, and 33. The second column contains the number of instances within the respective range. The column labeled "#" provides the number of "hard instances", that is the instances that could not be solved by the LP-solver followed by the approximation algorithm. The next four columns provide the number of hard instances that can be solved within 2, 10, 60, and 300 seconds by our algorithm that is based on the combination of FPT and LP and the columns labeled $k^{\mathrm{ST}}_{\mathrm{avg}}$ and $k^{\mathrm{ST}}_{\mathrm{max}}$ provide the average and maximum distance of the instances that could be solved by this approach. The columns $k^{\mathrm{Apro}}_{\mathrm{avg}}$ and $k^{\mathrm{Apro}}_{\mathrm{max}}$ provide the same values for the approximation algorithm for all (not only the hard) instances. Moreover, $^{\mathrm{AP}}_{\mathrm{OPT}}$ provides the number of instances that are solved optimal by the approximation algorithm and $^{\mathrm{AP}}_{\mathrm{TIME}}$ denotes its average running time for all instances. In the last two columns, $^{\mathrm{ILP}}_{\mathrm{OPT}}$ provides the number of hard instances that are solved optimally by the ILP-solver, and $^{\mathrm{ILP}}_{\mathrm{TIME}}$ denotes the average running time. The last subtable provides the results for our FPT-algorithm with branch and bound but without interleaving the data reduction rules.

c=33

range	total	#	2s	10s	60s	300s	$k^{\mathrm{ST}}_{\mathrm{avg}}$	$k^{\mathrm{ST}}_{\mathrm{max}}$	$k^{\mathrm{Apro}}_{\mathrm{avg}}$	$k^{\mathrm{Apro}}_{\mathrm{max}}$	$^{\mathrm{AP}}_{\mathrm{OPT}}$	$^{\mathrm{AP}}_{\mathrm{TIME}}$	$^{\mathrm{ILP}}_{\mathrm{OPT}}$	$^{\mathrm{ILP}}_{\mathrm{TIME}}$
$[0,30]$	3114	143	135	139	142	143	65	230	68	248	82	0.1	141	0.3
$(30,50]$	359	89	43	60	67	74	309	870	335	1033	10	0.3	89	9.5
$(50,70]$	210	57	3	16	21	22	467	673	608	1224	1	0.4	31	12.4
$[0,70]$	3683	289	181	215	230	239	177	870	256	1224	93	0.2	261	4.8

c=50

range	total	#	2s	10s	60s	300s	$k^{\mathrm{ST}}_{\mathrm{avg}}$	$k^{\mathrm{ST}}_{\mathrm{max}}$	$k^{\mathrm{Apro}}_{\mathrm{avg}}$	$k^{\mathrm{Apro}}_{\mathrm{max}}$	$^{\mathrm{AP}}_{\mathrm{OPT}}$	$^{\mathrm{AP}}_{\mathrm{TIME}}$	$^{\mathrm{ILP}}_{\mathrm{OPT}}$	$^{\mathrm{ILP}}_{\mathrm{TIME}}$
$[0,30]$	3114	210	189	196	207	208	44	169	44	1039	115	0.03	208	0.29
$(30,50]$	359	134	63	86	98	106	206	639	236	775	13	0.2	133	6.5
$(50,70]$	210	83	5	23	29	37	338	506	437	869	2	1.3	60	14.8
$[0,70]$	3683	427	257	305	334	351	124	639	186	1039	130	0.22	401	4.5

c=66

range	total	#	2s	10s	60s	300s	$k^{\mathrm{ST}}_{\mathrm{avg}}$	$k^{\mathrm{ST}}_{\mathrm{max}}$	$k^{\mathrm{Apro}}_{\mathrm{avg}}$	$k^{\mathrm{Apro}}_{\mathrm{max}}$	$^{\mathrm{AP}}_{\mathrm{OPT}}$	$^{\mathrm{AP}}_{\mathrm{TIME}}$	$^{\mathrm{ILP}}_{\mathrm{OPT}}$	$^{\mathrm{ILP}}_{\mathrm{TIME}}$
$[0,30]$	3114	178	132	158	172	172	30	177	35	309	66	0.02	178	0.7
$(30,50]$	359	147	4	13	53	106	108	226	130	342	11	0.11	144	3.6
$(50,70]$	210	88	0	0	3	14	194	303	270	498	4	0.37	73	15.6
$[0,70]$	3683	413	136	171	228	293	66	303	119	498	81	0.07	395	4.5

c=75

range	total	#	2s	10s	60s	300s	$k^{\mathrm{ST}}_{\mathrm{avg}}$	$k^{\mathrm{ST}}_{\mathrm{max}}$	$k^{\mathrm{Apro}}_{\mathrm{avg}}$	$k^{\mathrm{Apro}}_{\mathrm{max}}$	$^{\mathrm{AP}}_{\mathrm{OPT}}$	$^{\mathrm{AP}}_{\mathrm{TIME}}$	$^{\mathrm{ILP}}_{\mathrm{OPT}}$	$^{\mathrm{ILP}}_{\mathrm{TIME}}$
$[0,30]$	3114	128	123	125	127	127	15	231	18	282	110	0.03	128	1.2
$(30,50]$	359	117	77	81	97	107	68	165	76	243	56	0.2	117	2.3
$(50,70]$	210	92	29	49	56	70	130	278	151	322	27	0.6	92	9.2
$[0,70]$	3683	337	229	255	280	304	60	278	75	322	193	0.2	337	3.8

c=66 without data reduction rules

range	total	#	2s	10s	60s	300s	$k^{\mathrm{ST}}_{\mathrm{avg}}$	$k^{\mathrm{ST}}_{\mathrm{max}}$	$k^{\mathrm{Apro}}_{\mathrm{avg}}$	$k^{\mathrm{Apro}}_{\mathrm{max}}$	$^{\mathrm{AP}}_{\mathrm{OPT}}$	$^{\mathrm{AP}}_{\mathrm{TIME}}$	$^{\mathrm{ILP}}_{\mathrm{OPT}}$	$^{\mathrm{ILP}}_{\mathrm{TIME}}$
$[0,30]$	3114	178	101	125	147	159	28	177	33	177	68	0.02	178	0.5
$(30,50]$	359	147	3	8	37	60	109	226	131	344	12	0.12	137	7.3
$(50,70]$	210	88	0	0	2	5	170	276	271	495	5	0.5	69	22.5
$[0,70]$	3683	413	104	133	186	224	53	276	118	495	85	0.1	385	6.9

6.5.4 Conclusions and Recommendations

From the experiments with synthetic and protein similarity data, we draw the following conclusions and recommendations.

Conclusions from the Experiments with Synthetic Data. For $k < n$, one should use the simple search tree that is based on the combination of data reduction and the search tree algorithm. Since the ILP was not particularly fast in solving these types of instances, one should explore whether our data reduction rules can be incorporated into the ILP solving strategy to yield a speed-up. Furthermore, for $n < k < 2n$ most of the instances were solved immediately by solving the relaxed LP, since the LP solutions were integral. Hence, these instances seem to share a structural feature that makes them easier. A natural goal would be to develop data reduction rules that are able to solve all instances with $k < 2n$. In other words, the aim should be to "move" the onset of the combinatorial explosion for combinatorial FPT algorithms from $k \approx n$ to $k \approx 2n$.

Conclusions from the Experiments with Protein Similarity Data. A large fraction of the protein-similarity instances could be solved by running the LP-solver and the approximation algorithm. Hence, this approach should be applied before relying on more expensive approaches such as search trees or ILPs. Moreover, although outperformed by the ILP-solver for most instances, our approach solves a large fraction of the instances. With further improvements, our algorithms should form a serious competitor for state-of-the-art ILP-solvers for solving M-HTC. Finally, the data reduction rules are crucial for the performance of our search tree based approach. Thus, a main target for further algorithmic improvements of our approach should be to improve the data reduction rules, for example by improving the implementation of Reduction Rule 6.3 (which so far has not been employed in our systematic experiments).

6.6 Conclusion

We have initiated the study of M-Hierarchical Tree Clustering in the context of parameterized algorithmics and also provided first implementations for solving M-Hierarchical Tree Clustering. There are numerous topics for future research.

We have presented a first refinement of the simple $O(3^k)$-size search tree, yielding a fixed-parameter algorithm with running time $O(2.562^k \cdot n^3)$. It would be interesting to explore whether other branching strategies, such as for example the "vertex merging" strategy that has been proposed for Cluster Editing [24], lead to further improved search tree algorithms for M-Hierarchical Tree Clustering.

In this chapter, we have presented an $O(k^2)$-element and an $O(M \cdot k)$-element problem kernel with cubic worst-case running times. There are two questions immediately arising from our kernelization results: First, can the problem kernel size be improved; for example, is there an $O(k)$-element problem kernel (that is, a linear-element kernel whose size does not depend on M)? Second, can the running times of the kernelizations be improved? Note that Cluster Editing admits an $O(k^2)$-vertex problem kernel that can be computed in linear time [145].

In this chapter, we investigated M-Hierarchical Tree Clustering with respect to the standard parameter k, denoting the cost of the solution. As follows from our experiments, k is not really small for several real-world instances. Hence,

the investigation of M-Hierarchical Tree Clustering with respect to "nonstandard parameterizations" is desirable (as we have done for Cluster Editing, see Chapter 4). For example, a refined parameter (see Section 2.3) for parameter k for M-Hierarchical Tree Clustering would be the "element deletion distance to an ultrametric", that is, the minimum cardinality of an element set $X' \subseteq X$ such that $D[X \setminus X']$ is ultrametric.

From an applied point of view, further (heuristic) improvements are both conceivable and necessary to further increase the range of instances that can efficiently be solved.

Finally, we discuss two extensions of M-Hierarchical Tree Clustering.

First, it seems also worthwhile to study the parameterized complexity of the problem of "fitting ultrametrics" for other norms such as the ℓ_2 norm. The polynomial-time approximability of these problems is well-studied [5].

Second, overlapping clusters (see Chapter 5) have also been considered in context of hierarchical tree clustering. Observe that the t-clusters of an ultrametric over a set X form a partition of X (that is, any two t-clusters are disjoint). That is, at each level of an ultrametric tree there is no overlap between the clusters. Hence, models have been suggested that generalize ultrametrics to allow for overlap between clusters. For example, Jardine and Sibson [115, pages 65-71] suggest to "relax" the strong triangle inequality (see Definition 6.1) leading to the notion of s-ultrametrics; a distance function D over a set X is called an *s-ultrametric* if for all $i, j \in X$ and for all $S \subseteq X$ with $|S| = s$

$$D(i,j) \leq \max\{D(i',j') \mid i' \in S \cup \{i,j\}, j' \in S\}.$$

For $s = 1$, this is the same as the strong triangle inequality. As observed by Barthélemy and Brucker [13], an s-ultrametric D with maximum distance two corresponds to a graph fulfilling the "s-Zahn property", that is, any two maximal cliques in the 1-threshold graph of D overlap in at most $s - 1$ vertices (also see Section 5.1). The work of Barthélemy and Brucker [13] points to further generalizations of ultrametrics and corresponding clustering problems. The parameterized complexity of these generalizations seems unexplored so far.

Chapter 7

Minimum Flip Consensus Tree

7.1 Introduction

The Minimum Flip Consensus Tree problem arises in computational phylogenetics in the context of supertree construction. Given a binary matrix, the task is to "flip" a minimum number of entries of the matrix in order to obtain a binary matrix that admits what is called a *directed perfect phylogeny*. These are matrices from which a rooted phylogenetic tree can be inferred [100, 143].

In this chapter, we employ a graph-theoretic formulation of the problem, which was introduced by Chen et al. [45]: the binary input matrix A is represented by a bipartite graph $G = (V_c, V_t, E)$ where an edge between two vertices $i \in V_c$ and $j \in V_t$ is drawn if and only if $A_{i,j} = 1$ ($A_{i,j}$ denoting the jth entry in the ith row). The matrix then admits a directed perfect phylogeny if and only if the graph does not contain an *M-graph* as an induced subgraph. An M-graph is a path of five vertices with the first vertex belonging to V_t. An example of such an M-graph is depicted in Figure 7.1. The flipping of a matrix entry $A_{i,j}$ from 0 to 1 corresponds to the insertion of the edge $\{i, j\}$, and the flipping of $A_{i,j}$ from 1 to 0 corresponds to the deletion of the edge $\{i, j\}$.

The Minimum Flip Consensus Tree problem is then defined as follows.

Definition 7.1. Minimum Flip Consensus Tree (MFCT)
Input: A bipartite graph $G = (V_c, V_t, E)$ and an integer $k \geq 0$.
Question: Can G be transformed by applying up to k edge modifications into an M-free graph, that is, a graph without an induced M-graph?

We refer to a vertex $c \in V_c$ as c-vertex, and to a vertex $t \in V_t$ as t-vertex.

In a nutshell, other than previous work [27, 45] on fixed-parameter algorithms for MFCT, which mainly dealt with the development of depth-bounded search trees, here we mainly deal with polynomial-time data reduction, devising the first nontrivial kernelization algorithm for Minimum Flip Consensus Tree. Moreover, we present a further improved search algorithm for Minimum Flip Consensus Tree.

Minimum Flip Consensus Tree arises in an approach by Chen et al. [45] to aggregate the information of several input trees in a consensus tree. In the context

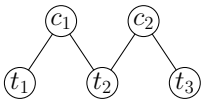


Figure 7.1: An M-subgraph with $t_1, t_2, t_3 \in V_t$ and $c_1, c_2 \in V_c$.

of this chapter, a rooted phylogenetic tree (also called directed phylogeny) is a rooted tree where each leaf corresponds to a taxon. In phylogenetics, a *taxon* refers to a group of organisms. Moreover, an inner node of a phylogenetic tree corresponds to a hypothetical last common ancestor of its descendents. Phylogenetic trees are inferred based on similarities and differences in their physical or genetic characteristics, either "by hand" or using computer-based approaches. Depending on the data and the methods that have been used to generate a phylogenetic tree, a set of given phylogenetic trees may disagree on the evolutionary relationship of the considered taxa. Since all methods used to infer phylogenetic trees have their advantages and drawbacks, the idea is that combining all information of the trees into a consensus tree gives a robust model of the evolutionary history. In this chapter, we focus on an approach of Chen et al. [45]. Given rooted phylogenetic trees $T_1 = (V_1, E_1), \ldots, T_\ell = (V_\ell, E_\ell)$ (all on the same set of taxa), a consensus tree is constructed in three phases. See Figure 7.2 for an illustration. In a first phase, the information of all input trees is represented in a bipartite graph $G = (V_c, V_t, E)$ as follows: The vertex set V_t contains a vertex for each taxon and the vertex set V_c is the (disjoint) union of the inner nodes of the input trees. Moreover, for each inner node c of an input tree T_i and each taxon t there is an edge $\{c, t\} \in E$ if and only if t is a descendent of c in T_i. If the input trees do not contain any conflicting information, then G is M-free [66, 100] (also see Figure 7.2). M-free graphs admit a directed perfect phylogeny, meaning that one can construct a rooted phylogenetic tree from an M-free graph G. If, however, the input trees contain contradicting information, then G contains induced M-graphs. Hence, in a second "error correction phase", the task is to solve Minimum Flip Consensus Tree (see Definition 7.1) to destroy all induced M-graphs. In a third phase, the consensus tree is then inferred from the M-free graph.

Known Results and Previous Work. The Minimum Flip Consensus Tree problem was introduced by Chen et al. [45]. They proved its NP-completeness and described a factor-$2d$ polynomial-time approximation algorithm for graphs with maximum degree d. Furthermore, they showed fixed-parameter tractability with respect to the number k of flips by describing a simple $O(6^k \cdot |V_t||V_c|)$ search tree algorithm that is based on the forbidden induced subgraph characterization with M-graphs. Subsequently, Böcker et al. [27] improved the running time to $O(4.42^k(|V_c|+|V_t|)+|V_c|\cdot|V_t|)$ by employing a refined branching strategy that leads to a search tree of size $O(4.42^k)$. This theoretically proven running time acceleration was also confirmed by computational experiments [27].

Minimum Flip Consensus Tree is a special case of the Flip Supertree problem, where the input matrix is allowed to have "uncertain" entries [45]. Experiments have shown that Flip Supertree compares favorably with other supertree-construction methods [44]. Very recently, it has been shown that Flip Supertree is $W[1]$-hard [26] with respect to the parameter "number of flips of certain entries". Chi-

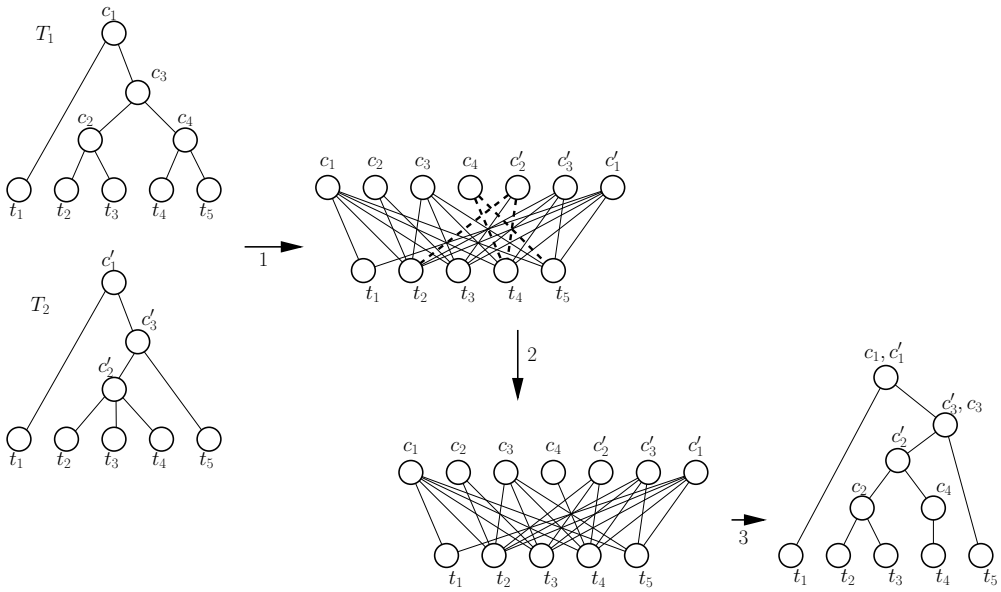


Figure 7.2: Illustration of the Minimum Flip Consensus Tree approach for two input trees. In a first step, the information of both input trees is represented in a bipartite graph. In this bipartite graph, there is a vertex for each taxon and for each inner tree node. An edge is drawn between an inner tree node and a taxon if and only if the taxon is a descendent of the inner tree node. In the given example, the two input trees contain conflicting information: In T_1, leaves t_5 and t_4 have a common ancestor (namely c_4) that is not an ancestor of t_2; however, in T_2, leaves t_2 and t_4 have a common ancestor (namely c'_2) that is not an ancestor of t_5: clearly both is not simultaneously possible. This contradicting information leads to M-graphs in G. One M-graph is highlighted by bold dashed lines. In a second phase, all induced M-graphs are destroyed by modifying edges. In the given example, it suffices to delete the edge $\{c_4, t_4\}$ to destroy all induced M-graphs. From the M-free bipartite graph, in a third phase the consensus tree is constructed.

mani, Rahmann, and Böcker [49] proposed an ILP-formulation for Flip Supertree and showed that optimal solutions can be found for instances with up to 100 taxa. An introduction into the topic of supertree construction methods is given by [23]. For a survey on fixed-parameter algorithms in phylogenetics we refer to [84].

From a graph-theoretic point of view, Minimum Flip Consensus Tree belongs to the class of so-called edge modification problems (see Section 3.3). Recall that in the case of edge modification problems the task is to modify the edge set of a given graph as little as possible to obtain a graph fulfilling a desired graph property. In particular, Minimum Flip Consensus Tree is Π-Editing (see Definition 3.1) where the desired graph property is to be M-free. Recall that an M-graph is a path on five vertices (a so-called P_5) with the first, middle, and last vertices from V_t. For an integer $\ell > 2$, Π-Editing for the graph property Π "being P_ℓ-free" is called P_ℓ-free Editing. Recall that Cluster Editing is P_3-free Editing (see Chapter 4). The currently smallest problem kernel for Cluster Editing contains at most $2k$

vertices [46]. P_4-free Editing in bipartite graphs is known as Bicluster Editing. Search tree algorithms and polynomial-size problem kernels have been presented for Bicluster Editing [93, 145]. For general graphs, Guillemot et al. [87] have recently shown that P_4-free Editing (also known as Cograph Editing) admits a cubic vertex kernel, whereas P_l-free Editing for $l \geq 13$ does not admit a polynomial-size problem kernel unless $PH = \Sigma_3$.

Damaschke [52] investigated kernelization in the context of enumerating all inclusion-minimal solutions of size at most k. In this scenario, when designing data reduction rules one has to guarantee that all inclusion-minimal solutions of size at most k are preserved. Kernels that fulfill these additional constraints are called *full kernels*. In this setting, Damaschke [52] presents a full kernel consisting of $O(6^k)$ matrix entries for the following problem closely related to Minimum Flip Consensus Tree: Given a binary matrix and a nonnegative integer k, enumerate all inclusion-minimal sets of at most k flips that transform the matrix into a matrix that admits an unrooted perfect phylogeny.

Our Contributions. In this chapter, we provide several polynomial-time data reduction rules for Minimum Flip Consensus Tree that lead to a problem kernel containing $O(k^3)$ vertices. This is the first nontrivial kernelization result for Minimum Flip Consensus Tree. Note that in the light of the non-kernelizability results for P_ℓ-free Editing for $\ell > 13$ by Guillemot et al. [87] and for H-free Editing for a specific graph H on seven vertices by Kratsch and Wahlström [124], it is a challenging task to prove the existence of polynomial-size problem kernels even for forbidden subgraphs of constant size. In addition, we present a search-tree algorithm with running time $O(3.68^k \cdot |V_c|^2|V_t|)$. Combining our kernelization algorithm with our search tree algorithm we achieve a running time of $O(3.68^k + |V_c|^2 \cdot |V_t| \cdot |E|)$ instead of the previous $O(4.42^k \cdot (|V_c| + |V_t|) + |V_c| \cdot |V_t|)$ [27].

This chapter is organized as follows. In Section 7.2, we introduce basic notation and some definitions used throughout this chapter. In Section 7.3, we present a decomposition property which yields the basis for one of the data reduction rules and for the identification of a polynomial-time solvable special case needed for the improved search tee strategy. In Section 7.4, we present one easy and two more intricate data reduction rules for MFCT. Based on these rules, in Section 7.5 we can show that MFCT admits a problem kernel with $O(k^3)$ vertices. Finally, in Section 7.6 we present an $O(3.68^k)$-size search tree for MFCT.

7.2 Preliminaries

Throughout this chapter, we use the following notation and definitions. See Section 2.6 for basic notation concerning graphs and Section 3.3.1 for basic notation concerning edge modification problems. In addition, we use the following notation.

For two sets X and Y with $X \cap Y = \emptyset$, let $E_{X,Y}$ denote the set $\{\{x, y\} \mid x \in X \wedge y \in Y\}$. As an abbreviation for $E_{\{x\},Y}$ we write $E_{x,Y}$. A bipartite graph $G = (X, Y, E)$ is called a *chain graph* if the neighborhoods of the vertices in X form a chain [163]. That is, there is an ordering of the vertices in X, say $x_1, x_2, \ldots, x_{|X|}$, such that $N_G(x_1) \subseteq N_G(x_2) \subseteq \ldots \subseteq N_G(x_{|X|})$. It is easy to see that the neighborhoods of Y also form a chain if G is a chain graph. Moreover, a bipartite graph is a chain graph if and only if

it is $2K_2$-free [163] (herein, a $2K_2$ is the graph that consists of two independent edges). Since every M-graph contains an induced $2K_2$, the set of chain graphs is contained in the class of M-free graphs. One of our data reduction rules is based on identifying and reducing the size of subgraphs of the input graphs that are chain graphs and additionally have a special neighborhood structure.

We use the following notation concerning rooted trees. A vertex of a tree is called *node*. For a rooted tree T let $L(T)$ denote the *leaves* of T (that is, the nodes of degree one). The nodes in $V(T) \setminus L(T)$ are denoted as *inner nodes*. The root of T is denoted by $r(T)$. Moreover, for a node $v \in V(T)$, the subtree rooted at v is denoted by T_v. We refer to a child of a node v as *leaf child* of v if it is a leaf; otherwise, it is called *non-leaf child* of v. We speak of the leaves (inner nodes) of a forest to refer to the union of the leaves (inner nodes) of the trees of the forest.

Induced M-graphs and M-freeness. Recall that a bipartite graph $G = (V_c, V_t, E)$ is called M-free if it does not contain an induced M-graph. An induced M-graph is denoted by a 5-tuple $(t_l, c_l, t_m, c_r, t_r)$, where $c_l, c_r \in V_c$ and $t_l, t_m, t_r \in V_t$. Two c-vertices c_l and c_r are said to be *in conflict* if there exists an induced M-graph containing both c_l and c_r. Clearly, for an induced M-graph $(t_l, c_l, t_m, c_r, t_r)$ we have $t_l \in N_G(c_l) \setminus N_G(c_r)$, $t_m \in N_G(c_l) \cap N_G(c_r)$ and $t_r \in N_G(c_r) \setminus N_G(c_l)$. Thus, two c-vertices $c_l, c_r \in V_c$ are in conflict if and only if

$$(N_G(c_l) \setminus N_G(c_r) \neq \emptyset) \wedge (N_G(c_l) \cap N_G(c_r) \neq \emptyset) \wedge (N_G(c_r) \setminus N_G(c_l) \neq \emptyset).$$

If $c_l, c_r \in V_c$ are in conflict we write $c_l \perp c_r$. In summary, for a bipartite graph $G = (V_c, V_t, E)$, the following statements are equivalent:

- G is M-free,
- no two c-vertices are in conflict, and
- for each pair of c-vertices c_l and c_r, it holds that $N(c_l) \cap N(c_r) = \emptyset$ or $N(c_l) \subseteq N(c_r)$ or $N(c_r) \subseteq N(c_l)$.

M-free graphs and rooted phylogenetic trees are closely related. Given a connected and M-free graph $G = (V_c, V_t, E)$, one can construct a rooted tree T with node set $V_t \cup V_c$ and with $L(T) = V_t$ such that $t_i \in V_t$ is a descendant of $c_j \in V_c$ if and only if $t_i \in N_G(c_j)$, see [45, 100, 143] for details. An example is given in Figure 7.3.

M-freeness and Critical Independent Set Preserving Graph Properties. Here, we recapitulate the findings of Section 3.4 relevant to Minimum Flip Consensus Tree. These findings are centered around the notion of critical independent sets.

Recall that a vertex set $I \subseteq V$ is called a *critical independent set* if for any two vertices $v, w \in I$ it holds that v and w are nonadjacent, $N_G(v) = N_G(w)$, and I is maximal with respect to this property (see Definition 2.5). Moreover, in Section 3.4 we defined so-called critical independent set preserving graph properties. A hereditary graph property Π is called 1-*critical independent set preserving* (1-*cisp*) whenever no forbidden induced subgraph F of Π contains a critical independent set of size at least two (that is, all critical independent sets of F have size one, see Definition 3.3). Note

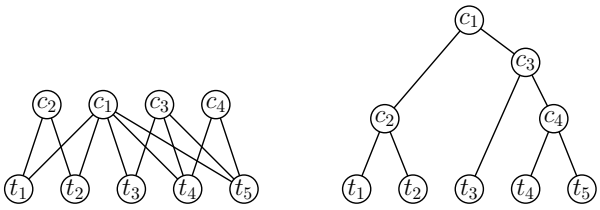


Figure 7.3: An M-free graph G and the corresponding tree. In a connected M-free graph there exists a "universal" c-vertex, that is, a c-vertex adjacent to all t-vertices. In this example, c_1 is a universal vertex. This universal vertex is the root of the corresponding tree. Thus, one can build a tree by applying the above argument recursively for the connected components of $G - c_1$.

that M-freeness is a 1-cisp graph property: all vertices in an induced M-graph have different neighborhoods. Therefore, Reduction Rule 3.1 and Lemma 3.5 apply directly to Minimum Flip Consensus Tree:

- Reduction Rule 3.1 shrinks large critical independent sets. More specifically, by Reduction Rule 3.1 it suffices to keep $k + 1$ "representatives" for each critical independent set.
- Lemma 3.5 implies that for Minimum Flip Consensus Tree there is an optimal solution that "treats" the vertices of a critical independent set equally, that is, if two nonadjacent vertices have an identical neighborhood in the input graph, then these two vertices have an identical neighborhood in the final target graph. In other words, critical independent sets are "preserved" by optimal solutions. This property is decisive for the correctness proof of Reduction Rule 7.4 in Section 7.4.

In our correctness proofs we rely on solutions that preserve critical independent set. A solution S for an instance (G, k) is called *regular* if every critical independent set of G is contained in a critical independent set of $G \Delta S$. By Lemma 3.5 we can assume that there exists a minimum-cardinality solution that is regular. Moreover, since the modification operations in the proof of Lemma 3.5 can be performed in polynomial time, we can compute a regular solution from a given (arbitrary) solution in polynomial time.

7.3 A Decomposition Property

In this section, we show that independent conflicts can be resolved independently. More specifically, we show that, given a set C of c-vertices such that no vertex in C is in conflict with any c-vertex outside of C, the conflicts involving the vertices in C can be resolved independently from the conflicts not involving vertices in C. This fact is used to prove the correctness of one of our data reduction rules in Section 7.4 and to identify a polynomial-time solvable special case of MFCT in Section 7.6 that is the basis for our improved search tree algorithm presented in Section 7.6.

For an input graph $G = (V_c, V_t, E)$, we consider the *conflict graph* $G_\perp = (V_c, F)$ with vertex set V_c and edge set $F := \{\{c', c''\} \mid c', c'' \in V_c \wedge c' \perp c''\}$. Roughly speaking, we show that for each connected component of the conflict graph, the conflicts can

be resolved independently. To show this property, we need the following lemma. Throughout this section, we identify a connected component by its vertex set.

Lemma 7.1. *Let C and C' be two connected components of the conflict graph $G_\perp$ with $N_G(C) \cap N_G(C') \neq \emptyset$. Then,*

- *for each $x \in C$ it holds that $N_G(C') \subseteq N_G(x)$ or $N_G(x) \cap N_G(C') = \emptyset$, or*
- *for each $y \in C'$ it holds that $N_G(C) \subseteq N_G(y)$ or $N_G(y) \cap N_G(C) = \emptyset$.*

Proof. Since $N_G(C) \cap N_G(C') \neq \emptyset$, there is a vertex $x \in C$ and a vertex $y \in C'$ with $N_G(x) \cap N_G(y) \neq \emptyset$. Furthermore, because x and y are in two different connected components of $G_\perp$, they are not in conflict, and, thus $N_G(x) \subseteq N_G(y)$ or $N_G(y) \subseteq N_G(x)$. Assume without loss of generality that $N_G(y) \subseteq N_G(x)$.

First, we show $N_G(C') \subseteq N_G(x)$. Assume towards a contradiction that $N_G(C') \setminus N_G(x) \neq \emptyset$. Let $A := \{y' \in C' \mid N_G(y') \subseteq N_G(x)\}$. Note that $y \in A$ and, hence, $A \neq \emptyset$. Furthermore, let $B := C' \setminus A$ and observe that $B \neq \emptyset$ by the assumption that $N_G(C') \setminus N_G(x) \neq \emptyset$. Since C' is a connected component of $G_\perp$, there are vertices $y' \in A$ and $y'' \in B$ being in conflict with each other. That is,

$$N_G(y') \setminus N_G(y'') \neq \emptyset \text{ and } N_G(y') \cap N_G(y'') \neq \emptyset \text{ and } N_G(y'') \setminus N_G(y') \neq \emptyset.$$

This directly implies that $N_G(x) \cap N_G(y'') \neq \emptyset$ (since $N_G(y') \subseteq N_G(x)$) and $N_G(x) \setminus N_G(y'') \neq \emptyset$ (since $N_G(y') \subseteq N_G(x)$). Furthermore, $y'' \in B$ implies $N_G(y'') \setminus N_G(x) \neq \emptyset$. As a consequence, x is in conflict with y'', contradicting the fact that x and y'' are contained in distinct connected components of $G_\perp$.

Next, we prove that for each $x' \in C$ it holds that $N_G(C') \subseteq N_G(x')$ or $N_G(x') \cap N_G(C') = \emptyset$. To this end, consider the following partition of C;

$$C_{\supseteq,\emptyset} := \{x' \in C \mid N_G(x') \supseteq N_G(C') \text{ or } N_G(x') \cap N_G(C') = \emptyset\},$$

$$C_\subset := \{x' \in C \mid N_G(x') \subset N_G(C')\},$$

$$C_r := C \setminus (C_{\supseteq,\emptyset} \cup C_\subset).$$

Note that $x \in C_{\supseteq,\emptyset}$, and, thus, $C_{\supseteq,\emptyset} \neq \emptyset$. To prove the above claim, we show that $C_\subset \cup C_r = \emptyset$. Assume towards a contradiction that $C_\subset \cup C_r \neq \emptyset$. We distinguish two cases based on whether $C_r = \emptyset$ or not.

Case 1: $C_r = \emptyset$**.** From assumption $C_\subset \cup C_r \neq \emptyset$ it follows that $C_\subset \neq \emptyset$. Furthermore, note that no vertex in $C_\subset$ is in conflict with a vertex in $C_{\supseteq,\emptyset}$. Since $C_{\supseteq,\emptyset} \neq \emptyset$ and $C_\subset \neq \emptyset$, this implies that C comprises at least two connected components; a contradiction.

Case 2: $C_r \neq \emptyset$**.** Let $x' \in C_r$ be arbitrarily chosen. Since $N_G(x') \setminus N_G(C') \neq \emptyset$ by the definition of C_r and x' is not in conflict with any vertex in C', for every vertex $y' \in C'$ we have either $N_G(y') \subseteq N_G(x')$ or $N_G(y') \cap N_G(x') = \emptyset$. Let $Y_1 := \{y' \in C' \mid N_G(y') \subseteq N_G(x')\}$ and $Y_2 := \{y' \in C' \mid N_G(y') \cap N_G(x') = \emptyset\}$. Note that $C' = Y_1 \cup Y_2$, $Y_1 \neq \emptyset$ (since $N_G(x') \cap N_G(C') \neq \emptyset$), and $Y_2 \neq \emptyset$ (since $N_G(C') \setminus N_G(x') \neq \emptyset$). Moreover, $N_G(Y_1) \cap N_G(Y_2) = \emptyset$ and, hence, no vertex in Y_1 is in conflict with a vertex in Y_2. Thus, graph $G_\perp[C']$ consists of at least two connected components; a contradiction. □

Recall that for a regular solution S it holds that each critical independent set of G is contained in a critical independent set of $G \Delta S$ (see Section 7.2). Next, we show that one obtains a minimum-cardinality solution by combining regular minimum-cardinality solutions for the subgraphs induced by the connected components of $G_\perp$ and their neighborhoods.

Proposition 7.1. *Let $C_1, C_2, \ldots, C_\ell$ denote the connected components of $G_\perp$ and define $G_i := G[C_i \cup N_G(C_i)]$. Let S_i denote a regular minimum-cardinality solution for G_i, $1 \leq i \leq \ell$. Then, $S := \bigcup_{i=1}^{\ell} S_i$ is a minimum-cardinality solution for G.*

Proof. The proof is organized as follows. First, we show that $|S|$ is a lower bound for an optimal solution, then we show that S is a solution, that is, $G \Delta S$ is M-free.

First, we show that $\sum_{i=1}^{\ell} |S_i|$ is a lower bound for the size of a minimum-cardinality solution for G (note that $|S| = \sum_{i=1}^{\ell} |S_i|$ since all S_i's are pairwise disjoint). To this end, observe that for every solution S' clearly $S'_i := S' \cap \{\{v_c, v_t\} \mid v_c \in C_i, v_t \in N_G(C_i)\}$ is a solution for G_i. Moreover, $|S_i| \leq |S'_i|$ since S_i is a minimum-cardinality solution for G_i. Since all S'_i's are pairwise disjoint, we thus have $|S| \leq |S'|$.

Second, we show that $G \Delta S$ is M-free. Assume towards a contradiction that there are two c-vertices c and c' that are in conflict in $G \Delta S$. Moreover, let C_i denote the connected component of $G_\perp$ containing c and let C_j denote the connected component of $G_\perp$ containing c'. Since $N_{G \Delta S}(c) \cap N_{G \Delta S}(c') \neq \emptyset$ and all edge insertions of S incident to c and c'' are between c and $N_G(C_i)$ and c' and $N_G(C_j)$, respectively, it follows that $N_G(C_i) \cap N_G(C_j) \neq \emptyset$. Thus, according to Lemma 7.1, we can assume without loss of generality that for every $x \in C_j$ it holds that either $N_G(C_i) \subseteq N_G(x)$ or $N_G(x) \cap N_G(C_j) = \emptyset$. As a consequence, $N_G(C_i) \subseteq N_G(C_j)$. Let $C_{j,1} := \{x \in C_j \mid N_G(C_i) \subseteq N_G(x)\}$ and $C_{j,2} := \{x \in C_j \mid N_G(C_i) \cap N_G(x) = \emptyset\}$. By Lemma 7.1, $C_j = C_{j,1} \cup C_{j,2}$. Hence, $N_G(C_i)$ is a critical independent set in G_j: for every vertex $t \in N_G(C_i)$ it holds that $N_{G_j}(t) = C_{j,1}$. Let t_1, t_2, t_3 denote three t-vertices that, together with c and c', induce an M-graph in $G \Delta S$. Without loss of generality, assume that $t_1 \in N_{G \Delta S}(c) \setminus N_{G \Delta S}(c')$, $t_2 \in N_{G \Delta S}(c) \cap N_{G \Delta S}(c')$, and $t_3 \in N_{G \Delta S}(c') \setminus N_{G \Delta S}(c)$. Next, we argue that $\{t_1, t_2\} \subseteq N_G(C_i)$: both t_1 and t_2 are neighbors of c in $G \Delta S$ and all edge modifications involving vertices in C_i are between C_i and $N_G(C_i)$. Finally, observe that $N_G(C_i)$ is not a critical independent set in $G_j \Delta S_j$ since $t_1 \notin N_{G_j \Delta S_j}(c')$ but $t_2 \in N_{G_j \Delta S_j}(c')$: this is a contradiction to the assumption that S_j is regular since $N_G(C_i)$ is a critical independent set in G_j. □

7.4 Data Reduction Rules

In this section, we present four polynomial-time data reduction rules for Minimum Flip Consensus Tree that produce an $O(k^3)$-vertex kernel.

As noted in Section 7.2, Reduction Rule 3.1 applies for MFCT. More specifically, since "being M-free" is a 1-cisp graph property, for MFCT Reduction Rule 3.1 reads at follows.

Reduction Rule 7.1. Let $I \subseteq V$ be a critical independent set. If $|I| > k + 1$, then delete $|I| - (k + 1)$ arbitrary vertices from I.

The first new data reduction rule is obvious.

Reduction Rule 7.2. Remove M-free connected components from the input graph.

Applying Reduction Rule 7.1 and a rule that removes all isolated bicliques already yields an $O(k^2)$-vertex kernel for Bicluster Editing [145]. However, for Minimum Flip Consensus Tree we need two further, more involved data reduction rules. The main difference here is that for M-free graphs, we have a much more complicated neighborhood structure than for P_4-free bipartite graphs (so-called bicluster graphs), where each connected component is a complete bipartite graph. That is, in contrast to bicluster graphs, where each connected component contains at most two critical independent sets, in case of M-free-graphs each connected component might contain an unbounded number of critical independent sets.

The next data reduction rule removes c-vertices from G that do not appear in any M-graph. The correctness of this rule follows from the decomposition property introduced in Section 7.3: a c-vertex c' that is not in any conflict forms a connected component $\{c'\}$ in the conflict graph $G_\perp$ whose associated MFCT–subinstance $G[\{c'\} \cup N_G(c')]$ is M-free. Thus, according to Proposition 7.1, the sizes of minimum solutions for G and $G - c'$ are equal.

Reduction Rule 7.3. Let $G = (V_c, V_t, E)$ be a bipartite graph. If there exists a vertex $c \in V_c$ that is not in conflict with any other vertex in V_c, then remove c.

Lemma 7.2. *Reduction Rule 7.3 is correct and can be exhaustively applied in* $O(|V_c|^2 \cdot |V_t|)$ *time.*

Proof. As discussed above, the correctness of Reduction Rule 7.3 follows directly from Proposition 7.1.

For the running time consider the following. To test the adjacency of two vertices in constant time, an adjacency matrix is built prior to the application of the rule. Then, for each pair of vertices $c_1, c_2 \in V_c$, we can determine in $O(V_t)$ time whether they are in conflict by checking for each vertex $t \in V_t$, whether it is adjacent to c_1, c_2, or both. Each c-vertex that is in conflict with some other vertex is marked. Finally, unmarked vertices are removed from the graph. This can be done in $O(|E|)$ time. The overall running time is thus $O(|V_c|^2 \cdot |V_t|)$. Note that every vertex that is in conflict prior to the application of the rule is also in conflict after the application of the rule. Thus, the above procedure results in an instance that is reduced with respect to Reduction Rule 7.3. □

The structurally "deepest" reduction rule shrinks induced subgraphs of the input graph that resemble "local" chain graphs. We call such a subgraph *P-structure*:

Definition 7.2. Let $G = (V_c, V_t, E)$ be a bipartite graph. A pair (C_P, T_P) of vertex sets $C_P \subseteq V_c$ and $T_P \subseteq V_t$ forms a *P-structure* if the following three properties are fulfilled:

1. $G[C_P \cup T_P]$ is a chain graph,
2. for all $c', c'' \in C_P$ it holds that $N(c') \setminus T_P = N(c'') \setminus T_P$, and
3. for all $t', t'' \in T_P$ it holds that $N(t') \setminus C_P = N(t'') \setminus C_P$.

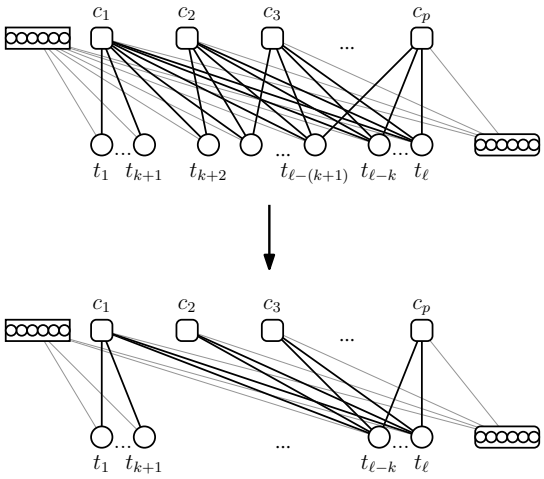


Figure 7.4: Illustration of the application of Reduction Rule 7.4. The edges within the P-structure are colored black and the edges between a vertex of the P-structure and a vertex outside the P-structure are colored grey. Reduction Rule 7.4 removes all but the first and last $k+1$ t-vertices of the P-structure.

For a P-structure (C_P, T_P) of a bipartite graph G the neighborhoods in G of the vertices in C_P (and T_P) also form a chain (since "outside" of the P-structure they have the same neighbors). Moreover, note that $G[C_P \cup T_P]$ is M-free.

Our last reduction rule shrinks large P-structures. See Figure 7.4 for an example.

Reduction Rule 7.4. Let $(G = (V_c, V_t, E), k)$ denote an MFCT-instance and let (C_P, T_P) be a P-structure in G, where $T_P = \{t_1, t_2, \ldots, t_\ell\}$ with $N_G(t_1) \subseteq N_G(t_2) \subseteq \ldots \subseteq N_G(t_\ell)$. If $\ell > 2(k+1)$, then remove $T_R := \{t_{k+2}, t_{k+3}, \ldots, t_{\ell-(k+1)}\}$ from G.

Next, we prove the correctness of Reduction Rule 7.4. After this, we show that it can be applied in polynomial time.

Lemma 7.3. *Reduction Rule 7.4 is correct.*

Proof. Let $G = (V_c, V_t, E)$, C_P, T_P, and T_R specified as in Reduction Rule 7.4. Let $C_P = \{c_1, c_2, \ldots, c_q\}$. Since (C_P, T_P) forms a P-structure, we can assume that $N_G(t_1) \subseteq N_G(t_2) \subseteq \ldots \subseteq N_G(t_\ell)$ and $N_G(c_1) \supseteq N_G(c_2) \supseteq \ldots \supseteq N_G(c_q)$. Furthermore, let $G' := G - T_R$ denote the reduced graph.

For the correctness of Reduction Rule 7.4, we prove the following.

Claim: (G, k) is a yes-instance $\iff$ (G', k) is a yes-instance.

"$\Rightarrow$": This direction follows directly from the fact that M-freeness is a hereditary graph property.

"$\Leftarrow$": Given a solution S' of size at most k for G', we show that one can construct a solution of size at most k for G. First, note that since $|S'| \leq k$ there exists an i with $1 \leq i \leq k+1$ such that t_i is not involved in any edge modification in S', that is, $N_{G'}(t_i) = N_{G' \Delta S'}(t_i)$. Analogously, there is a j with $\ell - k \leq j \leq \ell$ such that t_j is not involved in any edge modification in S', that is, $N_{G'}(t_j) = N_{G' \Delta S'}(t_j)$.

Note that $N_{G'}(t_i) \subseteq N_{G'}(t_j)$ and, hence, $N_{G'\Delta S'}(t_i) \subseteq N_{G'\Delta S'}(t_j)$. Furthermore, let $T := \{t_{i+1}, t_{i+2}, \ldots, t_{j-1}\}$ and $T' := T \setminus T_R$. Consider the edge modification set $S'' := S' \setminus E_{T',V_c}$. In other words, S'' contains all edge modifications from S' except those involving a vertex from T'. Clearly, S'' is a solution of size at most k for $G - T = G' - T'$. We show that from S'' one can build a solution of size at most k for G. To this end, we distinguish the cases that $N_G(t_i) = N_G(t_j)$ and $N_G(t_i) \subset N_G(t_j)$.

Case 1: $N_G(t_i) = N_G(t_j)$. This condition implies that $N_G(t') = N_G(t_i) = N_G(t_j)$ for every $t' \in T' \cup T_R$ since $N_G(t_i) \subseteq N_G(t') \subseteq N_G(t_j)$ and $N_G(t_i) = N_G(t_j)$. That is, in $G\Delta S''$, every vertex $t' \in T' \cup T_R$ has the same neighborhood as t_i and t_j (since t_i, t_j, and t' are not involved in any edge modification in S'') and, as a consequence of Lemma 3.1, $G\Delta S''$ is M-free (observe that $G \,\Delta\, S''$ results from $(G' - T') \,\Delta\, S''$ by adding the vertices in T step-by-step, making each vertex adjacent to every vertex in $N_G(t_i)$). Hence, S'' is a solution of size at most k for G.

Case 2: **$N_G(t_i) \subset N_G(t_j)$.** We need some observations on the structure of $G' - T'$ and $(G' - T') \,\Delta\, S''$.

First, we show that $N_G(t_j) \setminus N_G(t_i)$ is part of a critical independent set in $G' - T'$. By $N_G(t_i) \setminus C_P = N_G(t_j) \setminus C_P$ (Condition 3 of Definition 7.2), it follows that $N_G(t_j) \setminus N_G(t_i) \subseteq C_P$. Hence, all vertices from $N_G(t_j) \setminus N_G(t_i)$ have the same neighbors in $V_t \setminus T_P$ in $G' - T'$ according to Condition 2 of Definition 7.2. Moreover, every vertex in $N_G(t_j) \setminus N_G(t_i)$ is nonadjacent to all vertices in $\{t_1, \ldots, t_i\}$ since $N_G(t_1) \subseteq \ldots \subseteq N_G(t_i) \subset N_G(t_j)$ and is adjacent to all vertices in $\{t_j, \ldots, t_\ell\}$ since $N_G(t_i) \subset N_G(t_j) \subseteq \ldots \subseteq N_G(t_\ell)$. Thus, all vertices in $N_G(t_j) \setminus N_G(t_i)$ have an identical neighborhood in $G' - T'$.

Thus, by Lemma 3.5, there is a solution S of size at most k for $G' - T' = G - T$ such that $N_G(t_j) \setminus N_G(t_i)$ is contained in a critical independent set in $(G - T)\Delta S$ and neither t_i nor t_j are involved in any edge modification from S.

Next, we argue that $G_S := G\Delta S$ is M-free. Assume towards a contradiction that G_S contains an M-graph. We show that then G_S contains an M-graph $(t_l, c_l, t_m, c_r, t_r)$ such that

$$\text{exactly one of } t_l \text{ and } t_r \text{ is contained in } T \text{ and } t_m \notin T.$$

From the existence of such an M-graph we then derive a contradiction to the fact that $(G - T) \,\Delta\, S$ is M-free. To prove the existence of an M-graph as described above, consider an arbitrarily chosen M-graph $(t_l^*, c_l^*, t_m^*, c_r^*, t_r^*)$. First, we show that not both of t_l^* and t_r^* are contained in T. Observe that t_l^* has a neighbor not contained in $N_{G_S}(t_r^*)$ (namely c_l^*), and, vice versa, t_r^* has a neighbor not contained in $N_{G_S}(t_l^*)$ (namely c_r^*). Since, however, the neighborhoods of the vertices in T form a chain in G_S for any two vertices $t_x, t_y \in T$ either $N_{G_S}(t_x) \subseteq N_{G_S}(t_y)$ or $N_{G_S}(t_y) \subseteq N_{G_S}(t_x)$. Thus not both of t_l^* and t_r^* can be contained in T.

Next, assume that $t_m^* \in T$. We show that then $c_l^*, c_r^*, t_l^*, t_r^*$ and t_j (instead of t_m^*) induce an M-graph. Since $t_m^* \in T$ it holds that $N_{G_S}(t_m^*) \subseteq N_{G_S}(t_j)$, implying $t_r^* \neq t_j$ and $t_l^* \neq t_j$. Thus, since t_j is adjacent to c_l^* and c_r^* in G_S (this follows from $N_{G_S}(t_m^*) \subseteq N_{G_S}(t_j)$), the vertices c_l^*, c_r^*, t_l^*, t_r^*, and t_j (instead of t_m^*) induce an M-graph in G_S.

In summary, there is an M-graph $(t_l, c_l, t_m, c_r, t_r)$ in G_S such that not both t_l and t_r are contained in T and $t_m \notin T$. Since $(G - T) \,\Delta\, S$ is M-free, every M-graph in G_S contains at least one vertex from T. Thus, either $t_l \in T$ or $t_r \in T$. Without loss of generality assume that $t_l \in T$.

Finally, note that $c_r \notin N_{G_S}(t_i)$ because $c_r \notin N_{G_S}(t_l) \supseteq N_{G_S}(t_i)$. Moreover, note that $c_l \in N_{G_S}(t_j)$ since $N_{G_S}(t_l) \subseteq N_{G_S}(t_j)$. We distinguish the two cases $c_l \in N_G(t_i)$ and $c_l \in N_G(t_j) \setminus N_G(t_i)$ and in each case derive a contradiction.

Case 2.1: $c_l \in N_G(t_i)$**.** Then, $\{t_i, c_l, t_m, c_r, t_r\}$ induces an M-graph not containing a vertex from T because $c_l \in N_{G_S}(t_i)$ (since $c_l \in N_G(t_i)$ by the case condition) but $\{t_i, c_r\} \notin E(G_S)$ (since $c_r \notin N_{G_S}(t_i)$). This contradicts the assumption that $(G - T)\Delta S$ is M-free.

Case 2.2: $c_l \in N_G(t_j) \setminus N_G(t_i)$**.** Recall that by the discussion above $t_r, t_m \notin T$ and $c_r \notin N_G(t_i)$. Furthermore, we can assume that c_r is not contained in $N_G(t_j) \setminus N_G(t_i)$; otherwise, since $N_G(t_j) \setminus N_G(t_i)$ forms a critical independent set in $(G-T)\Delta S$ and $t_r, t_m \notin T$, it would follow that c_l and t_r are adjacent (however, $t_r \in N_{G_S}(c_r) \setminus N_{G_S}(c_l)$). Hence, $c_r \notin N_G(t_j)$ and, since c_r is adjacent to both t_m and t_r, it follows that $t_m \neq t_j$ and $t_r \neq t_j$. Thus, since $c_l \in N(t_l) \subseteq N(t_j)$ (by the case condition) but $c_r \notin N_G(t_j)$, the vertex set $\{t_j, c_l, t_m, c_r, t_r\}$ induces an M-graph in G_S not containing any vertex from T. This contradicts the M-freeness of $(G - T)\Delta S$. □

Finally, we show that Reduction Rule 7.4 can be applied in polynomial time. To this end, we first prove that a P-structure with a maximal number of t-vertices can be found in polynomial time.

Lemma 7.4. *A P-structure (C_P, T_P) such that $|T_P|$ is maximal can be found in $O(|V_c|^2 \cdot |E|)$ time.*

Proof. Assume that G contains a P-structure (C_P, T_P) where $C_P = \{c_1, c_2, \ldots, c_p\}$ and $N(c_1) \supseteq N(c_2) \supseteq \ldots \supseteq N(c_p)$. We show that given the two vertices $c_1, c_p \in C_P$ we can find in $O(|E|)$ time a P-structure (C, T) with $C = C_P$ such that $|T|$ is maximal. Hence, by trying all pairs $c', c'' \in V_c$, we can find a P-structure (C_P, T_P) for which $|T_P|$ is maximal in $O(|V_c|^2 \cdot |E|)$ time. We distinguish the cases $N(c_1) = N(c_P)$ and $N(c_1) \supset N(c_p)$.

Case 1: $N(c_1) = N(c_p)$**.** Any two vertices in C_P have an identical neighborhood. Hence, we can assume that $C_P = I$, where I denotes the critical independent set containing c_1 and c_p. Furthermore, since C_P is a critical independent set, Definition 7.2 implies that T_P is a critical independent set, too. Moreover, it is not hard to verify that I together with any critical independent set in $N(c_1)$ forms a P-structure. Note that the set of all critical independent sets of a graph can be computed in $O(|V_t|+|V_c|)$ time [133]. Hence, one can find a P-structure (I, T_P) such that $|T_P|$ is maximal in $O(|E|)$ time.

Case 2: $N(c_1) \supset N(c_p)$**.** We use the following notation. For two c-vertices c', c'' with $N(c'') \subseteq N(c')$ let

$$
\begin{aligned}
S(c', c'') &:= \{x \in V_c \mid N(c'') \subseteq N(x) \subseteq N(c')\},\\
T'(c', c'') &:= N(c') \setminus N(c''),\\
C_{\text{out}}(c', c'') &:= N(T'(c', c'')) \setminus S(c', c''),\\
T''(c', c'') &:= \{t \in N(c'') \mid N(t) = (S(c', c'') \cup C_{\text{out}}(c', c''))\}.
\end{aligned}
$$

Let (C_P, T_P) denote a P-structure of G where $C_P = \{c_1, c_2, \ldots, c_p\}$ with $N(c_1) \supseteq N(c_2) \supseteq \ldots \supseteq N(c_p)$ and $N(c_1) \supset N(c_p)$. We show that in this case (C_P, T_P) is uniquely determined by c_1 and c_p. More precisely, we show the following.

Claim: If (C_P, T_P) is a P-structure, then $C_P = S(c_1, c_p)$ and $T_P = T'(c_1, c_p) \cup T''(c_1, c_p)$.

First, we show that $C_P = S(c_1, c_p)$. From the definition of a P-structure it follows directly that $C_P \subseteq S(c_1, c_p)$. Hence, it remains to show that $S(c_1, c_p) \subseteq C_P$. Assume towards a contradiction that there exists a vertex $x \in S(c_1, c_p) \setminus C_p$. Note that, since $N(c_1) \setminus T_P = N(c_p) \setminus T_P$ (Condition 2 of Definition 7.2), it holds that $T'(c_1, c_p) \subseteq T_P$. Furthermore, observe that we can assume that x has not the same neighborhood as c_1 or c_p; since (C_P, T_P) is maximal, every vertex with the same neighborhood as c_1 or c_2 belongs to C_P. Thus, $N(c_1) \supset N(x) \supset N(c_p)$, and, consequently, there is a vertex $t' \in N(x) \setminus N(c_p)$ and a vertex $t'' \in N(c_1) \setminus N(x)$. Clearly, $t', t'' \in T'(c_1, c_p)$ and, thus, $t', t'' \in T_P$. Since by assumption $x \notin C_P$, it holds that $x \in N(t') \setminus C_P$ but $x \notin N(t'') \setminus C_P$; a contradiction to Condition 3 of Definition 7.2.

Second, we show that $T_P = T'(c_1, c_p) \cup T''(c_1, c_p)$. As argued above, $T'(c_1, c_p) \subseteq T_P$ and, thus, it remains to show that $T_P \setminus T'(c_1, c_p) = T''(c_1, c_p)$. Observe that $C_{\text{out}}(c_1, c_p) = N(t) \setminus C_P$ for each vertex $t \in T_P$ since $T'(c_1, c_p) \subseteq T_P$, $C_P = S(c_1, c_p)$, and due to Condition 3 of Definition 7.2. Hence, $T_P \setminus T'(c_1, c_p) \subseteq T''(c_1, c_p)$, since each vertex in $T_P \setminus T'(c_1, c_p)$ is adjacent to all vertices in $C_{\text{out}}(c', c'') \cup S(c', c'')$. Finally, note that $(C_P, T'(c_1, c_p) \cup T''(c_1, c_p))$ is a P-structure. Thus, $T_P = T'(c_1, c_p) \cup T''(c_1, c_p)$ by the maximality of (C_P, T_P). This concludes the proof of the above claim.

By the above claim, a P-structure (C_P, T_P) where $C_P = \{c_1, c_2, \ldots, c_p\}$ with $N(c_1) \supseteq N(c_2) \supseteq \ldots \supseteq N(c_p)$ and $N(c_1) \supset N(c_p)$ is uniquely determined by c_1 and c_p. Thus, it remains to show that computing the sets $C := S(c', c'')$ and $T := T'(c', c'') \cup T''(c', c'')$ and testing whether the found subgraph is a P-structure are doable in $O(|E|)$ time.

To compute the set $S(c', c'')$ proceed as follows. First, compute the sets $N(c')$ and $N(c'')$ and check whether $N(c'') \subseteq N(c')$. If so, in one iteration over E compute for every c-vertex x the values $a(x) := |N(c') \cap N(x)|$, $b(x) := |N(c'') \cap N(x)|$, and $c(x) := |N(x) \setminus N(c')|$. Clearly, $a(x) \leq |N(c')|$, $b(x) = |N(c'')|$, and $c(x) = 0$ if and only if $x \in S(c', c'')$. Using similar ideas, it is straightforward to verify that the sets $T'(c', c'')$ and $T''(c', c'')$ can be computed in $O(|E|)$ time.

Finally, we show that, given a tuple (C_P, T_P), it can be decided in $O(|E|)$ time whether it forms a P-structure. To this end, proceed as follows. First, compute the sets $T' := N(C_P) \setminus T_P$ and $C' := N(T_P) \setminus C_P$. Note that then in one iteration over E it is possible to check whether $N(c) \setminus T_P = T'$ for each $c \in C_P$. Analogously, check whether $N(t) \setminus C_P = C'$ for each $t \in C_P$. If this test is passed without conflicts, then it remains to verify that the neighborhoods of the vertices in C_P form a chain. To this end, sort the vertices in C_P in decreasing order of their degrees using bucket sort (in linear time). Let $C_P = \{c_1, \ldots, c_p\}$ where $\deg(c_1) \geq \ldots \geq \deg(c_p)$. Clearly, by first marking the neighbors of c_i and then iterating over c_{i-1}'s neighbors, it is possible to test whether $N(c_i) \subseteq N(c_{i-1})$ in time $O(\deg(c_{i-1}))$. Hence, checking whether the neighborhoods of the vertices in C_P form a chain is doable in $O(\sum \deg(c_i)) = O(|E|)$ time. □

Lemma 7.5. *In $O(|V_c|^2|E|)$ time, Reduction Rule 7.4 can be applied once or it can be decided that the instance is reduced with respect to Reduction Rule 7.4.*

Proof. By Lemma 7.4, a P-structure (C_P, T_P) such that $|T_P|$ is maximal can be computed in $O(|V_c|^2 \cdot |E|)$ time. Note that only the size of $|T_P|$ is important for the decision

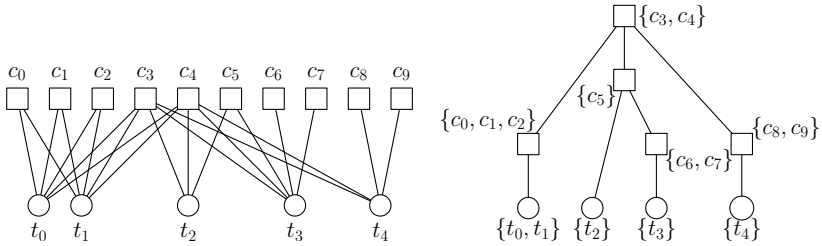


Figure 7.5: An M-free graph G and the corresponding tree $T_{\text{cis}}(G)$.

whether Reduction Rule 7.4 applies. Thus, if $|T_P| \leq 2(k+1)$, then the instance is reduced with respect to Reduction Rule 7.4. Otherwise, Reduction Rule 7.4 can be applied in $O(|E|)$ time. Thus, the overall running time is bounded by $O(|V_c|^2|E|)$. □

7.5 Analysis of the Problem Kernel Size

In this section, we bound the maximum number of vertices in an instance that is reduced with respect to Rules 7.1–7.4.

For the analysis of the problem kernel size, we make use of the representation of M-free graphs as rooted trees. Recall that, given a connected and M-free graph $G = (V_c, V_t, E)$, one can construct a rooted tree T with node set $V_t \cup V_c$ and with $L(T) = V_t$ such that $t_i \in V_t$ is a descendant of $c_j \in V_c$ if and only if $t_i \in N_G(c_j)$, see [45, 100, 143] for details.

Note that the critical independent set graph of an M-free graph is M-free. Hence, we can find a tree with the property that every leaf one-to-one corresponds to a critical independent set of the t-vertices and every inner vertex one-to-one corresponds to a critical independent set of the c-vertices. For a connected and M-free graph G, we denote this tree by $T_{\text{cis}}(G)$. Figure 7.5 shows an M-free graph G together with $T_{\text{cis}}(G)$.

The following easy observation is helpful in the analysis of the kernel size.

Observation 7.1. *For an M-free graph G the following holds.*

1. *Every inner node of $T_{cis}(G)$ has at most one leaf child, and*
2. *every inner node of $T_{cis}(G)$ with at most one non-leaf child has exactly one leaf child.*

Now, we arrive at our main result.

Theorem 7.1. Minimum Flip Consensus Tree *admits an $O(k^3)$-vertex problem kernel. The kernelization runs in $O(|V_c|^2 \cdot |V_t| \cdot |E|)$ time.*

Proof. Consider an instance $(G = (V_c, V_t, E), k)$ that is reduced with respect to Rules 7.1–7.4. We show that if (G, k) is a yes-instance, then the number of vertices in $V_c \cup V_t$ is $O(k^3)$.

Assume that (G, k) is a yes-instance and let S denote an optimal solution of size at most k for G. Moreover, let $G_S := G \Delta S$. Recall that the vertices that are involved in any edge modification of S are called *affected*. All other vertices are called *nonaffected*.

Let X_c denote the set of affected c-vertices and let Y_c denote the set of nonaffected c-vertices. Analogously, define X_t and Y_t as the set of affected and nonaffected t-vertices, respectively. Since every edge modification involves a c-vertex and a t-vertex, we have $|X_c| \leq k$ and $|X_t| \leq k$. Hence, it remains to bound $|Y_c \cup Y_t|$.

Let $G_{S,1}, G_{S,2}, \ldots, G_{S,p}$ denote the connected components of G_S. For each i, $1 \leq i \leq p$, let $T_i := T_{\text{cis}}(G_{S,i})$ denote the rooted tree corresponding to the critical independent set graph of $G_{S,i}$. Moreover, let T denote the forest comprising all T_i's. Recall that the leaves of T one-to-one correspond to the critical independent sets of V_t in G_S and that the inner nodes of T one-to-one correspond to the critical independent sets of V_c in G_S. For a node $z \in V(T)$, let $\mathcal{C}(z)$ denote the set of vertices contained in the critical independent set corresponding to z. Moreover, for $Z \subseteq V(T)$, define $\mathcal{C}(Z) := \bigcup_{z \in Z} \mathcal{C}(z)$. Finally, let $T' := T - L(T)$ denote the subforest of T induced by the critical independent sets corresponding to the c-vertices.

For the analysis of the kernel size, we partition the set of inner nodes into three sets A, B, and Q. The set A contains all inner nodes z of T for which at least one of the following two statements holds: $\mathcal{C}(z) \cap X_c \neq \emptyset$ or z has a leaf child w with $\mathcal{C}(w) \cap X_t \neq \emptyset$. Note that $|A| \leq 2k$ since there are at most $2k$ affected vertices. Moreover, let B be the set of the inner nodes that are not contained in A and that have at least two non-leaf children. Further, Q contains all inner nodes not contained in $A \cup B$. For a set of inner nodes Z, let L_Z denote the set of leaves incident to a vertex in Z.

First, we bound the number of the vertices contained in the critical independent sets corresponding to the nodes in $A \cup B$ and $L_A \cup L_B$. To this end, we show the following.

1. For every inner node $x \in B \cup Q$, there exists at least one node $y \in V(T_x)$ with $y \in A$.

2. The cardinality of B is at most $2k$.

3. The number of vertices contained in the critical independent sets corresponding to the nodes in $A \cup B \cup L_A \cup L_B$ is bounded by $O(k^2)$.

1.) Assume towards a contradiction that there is an inner node $x \in B \cup Q$ such that $V(T_x) \cap A = \emptyset$. That is, no vertex in $\mathcal{C}(V(T_x))$ is affected. Consider a vertex $c' \in \mathcal{C}(x)$. We show that c' is not contained in any conflict in G, contradicting the fact that G is reduced with respect to Reduction Rule 7.3. First, for every c-vertex y in $\mathcal{C}(V(T_x))$ it holds that $N_G(y) \subseteq N_G(c')$ since $N_{G_S}(y) \subseteq N_{G_S}(c')$ and S does not affect c or y. Second, for every c-vertex y in $\mathcal{C}(V(T) \backslash V(T_x))$ it holds that $N_{G_S}(c') \cap N_{G_S}(y) = \emptyset$ or $N_{G_S}(c') \subseteq N_{G_S}(y)$. As a consequence, since neither c' nor any t-vertex in $N_{G_S}(c')$ is affected, it follows that $N_G(c') \cap N_G(y) = \emptyset$ or $N_G(c') \subseteq N_G(y)$. This means that c' is not contained in any conflict in G.

2.) Recall that the forest T' results from deleting all leaves of T. Since each node of B has at least two non-leaf children in T, it has at least two children in T'. From 1.) it follows directly that the leaves of T' are contained in A and, hence, their number is bounded by $2k$. Since the number of inner nodes with at least two children is bounded by the number of leaves, we arrive at the bound $|B| \leq 2k$.

3.) First, note that $|A \cup B| \leq 4k$ since A and B each have cardinality at most $2k$. Moreover, $|L_A \cup L_B| \leq 4k$ since every inner node has at most one leaf child (see

Observation 7.1). For every node $y \in A \cup B \cup L_A \cup L_B$, define $\mathcal{C}'(y) := \mathcal{C}(y) \setminus (X_c \cup X_t)$ and observe that $\mathcal{C}'(y)$ forms a critical independent set in G since no vertex in $\mathcal{C}'(y)$ is affected. Thus, since G is reduced with respect to Reduction Rule 7.1, it follows that $|\mathcal{C}'(y)| \leq k+1$. Putting all together, we obtain

$$|\mathcal{C}(A \cup B \cup L_A \cup L_B)| \leq |X_c| + |X_t| + \sum_{y \in A \cup B \cup L_A \cup L_B} |\mathcal{C}'(y)| \leq 2k + 4k(k+1).$$

So far, we have bounded the number of vertices in $X_c \cup X_t$ and $\mathcal{C}(A \cup B \cup L_A \cup L_B)$ by $O(k^2)$. It remains to bound the number of vertices contained in $\mathcal{C}(Q \cup L_Q)$.

Observe that each inner node contained in Q (and hence not contained in $A \cup B$) has exactly one non-leaf child since $L(T') \subseteq A$ (see 2.) above). That is, in the the forest $T' = T - L(T)$ these vertices have degree two. Moreover, by Observation 7.1, each node contained in Q has exactly one leaf child. Recall that all leaves of T' are contained in A and hence $|L(T')| \leq 2k$. Consider a path $P = (\{x, y_1\}, \{y_1, y_2\}, \ldots, \{y_{l-1}, y_l\}, \{y_l, z\})$ in T' with $y_i \in Q$ for all $1 \leq i \leq l$ and $x, z \in A \cup B$. Such a path is called a *degree-two-path* in the following since by the above discussion $\deg_{T'}(y_i) = 2$ for all $1 \leq j \leq l$. Further, for every y_i, let w_i denote the leaf child of y_i in T. Note that in the forest T', there are at most $8k$ degree-two-paths since $L(T') \subseteq A$ and $|A \cup B| \leq 4k$. In the following, we bound the length of each degree-two-path by $2(k+1)$. Hence, for each such path we have

$$\sum_{i=1}^{l} (|\mathcal{C}(y_i)| + |\mathcal{C}(w_i)|) \leq l \cdot (2(k+1)) \leq (2(k+2)) \cdot 2(k+1)$$

vertices in G. Adding up over the at most $8k$ degree-two-paths, this amounts to $8k \cdot 2(k+1)(2(k+2)) \leq 32k(k+1)(k+2)$ vertices, yielding the bound of $O(k^3)$ vertices in total.

Next, we bound the length of each degree-two-path. To this end, consider such a degree-two-path $P = (\{x, y_1\}, \{y_1, y_2\}, \ldots, \{y_{l-1}, y_l\}, \{y_l, z\})$ in T', that is, $x, z \in A \cup B$ and $y_i \in Q$ for all $1 \leq i \leq l$. Without loss of generality, we assume that y_l is a descendent of y_1. For each y_i let w_i denote the one and only leaf child adjacent to y_i. See Figure 7.6 for an example. Let $C_P := \bigcup_{i=1}^{l} \mathcal{C}(y_i)$ and $T_P := \bigcup_{i=1}^{l} \mathcal{C}(w_i)$. We show that (C_P, T_P) forms a P-structure in G. First, note that $C_P \subseteq V_c$ and $T_P \subseteq V_t$. Recall that by definition all vertices in $C_P \cup T_P$ are unaffected. Next, observe that $G[C_P \cup T_P]$ forms a chain graph. This can be seen as follows. In G_S a vertex in $\mathcal{C}(y_1)$ is clearly adjacent to all vertices in T_P, a vertex in $\mathcal{C}(y_2)$ is adjacent to all vertices in $T_P \setminus \mathcal{C}(w_1)$, a vertex in $\mathcal{C}(y_3)$ is adjacent to all vertices in $T_P \setminus \mathcal{C}(\{w_1, w_2\})$, and so on. Hence, $G_S[C_P \cup T_P]$ is a chain graph and, since no vertex in C_P is involved in an edge modification, we have that $G[C_P \cup T_P]$ forms a chain graph, too (see Figure 7.6). Next, we show that C_P and T_P fulfill the second and third property of a P-structure. First, every vertex in C_P is adjacent in G_S to all vertices contained in the critical independent sets corresponding to the leaves in T_z and, hence, for all $c, c' \in C_P$, we have $N_{G_S}(c) \setminus T_P = N_{G_S}(c') \setminus T_P$. Since no vertex in C_P is affected, this implies that $N_G(c) \setminus T_P = N_G(c') \setminus T_P$ for all $c, c' \in C_P$. Second, every vertex $t \in T_P$ is adjacent in G_S (and hence in G) to all c-vertices contained in a critical independent set on the path from the root r to z. Hence, for any two vertices $t, t' \in T_P$ it holds that $N_G(t) \setminus T_P = N_G(t') \setminus T_P$. In summary, (C_P, T_P) forms a P-structure.

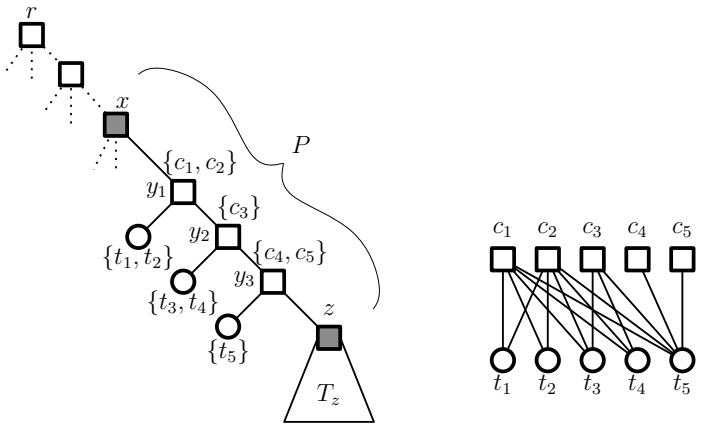


Figure 7.6: A degree-two-path P and the corresponding chain graph. Herein, $\mathcal{C}(y_1) = \{c_1, c_2\}$, $\mathcal{C}(y_2) = \{c_3\}$, and $\mathcal{C}(y_3) = \{c_4, c_5\}$.

Finally, we show that $l \leq 2(k+1)$. Assume towards a contradiction that $l > 2(k+1)$. This implies that $|T_P| > 2(k+1)$, too, since every y_i has exactly one leaf child that corresponds to a critical independent set of V_t. Hence, $|T_P| > 2(k+1)$ and thus all conditions to apply Reduction Rule 7.4 are fulfilled: a contradiction to the fact that G is reduced.

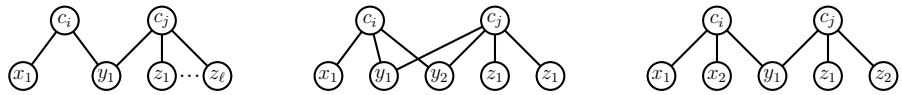
To prove the running time bound, we assume that the data reduction rules are applied as follows. First, we show that an instance reduced with respect to the Rules 7.3 and 7.4 can be computed in $O(|V_c|^2 \cdot |V_t| \cdot |E|)$ time. To this end, proceed as follows. Apply Reduction Rule 7.3 exhaustively. Then, check whether Reduction Rule 7.4 can be applied. If not, the instance is reduced with respect to both rules. Otherwise, apply Reduction Rule 7.3 exhaustively again and subsequently check whether Reduction Rule 7.4 can be applied, and so on. Note that Reduction Rule 7.4 is applied $O(|V_t|)$ times since each time (except for the last time) at least one t-vertex is removed. Hence, by Lemmas 7.2 and 7.5, one obtains a graph that is reduced with respect to Rules 7.3 and 7.4 in $O(|V_t| \cdot (|V_c|^2|V_t| + |V_c|^2|E|)) = O(|V_c|^2|V_t||E|)$ time.

Finally, observe that applying Reduction Rule 7.2 and subsequently Reduction Rule 7.1 to a graph reduced with respect to Rules 7.3 and 7.4 leaves a graph reduced with respect to Rules 7.3 and 7.4. Thus, the running time is dominated by the application of Rules 7.3 and 7.4. Hence, the kernelization runs in $O(|V_c|^2 \cdot |V_t| \cdot |E|)$ time. □

7.6 An $O(3.68^k)$-Size Search Tree

In this section, we present an $O(3.68^k)$-size search tree algorithm for Minimum Flip Consensus Tree, achieving a running time of $O(3.68^k + |V_c|^2 \cdot |V_t| \cdot |E|)$ instead of the previous $O(4.42^k \cdot (|V_c| + |V_t|) + |V_c| \cdot |V_t|)$ [27]. Basically, we use the same branching strategy as Böcker et al. [27]. Our main achievement is that exploiting Proposition 7.1 allows us to stop the branching process at an earlier stage.

For the presentation of our results, we use the following notation. Following Böcker



a) *b)* *c)*

Figure 7.7: The three cases for which new branching rules are designed.

et al. [27], for two c-vertices $c_i, c_j \in V_c$ we define $X(c_i, c_j) := N(c_i) \setminus N(c_j)$, $Y(c_i, c_j) := N(c_i) \cap N(c_j)$, and $Z(c_i, c_j) := N(c_j) \setminus N(c_i)$. Note that $c_i \perp c_j$ if and only if all three sets are nonempty. See Section 2.5 for the definition of branching rules and basic notation concerning depth-bounded search trees.

Böcker et al. [27] introduced a branching rule that, for two conflicting c-vertices c_i and c_j, branches into all possibilities to make one of the sets $X(c_i, c_j)$, $Y(c_i, c_j)$, and $Z(c_i, c_j)$ empty. To this end, they branch into $2^{|X(c_i,c_j)|} + 2^{|Y(c_i,c_j)|} + 2^{|Z(c_i,c_j)|}$ cases. In each of the first $2^{|X(c_i,c_j)|}$ branches, parameter k is decreased by $|X(c_i, c_j)|$, in the each of the subsequent $2^{|Y(c_i,c_j)|}$ branches parameter k is decreased by $|Y(c_i, c_j)|$, and in each of the last $2^{|Z(c_i,c_j)|}$ cases parameter k is decreased by $|Z(c_i, c_j)|$. We refer to [27] for any details and the correctness of this branching strategy. For example, if $|X(c_i, c_j)| = 3$, $|Y(c_i, c_j)| = 2$ and $|Z(c_i, c_j)| = 1$, then the branching vector is $(3, 3, 3, 3,\ 3, 3, 3, 3,\ 2, 2, 2, 2,\ 1, 1)$ leading to a branching number of 3.68. Indeed, using standard branching analysis tools it is easy to verify that the branching number is at most 3.68 if the size of one of these sets is at least 3 and the size of two of these sets is at least 2. Moreover, if $|X(c_i, c_j)| = |Y(c_i, c_j)| = |Z(c_i, c_j)| = 2$, then the branching vector is $(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2)$ giving a branching number of 3.47. Further, one can verify that the branching number is at most 3.47 if all of these sets have size at least two.

The basic idea behind the improved search tree algorithm is as follows. We will show that if for all conflicting c-vertices c_i and c_j it holds that $|X(c_i, c_j)| = |Z(c_i, c_j)| = 1$, then MFCT can be solved in polynomial time. Moreover, for all other cases we present branching rules with corresponding branching numbers better than 3.68. To this end, we use the branching strategy introduced by Böcker et al. as long as it yields a branching number better than 3.68. For the remaining cases we devise refined branching strategies, based on a simple structural observation.

We use the following branching rules.

1. If there are two conflicting c-vertices c_i, c_j such at least one of the three sets $X(c_i, c_j)$, $Y(c_i, c_j)$, and $Z(c_i, c_j)$ has cardinality three and at least two of the three sets $X(c_i, c_j)$, $Y(c_i, c_j)$, and $Z(c_i, c_j)$ have cardinality at least two, or if all three sets $X(c_i, c_j)$, $Y(c_i, c_j)$, and $Z(c_i, c_j)$ have cardinality at least two, then apply the branching strategy by Böcker et al. [27].

2. If there are two conflicting c-vertices c_i, c_j with $|X(c_i, c_j)| = |Y(c_i, c_j)| = 1$ and $|Z(c_i, c_j)| > 1$ (that is, c_i has degree two), then proceed as follows. See Figure 7.7 *a)* for notation. First, branch into the case to delete the edge $\{x_1, c_i\}$. Second, branch into the case to add the edge $\{x_1, c_j\}$. Third, branch into the case to delete the edge $\{y_1, c_j\}$. Fourth, branch into the case to delete all edges $\{\{z_p, c_i\} \mid 1 \leq p \leq \ell\}$.

3. If there are two conflicting c-vertices c_i, c_j with $|X(c_i, c_j)| = 1$ and $|Y(c_i, c_j)| = |Z(c_i, c_j)| = 2$, then proceed as follows. See Figure 7.7 *b*) for notation. First, branch into the case to delete edge $\{x_1, c_i\}$. Second, branch into the case to add $\{x_1, c_j\}$. Third, branch into the three cases to delete each time the two edges $S_{r,s} := \{\{y_1, c_r\}, \{y_2, c_s\}\}$ for $r \in \{i, j\}$ and $s \in \{i, j\}$ with $r = j$ or $s = j$. Fourth, branch into the three cases to modify each time the two edges $S_{r,s} := \{\{z_1, c_r\}, \{z_2, c_s\}\}$ for $r \in \{i, j\}$ and $s \in \{i, j\}$ with $r = j$ or $s = j$.

4. If there are two conflicting c-vertices c_i, c_j with $X|(c_i, c_j)| = 2$, $|Y(c_i, c_j)| = 1$, and $|Z(c_i, c_j)| = 2$, then proceed as follows. See Figure 7.7 *c*) for notation. First, branch into the case to delete $\{y_1, c_i\}$. Second, branch into the case to delete $\{y_1, c_j\}$. Third, branch into the two cases to modify each time the two edges $S_{r,s} := \{\{x_1, c_s\}, \{x_2, c_r\}\}$ for $r \in \{i, j\}$ and $s \in \{i, j\}$ with $r \neq s$. Fourth, branch into the two cases to modify each time the two edges $S_{r,s} := \{\{z_1, c_s\}, \{z_2, c_r\}\}$ for $r \in \{i, j\}$ and $s \in \{i, j\}$ with $r \neq s$.

In all branching rules, the parameter is decreased by the number of modified edges in each branch.

The correctness of all new branching rules is based on the following simple observation. A degree-one c-vertex is not part of any induced M-graph since the c-vertices of an M-graph have degree two. Hence, if a c-vertex c is involved in $\deg(c) - 1$ edge modifications, then we can assume that all these edge modifications are edge deletions. Otherwise, one can "undo" the edge modifications incident to c and instead delete $\deg(c) - 1$ arbitrary edges incident to c. Observe that this destroys all induced M-graphs containing c but does not create new induced M-graphs since the neighborhoods of the other c-vertices are not modified. Formally, for the correctness of the branching rules we employ the following observation.

Lemma 7.6. *Let (G, k) denote an MFCT-instance. Moreover, let $c \in V_c$ such that c is in conflict with some other c-vertex. Let $t \in N_G(c)$. If there is a solution S containing at least $\deg_G(c)-1$ edge modifications involving c, then there is a solution S', $|S'| \leq |S|$ with $\{c, t\} \in S'$.*

Proof. Let $G_S := G \,\Delta\, S$ and assume that $\{c, t\} \notin S$. Note that $\deg_G(c) > 1$ since c is contained in an induced M-graph. Let $S_c \subseteq S$ denote the edge modifications involving c. By the assumption of the lemma, we have that $|S_c| \geq \deg_G(c) - 1$.

Let E_c denote a set of $\deg_G(c) - 1$ edges incident to c in G containing $\{c, t\}$. Let $S' := (S \setminus S_c) \cup E_c$. That is, instead of the edge modifications of S involving c, S' contains all but one edge incident to c in G. Clearly $|S'| \leq |S|$. Finally, we argue that $G \,\Delta\, S'$ is M-free. To this end, observe that $N_{G_S}(c') = N_{G \,\Delta\, S'}(c')$ for each c-vertex $c' \in V_c \setminus \{c\}$. Hence, any two c-vertices $c', c'' \in V_c \setminus \{c\}$ are not in conflict. Moreover, c is not in conflict with any other c-vertex $c' \in V_c \setminus \{c\}$ since $\deg_{G \,\Delta\, S'}(c) = 1$. Thus, $G \,\Delta\, S'$ is M-free. □

Lemma 7.7. *The above branching rules are correct. The branching number of all branching rules is bounded from above by* 3.68.

Proof. For the correctness of Branching Rule 1 see Böcker et al. [27]. The correctness of the other three branching rules is based on Lemma 7.6. If $|X(c_i, c_j)| = |Y(c_i, c_j)| = |Z(c_i, c_j)| = 2$, then the branching number is bounded by 3.47 and if one of these sets

has cardinality three, one has cardinality two, and one has cardinality one, then the branching number is bounded by 3.68. Hence, the branching number of Branching Rule 1 is bounded by 3.68.

Correctness of Branching Rule 2. By Lemma 7.6, if there is a solution that contains an edge modification involving c_i, then we can assume that $\{c_i, x_1\}$ is deleted. The correctness follows by the fact that the branching rule branches into all cases to make one of the sets $X(c_i, c_j)$, $Y(c_i, c_j)$, and $Z(c_i, c_j)$ empty, omitting the cases that c_i is involved into an edge modification other than the deletion of $\{c_i, x_1\}$. The branching vector of Branching Rule 2 is $(1, 1, 1, x)$ with $x > 1$. Hence, the branching number is bounded by 3.31.

Correctness of Branching Rule 3. By Lemma 7.6, if there is a solution containing $\{\{y_1, c_i\}, \{y_2, c_i\}\}$ or $\{\{z_1, c_i\}, \{z_2, c_i\}\}$, then there exists a solution containing $\{x_1, c_i\}$. The case that $\{x_1, c_i\}$ is contained in the solution is considered by the rule. Hence, the correctness of the rule follows from the fact that the branching rule branches into all cases to make one of the sets $X(c_i, c_j)$, $Y(c_i, c_j)$, and $Z(c_i, c_j)$ empty, omitting the cases that c_i is involved into two edge modification (not containing $\{c_i, x_1\}$). The branching vector of Branching Rule 3 is $(1, 1, 2, 2, 2, 2, 2, 2)$, giving a branching number of 3.65.

Correctness of Branching Rule 4. By Lemma 7.6, if there is a solution that contains $\{\{x_1, c_i\}, \{x_2, c_i\}\}$, $\{\{z_1, c_i\}, \{z_2, c_i\}\}$, $\{\{x_1, c_j\}, \{x_2, c_j\}\}$, or $\{\{z_1, c_j\}, \{z_2, c_j\}\}$, then there is a solution containing either $\{y_1, c_i\}$ or $\{y_1, c_j\}$. Hence, these four cases must not be considered by Branching Rule 4. The branching vector of Branching Rule 4 is $(1, 1, 2, 2, 2, 2)$, leading to a branching number of 3.24. □

Next, we show that, by exploiting Proposition 7.1, an instance to which none of these branching rules applies can be solved in polynomial time. In what follows, a graph is called *conflict regular* if it fulfills the property that for any two conflicting c-vertices c_i and c_j it holds that $|X(c_i, c_j)| = 1$ and $|Z(c_i, c_j)| = 1$. It is easy to observe that if none of the four branching rules applies, then the graph is conflict regular. We show that for conflict regular graphs MFCT can be solved in polynomial time. This extends and generalizes an approach by Böcker et al. [27, Section 5.2] for instance where all c-vertices have degree three and two common neighbors.

An easy observation is that in a conflict regular graph any two conflicting vertices have the same degree. The next lemmas describe the structure of conflict regular graphs in more detail.

Lemma 7.8. *Let $G = (V_c, V_t, E)$ denote a conflict regular graph. Let c_1 and c_2 denote two conflicting c-vertices of degree at least three. Let $c' \in V_c$ with $N(c') \neq N(c_1)$. If c' is in conflict with c_2, then c' is in conflict with c_1.*

Proof. Consider a vertex c' with $c' \perp c_2$. Assume towards a contradiction that c' is not in conflict with c_1. Since G is conflict regular, all three vertices have the same degree and $N(c') \cap Y(c_1, c_2) \neq \emptyset$. Thus, since c' is not in conflict with c_1, it holds that $N(c') \subseteq N(c_1)$ or $N(c_1) \subseteq N(c')$. Since $N(c') \neq N(c_1)$, this is a contradiction to the fact that all three vertices have the same degree. □

Lemma 7.9. *Let $G = (V_c, V_t, E)$ denote a conflict regular graph and let C denote a connected component of $G_\perp$ with $|C| \geq 2$ and $\deg_G(c) \geq 3$ for all $c \in C$. Then, one of the following statements holds.*

1. *There is a set $Y \subseteq N_G(C)$ such that for any two vertices $c, c' \in C$ with $N_G(c) \neq N_G(c')$ it holds that $Y = Y(c, c')$.*

2. *For all $c \in C$, it holds that $|N_G(C) \setminus N_G(c)| = 1$.*

Proof. Let $X_1, \ldots, X_\ell$ denote the critical independent sets of G formed by the vertices in C. Note that Lemma 7.8 implies that any two vertices from different X_i's are in conflict. Moreover, observe that if $\ell = 2$, then the first statement of the lemma holds. Hence, in the following, we focus on the case $\ell > 2$.

Let $c_1 \in X_1$, $c_2 \in X_2$. Moreover, let $C' := \{c \in C \setminus (X_1 \cup X_2) \mid Y(c, c_1) = Y(c, c_2) = Y(c_1, c_2)\}$.

First, consider the case that $C' \neq \emptyset$. Assume without loss of generality that $C' = \bigcup_{i=3}^{s} X_i$ for some $s \geq 3$. We show that $s = \ell$ and, hence, the first statement of the lemma holds for $Y = Y(c_1, c_2)$. To this end, let $c_3 \in X_3$ and observe that $|N_G(c_i) \setminus (N_G(c_j) \cup N_G(c_k))| = 1$ for any three distinct $i, j, k \in \{1, 2, 3\}$. Hence, $|N(c_1) \cup N(c_2) \cup N(c_3)| \geq d + 2$, where d is the degree of the vertices in C (recall that all vertices in C have the same degree). Assume towards a contradiction that $s < \ell$ and let $c_\ell \in X_\ell$. Inductively applying Lemma 7.8, it follows that c_ℓ is in conflict with each of c_1, c_2, and c_3. Note that since $c_\ell \notin C'$ and G is conflict regular it holds that $|Y(c_1, c_2) \setminus N(c_\ell)| = 1$ and, except for this vertex, c_ℓ is adjacent to all vertices in $N(c_1) \cup N(c_2) \cup N(c_3)$; a contradiction to the fact that that $\deg_G(c_\ell) = d$.

Second, consider the case that $C' = \emptyset$. That is, for every vertex $c' \in \bigcup_{i=3}^{\ell} X_i$ it holds that $Y(c_1, c_2) \setminus N_G(c') \neq \emptyset$. By Lemma 7.8, c' is in conflict with both c_1 and c_2 and, hence, $\deg(c_1) = \deg(c_2) = \deg(c')$. Moreover, since G is conflict regular, c' is adjacent to all but one vertex in $N_G(c_1)$ and $N_G(c_2)$. Altogether, since c' is nonadjacent to a common neighbor of c_1 and c_2 this implies that $N(c') \subseteq N(c_1) \cup N(c_2)$. Hence, $N_G(C) \subseteq N_G(c_1) \cup N_G(c_2)$ and, since all vertices in C have the same degree, the second statement of the lemma holds. □

Theorem 7.2. Minimum Flip Consensus Tree *can be solved in $O(|V_c|^2 \cdot |V_t|)$ time for conflict regular graphs.*

Proof. Let $(G = (V_t, V_c, E), k)$ denote an MFCT-instance where G is a conflict regular graph. Moreover, let $G_\perp$ denote the conflict graph of G. By Proposition 7.1, we can resolve the conflicts for every connected component of $G_\perp$ independently. Let C denote a connected component of $G_\perp$. Clearly, all vertices of C have the same degree in G. If all vertices have degree two, then one can resolve all conflicts between vertices in C by the computation of a maximum-weight matching [27].

Hence, in the following we focus on the case that $\deg_G(c) = d \geq 3$ for all $c \in C$. We use the following notation. Let $G' := G[C \cup N_G(C)]$ and let $X_1, \ldots, X_\ell$ denote the critical independent sets of G' formed by the vertices in C. Assume without loss of generality that $|X_1| \leq \ldots \leq |X_\ell|$. We distinguish the cases that either the first or the second statement of Lemma 7.9 is true. In both cases we show how to construct a regular minimum-cardinality solution for G'.

Case 1: The first statement of Lemma 7.9 holds**.** That is, there is a set Y of t-vertices such that for any two vertices $c, c' \in C$ with $N_G(c) \neq N_G(c')$, we have $Y(c, c') = Y$. Note that $|Y| = d - 1 > 1$ and $|N(c) \setminus Y| = 1$ for all $c \in C$. The structure of G' is depicted in Figure 7.8.

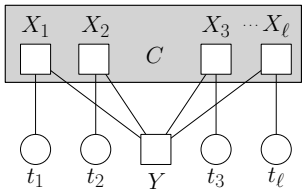


Figure 7.8: Structure of $G' := G[C \cup N(C)]$ if the second statement of Lemma 7.9 holds. The rectangular vertices represent critical independent sets of G', the circular vertices represent single vertices, and an edge represents all edges between the (sets) of vertices. Clearly, it is optimal to delete the edges between X_i and t_i for all but one X_i for which $|X_i|$ is maximal to destroy all M-graphs in G'.

In order to resolve the conflicts between the vertices in C delete the edges $S' := \bigcup_{i=1}^{\ell-1} E_{X_i, N(X_i)\setminus Y}$. Note that $|S'| = \sum_{i=1}^{\ell-1} |X_i|$ since $|N(X_i) \setminus Y| = 1$. Moreover, since $N_{G' \Delta S'}(X_i) = Y$ for all $1 \leq i \leq \ell - 1$ and $Y \subset N_{G' \Delta S'}(X_\ell)$, all M-graphs in $G' := G[C \cup N(C)]$ are destroyed by S'. Finally, we argue that S' is a regular minimum-cardinality solution for G'. By construction, for each X_i any two vertices of X_i have an identical neighborhood in $G' \Delta S'$. In this sense, S' applies the "same" edge modifications to all vertices in X_i and, hence, is regular. Moreover, since two vertices from different X_i's are in conflict, we have to modify the neighborhood structure of all but one X_i. Hence, for any affected X_i, one has to spend at least $|X_i|$ edge modifications (recall that each X_i is a critical independent set and, hence, there is an optimal solution that applies the "same" edge modifications to all vertices in X_i, see Lemma 3.5). Thus, $\sum_{i=1}^{\ell-1} |X_i| = |S'|$ is a lower bound for the solution size since $|X_\ell| \geq |X_i|$ for all $1 \leq i \leq \ell - 1$.

Case 2: The second statement of Lemma 7.9 holds**.** That is, $|N(C) \setminus N(c)| = 1$ for all $c \in C$. To resolve the conflicts between the vertices in C, add the edges $S' := \bigcup_{i=1}^{\ell-1} E_{X_i, N(C)\setminus N(X_i)}$. Note that $|S'| = \sum_{i=1}^{\ell-1} |X_i|$ since $|N(C) \setminus N(X_i)| = 1$. Since $N_{G' \Delta S'}(X_i) = N_G(C)$ for all $1 \leq i \leq \ell - 1$ and $N_{G' \Delta S'}(X_\ell) \subset N_G(C)$, all conflicts within C are resolved. Analogously to Case 1, one can show that S' is a regular minimum solution for G': Since two vertices from different X_i's are in conflict, we have to modify the neighborhood structure of all but one X_i (requiring at least $|X_i|$ edge modifications). Thus, $\sum_{i=1}^{\ell-1} |X_i|$ is a lower bound for the solution size.

For the running time note the following. The conflict graph $G_\perp$ can be built in $O(|V_c|^2 \cdot |V_t|)$ time. Moreover, when building $G_\perp$ one can simultaneously check whether the input graph is conflict regular.

Clearly, $G_\perp$ has $|V_c|$ vertices and at most $|V_c|^2$ edges. Hence, the connected components of $G_\perp$ can be found in $O(|V_c|^2)$ time. Let $C_1, \ldots, C_\ell$ denote the connected components of $G_\perp$. Clearly, for each C_i all changes can be performed in $O(|C_i| \cdot |V_t|)$ time. Hence, the total running time is bounded by $O(|V_c|^2 \cdot |V_t|)$. □

Combining Lemma 7.7 and Theorem 7.2 we arrive at the main result of this section.

Theorem 7.3. Minimum Flip Consensus Tree *can be solved in* $O(3.68^k \cdot |V_c|^2 |V_t|)$ *time.*

Proof. To solve MFCT, apply the above branching rules as long as possible. Note that, in $O(|V_c|^2 \cdot |V_t|)$ time, one can first check whether one of the branching rules

applies and second apply a respective branching rule (if possible). Moreover, if none of the branching rules applies, then the instance is conflict regular and, hence, can be solved in $O(|V_c|^2 \cdot |V_t|)$ time according to Theorem 7.2. Hence, for each search-tree node, all changes can be applied in $O(|V_c|^2 \cdot |E|)$ time. According to Lemma 7.7 this leads to a search-tree algorithm with running time $O(3.68^k \cdot |V_c|^2|V_t|)$. □

Applying the technique of interleaving (see Section 2.5) to our kernelization and the search tree algorithm, we obtain an "additive FPT" algorithm for Minimum Flip Consensus Tree.

Corollary 7.1. Minimum Flip Consensus Tree *can be solved in* $O(3.68^k + |V_c|^2 \cdot |V_t| \cdot |E|)$ *time.*

7.7 Conclusion

In this chapter, we have presented polynomial-time data reduction rules to obtain an $O(k^3)$-vertex kernel for Minimum Flip Consensus Tree. Moreover, we have presented a refined search tree algorithm that runs in $O(3.68^k \cdot |V_c|^2|V_t|)$ time.

Still, there are numerous tasks for future research. First of all, a natural next step is the implementation of our algorithms to see whether our theoretical improvements lead to faster running times in practice. Improving the polynomial running time of our data reduction rules is desirable. Obviously, obtaining data reduction rules that lead to a quadratic-vertex or linear-vertex kernel remains as an open question. Moreover, studying edge-weighted problem variants would be theoretically interesting.

Furthermore, can our data reduction rules be adapted to yield a full kernel (see [52]) for Minimum Flip Consensus Tree? Recall that in the case of full kernels the goal is to "preserve" all inclusion-minimal solutions of size at most k. The correctness proofs of some of our data reduction rules rely on the notion of regular solutions (that is, solutions preserving critical independent sets). However, it is not hard to see that an inclusion-minimal solution is not necessarily regular.

In the matrix representation (mentioned in the introduction) Minimum Flip Consensus Tree is the problem to destroy all "induced" occurrences of the 3×2-submatrix with rows 01, 10, and 11 by a minimum number of bit flips. The problem to destroy all "induced" occurrences of the 4×2-submatrix with rows 00, 01, 10, and 11 has been considered by Damaschke [52] in the context of enumeration all inclusion-minimal solutions of size at most k. Matrices that do not contain this 4×2-submatrix admit a *undirected* prefect phylogeny. This problem is a special case of the Perfect Phylogeny by Recoloring problem for binary character states [84, Definition 4.5]. Can our data reduction rules be generalized to yield a polynomial-size problem kernel for this problem?

Recall that Minimum Flip Consensus Tree is the problem to destroy—by a minimum number of edge modifications—all induced paths on five vertices (so-called P_5's) with the first vertex from V_t. For general graphs, Guillemot et al. [87] have recently shown that the problem to destroy all P_4's, called P_4-free Editing (also known as Cograph Editing), admits a cubic-vertex kernel whereas P_{13}-free Editing does not admit a polynomial-size problem kernel unless $PH = \Sigma_3$. To the best of our knowledge it is open whether P_5-free Editing in general graphs and P_6-free Editing in bipartite graphs admit polynomial-size problem kernels.

Recall that in the case of Minimum Flip Consensus Tree all input trees (all on the same taxa) are represented in a bipartite graph. In the case of supertree construction usually not all trees are on the same set of taxa, leading to "undecided edges" in the bipartite graph [45]. Then the problem Flip Supertree asks for an M-free graph with minimum edit distance to the bipartite graph, where inserting or deleting an "undecided edge" does not contribute to the edit distance. Recently, Böcker et al. [26] have shown that Flip Supertree is $W[1]$-hard with respect to the number of editing operations [26]. This stands in sharp contrast to Cluster Editing with "undecided edges" (also called Fuzzy Cluster Editing) which has recently been shown to be fixed-parameter tractable based on its close relationship to Multicut [132]. The $W[1]$-hardness of Flip Supertree motivates to invesitgate the influence of other (additional) parameters on its parameterized complexity.

For Minimum Flip Consensus Tree and Flip Supertree there are a lot of natural parameters for which the parameterized complexity seems unexplored so far. For example, it would be interesting to explore whether ideas from Bodlaender et al. [31] for Fuzzy Cluster Editing can be transferred to Flip Supertree, showing fixed-parameter tractability with respect to the combined parameter (k, r), where k denotes the "number of editing operations" and r denotes the "number of vertices needed to cover all undecided edges". The hope is that r is much smaller than the total number of undecided edges. Another practically relevant question is for example how the number of input trees influences the computational complexity of Minimum Flip Consensus Tree and Flip Supertree. This parameter seems so far unexplored. For Flip Supertree it would be interesting whether there is a reasonable notion of "overlap between the input trees" leading to fixed-parameter tractability. Moreover, does "high similarity" between the input trees helps to cope with the computational hardness of Minimum Flip Consensus Tree and Flip Supertree? Note that there are many distances that can be used to compare two trees (see for example [147, 11]). Betzler et al. [19] have recently shown that, for several consensus problems, if the the input structures are similar in average, then this can algorithmically be exploited. Clearly, there are other natural parameters such as the number of taxa or characters that deserve attention. Finally, the investigation of combinations of the mentioned parameters seems promising. In summary, considering the large amount of unexplored parameters, it certainly is a fruitful research direction to start a systematic multivariate complexity analysis (cf. [138] and Section 2.2.1) for Minimum Flip Consensus Tree and Flip Supertree.

Part III

Constrained Search Problems

This part is concerned with two constrained search problems that arise in molecular biology. In contrast to the previous part, the considered problems are not directly related and hence, instead of providing an introductory chapter of this part, the problems will be introduced in the two respective chapters. In the following, we briefly summarize the results.

Chapter 8. The NP-hard Interval Constrained Coloring (ICC) problem appears in the interpretation of experimental data in biochemistry dealing with protein fragments. We systematically perform the approach of "deconstructing intractability" to identify meaningful parameters. To this end, we thoroughly analyze a known NP-hardness proof for ICC and identify numerous parameters that naturally occur in ICC. Accordingly, we present several fixed-parameter tractability results exploiting various parameterizations. We substantiate the usefulness of this "multivariate algorithmics approach" by presenting experimental results with real-world data. Chapter 8 is based on [120].

Chapter 9. The task of the NP-hard Haplotype Inference by Parsimony (HIP) problem is to find a minimum-cardinality set of haplotypes that explain a given set of genotypes. We also consider a constrained version of parsimony haplotyping, where the explaining haplotype strings must be chosen from a given pool of plausible haplotype strings. The main motivation for parsimony haplotyping is that it is easier and cheaper to obtain the genotype information of an organism, though the haplotype information is of greater use. We propose improved and partially simplified fixed-parameter tractability results with respect to the parameter "size of the target haplotype set" k by presenting a $k^{4k}\,\mathrm{poly}(n,m)$-time algorithm. The algorithm also applies to the constrained case. The previous algorithms had exponential running time factors k^{k^2+k} for HIP and $k^{O(k^2)}$ for the constrained case. Chapter 9 is based on [76].

Chapter 8

Interval Constrained Coloring

8.1 Introduction

Althaus et al. [8, 9] identified Interval Constrained Coloring as an important combinatorial problem in the context of automated mass spectrometry and the determination of the 3-dimensional structure of proteins. It builds the key to replace a manual interpretation of exchange data for peptic fragments with computer-assisted methods, see Althaus et al. [8] for more on the biochemical background and further motivation. The NP-complete decision problem Interval Constrained Coloring (ICC) deals with matching color multisets with integer intervals and can be formalized as follows.[1] To this end, for two positive integers i, j with $i \leq j$, let $[i, j] := \{k \in \mathbb{N} \mid i \leq k \leq j\}$. In addition, for $i \geq 1$ let $[i]$ denote the interval $[1, i]$.

Definition 8.1. Interval Constrained Coloring (ICC)
Input: An integer $n \geq 1$, a multiset of m integer intervals $\mathcal{F} = \{F_1, \ldots, F_m\}$, all within $[n]$, and a multiset of m multisets of colors $\mathcal{C} = \{C_1, \ldots, C_m\}$ over k different colors.
Question: Is there a coloring $c : [n] \to [k]$ such that for each interval $F_i \in \mathcal{F}$ it holds that $C_i = c(F_i)$?

Herein, $c(F_i)$ denotes the *multiset* of colors assigned by c to the integer points in the interval F_i. Throughout this chapter, we assume that the input intervals cover $[n]$, since otherwise the input instance can be decomposed into independent subinstances. Moreover, we say that a coloring $c : [n] \to [k]$ *satisfies* an input interval F_i if $C_i = c(F_i)$. Finally, a coloring satisfying all input intervals is called *proper*.

From a biochemical point of view, the intervals correspond to (typically overlapping) fragments of a protein with n residues, and the k colors correspond to k different exchange rates that need to be assigned consistently to the n residues [8, 9]. The color multisets correspond to experimentally found bulk information that needs to be matched with the residues and can be interpreted as constraints that describe a set of valid colorings of the interval $[n]$. Note that, from an applied point of view, if not

[1] Compared with Althaus et al. [8, 9] we choose a somewhat different but equivalent formalization here; this problem definition turns out to be more suitable for our subsequent studies.

all constraints (that is, intervals that completely match with a given color multiset) can be fulfilled, then it is also important to investigate the corresponding optimization problems where one wants to maximize the number of fulfilled constraints [9] or to minimize a specific error function [8]. However, we mainly focus on analyzing the complexity of the decision problem. In the case of yes-instances, most of our algorithms can be easily adapted to provide a corresponding coloring.

Known Results. The algorithmic study of ICC has been initiated by Althaus et al. [8, 9]. ICC has been shown to be NP-complete by a reduction from the Exact Cover problem [9]. In a more applied paper [8], besides first introducing and formalizing the problem, an algorithm based on integer linear programming and branch-and-bound was presented that enumerates all valid (fulfilling all constraints) color mappings c. Moreover, it was shown that in the case of $k = 2$ colors a direct combinatorial algorithm leads to polynomial-time solvability. The computational complexity of the case $k = 3$ was left open by Althaus et al. [8]. Byrka et al. [35] filled this gap by showing the NP-completeness of ICC for $k = 3$. The corresponding reduction is from 3-Satisfiability. Moreover, concerning optimization, in a similar way they also showed that the "gap version" of this restricted case is NP-hard, also implying its APX-hardness. Successful experiments with real-world instances with $n < 60$, $m \leq 50$, $k = 3$ and randomly generated instances with $n \leq 1000$, $m = n/2$, and $k = 3$ have been performed [8]. In a more theoretical paper [9], besides the NP-completeness proof, the preceding work [8] has been continued by providing results concerning polynomial-time approximability. In particular, there is an algorithm producing a coloring where all requirements are matched within an additive error of one if the LP-relaxation of the presented integer program for ICC has a feasible solution. This algorithm is based on a sophisticated polyhedral approach combined with recent randomized rounding techniques. Furthermore, Althaus et al. [9] introduced a weighted version of ICC, called Max Coloring, where each interval is associated with a nonnegative weight and the goal is to find a coloring that maximizes the total weight of the satisfied intervals. For Max Coloring they showed that, if one is allowed to relax the coloring requirements by a factor of $(1 + \epsilon)$, then a coloring "satisfying" the optimal number of intervals can be found in $n^{O(\frac{k^2}{\epsilon} \log n \log m)}$ time. Moreover, Canzar et al. [38] provided a new method (using linear programming and backtracking) for enumerating all exact and further approximate solutions with polynomial delay between two successive outputs and using polynomial space. They confirmed the practical use of their approach by experiments. Finally, for a constant number of colors, Canzar et al. [39] presented a factor-$O(\sqrt{|\mathrm{Opt}|})$ polynomial-time approximation algorithm for Max Coloring, where Opt denotes a minimum-cardinality maximum-weight solution. Moreover, they showed that this optimization version is APX-hard even for two colors.

Our Contributions. In this chapter, we propose a fresh view on ICC and the development of exact algorithms for NP-hard combinatorial problems in general. The fundamental starting point here is to deconstruct proofs of NP-hardness in order to obtain new insights into the combinatorial structure of problems. The point is to analyze how different parameters occurring in a problem contribute to its computational complexity. This is where parameterized algorithmics comes into play. Indeed, as it turns out, ICC gives a prime example for the continuing evolution of parameterized

algorithmics into multivariate algorithmics (see Section 2.2.1). For ICC, there is a big number of useful parameterizations, all naturally deduced from deconstructing the NP-hardness proof due to Althaus et al. [8]. In this line, for instance, we can show a fixed-parameter tractability result with respect to the parameter "maximum interval length". Whereas, unless P = NP, the problem is not fixed-parameter tractable with respect to the color parameter k alone [35], it is with respect to the combined parameter (n, k); that is, there is an algorithm with time complexity $(k-1)^n \cdot \text{poly}(n, m)$. These algorithms are of practical interest when the corresponding parameter values are sufficiently small. For instance, note that all experiments of Althaus et al. [8] were performed having $k = 3$ and $n \leq 60$ for real-world instances. Indeed, in the already NP-complete case of $k = 3$ we can further improve the running time to $1.89^n \cdot \text{poly}(n, m)$. In this spirit, in Section 8.4 we investigate a number of "single parameterizations", and in Section 8.5 we consider "combined parameterizations". Moreover, whereas ICC is NP-complete for "cutwidth" three [9], we present a combinatorial polynomial-time algorithm for cutwidth two.[2] Tables 8.1 and 8.2 in Sections 8.4 and 8.5 survey the current state of the art and our new results concerning (combinatorial) algorithms that can efficiently solve ICC in case of favorable parameter constellations. Finally, in Section 8.6, we report positive experimental results based on implementations of some of our new algorithms. We conclude with a discussion and some open questions in Section 8.7.

8.2 Parameterization and the Deconstruction of NP-Hardness

In parameterized algorithmics or multivariate algorithmics the identification of meaningful parameters is a fundamental and nontrivial task (also see Section 2.3). A standard question of people unfamiliar with parameterized algorithmics is how to define respectively find "the" parameter for an NP-hard problem. There are the following (partly overlapping) "standard answers" to this question:

1. The standard parameterization classically refers to the size of the solution set of the underlying problem (whenever applicable).

2. A parameter describes a structural property of the input; for instance, the treewidth of a graph or the number of input strings.

3. A parameter may restrict the "dimensionality" of the input; for instance, in the case of problems from computational geometry.

4. Finding useful parameters to some extent is an "art" based on analyzing what typical real-world instances look like.

Perhaps the most natural and constructive answer, however, is to look at the corresponding proof(s) of NP-hardness and what "parameter assumptions" they (do not) make use of. Indeed, this is what we refer to by *deconstructing NP-hardness proofs for parameter identification*. In this chapter, we deconstruct Althaus et al.'s [9]

[2]The cutwidth denotes the size of a maximum-cardinality set of pairwise overlapping intervals.

NP-hardness proof for ICC and gain a rich scenario of combinatorially and practically interesting parameterizations.

Let us now take a closer look at the NP-hardness of ICC. We first have to briefly review the many-one reduction from Exact Cover due to Althaus et al. [9]: The input of Exact Cover is a set $\mathcal{S}$ of subsets of a ground set $U := \{1, 2, \ldots, u\}$ and a positive integer t, and the question is whether there are t subsets from $\mathcal{S}$ such that every element from U is contained in exactly one such subset. Althaus et al.'s polynomial-time many-one reduction (using an approach by Chang et al. [42]) from Exact Cover to ICC works as follows.

1. The number of colors k is set to $s := |\mathcal{S}|$.
2. The interval range n is set to $u \cdot s$.
3. For each element from U, there are three corresponding integer intervals. Indeed, one can speak of three types of intervals, and all intervals of one type can be placed consecutively into one interval $[n]$ without overlap.
   (a) Type 1: Intervals of the form $[(i-1)s+1, is]$ for all $1 \leq i \leq u$.
   (b) Type 2: Intervals of the form $[is-t+1, (i+1)s-t]$ for all $1 \leq i \leq u-1$.
   (c) Type 3: Intervals of the form $[is-t-f_i+1, is-t+1]$ for all $1 \leq i \leq u$, where f_i denotes the number of occurrences of u in the sets of $\mathcal{S}$.
4. Every type-1 and every type-2 interval is assigned the color set $\{1, \ldots, k\}$. A type-3 interval corresponding to $i \in U$ is assigned the color set consisting of the colors associated with the subsets in $\mathcal{S}$ that contain i.

We remark that this proof of NP-hardness actually works with just using sets instead of multisets in the constructed ICC instance. After having described the construction employed in the NP-hardness proof, the deconstruction begins by making several observations about its properties:

1. The interval range n and the number m of intervals both are unbounded.
2. The number of colors k is s, hence unbounded, but all color multisets indeed are sets. That is, no interval shall be assigned the same color twice.
3. The maximum interval length is s, hence unbounded.
4. The maximum overlap between intervals is $\max\{t, s-t\}$, hence unbounded.
5. Recall that there are three types of intervals and that any two intervals of the same type are disjoint. Hence for each $i \in [n]$ there are at most three intervals containing i. Thus, the *cutwidth*[3] of the constructed instance is bounded by three.

From the last observation we can conclude that there is no hope for fixed-parameter tractability with respect to the parameter "cutwidth" unless P=NP. Referring to the second observation, the same holds true for the parameter k (number of colors) because Byrka et al. [35] showed NP-hardness even for the case $k = 3$. On the positive side,

[3]Formally, the cutwidth is defined as $\max_{1 \leq i \leq n} |\{F \in \mathcal{F} : i \in F\}|$.

we will show that ICC is polynomial-time solvable for cutwidth two. However, from the other three observations we directly obtain the following questions concerning a parameterized complexity analysis of ICC.

1. Is ICC fixed-parameter tractable with respect to the parameters n (interval range) or m (number of intervals)?
2. Is ICC fixed-parameter tractable with respect to the parameter "maximum interval length"?
3. Is ICC fixed-parameter tractable with respect to the parameter "maximum overlap between intervals"?

The central point underlying the above derived algorithmic questions is that whenever a quantity (that is, parameter) in an NP-hardness proof is unbounded (non-constant), then it is natural to investigate what happens if this quantity is constant or considered to be small compared to the overall input size. Clearly, one way to answer is to provide a different proof of NP-hardness where this quantity is bounded. Indeed, this now has happened with respect to the parameter k in the sense that in the NP-hardness proof due to Althaus et al. [9] k is unbounded whereas in the new NP-hardness proof due to Byrka et al. [35] we have $k = 3$. Otherwise, the main tool in answering such questions is parameterized algorithmics. Indeed, the story goes even further by also combining different parameterizations. More specifically, it is, for instance, natural to ask whether ICC is fixed-parameter tractable when parameterized by *both* cutwidth and the number k of colors (the answer is open), or whether it is fixed-parameter tractable when parameterized by both n and k (the answer is "yes") and what the combinatorial explosion $f(n, k)$ then looks like. In this way, one ends up with an extremely diverse and fruitful ground to develop practically relevant combinatorial algorithms.

In the remainder of this chapter, besides the already defined parameters n (range), m (number of intervals), and k (number of colors), we will consider the following parameters and combinations thereof:

- maximum interval length l;
- cutwidth $\mathrm{cw} := \max_{1 \leq i \leq n} |\{F \in \mathcal{F} : i \in F\}|$;
- maximum pairwise overlap between intervals $o := \max_{1 \leq i < j \leq m} |F_i \cap F_j|$;
- maximum number of different colors Δ in the color multisets.

Note that the NP-hardness result of Byrka et al. [35] for $k = 3$ also implies the NP-hardness for $\Delta = 3$ but $\Delta = 2$ is yet unclassified. In addition, one of the integer linear programs devised by Althaus et al. [8] has $O(m \cdot k)$ variables. Using Lenstra's famous result [128] on the running time of integer linear programs with a fixed number of variables then implies that ICC is fixed-parameter tractable with respect to the (combined) parameter (m, k). However, even after several improvements (for example by Frank and Tardos [79]), the combinatorial explosions in Lenstra's theorem remains huge. This fixed-parameter tractability result is thus of purely theoretical interest and more efficient combinatorial algorithms are desirable (see [85, 74, 75] for similar classification results using integer linear programs).

In what follows, we present several fixed-parameter tractability results with respect to the above parameters (Section 8.4) and some combinations of them (Section 8.5).

8.3 A Simple Normal Form Observation

Here, we observe that there is a "normal form" that one may assume without loss of generality for all ICC input instances. More specifically, based on simple and efficient preprocessing rules, one can perform a data reduction that yields this normal form.

Proposition 8.1. (Normal form for ICC)
In $O(lmn)$ time, one can transform every ICC *instance into an equivalent one such that*

1. *at every position $i \in [n]$, there is at most one interval starting at i and at most one interval ending at i, and*
2. *if the maximum interval length is l, then every position $i \in [n]$ is contained in at most l intervals.*

Proof. To achieve the claimed normal form, exhaustively perform the following two preprocessing rules.

1. If there are two intervals $F_i = [s_i, t_i]$ and $F_j = [s_j, t_j]$ with $s_i = s_j$, $t_i = t_j$, and $C_i \neq C_j$, then return "No". Otherwise, remove F_i and C_i.
2. If there are two intervals $F_i = [s_i, t_i]$ and $F_j = [s_j, t_j]$ with $s_i = s_j$ and $t_i < t_j$, then set $F_j := [t_i + 1, t_j]$ and $C_j := C_j \setminus C_i$.[4] If $|C_j| \neq |F_j|$, then return "No". The case $s_i < s_j$ and $t_i = t_j$ is handled analogously.

Obviously, the two rules directly imply normal form property 1, which again immediately implies normal form property 2. For the correctness of the first rule, observe that no coloring can simultaneously satisfy F_i and F_j. For the correctness of the second rule note that a proper coloring for the original instance is a proper coloring of the instance that results by one application of the rule, and vice versa. Hence, if the new instance obviously is a no-instance (i.e. $|C_j| \neq |F_j|$), then it is correct to reject the instance.

Next, we give an analysis of the running time. We use the following strategy. Keep an array A that holds for every position $i \in [n]$ a list with the intervals starting at i (and another list with the intervals ending at i). This array is initialized before the application of the rules in $O(nm)$ time. To decide whether a rule can be applied, iterate over the array to find a position at which two intervals start (or end). For two intervals F_i and F_j, the necessary changes can be performed in $O(n)$ time (if one implements the color multisets by an array of size $k \leq n$). Also note that we can update array A within this time bound. Hence, one application of a rule and the update of array A take $O(n)$ time. Finally, note that for every interval the first rule can be applied at most once, and the second rule at most l times. Hence, the rules can be exhaustively applied in $O(nm) + O(lmn) = O(lmn)$ time. □

Clearly, Proposition 8.1, property 1, implies that after preprocessing the "reduced equivalent instance" contains at most n intervals and n multicolor sets, which can be interpreted as kernelization with respect to the parameter n.

[4] The setminus operation here has to be adapted to multisets, that is, for example, $\{a, a, b\} \setminus \{a, b\} = \{a\}$.

Table 8.1: Complexity of ICC for one-dimensional parameterizations. Herein, "P" means that the problem is polynomial-time solvable, "NPc" means that the problem is NP-complete, and "?" means that the complexity is unknown. For fixed-parameter algorithms, we only give the function of the exponential term, omitting polynomial factors. The results for $k = 2$ and $\mathrm{cw} = 3$ are due to Althaus et al. [9, 8], the results for $k \geq 3$ and $\Delta \geq 3$ are due to Byrka et al. [35], the rest is new.

Parameter	k	Δ	l	cw	m	n	o
Complexity	$k = 2$: P $k \geq 3$: NPc	$\Delta = 2$: ? $\Delta \geq 3$: NPc	$l!$	$\mathrm{cw} = 2$: P $\mathrm{cw} = 3$: NPc	?	$n!$	$o = 1$: P $o \geq 2$: ?

8.4 Single Parameters

In Section 8.2, we identified various parameters as meaningful "combinatorial handles" to better assess the computational complexity of ICC. Whereas ICC is NP-complete for cutwidth $\mathrm{cw} = 3$ [9], we will show that it is polynomial-time solvable for $\mathrm{cw} = 2$. Obviously, the maximum length l fulfills $l \leq n$, so the fixed-parameter tractability with respect to l (as we will prove subsequently) implies the fixed-parameter tractability with respect to n. Table 8.1 surveys known and new results with respect to single parameters.

Parameter Maximum Interval Length l. Our first algorithm exploits the parameter "maximum interval length l". The rough idea is that the coloring at position i does not affect any intervals that overlap with position $i + l$. This leads to a dynamic programming algorithm that keeps track of all possible colorings of the "last" interval (which has at most l positions).

Theorem 8.1. ICC *can be solved in* $O(l! \cdot l \log l \cdot mn)$ *time.*

Proof. We present a dynamic programming algorithm. To this end, we use the following notation. Let $\mathcal{K} = \{1, \ldots, k\}$ denote the set of all colors. For an interval $[s, t]$, a coloring c is represented by a tuple $(c_1, \ldots, c_{t-s+1}) \in \mathcal{K}^{t-s+1}$, meaning that $c(s) = c_1$, $c(s + 1) = c_2$, and so on. We say that a coloring c' *satisfies* an input interval $F_i \in \mathcal{F}$ if $c'(F_i) = C_i$. For an input interval $F_i \in \mathcal{F}$, the set K_i of all satisfying colorings is given by

$$K_i := \{c' \in \mathcal{K}^{|F_i|} \mid c' \text{ satisfies } F_i\}.$$

Note that there are at most $|C_i|!$ satisfying colorings of an input interval F_i (the worst case arises when every color occurs at most once in the multiset C_i since then every permutation of the colors in C_i represents a satisfying coloring). Finally, let A denote the set of intervals completely contained in some other intervals, that is,

$$A := \{F \in \mathcal{F} \mid \exists_{F' \in \mathcal{F}} : F \subseteq F'\},$$

and $B := \mathcal{F} \setminus A$. We assume that the intervals in B are ordered in increasing order of their start points (and, hence, also in increasing order of their endpoints). Let $B = \{B_1, \ldots, B_{m'}\}$ and $B_j = [s_j, t_j]$ for all $1 \leq j \leq m'$. Note that $m' \leq n$ and that the

intervals in B cover $[n]$, that is, $\bigcup_{j=1}^{m'} B_j = [n]$ (as discussed in Section 8.1 we assume that the input intervals cover $[n]$).

Now, we are ready to describe the algorithm. The algorithm traverses the B_j's in increasing order of j, $1 \leq j \leq m'$. For every B_j, the algorithm maintains a table T_j with an entry for every satisfying coloring c of B_j. Informally speaking, this entry indicates whether there exists a coloring of the interval $[1, t_j]$ that agrees with c in $[s_j, t_j]$ and satisfies all intervals seen so far. More specifically, the goal of the dynamic programming procedure is to fill these tables to match the following definition. For every coloring $c' = (c'_1, \ldots, c'_{|B_j|}) \in K_j$, we have $T_j(c') = \text{true}$ if and only if there exists a coloring $c'' = (c''_1, \ldots, c''_{t_j}) \in \mathcal{K}^{t_j}$ of the interval $[t_j]$ with $(c''_{s_j}, \ldots, c''_{t_j}) = c'$ such that c'' satisfies each interval $F \in \mathcal{F}$ with $F \subseteq [t_j]$. Obviously, if the algorithm correctly computes the tables according to this definition, then $T_{m'}$ contains a true entry if and only if the input is a yes-instance.

Table T_1 is computed as follows. For every $c' \in K_1$, set $T_1(c') :=$ true if and only if c' satisfies every interval $[s, t] \in A$ with $[s, t] \subseteq [s_1, t_1]$.

For $j \geq 2$, table T_j is computed based on T_{j-1}, as described in the following. We say that a coloring $c' = (c'_1, \ldots, c'_{|B_j|}) \in K_j$ for B_j is consistent with a coloring $c'' = (c''_1, \ldots, c''_{|B_{j-1}|}) \in K_{j-1}$ for B_{j-1} if c' and c'' agree in $B_{j-1} \cap B_j$, that is, $(c''_{s_j - s_{j-1}+1}, \ldots, c''_{|B_{j-1}|}) = (c'_1, \ldots, c_{t_{j-1}-s_j+1})$. We write $c'|c''$ to denote that c' is consistent with c''. To compute the entries of T_j, proceed as follows. For j from 2 to m' and for every $c' = (c'_1, \ldots, c'_{|B_j|}) \in K_j$, set

$$T_j(c') = \text{true} \iff c' \text{ satisfies all } F \in A \text{ with } F \subseteq B_j \text{ and } \exists c'' \in K_{j-1},\ c'|c'' : T_{j-1}(c'') = \text{true}. \tag{8.1}$$

Finally, the algorithm returns "Yes" if T'_m contains a true entry and "No", otherwise.

For the correctness of the algorithm, we show by induction that for every j, $1 \leq j \leq m'$, table T_j meets above definition, that is, table entry $T_j(c')$ is true if and only there exists a coloring of $[t_j]$ with "suffix" c' satisfying all intervals $F \in \mathcal{F}$ with $F \subseteq [t_j]$. Clearly, T_1 is computed in accordance with this definition, yielding the induction base.

For the induction step, we show that Recursion (8.1) computes T_j according to the above definition assuming that T_{j-1} has been correctly computed (induction hypothesis). That is, we show that there is a coloring of $[t_j]$ with suffix c' satisfying all intervals $F \in \mathcal{F}$ with $F \subseteq [t_j]$ if and only if c' satisfies all $F \in A$ with $F \subseteq B_j$ and there is a $c'' \in K_{j-1}$ with $c'|c''$ such that $T_{j-1}(c'') = \text{true}$. The "$\Rightarrow$"-direction is obvious. For the "$\Leftarrow$"-direction, observe the following. The algorithm combines a coloring of $[t_{j-1}]$ satisfying all $F \in \mathcal{F}$ with $F \subseteq [t_{j-1}]$ with a coloring c' of $[s_j, t_j]$ that is consistent with c'' and satisfies all $F \in \mathcal{F}$ with $F \subseteq [s_j, t_j]$. This yields a coloring for $[t_j]$ that satisfying all $F \in \mathcal{F}$ with $F \subseteq [t_j]$; all $F \in \mathcal{F}$ with $F \subseteq [t_{j-1}]$ are clearly also satisfied by this coloring. Moreover, all other $F \in \mathcal{F}$ with $F \subseteq [t_j]$ are satisfied since for every input interval $[s, t] \in \mathcal{F}$ with $t_{j-1} < t \leq t_j$ it holds that $[s, t] \subseteq [s_j, t_j]$.

As to the running time, there are at most $|B_j|!$ satisfying colorings of B_j; at most one for every permutation of the associated color multiset. Hence, one has to consider at most $l!$ colorings for every B_j. For every $j = 1, \ldots, m' - 1$, the algorithm proceeds as follows.

When building table T_j, the algorithm computes an auxiliary table Q_j containing one entry for all $c' \in K_j$ with the same length-$(t_j - s_{j+1} + 1)$ suffix, indicating

whether T_j contains a true entry for one of these colorings. Table Q_j can for example be realized by a dictionary for which the addition and the lookup of a key requires $O(\log(s))$ comparisons, where s is the size of the dictionary. Then, to check whether $\exists c'' \in K_{j-1}$, $c'|c'' : T_{j-1}(c'') = \text{true}$ for a $c' = (c'_1, \ldots, c'_{|B_j|}) \in K_j$, the algorithm can check whether $Q_{j-1}(c'_1, \ldots, c'_{t_j - s_{j-1}+1}) = \text{true}$ in $O(l \log l)$ time (note that the size of Q_j does not exceed $l!$). Hence, for every position $1 \leq j \leq m'$, it needs at most $O(l! \cdot (l \log l + lm))$ time, where the factor lm is due to checking whether c' satisfies all $F \in \mathcal{F}$ with $F \subseteq [s_j, t_j]$. In summary, since $m' \leq n$ the total running time is $O(l! \cdot l \log lmn)$. □

Parameter Cutwidth cw. Here, we show that ICC is solvable in $O(n^2)$ time for cutwidth $\text{cw} = 2$. This contrasts the case $\text{cw} = 3$ shown to be NP-complete [9]. Our algorithm is based on four data reduction rules that are executable in polynomial time. The application of these rules either leads to a much simplified instance that can be colored without violating any interval constraints or shows that the instance is a no-instance.

In the following, we say that a reduction rule is *correct* if the instance after applying this rule has a proper coloring if and only if the original instance has a proper coloring. Some of the subsequent data reduction rules are based on identifying positions for which we can decide which color they will have in a proper coloring. In this context, we use the following notation. If we can decide that a position i is colored by color c_x in a proper coloring, we write $c(i) = c_x$, meaning that we simplify the instance as follows; for all $F_j = [s, t]$ with $s \leq i \leq t$, we set $C_j := C_j \setminus \{c_x\}$ and $t := t - 1$. For all $F_j = [s, t]$ with $i < s$, we set $s := s - 1$ and $t := t - 1$. "Empty" intervals F_j with $C_j = \emptyset$ are removed from the input. Finally, we call an instance reduced with respect to one or more data reduction rules if none of these rules applies.

We use the following notation. For a color multiset C and a color c_x let $\text{occ}(c_x, C)$ denote the multiplicity of c_x in C. For two color multisets A and B the *intersection* of A and B, denoted by $A \cap B$, is the multiset that contains each color c with $\text{occ}(c, A) > 0$ and $\text{occ}(c, B) > 0$ exactly $\min(\text{occ}(c, A), \text{occ}(c, B))$ times. For example, $\{a, b, b, b, c\} \cap \{a, a, b, b\} = \{a, b, b\}$. We start with a basically straightforward data reduction rule.

Reduction Rule 8.1. For any two intervals F_i and F_j and their corresponding color multisets C_i and C_j,

- if $|F_i \cap F_j| = |C_i \cap C_j|$, then set $c(F_i \cap F_j) = C_i \cap C_j$;
- if $|F_i \cap F_j| > |C_i \cap C_j|$, then return "No".

Reduction Rule 8.1 is obviously correct: if two intervals "share" more positions than color elements, then there is no coloring that satisfies both intervals, and if the number of shared positions is equal to the number of shared color elements, then one has to color the overlapping intervals exactly with the corresponding colors. Moreover, since the cutwidth is two there is no further interval containing a position in $F_i \cap F_j$. Hence, we can color the positions of $F_i \cap F_j$ in arbitrary order with the colors in $C_i \cap C_j$.

After exhaustive application of Reduction Rule 8.1 we can assume that no interval is completely contained in any other interval.

In the following, assume that the intervals are ordered with respect to their startpoints, that is, for $F_i = [s_i, t_i]$ and $F_j = [s_j, t_j]$ with $i < j$ we have $s_i < s_j$. Consider a

position i, $1 \leq i < n$. If there is no input interval $[s_j, t_j]$ with $s_j \leq i$ and $t_j > i$, then we can color $[i]$ independently from $[i+1, n]$. Together with Reduction Rule 8.1 and the fact that $\mathrm{cw} = 2$, we can thus assume that all intervals except for F_1 and F_m overlap with exactly two other intervals. Hence, we can partition each interval F_j, $1 < j < m$, into at most three subintervals: the first subinterval overlaps with F_{j-1}, the second (possibly empty) subinterval does not overlap with any other interval, and the third subinterval overlaps with F_{j+1}. The following notation describes this structural property. For an interval F_j, $1 < j < m$, define

$$F_j^1 := F_j \cap F_{j-1},\ F_j^3 := F_j \cap F_{j+1}, \text{ and } F_j^2 := F_j \setminus (F_j^1 \cup F_j^3).$$

For a coloring c' of $[n]$ and j, $1 < j < m$, let $C_j^1 := c'(F_j^1)$. Define C_j^2 and C_j^3 accordingly. For F_1, define $F_1^3 := F_2^1$ and $F_1^2 := F_1 \setminus F_1^3$; for F_m, define $F_m^1 := F_m \cap F_{m-1}$ and $F_m^2 := F_m \setminus F_m^1$; C_1^3, C_1^2, C_m^1, and C_m^2 are defined analogously. Whether a coloring violates an interval F_j only depends on the sets C_j^1, C_j^2, and C_j^3. Hence, when we know that a color c_x must belong to some C_j^l, $1 \leq l \leq 3$, then we can color an arbitrary $i \in F_j^l$ with c_x. Recall that for a color multiset C and a color c_x, $\mathrm{occ}(c_x, C)$ denotes the multiplicity of c_x in C.

The next rule reduces intervals F_j that have no "private" middle interval F_j^2 but more elements of some color c_x than the previous interval.

Reduction Rule 8.2. For any interval F_j, if $F_j^2 = \emptyset$ and there is a color c_x such that $\mathrm{occ}(c_x, C_{j-1}) < \mathrm{occ}(c_x, C_j)$, then for some arbitrary $i \in F_j^3$ set $c(i) = c_x$.

The rule is correct because in a proper coloring at most $\mathrm{occ}(c_x, C_{j-1})$ many positions in F_j^1 (which is the intersection of F_j and F_{j-1}) can be colored with c_x. Hence, in order to satisfy constraint C_j, all other occurrences of c_x must be at positions in F_j^3. Again, since the cutwidth is two, we can choose an arbitrary position of F_j^3. After the exhaustive application of Reduction Rule 8.2, for every interval F_j with $F_j^2 = \emptyset$, we have $C_{j-1} \supseteq C_j$. Next, we reduce triples of intervals F_{j-1}, F_j, F_{j+1} that have identical color multisets in case $F_j^2 = \emptyset$.

Reduction Rule 8.3. For intervals F_{j-1}, F_j, and F_{j+1} such that $C_{j-1} = C_j = C_{j+1}$ and $F_j^2 = \emptyset$, remove F_{j-1} and F_j from the input and for all intervals $F_{j+l} =: [s, t]$ with $l \geq 1$ set $F_{j+l} = [s', t']$, where $s' := s - |F_j|$ and $t' := t - |F_j|$.

Lemma 8.1. *Reduction Rule 8.3 is correct.*

Proof. Let I be an instance to which Reduction Rule 8.3 is applied, and let I' be the resulting instance. We show that I is a yes-instance if and only if I' is a yes-instance. Let I be a yes-instance, let c' be a proper coloring of I, and for each input interval $F_l \in \mathcal{F}$ let C_l^1, C_l^2, C_l^3 be the color multisets according to c'. Since $C_{j-1} = C_j = C_{j+1}$, we have $C_{j-1}^1 \subseteq C_{j+1}^1$ and $C_{j+1}^3 \subseteq C_{j-1}^3$. In I', F_{j+1} overlaps with F_{j-2} and F_{j+2}. Coloring F_{j+1}^1 with the colors of C_{j-1}^1 and F_{j+1}^2 with the colors of $C_j \setminus (C_{j-1}^1 \cup C_{j+1}^3)$ yields a proper coloring for I' since C_{j+1} is not violated and for the other intervals the coloring has not changed. Hence, if I is a yes-instance, then I' is a yes-instance. The other direction can be shown analogously. □

The following is our final data reduction rule.

Reduction Rule 8.4. Let I be an instance that is reduced with respect to Reduction Rules 8.1, 8.2, and 8.3, and let F_j be the first interval of I such that there is a color c_x with $\text{occ}(c_x, C_j) > \text{occ}(c_x, C_{j+1})$. Then,

- if $j = 1$, then set $c(i) = c_x$ for some arbitrary $i \in F_1^2$;
- if $j > 1$ and $c_x \notin C_{j-1}$, then set $c(i) = c_x$ for some arbitrary $i \in F_j^2$ in case $F_j^2 \neq \emptyset$ and otherwise return "No";
- if $j > 1$ and $c_x \in C_{j-1}$, then set $c(i) = c_x$ for some arbitrary $i \in F_j^1$.

Lemma 8.2. *Reduction Rule 8.4 is correct.*

Proof. Let I be an instance that is reduced with respect to Reduction Rules 8.1, 8.2, and 8.3 to which Reduction Rule 8.4 is applied, and let I' be the resulting instance. We show that I is a yes-instance if and only if I' is a yes-instance. We only show that if I is a yes-instance, then I' is a yes-instance, since the other direction trivially holds.

If $j = 1$, this is easy to see: since c_x occurs more often in F_1 than in F_2, one of the positions in $F_1 \setminus F_2$ must be colored with c_x.

If $j > 1$ and $c_x \notin C_{j-1}$, then it is clear that one of the positions in F_j^2 must be colored with c_x. We either perform this forced coloring or return "No" if this is not possible.

Finally, if $j > 1$ and $c_x \in C_{j-1}$, the situation is more complicated. Let c' be a proper coloring of I. If there is a position $i \in F_j^1$ such that $c'(i) = c_x$, then the claim obviously holds. Otherwise, we show that we can transform c' into an alternative coloring c'' that is proper and there is an $i \in F_j^1$ such that $c''(i) = c_x$. Whether coloring c'' is proper can be determined from the multisets C_l^1, C_l^2, and C_l^3, $1 \leq l \leq m$, defined by the coloring function c''. Hence, we describe the transformation applied to c' with respect to these multisets. Note that we do not modify the sets C_l^y, $y \in \{1, 2, 3\}$, for any $l > j$.

We face the following situation: $c_x \notin C_j^1$, but since c' is a coloring that does not violate any interval constraints and by the precondition of Reduction Rule 8.4, $c_x \in C_j^2$. By the choice of j in Reduction Rule 8.4, we have $C_1 \subseteq C_2 \subseteq \ldots \subseteq C_j$. We show that we can always find a series of "exchange operations" such that the resulting coloring is proper and $c_x \in C_j^1$. We perform a case distinction.

Case 1: $F_{j-1}^2 \neq \emptyset$. There are three subcases of this case.

Case 1.1: $c_x \in C_{j-1}^2$. In this case, we exchange c_x with some arbitrary $c_l \in C_j^1$. Furthermore, we remove c_x from C_j^2 and add c_l to C_j^2. The exchange is shown in Figure 8.1a; the resulting coloring is clearly proper and $c_x \in C_j^1$.

Case 1.2: $c_x \in C_{j-1}^1$ and $F_{j-2}^2 \neq \emptyset$. Clearly, C_{j-2} must be involved in the exchange. We choose an arbitrary element $c_l \in C_{j-2}^2$. Since $C_{j-2} \subseteq C_{j-1}$, we also have $c_l \in C_{j-1} \setminus C_{j-1}^1$. We distinguish two further subcases.

Case 1.2.1: $c_l \in C_{j-1}^3$. We perform a *direct* exchange of c_l and c_x between C_{j-2}^2 and C_{j-1}^1 and also between C_j^1 and C_j^2. The exchange is shown in Figure 8.1b; the resulting coloring is clearly proper and $c_x \in C_j^1$.

Case 1.2.2: $c_l \in C_{j-1}^2$. We remove c_l from C_{j-2}^2 and add c_x to C_{j-2}^2. Furthermore, we perform a *circular* exchange between C_{j-1}^1, C_{j-1}^2, and C_{j-1}^3: move c_x from C_{j-1}^1 to C_{j-1}^3, move an arbitrary element c_f from C_{j-1}^3 to C_{j-1}^2, and move c_l from C_{j-1}^2

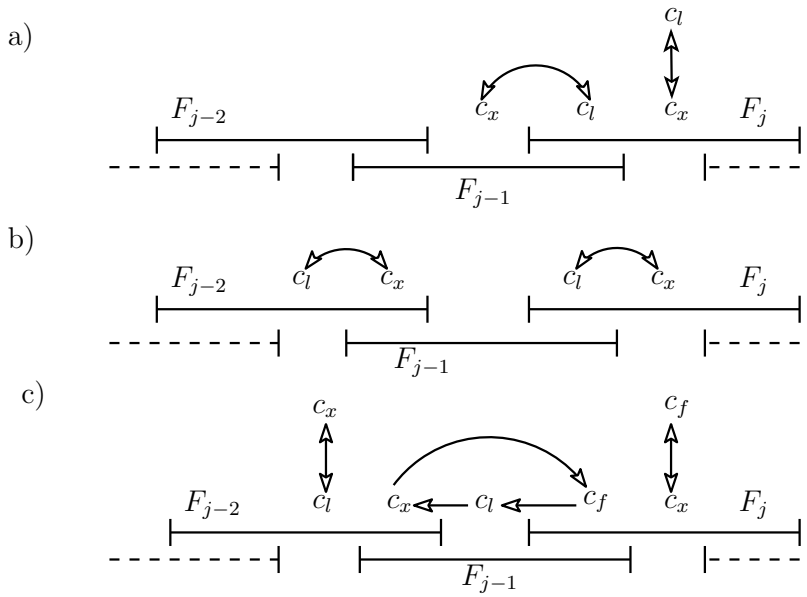


Figure 8.1: Exchange operations used in the proof of Lemma 8.2. Intervals are shown as lines. The segments of an interval F_j where no other interval starts or ends correspond to the color multisets C_j^1, C_j^3, and (if present) C_j^2. A vertical double arrow means removing the element shown at the bottom from the corresponding multiset and adding the element shown at the top of the arrow to the multiset; a double arrow between two multisets means exchanging the elements between the corresponding multisets; a simple arrow means moving an element from one multiset to another in the indicated direction.

to C_{j-1}^1. Finally, we remove c_x from C_j^2 and add c_f to C_j^2. The exchange is shown in Figure 8.1c; the resulting coloring is clearly proper and $c_x \in C_j^1$.

Case 1.3: $c_x \in C_{j-1}^1$ **and** $F_{j-2}^2 = \emptyset$**.** As stated above, we have $C_{j-3} \subseteq C_{j-2}$. Furthermore, since $F_{j-2}^2 = \emptyset$ and Reduction Rule 8.2 does not apply, we have $C_{j-3} = C_{j-2}$. Hence, $F_{j-3}^2 \neq \emptyset$ since otherwise F_{j-3} would have been removed by Reduction Rule 8.3.

Case 1.3.1: $c_x \in C_{j-3}^2$**.** We pick an arbitrary element $c_l \in C_{j-3}^3$ and exchange it with $c_x \in C_{j-3}^2$. If $c_l \in C_{j-1}^3$, then we perform a direct exchange of c_x and c_l between C_{j-1}^1 and C_{j-1}^3. If $c_l \in C_{j-1}^2$, we perform a circular exchange between C_{j-1}^1, C_{j-1}^2, and C_{j-1}^3 using some arbitrary $c_f \in C_{j-1}^3$. Furthermore, we remove c_x from C_j^2 and insert c_l into C_j^2. Figure 8.2a shows the more complicated case where $c_l \in C_{j-1}^2$.

Case 1.3.2: $c_x \in C_{j-3}^1$**.** Clearly, any change must also involve F_{j-4}. Furthermore, we can have a long "chain" of alternating intervals F_{j-2i} and F_{j-2i-1}, $1 \leq i < j/2$, such that $F_{j-2i}^2 = \emptyset$ and $F_{j-2i-1}^2 \neq \emptyset$. Since the instance is reduced with respect to Reduction Rules 8.2 and 8.3 we have $C_{j-2i-1} = C_{j-2i} \subseteq C_{j-2i+1}$. Let F_h be the first (with lowest index) interval of the chain, that is, $F_h^2 \neq \emptyset$, $F_{h+1}^2 = \emptyset$, and either $F_h = F_1$ or $F_{h-1}^2 \neq \emptyset$. There is either some rightmost (with highest index) interval F_g such that $c_x \in C_g^2$, or for all intervals F_i of the chain we have $c_x \notin C_i^2$. We show that in the first case, for all intervals F_{g+2i}, $g + 2i \leq j - 3$, we have $c_x \in C_{g+2i}^1$, and in the

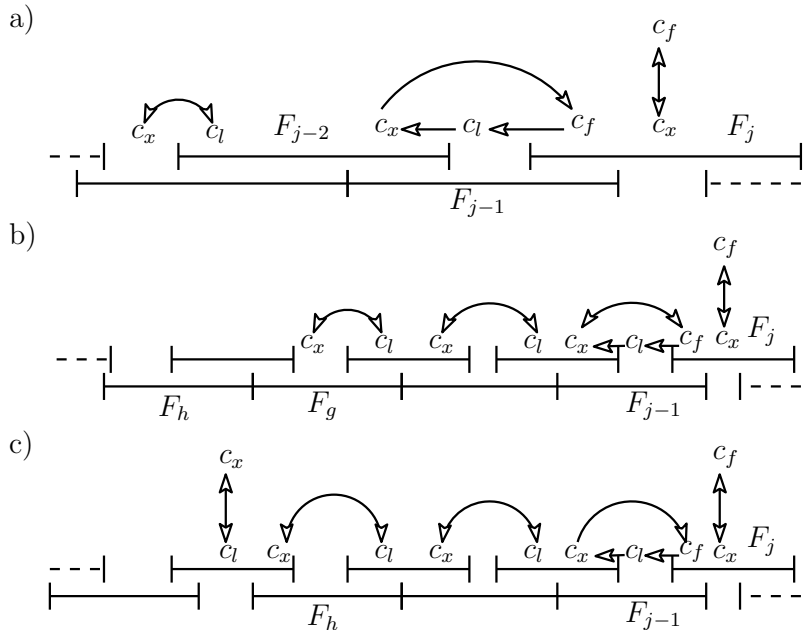


Figure 8.2: Exchange operations used in Case 1.3 of the proof of Lemma 8.2. Intervals are shown as lines. Two intervals $[s,t]$ and $[s',t']$ are drawn adjacent if $t = s'-1$ or vice versa. The segments of an interval F_j where no other interval starts or ends correspond to the color multisets C_j^1, C_j^3, and (if present) C_j^2. A vertical double arrow means removing the element shown at the bottom from the corresponding multiset and adding the element shown at the top of the arrow to the multiset; a double arrow between two multisets means exchanging the elements between the corresponding multisets; a simple arrow means moving an element from one multiset to another in the indicated direction.

second case for all F_{h+2i}, $h+2i \leq j-3$, we have $c_x \in C_{h+2i}^1$. This can be seen as follows. We have $c_x \in C_{j-3}^1$ and thus $c_x \in C_{j-5} \setminus C_{j-4}^1$, since $C_{j-4} = C_{j-5}$. We either have $c_x \in C_{j-5}^2$ (implying $F_g = F_{j-5}$) or $c_x \in C_{j-5}^1$ (in which case we can apply the same arguments showing that either $F_g = F_{j-7}$ or $c_x \in C_{j-7}^1$ and so on). We now sketch the exchange operations that we perform.

First, consider the case that there is some F_g with $c_x \in C_g^2$. We perform an exchange similar to the one shown in Figure 8.2a. That is, we exchange $c_x \in C_g^2$ and some $c_l \in C_g^3$. Then we remove c_x from C_{g+2}^1, and add it to C_{g+2}^3. We also need to add c_l to C_{g+2}^1, which is possible since $c_l \in C_{g+2} \setminus C_{g+2}^1$. Depending on whether $c_l \in C_{g+2}^2$ or $c_l \in C_{g+2}^3$, we perform a circular or a direct exchange. These exchange operations are carried on (for F_{g+2i} for increasing $i \geq 1$) until we have reached F_j, that is, we move c_x from C_{g+2i}^1 to C_{g+2i}^3 and some c_y (depending on the previous exchange) from $C_{g+2i} \setminus C_{g+2i}^1$ to C_{g+2i}^1. An example of this exchange is shown in Figure 8.2b. Note that this also includes the case where $F_h = F_1$.

Second, we consider the case where for all intervals F_i of the chain we have $c_x \notin C_i^2$. We start the exchange operation at the first (with lowest index) interval F_h of the chain,

that is, the first interval F_h of the chain such that $F_h^2 \neq \emptyset$ and $F_{h-1}^2 \neq \emptyset$. Note that, since we have already considered the case $F_1 = F_h$, such an interval must exist. Then we perform the exchange operations as sketched in Figure 8.2c. That is, we remove an arbitrary element c_l from C_{h-1}^2 and add c_x to C_{h-1}^2. Then we perform either a circular or a direct exchange of c_l and c_x in F_h, which is possible since $c_l \in C_h \setminus C_h^1$. We continue with these circular or direct exchanges for F_{h+2i} for increasing $i \geq 1$ until we have reached F_{j-1}. Finally, we remove c_x from C_{j-2} and add c_f to C_{j-2}.

Case 2: $F_{j-1}^2 = \emptyset$**.** As shown in Case 1.3, we can assume that $F_{j-2}^2 \neq \emptyset$ since otherwise Reduction Rule 8.3 would apply. Hence, there is some $c_l \in C_{j-2}^2$. Since $C_{j-2} = C_{j-1}$ and $C_{j-1}^2 = \emptyset$, we also have $c_l \in C_{j-1}^3$. Hence, this case is similar to Case 1.2.1 and we can perform an exchange as shown in Figure 8.1b.

In all cases, we construct an alternative coloring c'' that is proper and there is an $i \in F_j^1$, such that $c''(i) = c_x$. This means that we can assume that if I is a yes-instance, then there is some $i \in F_j^1$ such that $c'(i) = c_x$. In summary, this shows that I is a yes-instance if and only if I' is a yes-instance. □

With these four reduction rules at hand, we can describe a simple quadratic-time algorithm (for constant number k of colors) for ICC with cutwidth two.

Theorem 8.2. ICC *can be solved in* $O(kn^2)$ *time when the input has cutwidth two.*

Proof. The algorithm starts with exhaustively applying Reduction Rules 8.1 to 8.4. Note that before applying Reduction Rule 8.4 we always have to check whether Reduction Rule 8.1, Reduction Rule 8.2 , or Reduction Rule 8.3 can be applied, because it is only correct to apply Reduction Rule 8.4 when the instance is reduced with respect to the other rules. The rules either return "No" or we obtain an instance that is reduced with respect to all reduction rules. In such an instance we have $C_1 \subseteq C_2 \subseteq \ldots \subseteq C_m$. Otherwise, Reduction Rule 8.4 would apply, because there would be some $F_i \in \mathcal{F}$ and a color c_x such that $\mathrm{occ}(c_x, C_i) > \mathrm{occ}(c_x, C_{i+1})$. This instance can be easily colored as follows. For the first interval F_1, we choose an arbitrary coloring that does not violate C_1. Since $C_1 \subseteq C_2$, this coloring also does not violate C_2. Then we remove the colored parts from the input, adjust the color multisets and intervals accordingly, and choose an arbitrary coloring that does not violate C_2. Clearly, this does not violate C_3, since $C_2 \subseteq C_3$. After this, we again reduce the colored parts and continue with coloring F_3. This is repeated until all positions are colored and produces a coloring that does not violate any interval constraints. This proves the correctness of the algorithm.

For the running time of the algorithm consider the following. First, since the input has cutwidth two, the number m of intervals is $O(n)$. For each reduction rule, checking whether it can be applied and the application itself can be performed in $O(kn)$ time. Furthermore, the application of any of the reduction rules removes at least one position from the interval $[n]$. Overall, the rules can thus be applied at most n times. Together with the $O(n)$ steps that are clearly sufficient for coloring any instance that is reduced with respect to the reduction rules, this leads to a total running time of $O(kn^2)$. □

Using the previous algorithm, we also obtain polynomial-time solvability in case the maximum overlap o between intervals is at most one. This follows from the observation that after achieving the normal form of the instance (see Proposition 8.1), each instance with overlap at most one also has cutwidth at most two, which can be seen as follows. Suppose an instance that has the normal form has overlap one and cutwidth at least

Table 8.2: Complexity of ICC for combined parameters. We only give the function of the exponential term, omitting polynomial factors. Herein, $(k, *)$ and $(k, *, *)$ refer to combined parameters that feature k and one or two additional parameters, $(l, *)$ refers to combined parameters that feature l and one additional parameter. The result for parameter (k, m) is due to Althaus et al. [8], the rest is new.

Parameter	Running times
$(k, *)$	k^l, $(k-1)^n$, $f(k,m)$ (ILP)
$(k, *, *)$	$l^{\mathrm{cw}\cdot(k-1)}$, $n^{\mathrm{cw}\cdot(k-1)}$
$(l, *)$	Δ^l, $(\mathrm{cw}+1)^l$

three. Then there must be a position i such that at least three intervals F, F', and F'' overlap at i. By Proposition 8.1, at most one of these three intervals, say F, starts at i. This, however, means that F' and F'' have overlap at least two.

Corollary 8.1. ICC *can be solved in* $O(n^2)$ *time when the input has overlap one.*

8.5 Combined Parameters

In the following, as already indicated in Section 8.2, we turn to the study of some relevant pairs of single parameters which form a "combined parameter". Table 8.2 summarizes our current knowledge about combined parameterizations of ICC—there are many questions left open.

First, we present a dynamic programming strategy for solving ICC in $O(k^l \cdot (k + l)mn)$ time. This algorithm uses similar ideas as the algorithm presented in the proof Theorem 8.1. Note that, for a small number k of colors this algorithm is more efficient than the algorithm presented in the proof of Theorem 8.1.

Theorem 8.3. ICC *can be solved in* $O(k^l \cdot (k + l)mn)$ *time.*

Proof. We present a dynamic programming algorithm. The basic idea of the algorithm is to maintain for every length-l subinterval of $[n]$ a table with an entry for every possible coloring of that interval indicating whether this coloring can be extended to a proper coloring of $[n]$. These tables are built by a "left to right" dynamic programming procedure. The details follow.

For every $1 \leq i \leq n-l+1$, the algorithm maintains a table T_i with an entry for every possible coloring of the interval $I_i := [i, i + l - 1]$. Note that there are k^l possibilities to color a size-l interval with k colors. In the following, let $\mathcal{K} := \{1, 2, \ldots, k\}$ be the set of all colors. A coloring of a length-l interval $[i, i + l - 1]$ is represented by a vector $c' = (c'_1, \ldots, c'_l) \in \mathcal{K}^l$, meaning that $c'(i) = c'_1$, $c'(i+1) = c'_2$, and so on. Recall that a coloring c' *satisfies* (the constraint of) an input interval $F_j \in \mathcal{F}$ if $C_j = c'(F_j)$. Finally, for $1 \leq i \leq n$ we say that a coloring of $[i]$ is *proper*, if it satisfies all input intervals contained in $[i]$.

The goal of the dynamic programming procedure is to fill the tables T_i in accordance with the following definition. For every $1 \leq i \leq n - l + 1$ and for every coloring $c' \in \mathcal{K}^l$, we have $T_i(c') =$ true if and only if there exists a coloring $c'' = (c''_1, \ldots, c''_{i+l-1}) \in \mathcal{K}^{i+l-1}$ of the interval $[i + l - 1]$ with $(c''_i, \ldots, c''_{i+l-1}) = c'$

satisfying each interval $F \in \mathcal{F}$ with $F \subseteq [i+l-1]$. That is, $T_i(c') = \text{true}$ if and only if there exists a proper coloring c'' of the interval $[i+l-1]$ that is an extension of c'.

For $i = 1$ and for every $c' \in \mathcal{K}^l$, this is achieved by setting $T_1(c') := \text{true}$ if and only if c' satisfies every interval $F \in \mathcal{F}$ with $F \subseteq [l]$.

For $i > 1$, table T_i is computed based on T_{i-1} as follows. For $i = 2$ to $n - l + 1$ and for every $c' = (c'_1, \ldots, c'_l) \in \mathcal{K}^l$, set

$$T_i(c') = \text{true} \iff c' \text{ satisfies every } [s,t] \in \mathcal{F} \text{ with } [s,t] \subseteq [i, i+l-1] \text{ and } \exists z \in \mathcal{K} : T_{i-1}((z, c'_1, c'_2, \ldots, c'_{l-1})) = \text{true}. \tag{8.2}$$

Finally, output "Yes" if there exists a coloring $c' \in \mathcal{K}^l$ with $T_{n-l+1}(c') = \text{true}$, and "No", otherwise. This completes the description of the algorithm.

The correctness of the algorithm follows by induction on i. More specifically, we show that T_i meets the above definition for every $1 \leq i \leq n - l + 1$. That is, we show that $T_i(c') = \text{true}$ if and only if there is a proper coloring of the interval $[i + l - 1]$ with "suffix" c'. Obviously, this holds for $i = 1$.

For the induction step we show the correctness of Recursion (8.2). That is, we show that there is a proper coloring of $[i + l - 1]$ with suffix c' (that is, $T_i(c') = \mathit{true}$) if and only if c' satisfies all input intervals contained in $[i, i + l - 1]$ and there is a proper coloring of $[i + l - 2]$ with suffix $(z, c'_1, c'_2, \ldots, c'_{l-1})$ for some $z \in \mathcal{K}$ (that is, $T_{i-1}((z, c'_1, c'_2, \ldots, c'_{l-1})) = \text{true}$).

The "$\Rightarrow$"-direction is straightforward. For the "$\Leftarrow$"-direction, note the following. The existence of a $z \in \mathcal{K}$ with $T_{i-1}((z, c'_1, c'_2, \ldots, c'_{l-1})) = \text{true}$ means that there is a coloring $c'' = (c''_1, \ldots, c''_{i-2}, z, c'_1, \ldots, c'_{l-1})$ of $[i + l - 2]$ satisfying all $F \in \mathcal{F}$, $F \subseteq [i + l - 2]$. Thus, the coloring $c^* = (c''_1, \ldots, c''_{i-2}, z, c'_1, \ldots, c'_{l-1}, c'_l)$ clearly satisfies all input intervals $[s,t] \in \mathcal{F}$ with $t \leq i + l - 2$. Moreover, c^* satisfies all input intervals $[s,t] \in \mathcal{F}$ with $t = i + l - 1$ since every input interval that ends at position $i + l - 1$ must be completely contained in $[i, i + l - 1]$.

As to the running time, for every $i \in [n]$ and for every $c' \in \mathcal{K}^l$ the computation of $T_i(c')$ according to Recursion (8.2) can be performed in $O(m(k+l))$ time. This can be seen as follows. For every input interval completely contained in $[i, i + l - 1]$, check whether it is satisfied by c', which is doable in $O(k+l)$ time (if we realize the multisets by size-k arrays). In addition, to check whether $\exists z \in \mathcal{K} : T_{i-1}((z, c_1, c_2, \ldots, c_{l-1})) = \text{true}$, try all k colors for z; hence, the running time for the above recursion is $O(m(k + l))$. This leads to a total running time of $O(k^l \cdot (k + l)mn)$ since for every position i, $1 \leq i \leq n - l + 1$, we have to try all k^l possible colorings of a length-l interval. □

Next, we present an alternative solution strategy also based on dynamic programming. The running time of this algorithm can be bounded by $(\text{cw} + 1)^l \cdot \text{poly}(n, m)$ or $l^{\text{cw} \cdot (k-1)} \cdot \text{poly}(n, m)$. To explain the basic idea of the algorithm, consider the following. Assume that we are given a coloring c' satisfying all intervals. Consider a position i. With respect to i coloring c' partitions a color multiset C of an interval F intersecting with i into two multisets: one containing the colors that c' uses for the positions $j \in F$ with $j \leq i$ and the other containing all other colors of the color multiset. The basic idea of the dynamic programming algorithm is to traverse the instance from "left to right" and to try for every position i all partitions of the color multisets of the intervals intersecting with i. Roughly speaking, for every such partition, the algorithm remembers whether this partition is consistent with a coloring satisfying all

input interval seen so far. In the proof of the next theorem, we will show that, for a position i and a partition of the color multisets intersecting with i, this decision can be made based on the stored information for position $i-1$.

Theorem 8.4. ICC *can be solved in* $O((cw+1)^l \cdot l \cdot (k \cdot cw)^2 \cdot \log cw \cdot n)$ *time and* $O(l^{cw\cdot(k-1)} \cdot (k \cdot cw)^3 \log l \cdot n)$ *time, respectively.*

Proof. We present a dynamic programming algorithm that yields both claimed running times. We use the following notation. For every position i, $1 \leq i \leq n$, let $\mathcal{F}_i = \{F_{i_1}, \ldots, F_{i_{n_i}}\}$ denote the input intervals containing i. Further, let $\mathcal{C}_i = \{C_{i_1}, \ldots, C_{i_{n_i}}\}$ denote the color multisets associated with the intervals in $\mathcal{F}_i$, where C_{i_j} is the color multiset associated with F_{i_j}, $1 \leq j \leq n_i$. Note that $n_i \leq$ cw. Let $F_{i_j} = [s_{i_j}, t_{i_j}]$ for all $1 \leq j \leq n_i$. By $\mathcal{K} = \{1, \ldots, k\}$ we refer to the set of all colors. In addition, a tuple $(M_1, \ldots, M_q)$ of multisets is called a *chain* if there exists a permutation π of $\{1, \ldots, q\}$ such that $M_{\pi(1)} \subseteq M_{\pi(2)} \subseteq \ldots \subseteq M_{\pi(q)}$. Finally, for $1 \leq i \leq n$ we say that a coloring of $[i]$ is *proper* if it satisfies all input intervals contained in $[i]$.

For every position i, the algorithm maintains a table T_i with an entry for every possible tuple of color multisets $(A_1, \ldots, A_{n_i})$ with $A_j \subseteq C_{i_j}$ and $|A_j| = i - s_{i_j} + 1$ for all $1 \leq j \leq n_i$. Informally speaking, this entry indicates whether there exists a coloring of the interval $[i]$ that uses for every j, $1 \leq j \leq n_i$, the colors in A_j for the subinterval $[s_{i_j}, i]$ and satisfies all intervals that end before position i. More specifically, the goal of the dynamic programming procedure is to compute these tables in accordance with the following definition: $T_i(A_1, \ldots, A_{n_i}) =$ true if and only if there exists a proper coloring $c' : [i] \to \mathcal{K}$ such that $c'([s_{i_j}, i]) = A_j$ for all $1 \leq j \leq n_i$ and, for every $F_l \in \mathcal{F}$ with $F_l \subseteq [i]$, it holds that $c'(F_l) = C_l$. Such a coloring is called proper with respect to $(A_1, \ldots, A_{n_i})$. Note that an instance is a yes-instance if and only if T_n contains an entry set to true.

For position $i = 1$, initiate the table T_i as follows. According to Proposition 8.1, at each position in $[n]$ there starts at most one input interval and ends at most one input interval. Hence, there is exactly one interval in $\mathcal{F}_1$. Let $\mathcal{F}_1 = \{F\}$ and let C be the color multiset associated with F. Set $T_1(\{c'\}) =$ true for every $c' \in C$.

For a position $i > 1$ compute the table T_i based on T_{i-1} as described next. By Proposition 8.1, for every position i, $2 \leq i \leq n$, there is at most one interval in $\mathcal{F}_{i-1} \setminus \mathcal{F}_i$ and at most one in $\mathcal{F}_i \setminus \mathcal{F}_{i-1}$. Thus, assume that $\mathcal{F}_{i-1} = \{F', F_1, \ldots, F_q\}$ and $\mathcal{F}_i = \{F_1, \ldots, F_q, F''\}$, that is, $\mathcal{F}_{i-1} \cap \mathcal{F}_i = \{F_1, \ldots, F_q\}$ (if $\mathcal{F}_{i-1} \setminus \mathcal{F}_i = \emptyset$ or $\mathcal{F}_i \setminus \mathcal{F}_{i-1} = \emptyset$, then skip F' or F'' and the respective color (sub)multisets in the following formulas). Let $F_j = [s_j, t_j]$ for all $1 \leq j \leq q$.

For every tuple $(A_1, \ldots, A_q, A'')$ that forms a chain and fulfills $A_j \subseteq C_j$ with $|A_j| = i - s_j + 1$ for $1 \leq j \leq q$ and $A'' \subseteq C''$ with $|A''| = 1$, set

$$T_i(A_1, \ldots, A_q, A'') = \text{ true } \iff \exists x \in \left(\bigcap_{j=1}^{q} A_j\right) \cap A'' : T_{i-1}(C', A_1 \setminus \{x\}, \ldots, A_q \setminus \{x\}) = \text{true}. \tag{8.3}$$

Using Recursion (8.3), the algorithm computes the tables T_i for increasing values of i (starting with $i = 2$). Finally, it outputs "Yes" if T_n contains a true entry and "No", otherwise. This completes the description of the algorithm.

For the correctness we show by induction on i that T_i is computed in accordance with the above definition. Clearly, this is the case for $i = 1$.

For the correctness of the case $i > 1$, first note that we only consider tuples $(A_1, \ldots, A_q, A'')$ that form chains. This is correct since for a coloring $c' : [i] \to \mathcal{K}$ the tuple $(c'([s_1, i]), \ldots, c'([s_j, i]), c'([i, i]))$ forms a chain. For the induction step we show the correctness of Recursion (8.3). That is, we show that there exists a proper coloring c of $[i]$ with respect to $(A_1, \ldots, A_q, A'')$ (that is, $T_i(A_1, \ldots, A_q, A'') =$ true) if and only there is a proper coloring c' of $[i-1]$ with respect to $(C', A_1 \setminus \{x\}, \ldots, A_q \setminus \{x\})$ for some $x \in (\bigcap_{j=1}^{q} A_j) \cap A''$.

For the "⇒-direction" note that, if there is a proper coloring c of $[i]$ with respect to $(A_1, A_2, \ldots, A_q, A'')$, then c restricted to $[i-1]$ clearly is a proper coloring of $[i-1]$ with respect to $(C', A_1 \setminus \{c'(i)\}, \ldots, A_q \setminus \{c'(i)\})$.

For the "⇐-direction", note that if there is an $x \in (\bigcap_{j=1}^{q} A_j) \cap A''$ and a proper coloring c' of $[i-1]$ with respect to $(C', A_1 \setminus \{x\}, \ldots, A_q \setminus \{x\})$, then the extension c of c' with $c(j) := c'(j)$ for $1 \leq j < i$ and $c(i) := x$ is a proper coloring of $[i]$ with respect to $(A_1, \ldots, A_q, A'')$.

Next, we show that the running time of the algorithm can be bounded by $O((\mathrm{cw}+1)^l \cdot l \cdot (k \cdot \mathrm{cw})^2 \cdot \log \mathrm{cw} \cdot n)$. To this end, note that for every position there are at most $(\mathrm{cw}+1)^l$ tuples of color multisets $(A_1, \ldots, A_{n_i})$ with $A_j \subseteq C_{i_j}$ and $|A_j| = i - s_{i_j} + 1$, $1 \leq j \leq n_i$, that form a chain. This can be seen as follows. Let F_z denote the interval in $\mathcal{F}_i$ with the smallest starting point. Note that a tuple of color multisets $(A_1, \ldots, A_{n_i})$ that forms a chain corresponds to a partition of C_z into (n_i+1) subsets. Since $n_i \leq \mathrm{cw}$ and for every color in C_z there are at most $(\mathrm{cw}+1)$ choices, there are at most $(\mathrm{cw}+1)^l$ such partitions. Next, we show that in $O(l \cdot (k \cdot \mathrm{cw})^2 \cdot \log \mathrm{cw})$ time one can determine whether there exists an $x \in (\bigcap_{j=1}^{q} A_j) \cap A''$ such that $T_{i-1}(C', A_1 \setminus \{x\}, \ldots, A_q \setminus \{x\}) =$ true. To this end, we implement the dynamic programming tables T_i by dictionaries for which the addition and the lookup of a key requires $O(\log(s))$ comparisons, where s is the size of the dictionary. Such a dictionary can, for example, be realized by a balanced binary search tree. Then, for computing an entry of T_i using Recursion (8.3), one first determines the colors in $(\bigcap_{j=1}^{q} A_j) \cap A''$, which is doable in $O(k \cdot \mathrm{cw})$ time if the multisets are realized by size-k arrays. Then, the lookup in T_{j-1} needs at most $O(l \cdot \log(\mathrm{cw}))$ comparisons. To compare two tuples one can iterate over the two tuples in parallel until finding a first pair of multisets that are different and return the result of the comparison between these multisets. Analogously, two multisets can be compared by comparing the occurrence numbers of the colors. In total, one comparison of two tuples of color multisets takes $O(\mathrm{cw} \cdot k)$ time. Hence, for a given tuple of color multisets the computations can be done in $O(l \cdot (k \cdot \mathrm{cw})^2 \cdot \log \mathrm{cw})$ time. This leads to a total running time of $O((\mathrm{cw}+1)^l \cdot l \cdot (k \cdot \mathrm{cw})^2 \cdot \log \mathrm{cw} \cdot n)$.

Finally, to prove the second running time claimed in Theorem 8.4, we perform an alternative analysis of the running time of the above algorithm. To this end, we need the following observation. For a multiset M that contains k different colors and for an integer $q \geq 1$, there are at most $(q+1)^{k-1}$ size-q submultisets of M; first, note that for every color there are $q+1$ choices for the number of occurrences of this color in the subset (between 0 and q times). Second, note that choosing the occurrence numbers of the first $k-1$ colors in a size-q subset (there are at most $(q+1)^{k-1}$ choices) determines the occurrence number of the kth color. With this observation, it is not hard to verify that for each position one has to consider at most $(l^{(k-1)})^{\mathrm{cw}} = l^{\mathrm{cw} \cdot (k-1)}$ tuples of multisets. Thus, the dynamic programming tables are of this size and with the same analysis as above the total running time can be bounded by $O(l^{\mathrm{cw} \cdot (k-1)} \cdot (k \cdot \mathrm{cw})^3 \log l \cdot n)$. □

Trivially, one can solve ICC in $k^n \cdot \text{poly}(n, m)$ time by trying all k colors for all n positions. Subsequently, we show that we can improve on this running time bound by exploiting the fact that for two colors the problem is polynomial-time solvable [8] (whereas it is NP-complete for three colors [35]). The idea is to "guess" only $k-2$ colors and the positions that have one of the two remaining colors. For these positions, we then use the polynomial-time algorithm for ICC with two colors, giving the following result.

Proposition 8.2. ICC *can be solved in* $O((k-1)^n \cdot g(n, m))$ *time, where* $g(n, m)$ *is the time needed to solve* ICC *for* $k = 2$.

Proof. For each position $1 \leq i \leq n$, we branch into $k-1$ cases. The first case corresponds to i being assigned color 1 *or* 2. The other $k-2$ cases each correspond to i being assigned one of the other $k-2$ colors. When there is no more position that we can branch on, we first check whether any of the interval constraints has been violated so far. That is, we check whether there is an interval F with associated color multiset C and a color $x \in \{3, \ldots, k\}$ such that the occurrence number of x in C is different from the number of positions in F that are colored by x. If this is the case, then this branch does not lead to a proper coloring. Otherwise, the positions with fixed colors, that is, the positions that have been assigned a color from 3 to k, are removed from $[n]$ and the intervals with their color constraints are updated accordingly (that is, we ignore the colors $3, \ldots, k$ in the color constraints). For the remaining positions, we can only assign colors 1 or 2. This problem can be solved in polynomial time [8].

Overall, we branch into $(k-1)^n$ possibilities and for each of them the problem can be solved in polynomial time. □

For the practically relevant [8] NP-complete [35] case where $k = 3$, we can achieve a further speed-up by the following simple observation: At least one of the colors appears at most on $n/3$ positions.

Proposition 8.3. *For* $k = 3$, ICC *can be solved in* $O(1.89^n \cdot g(n, m))$ *time, where* $g(n, m)$ *is the time needed to solve* ICC *for* $k = 2$.

Proof. For each of the three colors, we solve the problem of finding a coloring in which this particular color is assigned to at most $n/3$ positions. We try all possibilities of selecting the positions, and since at most $n/3$ positions have to be selected, the number of these possibilities is $\sum_{0 \leq i \leq n/3} \binom{n}{i}$. Using Stirling's approximation of factorials, we obtain an upper bound of $O(1.89^n)$ for this number. For each of these possibilities, we then solve ICC for the remaining two colors in polynomial time [8]. □

Beigel and Eppstein [14] gave a thorough study of exact exponential-time algorithms for the NP-complete 3-Coloring problem. It is tempting to investigate whether some of their tricks can be applied to ICC with three colors; in particular, a simple randomized strategy presented by Beigel and Eppstein might be promising. This is left as a challenge for future work.

8.6 Implementations and Experiments

We performed computational experiments on peptide fragment data that were also used by Althaus et al. [8] to find out whether our theoretical algorithms are valuable

in practice. We considered only the non-trivial instances with more than one fragment. Our aim was mainly to answer the following three questions: First, what do the parameters derived from the NP-hardness reduction look like in real-world data? Second, how do the presented algorithms behave on real-world data? Third, what can we conclude from these experiments; for example, can we find new promising parameterizations either from the data itself or from the behavior of our algorithms?

All experiments were run on an AMD Athlon 64 3700+ machine with 2.2 GHz, 1 M L2 cache, and 3 GB main memory running under the Debian GNU/Linux 4.0 operating system with Java version 1.5.0_14; the Java VM was invoked with 2 GB heap size.[5]

Aspects of the Data and Choice of Algorithms. First, we examined the data with respect to the parameters we identified from the NP-hardness proof in Section 8.2. The most obvious observation is that $k = 3$ in all instances; the other parameter values are shown in Table 8.3. Unfortunately, the difference between range n and maximum interval length l is not that big in many instances. This is usually due to one or two intervals that are very long in comparison to the other intervals. Furthermore, the cutwidth cw is usually much larger than k and only in trivial instances less than 3. Hence, we have not implemented our algorithm for $\mathrm{cw} = 2$ since it seems unattractive for the *available* real-world data. We decided to implement the dynamic programming algorithm with running time $k^l \cdot \mathrm{poly}(n, m)$ (called Algorithm 1 in the following) presented in the proof of Theorem 8.3 because it was conceptually the easiest and because, with $k = 3$, its running time of $3^l \cdot \mathrm{poly}(n, m)$ is much better than the running times of $l! \cdot \mathrm{poly}(n, m)$ and $(\mathrm{cw} + 1)^l \cdot \mathrm{poly}(n, m)$ of the algorithms from Theorems 8.1 and 8.4. Moreover, it is easy to extend this algorithm to solve the error minimization variant of ICC introduced by Althaus et al. [8]. Note, however, that the algorithm from Theorem 8.4 has an alternative running time bound which, for $k = 3$, is $l^{2\mathrm{cw}} \cdot \mathrm{poly}(n, m)$. Since in the data often $\mathrm{cw} \ll l$ it seems worthwhile to consider this algorithm, even though, compared to the algorithm from Theorem 8.3, it uses three parameters (l, k, and cw) instead of two (k and l). We thus implemented two algorithms that can be seen as variants of the algorithm from Theorem 8.4. The main idea of these two algorithms can be described as follows. We use dynamic programming. Both algorithms process in "left" to "right" order. The positions for which values are stored in the dynamic programming table are the start- and endpoints of the input intervals. For such a position i we store a description of each proper coloring of range $[i]$. Next, we describe this description in more detail. A segment of $[n]$ is a subinterval $[s, t] \subseteq [n]$ such that each position of $[s, t]$ is contained in exactly the same set of input intervals and $[s, t]$ is maximal with respect to this property. A description of a coloring for $[i]$ contains for each segment $[s, t]$ of $[i]$ and for each color c the number of positions of $[s, t]$ that are colored with c. When creating the table entries for position i, the algorithms try all possible combinations of extending a proper coloring stored for $[j]$, $j < i$, with colorings of the segment $[j+1, i]$, where j is the last position of the preceding segment. Basically, the algorithm as described so far (Algorithm 2) is an enumeration of all proper colorings. We have also implemented an adaption of this algorithm that, as long as there are two colorings that differ in general, but not in the segments that overlap with the last l positions,

[5] The Java program is free software and available from `http://theinf1.informatik.uni-jena.de/icc`

Table 8.3: Parameters and running times for peptide fragment data from Canzar et al. [38]. Algorithm 1 is the $k^l \cdot \text{poly}(n, m)$ algorithm, Algorithm 2 stores all equivalent colorings, Algorithm 3 stores only those colorings that differ in the segments overlapping with the last l positions. The shown parameters are range n, number of intervals m, maximum interval length l, and cutwidth cw; running times are given in milliseconds. In the column for Algorithm 1 an entry "—" means that the the respective instance could not be solved because the space consumption exceeded 2GB. In the columns for Algorithms 2 and 3 an entry "—" means that the respective instance could not be solved within a time limit of one hour.

Instance	n	m	l	cw	Algorithm 1	Algorithm 2	Algorithm 3
Cabin	78	34	74	21	—	—	—
CytoCA	27	6	16	5	11,511,587	1,225	1,642
CytoCB	26	6	26	4	—	1,690	1,649
CytoCC	15	5	14	4	88,684	427	528
FKBP-both-A	49	24	25	19	—	13,499	17,163
FKBP-both-B	11	4	7	4	403	50	46
FKBP-both-C	25	26	25	23	—	—	—
FKBP-both-D	4	2	3	2	3	33	31
FKBP-ilp-A	35	12	21	8	—	11,074	14,074
FKBP-ilp-B	16	5	9	3	1,838	237	219
FKBP-ilp-C	36	14	23	8	—	966,583	26,758
FKBP-mem-A	35	12	21	8	—	5,494	7,127
FKBP-mem-B	16	5	9	3	1,857	133	122
FKBP-mem-C	36	14	23	8	—	3,241,131	53,662
FKBP-xiii-A	22	16	21	16	—	1,453	2,224
FKBP-xiii-B	10	4	10	4	235	106	97
FKBP-xiii-C	11	4	7	4	440	48	45
FKBP-xiii-D	25	22	25	19	—	8,334	15,626
HorseHeart-A	17	10	17	7	1,047,098	2,831	4,197
HorseHeart-B	12	4	12	4	879	58	55
HorseHeart-C	22	8	11	7	32,750	880	1,669
HorseHeart-D	37	17	23	10	—	—	6,443
HorseHeart-F	21	6	19	6	—	760	760

removes one of these colorings from the dynamic programming table (Algorithm 3). This latter algorithm can be shown to have a running time of $l^{(k-1)\mathrm{cw}} \cdot \mathrm{poly}(n, m)$, as it stores at most as many combinations as the algorithm from Theorem 8.4. Finally, we implemented both algorithms to not only store proper colorings, but also colorings whose total sum of errors is below an error threshold ϵ, where the sum of errors is defined as by Althaus et al. [8]. This way an optimal solution can be found by incrementally increasing the error threshold (starting with $\epsilon = 0$ and increasing ϵ by one in each step) until one coloring that has an error below the considered threshold is found.

Evaluation. Algorithm 1 can compute the minimum error for all given instances with $l \leq 17$ (11 of 23 instances in total). However, one can observe an explosion in the running time for growing values of l. This is expected since for every length-l subinterval this algorithm enumerates all 3^l colorings. Note that for instances with $l > 17$, the space consumption of our implementation is too large and the algorithm terminates immediately.

In terms of running time, for most instances there is no big difference between Algorithms 2 and 3 and both algorithms outperform Algorithm 1. Algorithms 2 and 3 also have a moderate space consumption for all instances. Moreover, Algorithms 2 and 3 can solve all but two instances within one hour. For many instances, however, the performance is much better. For example, 14 out of 23 instances could be solved in less than four seconds. Interestingly, the fact that Algorithm 3 stores only feasible colorings if they differ in the last l positions and that Algorithm 2 stores all feasible colorings does not result in better running times of Algorithm 2 for most instances. Recall that Algorithms 2 and 3 check for an increasing error value ϵ whether the input instance admits a coloring with error ϵ. The case $\epsilon = 0$ corresponds to the decision version as introduced in Section 8.1. For the case that $\epsilon = 0$ Algorithms 2 and 3 can solve all instances in less than one second (not shown here).

So far our algorithms are not competitive with the state-of-the-art algorithms for ICC such as the ILP based approach by Althaus et al. [8] that solves every instance in less than 10 seconds or a polynomial-delay algorithm by Canzar et al. [38] that solves every instance in less than 57 seconds. In particular, Algorithm 1 is rather slow and has a high space consumption. Algorithms 2 and 3 perform much better but are still slower than the polynomial-delay algorithm due to Canzar et al. [38]. However, there is one instance (FKBP-mem-A) where Algorithm 2 (running time 5.5 sec) is competitive with the polynomial-delay algorithm by Canzar et al. [38] (running time 7.81 sec) and one instance (FKBP-mem-C) where Algorithm 3 (running time 53.66 sec) is competitive with the polynomial-delay algorithm [38] (running time 56.31 sec).

In summary, for favorable parameter constellations our algorithms solve ICC within seconds. However, for some instances (such as FKBP-both-C) none of the considered parameters are sufficiently small. Accordingly, none of the implemented algorithms solve this instance within acceptable running time.

Conclusions from the Experiments. The experiments show that the approach by deconstructing an NP-hardness proof can lead to algorithms that are capable of solving real-world instances. However, it is also obvious that the theoretical algorithms if implemented straightforwardly, such as Algorithm 1, may be inefficient. It also

becomes clear that the parameters should not only be derived from deconstructing intractability but also from examining the structure of the data. For example, cw is often much smaller than l which might be a reason for the relatively good performance of Algorithms 2 and 3 in comparison with Algorithm 1. Also note that often $l \approx n$, but the number of long intervals is usually very small. Hence, it is intriguing to consider the parameter "number of long intervals" in combination with other parameters.

8.7 Conclusion

Through deconstructing intractability and using methods of parameterized algorithmics, we started a multivariate complexity analysis of Interval Constrained Coloring. Refer to Tables 8.1 and 8.2 in Sections 8.4 and 8.5 for an overview and several challenges for future research. Next, we highlight some possible future research directions.

First, we emphasize our specific interest in the parameter m (number of intervals), which takes relatively small values for several of the considered real-world instances (see Table 8.3). The parameterized complexity with respect to parameter m is left open. Recall that for the combined parameter (k, m) fixed-parameter tractability follows by Lenstra's famous result [128] on the running time of integer linear programs with a fixed number of variables (which is of purely theoretical interest). Hence, direct combinatorial fixed-parameter algorithms for the combined parameter (k, m) are desirable.

Also the combined parameter "cutwidth cw and number of colors k" deserves further investigations since the cutwidth and the number of colors are small for most of the considered real-world instances. Recall that ICC is NP-hard for constant cutwidth or a constant number of colors and polynomial-time solvable if both values are constant (the degree of the polynomial depending on cw and k). However, we left open whether ICC is fixed-parameter tractable for the combined parameter.

Beyond that, there remain many further tasks. For instance, also combinations of three or more parameters may be relevant. Moreover, already for combinations of two parameters there are several qualitatively different fixed-parameter tractability results one can strive for and which typically are independent from each other. For instance, for a combined parameter (p_1, p_2) the incomparable combinatorial explosions $p_1^{p_2}$ and $p_2^{p_1}$ can both be useful for solving specific real-world instances.

Although polynomial-time executable data reduction rules played a significant role in this chapter, we achieved no nontrivial problem kernelization results for fixed-parameter tractable problem variants.

In our theoretical algorithms, we focussed attention on the decision version and corresponding exact solutions; the investigations should be extended to the optimization variants. So far, two optimization criteria have been proposed. First, Althaus et al. [8] suggested an error minimization version. We use the respective error function in our experiments as well. Second, Althaus et al. [9] and Canzar et al. [39] investigated a weighted version, where the goal is to find a coloring maximizing the total weight of satisfied intervals.

Finally, our experimental work indicates the practical potential of a multivariate approach to the design of combinatorial algorithms for NP-hard problems such as Interval Constrained Coloring. However, from our experimental studies it has

also become clear that a "deconstructing intractability study" should always come along with a *data-driven* algorithm design process. For example, we have identified the maximum interval length l as a theoretically interesting parameter. However, for most of the considered real-world instances we have $l \approx n$. A more precise analysis of the data shows that this is caused by only few "large" intervals, giving raise to the following questions. What is a reasonable concept of "large intervals"? Can we find improved fixed-parameter algorithms when we consider "the number of large intervals" as an (additional) parameter? Finally, can we speed up our algorithms to become practically competitive by exploiting the fact that there are relatively few large intervals?

Chapter 9

Parsimony Haplotyping

9.1 Introduction

Over the last few years, haplotype inference has become a central problem in algorithmic bioinformatics [103, 41]. Besides applications in the investigation of diseases and genetic mutations, haplotype inference plays an important role in the inference of population histories, drug design, and pharmacogenetics. A brief introduction follows.

The DNA sequences of the individuals of a population are almost identical except for the nucleotides at so-called *single nucleotide polymorphism* (SNP) sites. Moreover, at each SNP site there almost always appear only two (of the four possible) nucleotides that, hence, are usually encoded by the two states "0" and "1". Diploid organisms such as mammals have two not necessarily identical copies of the DNA sequence, each called a *haplotype*. In particular, the haplotypes can differ at the SNP sites. The *genotype* of an organism is the combination of the two haplotypes, and can be encoded by a string consisting of "0", "1", and "2" entries, where a "0" ("1") means that both haplotypes have state "0" ("1") at the respective SNP site and "2" means that the haplotypes differ at the respective SNP site. A site is called *homozygous* if the two haplotypes agree in this site; otherwise it is called *heterozygous*. Commonly used sequencing methods produce such an encoding of the genotype using "2" for heterozygous sites. Observe that, however, in this encoding for a heterozygous site the information which of the two haplotypes has state "0" and which has "1" is lost. Now, given the genotypes of the individuals of a population, the task in *haplotype inference* is to "reconstruct" the haplotypes. The central motivation for haplotype inference is that it is easier and cheaper to obtain the genotype information of an organism, though the haplotype information is of greater use [103]. See for example the surveys by Catanzaro and Labbé [41] and Gusfield and Orzack [103] for the biochemical background.

In summary, a genotype can be seen as a length-m string over the alphabet $\{0, 1, 2\}$, while a haplotype can be seen as a length-m string over the alphabet $\{0, 1\}$ and, in haplotype inference, the goal is to extract the haplotype information from the given the genotype information of a population. One of the major approaches to haplotype inference is *parsimony haplotyping*: Given a set of genotypes, the task is to find a *minimum-cardinality* set of haplotypes that explains the input set of genotypes. The

task to select as few haplotypes as possible (parsimony criterion) is motivated by the observation that in natural populations the number of haplotypes is much smaller than the number of genotypes and, hence, there is a good chance that a minimum-cardinality set of haplotypes corresponds to the "true" biological haplotypes [41].

Next, we provide basic definitions that are necessary to formally introduce the problems considered in this chapter.

Problem Statement. A set H of haplotypes *explains*, or *resolves*, a set G of genotypes if for every $g \in G$ there is either an $h \in H$ with $g = h$ (trivial case), or there are two haplotypes h_1 and h_2 in H such that, for all $i \in \{1, \ldots, m\}$,

- if g has letter 0 or 1 at position i, then both h_1 and h_2 have this letter at position i, and
- if g has letter 2 at position i, then one of h_1 and h_2 has letter 0 at position i while the other one has letter 1.

For example, $H = \{00100, 01110, 10110\}$ resolves $G = \{02120, 20120, 22110\}$; the first genotype 02120 is resolved by the two haplotypes 00100 and 01110, and so on. For a given set H of haplotypes, let $\operatorname{res}(H)$ denote the set of genotypes resolved by H. We consider the following haplotype inference problem parameterized by the size of the haplotype set H to be computed.

Definition 9.1. Haplotype Inference by Parsimony (HIP)
Input: A set G of length-m genotypes and an integer $k \geq 0$.
Question: Is there a set H of length-m haplotypes such that $|H| \leq k$ and $G \subseteq \operatorname{res}(H)$?

HIP is NP-hard, and numerous algorithmic approaches based on heuristics and integer linear programming methods are applied in practice [41]. There is also a growing list of combinatorial approaches (with provable performance guarantees) including the identification of polynomial-time solvable special cases, approximation algorithms, and fixed-parameter algorithms [65, 72, 126, 127, 151, 111].

Fellows et al. [72] introduced a constrained version of HIP, where a pool of plausible haplotypes $\tilde{H}$ is given together with the set of genotypes, and the goal is to find a set $H \subseteq \tilde{H}$ that resolves the genotypes:

Definition 9.2. Constrained Haplotype Inference by Parsimony (CHIP)
Input: A set G of length-m genotypes, a set $\tilde{H}$ of length-m haplotypes, and an integer $k \geq 0$.
Question: Is there a set $H \subseteq \tilde{H}$ of length-m haplotypes such that $|H| \leq k$ and $G \subseteq \operatorname{res}(H)$?

In this chapter, we improve the running times of the existing fixed-parameter algorithms for HIP and CHIP and devise a simple kernelization for HIP. Before describing our new results, we summarize previous results that are relevant for this work.

Previous Work. Lancia and Rizzi [127] showed that parsimony haplotyping can be solved in polynomial time if every genotype string contains at most two letters 2, while the problem becomes NP-hard if genotypes may contain three letters 2 [126]. Sharan et al. [151] proved that parsimony haplotyping is APX-hard even in very restricted

cases. They identified instances with a specific structure that allow for polynomial-time exact solutions or constant-factor approximations. Moreover, they showed that the problem is fixed-parameter tractable with respect to the parameter $k =$"number of haplotypes in the solution set". The corresponding exact algorithm has running time $O(k^{k^2+k} \cdot m)$. These results were further extended by van Iersel et al. [111] to cases where the *genotype matrix* (the rows are the genotypes and the columns are the m positions in the genotype strings) has restrictions on the number of 2's in the rows and/or columns. They identified various special cases of haplotyping with polynomial-time exact or approximation algorithms with approximation factors depending on the numbers of 2's per column and/or row, leaving open the complexity of the case with at most two 2's per column (and an unbounded number of 2's per row). Further results in this direction have been provided by Cicalese and Milanič [50].

Finally, parsimony haplotyping has been extended to incorporate prior knowledge on which haplotypes are available. First, Fellows et al. [72] introduced the *constrained parsimony haplotyping problem* where the set of haplotypes may not be chosen arbitrarily from $\{0,1\}^m$ but only from a pool $\tilde{H}$ of plausible haplotypes. Using an intricate dynamic programming algorithm, they extended the fixed-parameter tractability result of Sharan et al. [151] to the constrained case, proving a running time of $k^{O(k^2)} \cdot \text{poly}(m, |\tilde{H}|)$.

Second, Elberfeld and Tantau [65] further refined the model by allowing to specify a pool of plausible haplotypes for each given genotype separately. They showed that this problem is W[2]-complete with respect to the parameter k (denoting the number of distinct haplotypes in the solution) but fixed-parameter tractable when parameterized by k and l, where l denotes the number of duplicated genotypes with "pairwise incomparable pool constraints".

Our Results. We improve the fixed-parameter tractability results of Sharan et al. [151] and Fellows et al. [72] by proposing fixed-parameter algorithms for the constrained and unconstrained versions of parsimony haplotyping that run in $O(k^{4k} \cdot k^3 \cdot m \cdot |\tilde{H}|^2)$ and in $O(k^{4k} \cdot k^3 \cdot m)$ time, respectively. This is a significant exponential speed-up over previous algorithms, with exponential running time factors k^{k^2+k} and $k^{O(k^2)}$. Moreover, we develop a polynomial-time data reduction rule that yields a problem kernel of size at most $2^k k^2$ for HIP.

Preliminaries. Throughout this chapter, we consider *genotypes* as strings of length m over the alphabet $\{0,1,2\}$, while *haplotypes* are considered as strings of length m over the alphabet $\{0,1\}$. If s is a string, then $s[i]$ denotes the letter of s at position i. This applies to both haplotypes and genotypes.

Two haplotypes h_1 and h_2 *resolve* a genotype g, denoted by $\text{res}(h_1, h_2) = g$, if, for all positions i, either $h_1[i] = h_2[i] = g[i]$, or $g[i] = 2$ and $h_1[i] \neq h_2[i]$. For a given set H of haplotypes, let $\text{res}(H) := \{\text{res}(h_1, h_2) \mid h_1, h_2 \in H\}$ denote the set of genotypes resolved by H. We say a set H of haplotypes *resolves* a given set G of genotypes if $G \subseteq \text{res}(H)$. Note that with k haplotypes one can resolve at most $\binom{k}{2} + k$ genotypes. Hence, throughout this chapter, we assume that $|G|$ is bounded from above by $\binom{k}{2} + k \leq k^2$.

For the presentation of our results, we use notation from graph theory (see Section 2.6). In this chapter, we consider undirected graphs that may contain self-loops

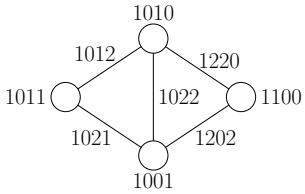


Figure 9.1: A solution graph resolving the genotypes 1012, 1021, 1022, 1202, and 1220, by the haplotypes 1001, 1010, 1011, and 1100.

(that is, edges of the form $\{v, v\}$). Furthermore, a graph $G' = (V, E')$ is called a *spanning subgraph* of a graph $G = (V, E)$ if $E' \subseteq E$ and, for every connected component (V'', E'') of G, $G'[V'']$ is connected.

9.2 Improved Fixed-Parameter Algorithms for Haplotype Inference by Parsimony

In this section, we present algorithms to solve the parsimony haplotyping problem for the unconstrained (HIP) and the constrained versions (CHIP) in $O(k^{4k} \cdot k^3 \cdot m)$ and $O(k^{4k} \cdot k^3 \cdot m \cdot |\tilde{H}|^2)$ time, respectively. This improves and partially simplifies previous fixed-parameter tractability results [72, 151]. In addition, we provide a simple kernelization for the unconstrained version.

We start with some preliminary considerations. Given a set H of haplotypes resolving a given set G of genotypes, the relation between the haplotypes and the genotypes can be depicted by an undirected graph, in which the edges are labeled by the genotypes and every vertex v is labeled by a haplotype h_v. If an edge $\{u, v\}$ is labeled by genotype g, we require that $g = \text{res}(h_u, h_v)$. Such a graph is called a *solution graph* for G. See Figure 9.1 for an illustration.

A graph with edge labels of length-m strings over the alphabet $\{0, 1, 2\}$ (corresponding to genotypes) is called an *inference graph*. For an inference graph a vertex labeling by haplotypes is *consistent* if for each edge $e = \{v, w\}$ it holds that $g_e = \text{res}(h_u, h_v)$, where g_e is the genotype associated with e and h_u and h_v are the haplotypes associated with u and v. Note that a consistent vertex labeling does not necessarily exist. Solution graphs as well as the inference graphs may contain self-loops (to model that a genotype that does not contain a letter 2 can be resolved by a single haplotype).

We first present our results for HIP. Then, we show that our algorithm can be adapted for CHIP.

9.2.1 Haplotype Inference by Parsimony

The basic idea of the previous fixed-parameter algorithm for HIP is to enumerate all $O(k^{k^2})$ inference graphs for the given set of genotypes, without explicitly using the notion of inference graphs. Here, we show that it is sufficient to enumerate only the at most $O(k^{4k})$ inference graphs for all size-k subsets of the given genotypes.

As we will see, one of these inference graphs corresponds to a spanning subgraph of a solution graph (if one exists) and contains a sufficient amount of information to

construct the solution graph for all genotypes. This can be shown with the help of the three following lemmas[1].

Lemma 9.1. *Let G be a set of genotypes and let $\Gamma = (V, E)$ be a connected inference graph for G. For each position i, $1 \leq i \leq m$, if there is a genotype $g \in G$ with $g[i] \neq 2$, then one can, in $O(|V| + |E|)$ time, uniquely infer the letters of all haplotypes at position i or report that no consistent vertex labeling exists.*

Proof. Let g be a genotype with $g[i] \neq 2$ and let $e_g = \{u, v\}$ denote the edge associated with g. For every consistent vertex labeling it is required that $\text{res}(h_u, h_v) = g$, implying that $h_u[i] = h_v[i] = g[i]$. Thus, let $h_u[i] := g[i]$ and $h_v[i] := g[i]$. Now, consider an arbitrary edge $e_{g'} = \{x, y\} \in E$ (associated with $g' \in G$) for which $h_x[i]$ is already known. Then, one can infer $h_y[i]$ as follows. If $g'[i] = 2$, then set $h_y[i] := 1 - h_x[i]$; otherwise, set $h_y[i] := h_x[i]$, and if $g'[i] \neq h_x[i]$, then report that there is no consistent vertex labeling. Hence, in a depth-first traversal starting at u (note that $h_u[i]$ is known) one can determine the letter at position i of all haplotypes in $O(|V| + |E|)$ time. Observe that there is no freedom of choice in any step. Hence, the letters of the haplotypes at position i are uniquely determined. □

Next, we show that for non-bipartite inference graphs (that is, the inference graph contains an odd-length cycle) the haplotypes are uniquely determined.

Lemma 9.2. *Let $\Gamma = (V, E)$ be a connected inference graph for a set G of genotypes. If Γ contains an odd-length cycle, then there exists at most one consistent vertex labeling. Furthermore, one can compute in $O(m \cdot (|V| + |E|))$ time a consistent vertex labeling or report that no consistent vertex labeling exists.*

Proof. We show that if there exists a consistent vertex labeling, then for every position i there exists at least one genotype on every odd cycle with a letter different from 2 at position i. Then, the claim follows directly from Lemma 9.1. Assume towards a contradiction that there exists an odd-length cycle such that all genotypes on this cycle have letter 2 for some position i, $1 \leq i \leq m$. However, $h_x[i] \neq h_y[i]$ for every edge $\{x, y\}$ on this cycle, which is impossible for an odd-length cycle. □

For bipartite inference graphs there may exist several consistent vertex labels. The crucial observation that we use in our algorithms is as follows. Consider a vertex u from the first and a vertex v from the second partition of a bipartite graph. For every consistent vertex labeling the same genotype is resolved by the haplotypes corresponding to u and v.

Lemma 9.3. *Let $\Gamma = (V_a, V_b, E)$ be a connected bipartite inference graph for a set G of length-m genotypes. Then, the following holds.*

1. *One can compute in $O(m \cdot (|V_a| + |V_b| + |E|))$ time a consistent vertex labeling or report that no consistent vertex labeling exists.*
2. *For every $u \in V_a$ and $w \in V_b$ consider two arbitrary consistent vertex labelings. Assume that the first one assigns h_u to u and h_v to v and the second one assigns h'_u to u and h'_v to v. Then, it holds that $\text{res}(h_u, h_v) = \text{res}(h'_u, h'_v)$.*

[1] Damaschke [51] uses similar graph-theoretic observations in the context of incremental haplotype inference.

Input: A set of genotypes $G \subseteq \{0,1,2\}^m$ and an integer $k \geq 0$.
Output: Either a set of haplotypes H with $|H| \leq k$ and $G \subseteq \operatorname{res}(H)$, or "no" if there is no such H.

```
1  forall size-k subsets G' ⊆ G do
2      forall inference graphs Γ for G' on k vertices and k edges do
3          forall non-bipartite connected components of Γ do
4              if possible, compute a consistent vertex labeling of the component
               (Lemma 9.2),
5              otherwise try the next inference graph (goto line 2);
6          end
7          forall bipartite connected components of Γ do
8              if possible, compute a consistent vertex labeling of the component
               (Lemma 9.3),
9              otherwise try the next inference graph (goto line 2);
10         end
11         Let H denote the inferred haplotypes (vertex labels);
12         if G ⊆ res(H) then return H;
13     end
14 end
15 return "no";
```

Algorithm 4: An algorithm solving HIP in $O(k^{4k} \cdot k^3 \cdot m)$ time.

Proof. By Lemma 9.1, one can uniquely infer for all vertex labels the letter at position i if there exists a genotype $g \in G$ with $g[i] \neq 2$. Thus, consider a position i at which all genotypes in G have letter 2. Then, $h_x[i] \neq h_y[i]$ for every edge $\{x, y\} \in E$. Hence, all haplotypes associated with the vertices in V_a must have the same letter at position i, which must be different from the letter at position i of all haplotypes associated with the vertices in V_b. Thus, there are two ways to set position i of the haplotypes in Γ in order to obtain a consistent vertex labeling of Γ. Either all haplotypes associated with vertices in V_a are 0 and in V_b are 1 at position i, or vice versa. This shows part (1) of the lemma. Part (2) follows because all other positions of the haplotypes are uniquely determined. □

Based on the previous lemmas, we can now show the first main result of this section.

Theorem 9.1. Haplotype Inference by Parsimony *can be solved in* $O(k^{4k} \cdot k^3 \cdot m)$ *time.*

Proof. We describe the algorithm for HIP, see Alg. 4. We first select a size-k subset of genotypes (line 1 of Alg. 4) and then enumerate all inference graphs on k vertices containing exactly k edges labeled by the k chosen genotypes (line 2 of Alg. 4). Assume that there exists a solution graph for G. Out of all inference graphs on k vertices and k edges, consider one with the following properties:

- it contains a spanning subgraph of every connected component of the solution graph, and
- the spanning subgraph of any non-bipartite connected component contains an odd cycle.

Obviously, this inference graph exists and is considered by Alg. 4 (lines 1 and 2). For every non-bipartite connected component of this inference graph the algorithm infers the vertex labels based on Lemma 9.2 (see line 5 of Alg. 4). By Lemma 9.2, the vertex labels for all non-bipartite connected components of the inference graph (containing an odd-length cycle) are uniquely determined. Hence, these haplotypes are exactly the haplotypes in the respective non-bipartite connected component of the solution graph.

Moreover, for every bipartite component, the algorithm can compute a consistent vertex labeling based on Lemma 9.3 (see line 9 of Alg. 4). In such a bipartite component, by Lemma 9.3, for any two vertices $u \in V_a$ and $v \in V_b$, the genotypes resolved by the corresponding haplotypes are identical for every consistent vertex labeling. Thus, if considering a spanning subgraph of a solution graph, then the haplotypes resolve all genotypes contained in the respective bipartite component of the solution graph.

In summary, if the given instance is a yes-instance, then our algorithm will find a set of at most k haplotypes resolving the given genotypes.

It remains to analyze its running time. First, there are $O(\binom{|G|}{k})$ size-k subsets G' of G. Second, there are $O(k^{2k})$ inference graphs on k vertices containing exactly k edges labeled by the genotypes in G': for every genotype $g \in G'$ we have k^2 choices for the endpoints of the edge labeled by g since self-loops may occur. For each of those inference graphs, applying Lemma 9.2 and Lemma 9.3 to its connected components takes $O(k \cdot m)$ time and the test whether $G \subseteq \operatorname{res}(H)$ is doable in $O(k^3 m)$ time. Thus, the overall running time of Alg. 4 is bounded by $O(\binom{|G|}{k} \cdot k^{2k} \cdot m \cdot k^3)$. Finally, since $|G| \leq k^2$, HIP can be solved in $O(k^{4k} \cdot k^3 \cdot m)$ time. □

9.2.2 Constrained Haplotype Inference by Parsimony

Now, we argue that Alg. 4 can be adapted to solve the constrained version CHIP.

Theorem 9.2. Constrained Haplotype Inference by Parsimony *can be solved in* $O(k^{4k} \cdot k^3 \cdot m \cdot |\tilde{H}|^2)$ *time.*

Proof. First, we explain the modified algorithm. As for the unconstrained version, it enumerates all size-k subsets $G' \subseteq G$ and all inference graphs for G'.

For the non-bipartite components the only difference is to check whether the inferred haplotypes (which by Lemma 9.2 are uniquely determined) are contained in the given haplotype pool $\tilde{H}$ (otherwise, try the next inference graph).

The only substantial difference is how to process the bipartite components of the inference graph. Let (V_a, V_b, F) be a connected bipartite component of the current inference graph. Instead of choosing an arbitrary consistent vertex labeling as done in Alg. 4, proceed as follows. Choose an arbitrary vertex $v \in V_a \cup V_b$ and check for every haplotype $h \in \tilde{H}$ whether there exists a consistent vertex labeling for this component where v is labeled by h. It is easy to verify that fixing the vertex label for v implies the existence of at most one consistent vertex labeling of (V_a, V_b, F). If existing, this labeling can be computed by a depth-first traversal starting at v. If for a haplotype h there exists a consistent vertex labeling of (V_a, V_b, F) such that all labels are contained in $\tilde{H}$, then proceed with the next bipartite component. Otherwise, one can conclude that for the current inference graph there is no consistent vertex labeling using only the given haplotypes from $\tilde{H}$, and, hence, one can proceed with the next inference graph.

As to the correctness, assume that there is a solution graph and that the considered inference graph is a spanning subgraph of it that contains for every (non)bipartite connected component of the solution graph a (non)bipartite spanning subgraph. Note that the haplotypes inferred for the bipartite components of the inference graphs are not necessarily the haplotypes of the respective bipartite connected component of the solution graph. However, by Lemma 9.3, these haplotypes resolve all genotypes of the respective bipartite connected component of the solution graph.

For the running time note the following. For a non-bipartite connected component (V, E) finding a consistent vertex labeling (see Lemma 9.2) and testing whether the found haplotypes are contained in $\tilde{H}$ is doable in $O(|\tilde{H}||E|m)$ time. Moreover, for a bipartite component (V_a, V_b, F) and a vertex $v \in V_a \cup V_b$ checking whether for a $h \in \tilde{H}$ there is a consistent vertex labeling with $h_v = h$ with all labels from $\tilde{H}$ is clearly doable in $O(|\tilde{H}|^2|F|m)$ time. Hence, with the same arguments as in the proof of Theorem 9.1 the overall running time can be bounded by $O(k^{4k} \cdot k^3 \cdot m \cdot |\tilde{H}|^2)$. □

9.3 Problem Kernelization for Haplotype Inference by Parsimony

In this section, we show that HIP admits an exponential-size problem kernel. To this end, we assume the input G to be in the matrix representation that is mentioned in the introduction; that is, each row represents a genotype while each column represents a position. Since it is obvious that we can upper-bound the number n of genotypes in the input by k^2, it remains to bound the number m of columns (positions) in the input. To this end, we employ one simple data reduction rule that deletes one of two identical columns.

Reduction Rule 9.1. Let (G, k) be an instance of HIP. If two columns of G are equal, then delete one of them.

The correctness of Reduction Rule 9.1 follows by the observation that, given at most k haplotypes resolving the genotypes in the reduced instance, we can easily find a solution for the original instance by copying the respective haplotype positions.

Lemma 9.4. *Reduction Rule 9.1 is correct and can be exhaustively applied in* $O(n \cdot m \cdot \log m)$ *time.*

Proof. Let columns i and j of G be equal. Let G' be the genotype matrix resulting from the deletion of column j according to Reduction Rule 9.1. If one can resolve all genotypes in G with k haplotypes, then one can also resolve all genotypes in G' by deleting position j of each of the k haplotypes. If there are k haplotypes resolving G', then one can also construct k haplotypes resolving G by inserting a copy of position i at position j in each haplotype. To implement the rule, we need to sort the m column strings (each of length n), which takes $O(n \cdot m \cdot \log m)$ time. □

Next, we bound the number of columns in a reduced instance. This allows us to bound the size of the whole instance by a function in k.

Lemma 9.5. *Let* (G, k) *be a yes-instance of* HIP *that is reduced with respect to Reduction Rule 9.1. Then,* G *has at most* 2^k *columns.*

Proof. Let H denote a matrix of k haplotypes resolving G. It is obvious that if two columns i and j of H are equal, then columns i and j of G are equal. Now, since G does not contain a pair of equal columns, neither does H. Since there are only 2^k different strings in $\{0,1\}^k$, it is clear that H cannot contain more than 2^k columns and, thus, neither can G. □

Since the number n of genotypes can be upper-bounded by k^2 and the number m of columns can be upper-bounded by 2^k (Lemma 9.5), one directly obtains Proposition 9.1.

Proposition 9.1. Haplotype Inference by Parsimony *admits a problem kernel of size at most* $2^k \cdot k^2$ *that can be constructed in* $O(n \cdot m \cdot \log m)$ *time.*

Combining Proposition 9.1 and Theorem 9.1, we achieve the following.

Corollary 9.1. Haplotype Inference by Parsimony *can be solved in* $O(k^{4k}k^3 \cdot 2^k + n \cdot m \cdot \log m)$ *time.*

Finally, we remark that Reduction Rule 9.1 is incorrect for the constrained version: Copying a position in a haplotype may lead to a haplotype that is not contained in the given pool $\tilde{H}$. Hence, in case of CHIP, we are not allowed to delete one of two identical columns of the genotype matrix. Moreover, a challenging task for devising a (nontrivial) problem kernel for CHIP is to upper-bound the number of haplotypes in the pool.

9.4 Further Results and Conclusions

In the previous sections, we contributed improved fixed-parameter algorithms for Haplotype Inference by Parsimony and Constrained Haplotype Inference by Parsimony. In this section, we briefly discuss further results from our paper [76] and highlight some questions for future research.

A further substantial result in our paper [76] is the identification of a polynomial-time solvable special case for HIP and CHIP. More precisely, we consider the case that the given set of genotypes is *complete*, that is, contains all possible genotypes that can be explained by the set of haplotypes. We call this special case *induced parsimony haplotyping*. More formally, we consider the following problem version.

> Induced (Constrained) Haplotype Inference by Parsimony ((C)IHIP):
> **Input**: A set G of length-m genotypes each containing at least one letter 2 (and a set $\tilde{H}$ of length-m haplotypes).
> **Question**: Is there a set H ($\subseteq \tilde{H}$) of length-m haplotypes such that $G = \text{res}(H) \setminus H$?

We could show that IHIP and CIHIP can be solved in $O(k \cdot m \cdot |G|)$ and $O(k \cdot m \cdot (|G| + |\tilde{H}|))$ time, respectively [76]. Note that many previous polynomial-time solvable cases [50, 127, 151, 111] require a bound on the number of 2's in the genotype matrix. In contrast, complementing these cases, IHIP does not require such a bound on the number of 2's.

Our results also lead to several new questions for future research.

- Polynomial-time special cases are particularly interesting to pursue a "distance from triviality" approach [94]. The idea here is to identify and exploit parameters that measure the distance of general instances of HIP or CHIP to the "trivial" (that is, polynomial-time solvable) induced cases. For example, it is an interesting open question whether HIP and CHIP are efficiently solvable for "almost induced" cases and how the distance to the induced case influences the computational complexity of (C)HIP. More specifically, one concrete question in this direction is whether the following problem version of HIP is fixed-parameter tractable, when parameterized by a "distance parameter" p: given a set G of length-m genotypes and an integer $p \geq 0$, is there a set H of length-m haplotypes such that $G \subseteq \operatorname{res}(H)$ and $|\operatorname{res}(H) \setminus G| \leq p$?

- It remains an interesting open problem to find fixed-parameter algorithms for HIP and CHIP with an exponential factor of the form c^k for some constant c.

- Our kernelization for HIP yields a problem kernel of exponential size. It would be interesting to know whether a polynomial-size problem kernel exists. This may also be seen in the light of recent breakthrough results on methods to prove the non-existence of polynomial-size kernels [30, 78]. Finally, as mentioned in Section 9.3, the presented data reduction rule cannot be used for CHIP. Devising a nontrivial problem kernel for CHIP is left open.

Besides the parsimony approach, a second main approach to haplotype inference is Perfect Phylogeny Haplotyping (PPH). Here the goal is to find a set of explaining haplotypes that defines a perfect phylogeny (perfect phylogeny assumption). Roughly speaking, a set of haplotypes admits a prefect phylogeny if there exists a tree (where the nodes are bijectively labeled with the haplotypes) explaining the evolutionary relationship of the haplotypes. Technically, this implies that the haplotype matrix (the rows corresponding to the haplotypes) does not contain a 4×2-submatrix with rows 00, 01, 10, and 11 (in arbitrary order). In contrast to HIP, PPH is solvable in polynomial-time [102]. Moreover, the maximum parsimony and the perfect phylogeny assumptions have been combined, leading to an NP-hard problem again [10]. Recently, Elberfeld and Tantau [65] presented $2^{2k^2} \cdot \operatorname{poly}(n, m)$-time algorithms for the unconstrained and constrained haplotype inference problem with combined assumptions. These algorithms are based on enumerating all inference graphs (called "empty sharing plans") for the given genotypes. It is an interesting question whether our approach can be adapted to speed up these algorithm.

Part IV

Conclusion

Chapter 10

Conclusion

The thesis presented several new fixed-parameter algorithms for combinatorial hard problems with applications in the field of computational biology. We refer to Section 1.2 for an overview of the considered problems and the obtained results.

Here, we highlight some results and aspects that are of general interest for future research. We refer to the concluding sections of Chapters 4 to 9 for several problem-specific comments, open problems, and future research challenges.

Nonstandard Parameterizations, Multivariate Algorithmics, and Data-Driven Parameterization

One aspect of the thesis was to suggest that parameterized complexity studies with respect to a standard parameter should be complemented by a systematic study of the influence of several further parameters and their combination on the computational complexity of a problem. This was driven by the inquisitiveness to better understand the computational complexity of a problem and was further motivated by the observation that the standard parameter is often not really small for considered real-world instances. There is no reason why parameterized complexity studies should be restricted to only few single parameters. In particular, for practically relevant problems a systematic study of several parameters and their combination seems unavoidable if the goal is to significantly increase the range of instances that can be solved in practice.

In this line, we proposed to consider refined parameters and suggested the approach of deconstructing intractability for the identification of meaningful parameters (see Section 2.3). Chapter 4 presents fixed-parameter algorithms for Cluster Editing and Cluster Deletion with respect to a refined parameter, namely the "cluster vertex deletion number" and also discussed further nonstandard parameterizations. In Chapter 8, we performed a systematic "deconstruction of intractability" approach for the identification of meaningful parameters and initiated a multivariate complexity analysis of Interval Constrained Coloring. We substantiated the usefulness of the deconstructing intractability approach by presenting encouraging results for real-world data for Interval Constrained Coloring.

Many problems with applications in algorithmic bioinformatics still await a sys-

tematic multivariate complexity analysis. For example, in Section 7.7 we argued that in case of Minimum Flip Consensus Tree and the closely related Flip Supertree problem a systematic multivariate complexity analysis is desirable and should be explored in future studies.

Having identified theoretically interesting parameters a logical next step is to closely inspect the real-world data in order to see whether these parameters are small in practice. If this is not the case, then it might be interesting to analyze whether the instances are close to instances with small parameter values. For Interval Constrained Coloring, in Chapter 8, we identified the "maximum interval length" as an interesting parameter that allows for fixed-parameter tractability. However, this parameter turned out to be relatively large for many real-world instances. The data showed further that this is caused by only few "large" intervals and, hence, it seems natural to investigate whether this aspect can be exploited algorithmically. Such considerations suggest that a multivariate complexity analysis is particularly interesting in combination with a "data-driven" algorithm design. In the spirit of algorithm engineering, the idea is to start by analyzing real-world data in order to identify parameters with small values and then to perform a multivariate complexity analysis for the identified parameters. We are not aware of a systematic work in this direction.

Note that there are several other interesting approaches for parameter identification. For an overview, we refer to the survey by Niedermeier [138].

Polynomial-Time Data Reduction and Kernelization

Kernelization has developed into one of the most active research areas within parameterized algorithmics and is also the main technique used in this thesis. We refer to Section 2.4 for an short introduction to the concept of kernelization. Altogether, the thesis contributes to this field by the following results. The main results in Chapter 6 and Chapter 7 are polynomial-size problem kernels. Two further polynomial-size problem kernels are presented in Section 5.5 and a simple exponential-size kernel is given in Section 9.3. In addition, polynomial-time data reduction played an important role for showing that Interval Constrained Coloring is polynomial-time solvable for instances with cutwidth two (Section 8.4). In the following, we discuss some future research challenges in the field of kernelization.

Kernelization for Edge Modification Problems. The investigation of the existence of polynomial-size problem kernels for edge modification and closely related problems has recently attracted special attention (see Section 3.3). This includes many polynomial-size kernelization results for Π-Editing, where Π denotes a hereditary graph property. One may ask whether Π-Editing always allows for polynomial-size problem kernels if Π can be characterized by finite forbidden subgraphs. However, Kratsch and Wahlström [124] showed that there is a graph H on seven vertices for which there is little hope that H-free Editing admits a polynomial-size problem kernel. This contrasts the case of vertex deletion where for every fixed forbidden subgraph H, H-free Vertex Deletion admits a polynomial-size problem kernel. In this context, the question arises for which graph properties Π the Π-Editing problem admits a polynomial-size problem kernel.

To obtain general kernelization results it seems desirable to design data reduction rules for classes of properties. In this line, we presented a universal data reduction

rule for a whole class of edge modification problems. We employed this rule (in combination with other, problem-specific data reduction rules) to obtain kernelizations for several edge modification problems (Section 3.4). The applicability of this rule only depends on the maximum cardinality of a critical clique or critical independent set in a forbidden subgraph describing the desired graph property. It is desirable to "unify" other commonly used data rules reduction for edge modification problems. and to design new data reduction rules that are applicable to a large class of edge modification problems. We discuss one concrete question in this context.

As pointed out by Guo [88], several polynomial-size problem kernels for edge modification problems make use of a data reduction rule that removes vertices that are not contained in any forbidden induced subgraph. Indeed, Reduction Rule 6.2 and Reduction Rule 7.3 for M-Hierarchical Tree Clustering and Minimum Flip Consensus Tree, respectively, are of this type. This raises the question for which graph properties such a data reduction rule is correct. Notably, we could show that such a rule is incorrect even for some 1-critical clique preserving graph properties: In Section 5.5.2, we presented an example for 2-Vertex-Overlap Deletion showing that deleting a "satisfied vertex", that is, a vertex not contained in any forbidden induced subgraph, does not lead to an equivalent instance. We note, however, that we employed other data reduction rules (Reduction Rule 5.1 and Reduction Rule 5.6), which can be seen as a generalization of a rule that removes "satisfied vertices". Here, the idea was to also consider the neighborhood structure of a satisfied vertex. A data reduction rule that unifies all these mentioned rules seems necessary to obtain "generalized" kernelization results for edge modification problems.

Average-Case Analysis. From our experiments in Chapter 6 for M-Hierarchical Tree Clustering, we have concluded that our fixed-parameterized algorithms based on a simple search tree interleaved with data reduction is the method of choice for parameter values $k < |X|$ (recall that X denotes the set of elements to be clustered and k is the cost of the solution). We could explain this behavior by the fact that most instances with $k < |X|$ could be solved without branching, just by applying the data reduction rules. For this behavior, it was decisive that $k < |X|$ whereas the absolute value of k seems not important. Note that it is clear from the analysis of the kernel size that the data reduction rules must take effect if $k << |X|$. However, often the instances are (completely) solved by applying the kernelization even if k is only slightly smaller than $|X|$.

A theoretical study of this behavior could further substantiate the usefulness of kernelization. To this end, the following questions seem worth to be investigated (under reasonable input distributions).

- What is the expected kernel size for instances of M-Hierarchical Tree Clustering with $k < |X|$?
- What is the expected percentage of instances of M-Hierarchical Tree Clustering with $k < |X|$ that can be solved to optimality just by applying the data reduction rules?
- What is the expected running time of the search tree algorithm interleaved with the data reduction for instances with $|X| < k$?

Clearly, such considerations are interesting not only for M-Hierarchical Tree Clustering but seem to be of general relevance.

Kernelizations for Parameters that Measure the Tree-Likeness of a Graph. Here, we discuss some aspects of kernelizations for parameters measuring the "tree-likeness" of a graph. The treewidth of a graph is the most prominent example of a parameter measuring the tree-likeness of a graph. Fixed treewidth allows for fixed-parameter algorithms for many relevant graph problems. However, there are problems that are fixed-parameter intractable even when parameterized by the treewidth. Also, there is evidence that most graph problems do not admit polynomial-size problem kernels when parameterized by the treewidth [30]. Altogether, this motivates the study of parameters that impose a stronger restriction on the input graph than treewidth, for example the "vertex cover number" of a graph [18, 74, 75, 136]. Other parameters considered in this context are the "feedback vertex/edge set number" [18, 113, 136, 155]. Note that the treewidth is upper-bounded by the "feedback vertex set number" which, in turn, is upper-bounded by the "vertex cover number". It turned out that some problems admit polynomial-size problem kernels with respect to the "feedback vertex set number" [113] whereas other problems do not admit polynomial-size problem kernels even when parameterized by the "vertex cover number" [18, 61, 136]. For a specific problem it is hence interesting to investigate for which "tree-like" parameters it admits polynomial-size problem kernels.

Kernelizations for Nonstandard Parameters. A so far hardly explored research direction in the field of kernelization is the investigation of polynomial-size problem kernels for nonstandard parameterizations. This comprises the study of refined parameters as well as of combined parameters. In particular, we are only aware of two recent works [113, 155] that explicitly deal with polynomial-size problem kernels for refined parameters (see Section 2.3). We discuss two open questions arising from this work. In Chapter 4, we investigated the parameterized complexity of Cluster Editing and Cluster Deletion with respect to the refined parameter "cluster vertex deletion number"c. We left open the existence of polynomial-size kernelizations for these two problems. Very recently, Jansen [112] showed that there is little hope for polynomial-size problem kernels for Cluster Deletion parameterized by the "cluster vertex deletion number". For Cluster Editing an analogous result seems plausible but is still open. In case that a refined parameter does not lead to a polynomial-size problem kernel, it still might be useful to obtain a polynomial-size problem kernel with respect to a combined parameter in the spirit of a multivariate algorithmics analysis. For example, in case of Cluster Editing or Cluster Deletion we have identified the maximum number t of edge modifications involving a vertex as an interesting parameter (see Section 4.3). However Cluster Editing or Cluster Deletion have turned out to be NP-hard even for constant values of t. Altogether, this motivates the investigation whether (d,t)-Constrained-Cluster Editing and (d,t)-Constrained-Cluster Deletion (Definition 4.2) admit polynomial-size problem kernels for the combined parameter (c,t).

Bibliography

[1] F. N. Abu-Khzam. A kernelization algorithm for d-hitting set. *Journal of Computer and System Sciences*, 76(7):524–531, 2009. Cited on p. 25.

[2] F. N. Abu-Khzam and H. Fernau. Kernels: Annotated, proper and induced. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC '06)*, volume 4169 of *LNCS*, pages 264–275. Springer, 2006. Cited on p. 25.

[3] R. Agarwala, V. Bafna, M. Farach, B. Narayanan, M. Paterson, and M. Thorup. On the approximability of numerical taxonomy (fitting distances by tree matrices). *SIAM Journal on Computing*, 28(3):1073–1085, 1999. Cited on p. 84.

[4] N. Ailon, N. Avigdor-Elgrabli, and E. Liberty. An improved algorithm for bipartite correlation clustering. *CoRR*, abs/1012.3011, 2010. Cited on pp. 35 and 80.

[5] N. Ailon and M. Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS '05)*, pages 73–82. IEEE Computer Society, 2005. Cited on pp. 21, 24, 81, 82, 83, 84, 85, 100, and 106.

[6] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5), 2008. Cited on pp. 34 and 84.

[7] N. Alon and S. Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009. Cited on p. 13.

[8] E. Althaus, S. Canzar, C. Ehrler, M. R. Emmett, A. Karrenbauer, A. G. Marshall, A. Meyer-Bäse, J. D. Tipton, and H. Zhang. Computing H/D-exchange rates of single residues from data of proteolytic fragments. *BMC Bioinformatics*, 11(424), 2010. Cited on pp. 133, 134, 135, 137, 139, 147, 151, 152, 154, and 155.

[9] E. Althaus, S. Canzar, K. Elbassioni, A. Karrenbauer, and J. Mestre. Approximation algorithms for the interval constrained coloring problem. *Algorithmica*. Available online. Cited on pp. 133, 134, 135, 136, 137, 139, 141, and 155.

[10] V. Bafna, D. Gusfield, S. Hannenhalli, and S. Yooseph. A note on efficient computation of haplotypes via perfect phylogeny. *Journal of Computational Biology*, 11(5):858–866, 2004. Cited on p. 166.

[11] M. S. Bansal, J. Dong, and D. Fernández-Baca. Comparing and aggregating partially resolved trees. In *Proceedings of the 8th Latin American Symposium on Theoretical Informatics (LATIN '08)*, volume 4957 of *LNCS*, pages 72–83. Springer, 2008. Cited on p. 130.

[12] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1–3):89–113, 2004. Cited on pp. 20, 33, 34, 35, 50, and 84.

[13] J.-P. Barthélemy and F. Brucker. NP-hard approximation problems in overlapping clustering. *Journal of Classification*, 18(2):159–183, 2001. Cited on pp. 55, 58, 60, and 106.

[14] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005. Cited on p. 151.

[15] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–292, 1999. Cited on pp. 20, 33, 34, and 84.

[16] S. Bessy, F. V. Fomin, S. Gaspers, C. Paul, A. Perez, S. Saurabh, and S. Thomassé. Kernels for feedback arc set in tournaments. *Journal of Computer and System Sciences*, 2011. Available online. Cited on p. 25.

[17] S. Bessy, C. Paul, and A. Perez. Polynomial kernels for 3-leaf power graph modification problems. *Discrete Applied Mathematics*, 158(16):1732–1744, 2010. Cited on pp. 25, 30, 31, and 32.

[18] N. Betzler, R. Bredereck, R. Niedermeier, and J. Uhlmann. On making a distinguished vertex minimum degree by vertex deletion. In *Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*, volume 6543 of *LNCS*, pages 123–134. Springer, 2011. Cited on pp. iii and 172.

[19] N. Betzler, J. Guo, C. Komusiewicz, and R. Niedermeier. Average parameterization and partial kernelization for computing medians. *Journal of Computer and System Sciences*, 77(4):774–789, 2011. Cited on p. 130.

[20] N. Betzler, R. Niedermeier, and J. Uhlmann. Tree decompositions of graphs: Saving memory in dynamic programming. *Discrete Optimization*, 3(3):220–229, 2006. Cited on p. iii.

[21] N. Betzler and J. Uhlmann. Parameterized complexity of candidate control in elections and related digraph problems. *Theoretical Computer Science*, 410(52):5425–5442, 2009. Cited on p. iii.

[22] R. van Bevern, H. Moser, and R. Niedermeier. Approximation and tidying—a problem kernel for s-plex cluster vertex deletion. *Algorithmica*, 2011. Available online. Cited on p. 79.

[23] O. Bininda-Emonds, editor. *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*. Kluwer Academic, 2004. Cited on p. 109.

[24] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009. Cited on pp. 20, 23, 25, 35, 84, and 105.

[25] S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011. Cited on pp. 20, 23, 34, 35, 36, 79, and 84.

[26] S. Böcker, Q. B. A. Bui, F. Nicolas, and A. Truss. Tree compatibility is easy, flip supertree is not. Unpublished manuscript. Cited on pp. 108 and 130.

[27] S. Böcker, Q. B. A. Bui, and A. Truss. Improved fixed-parameter algorithms for minimum-flip consensus trees. *ACM Transactions on Algorithms*, 2009. Accepted for publication. Cited on pp. 6, 24, 107, 108, 110, 123, 124, 125, 126, and 127.

[28] S. Böcker and P. Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011. Cited on pp. 20, 23, 35, and 84.

[29] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *LNCS*, pages 17–37. Springer, 2009. Cited on pp. 4 and 13.

[30] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. Cited on pp. 14, 166, and 172.

[31] H. L. Bodlaender, M. R. Fellows, P. Heggernes, F. Mancini, C. Papadopoulos, and F. A. Rosamond. Clustering with partial information. *Theoretical Computer Science*, 411(7-9):1202–1211, 2010. Cited on pp. 35 and 130.

[32] H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA '09)*, volume 5757 of *LNCS*, pages 635–646. Springer, 2009. Cited on p. 14.

[33] D. Brügmann, C. Komusiewicz, and H. Moser. On generating triangle-free graphs. *Electronic Notes in Discrete Mathematics*, 32:51–58, 2009. Cited on p. 25.

[34] P. Burzyn, F. Bonomo, and G. Durán. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006. Cited on p. 25.

[35] J. Byrka, A. Karrenbauer, and L. Sanità. The interval constrained 3-coloring problem. In *Proceedings of the 9th Latin American Theoretical Informatics Symposium (LATIN'10)*, volume 6034 of *LNCS*, pages 591–602. Springer, 2010. Cited on pp. 134, 135, 136, 137, 139, and 151.

[36] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. Cited on p. 62.

[37] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997. Cited on p. 13.

[38] S. Canzar, K. Elbassioni, and J. Mestre. A polynomial delay algorithm for enumerating approximate solutions to the interval constrained coloring problem. In *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10)*, pages 23–33. SIAM, 2010. Cited on pp. 134, 153, and 154.

[39] S. Canzar, K. M. Elbassioni, A. Elmasry, and R. Raman. On the approximability of the maximum interval constrained coloring problem. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC '10)*, volume 6507 of *LNCS*, pages 168–179. Springer, 2010. Cited on pp. 134 and 155.

[40] Y. Cao and J. Chen. Cluster editing: Kernelization based on edge cuts. In *Proceedings of the 5th International Symposium on Parameterized and Exact Computation (IPEC '10)*, volume 6478 of *LNCS*, pages 60–71. Springer, 2010. Cited on p. 25.

[41] D. Catanzaro and M. Labbé. The pure parsimony haplotyping problem: Overview and computational advances. *International Transactions in Operational Research*, 16(5):561–584, 2009. Cited on pp. 157 and 158.

[42] J. Chang, T. Erlebach, R. Gailis, and S. Khuller. Broadcast scheduling: Algorithms and complexity. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA '08)*, pages 473–482. ACM-SIAM, 2008. Cited on p. 136.

[43] M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005. Cited on p. 34.

[44] D. Chen, O. Eulenstein, D. Fernández-Baca, and J. G. Burleigh. Improved heuristics for minimum-flip supertree construction. *Evolutionary Bioinformatics*, 2:347–356, 2006. Cited on p. 108.

[45] D. Chen, O. Eulenstein, D. Fernández-Baca, and M. Sanderson. Minimum-flip supertrees: Complexity and algorithms. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):165–173, 2006. Preliminary version presented at *COCOON '02*. Cited on pp. 21, 24, 107, 108, 111, 120, and 130.

[46] J. Chen and J. Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 2011. Available online. Cited on pp. 20, 23, 25, 35, 84, and 110.

[47] Z.-Z. Chen, T. Jiang, and G. Lin. Computing phylogenetic roots with bounded degrees and errors. *SIAM Journal on Computing*, 32(4):864–879, 2003. Cited on pp. 34 and 50.

[48] E. J. Chesler, L. Lu, S. Shou, Y. Qu, J. Gu, J. Wang, H. C. Hsu, J. D. Mountz, N. E. Baldwin, M. A. Langston, D. W. Threadgill, K. F. Manly, and R. W. Williams. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. *Nature Genetics*, 37(3):233–242, 2005. Cited on p. 36.

[49] M. Chimani, S. Rahmann, and S. Böcker. Exact ILP solutions for phylogenetic minimum flip problems. In *Proceedings of the 1st ACM International Conference On Bioinformatics and Computational Biology (ACM-BCB '10)*, pages 147–153. ACM, 2010. Cited on p. 109.

[50] F. Cicalese and M. Milanič. On parsimony haplotyping. Technical Report 2008-04, Universität Bielefeld, 2008. Cited on pp. 159 and 165.

[51] P. Damaschke. Incremental haplotype inference, phylogeny and almost bipartite graphs. In *2nd RECOMB Satellite Workshop on Computational Methods for SNPs and Haplotypes*, pages 1–11, 2004. See `http://www.cse.chalmers.se/~ptr/haploincrj.pdf` for an extended version. Cited on p. 161.

[52] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006. Cited on pp. 110 and 129.

[53] P. Damaschke. Bounded-degree techniques accelerate some parameterized graph algorithms. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *LNCS*, pages 98–109. Springer, 2009. Cited on pp. 20, 23, and 35.

[54] P. Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory of Computing Systems*, 46(2):261–283, 2010. Cited on pp. 20, 35, 36, and 55.

[55] S. Dasgupta and P. M. Long. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555–569, 2005. Cited on pp. 21 and 81.

[56] W. H. E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49(4):461–467, 1987. Cited on p. 84.

[57] F. Dehne, M. A. Langston, X. Luo, S. Pitre, P. Shaw, and Y. Zhang. The cluster editing problem: Implementations and experiments. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC '06)*, volume 4169 of *LNCS*, pages 13–24. Springer, 2006. Cited on pp. 20, 23, 35, 53, 78, 79, and 84.

[58] E. D. Demaine, M. Hajiaghayi, and D. Marx. Open problems – parameterized complexity and approximation algorithms. In *Parameterized Complexity and Approximation Algorithms*, volume 09511 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010. Cited on pp. 5, 36, and 37.

[59] J. Díaz and D. M. Thilikos. Fast fpt-algorithms for cleaning grids. In *Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science (STACS '06)*, LNCS, pages 361–371, 2006. Cited on p. 25.

[60] R. Diestel. *Graph Theory*. Springer, 3rd edition, 2005. Cited on p. 15.

[61] M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and IDs. In *Proceedings of the 36th International Colloquium on Automata, Languages, and Programming (ICALP '09)*, volume 5555 of *LNCS*, pages 378–389. Springer, 2009. Cited on pp. 14 and 172.

[62] B. Dorn, F. Hüffner, D. Krüger, R. Niedermeier, and J. Uhlmann. Exploiting bounded signal flow for graph orientation based on cause-effect pairs. In *Proceedings of the 1st International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems (TAPAS '11)*, volume 6595 of *LNCS*, pages 104–115. Springer, 2011. Long version to appear in *Algorithms for Molecular Biology*. Cited on p. iii.

[63] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. Cited on pp. 3, 10, 11, 62, and 63.

[64] R. G. Downey, M. R. Fellows, and M. A. Langston. The computer journal special issue on parameterized complexity: Foreword by the guest editors. *The Computer Journal*, 51(1):1–6, 2008. Cited on p. 10.

[65] M. Elberfeld and T. Tantau. Phylogeny- and parsimony-based haplotype inference with constraints. In *Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching (CPM '10)*, volume 6129 of *LNCS*, pages 177–189. Springer, 2010. Cited on pp. 158, 159, and 166.

[66] G. F. Estabrook and F. R. McMorris. When is one estimate of evolutionary relationships a refinement of another? *Journal of Mathematical Biology*, 10:367–373, 1980. Cited on p. 108.

[67] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13:155–179, 1995. Cited on pp. 21, 83, and 84.

[68] M. R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC '06)*, volume 4169 of *LNCS*, pages 276–277. Springer, 2006. Cited on pp. 4 and 13.

[69] M. R. Fellows. Towards fully multivariate algorithmics: Some new results and directions in parameter ecology. In *Proceedings of the 20th International Workshop on Combinatorial Algorithms (IWOCA'09)*, volume 5874 of *LNCS*, pages 2–10. Springer, 2009. Cited on pp. 4, 11, and 12.

[70] M. R. Fellows, J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Graph-based data clustering with overlaps. In *Proceedings of the 15th Annual International Computing and Combinatorics Conference (COCOON '09)*, volume 5609 of *LNCS*, pages 516–526. Springer, 2009. Cited on p. iv.

[71] M. R. Fellows, J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011. Cited on pp. iv, 5, 25, and 36.

[72] M. R. Fellows, T. Hartman, D. Hermelin, G. M. Landau, F. Rosamond, and L. Rozenberg. Haplotype inference constrained by plausible haplotype data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2010. Available online. Cited on pp. 7, 158, 159, and 160.

[73] M. R. Fellows, M. A. Langston, F. A. Rosamond, and P. Shaw. Efficient parameterized preprocessing for Cluster Editing. In *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory (FCT '07)*, volume 4639 of *LNCS*, pages 312–321. Springer, 2007. Cited on pp. 20, 23, 25, 35, and 84.

[74] M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC '08)*, volume 5369 of *LNCS*, pages 294–305. Springer, 2008. Cited on pp. 137 and 172.

[75] J. Fiala, P. A. Golovach, and J. Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theoretical Computer Science*, 412(23):2513–2523, 2011. Cited on pp. 137 and 172.

[76] R. Fleischer, J. Guo, R. Niedermeier, J. Uhlmann, Y. Wang, M. Weller, and X. Wu. Extended islands of tractability for parsimony haplotyping. In *Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching (CPM '10)*, volume 6129, pages 214–226. Springer, 2010. Cited on pp. v, 7, 131, and 165.

[77] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. Cited on pp. 3, 10, and 11.

[78] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011. Cited on pp. 14 and 166.

[79] A. Frank and É. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. Cited on p. 137.

[80] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987. Cited on p. 16.

[81] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979. Cited on p. 9.

[82] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004. Cited on p. 35.

[83] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005. Cited on pp. 20, 23, 24, 25, 35, 51, 84, and 90.

[84] J. Gramm, A. Nickelsen, and T. Tantau. Fixed-parameter algorithms in phylogenetics. *The Computer Journal*, 51(1):79–101, 2008. Cited on pp. 109 and 129.

[85] J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for Closest String and related problems. *Algorithmica*, 37(1):25–42, 2003. Cited on p. 137.

[86] D. L. Greenwell, R. L. Hemminger, and J. B. Klerlein. Forbidden subgraphs. In *Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 389–394. Utilitas Mathematica, 1973. Cited on p. 26.

[87] S. Guillemot, C. Paul, and A. Perez. On the (non-)existence of polynomial kernels for P_l-free edge modification problems. In *Proceedings of the 5th International Symposium on Parameterized and Exact Computation (IPEC '10)*, volume 6478 of *LNCS*, pages 147–157. Springer, 2010. Cited on pp. 79, 110, and 129.

[88] J. Guo. Problem kernels for NP-complete edge deletion problems: split and related graphs. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC '07)*, volume 4835 of *LNCS*, pages 915–926. Springer, 2007. Cited on pp. 25 and 171.

[89] J. Guo. A more effective linear kernelization for Cluster Editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009. Cited on pp. 16, 20, 23, 24, 25, 32, 35, 84, 92, and 93.

[90] J. Guo, S. Hartung, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Exact algorithms and experiments for hierarchical tree clustering. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*, pages 457–462, 2010. Cited on pp. iv and 6.

[91] J. Guo, S. Hartung, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Data reduction, exact algorithms, and experiments for hierarchical tree clustering. Manuscript. Submitted to the *Journal of Classification*, 2011. Cited on p. iv.

[92] J. Guo, F. Hüffner, E. Kenar, R. Niedermeier, and J. Uhlmann. Complexity and exact algorithms for vertex multicut in interval and bounded treewidth graphs. *European Journal of Operational Research*, 186(2):542–553, 2008. Cited on p. iii.

[93] J. Guo, F. Hüffner, C. Komusiewicz, and Y. Zhang. Improved algorithms for bicluster editing. In *Proceedings of the 5th Annual Conference on Theory and Applications of Models of Computation (TAMC '08)*, volume 4978 of *LNCS*. Springer, 2008. Cited on pp. 25, 35, 80, and 110.

[94] J. Guo, F. Hüffner, and R. Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *LNCS*, pages 162–173. Springer, 2004. Cited on p. 166.

[95] J. Guo, I. A. Kanj, C. Komusiewicz, and J. Uhlmann. Editing graphs into disjoint unions of dense clusters. *Algorithmica*, 2011. Available online. Cited on pp. iii and 36.

[96] J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM Journal on Discrete Mathematics*, 24(4):1662–1683, 2010. Cited on pp. iii, 25, 36, and 54.

[97] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007. Cited on pp. 4 and 13.

[98] J. Guo, R. Niedermeier, and J. Uhlmann. Two fixed-parameter algorithms for vertex covering by paths on trees. *Information Processing Letters*, 106(2):81–86, 2008. Cited on p. iii.

[99] J. Guo and J. Uhlmann. Kernelization and complexity results for connectivity augmentation problems. *Networks*, 56(2):131–142, 2010. Cited on pp. iii and 25.

[100] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991. Cited on pp. 107, 108, 111, and 120.

[101] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997. Cited on p. 84.

[102] D. Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. In *Proceedings of the 6th Annual International Conference on Computational Molecular Biology (RECOMB '02)*, pages 166–175, 2002. Cited on p. 166.

[103] D. Gusfield and S. H. Orzack. Haplotype inference. CRC Handbook on Bioinformatics, chapter 1, pages 1–25. CRC Press, 2005. Cited on p. 157.

[104] B. Harb, S. Kannan, and A. McGregor. Approximating the best-fit tree under L_p norms. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX '05)*, volume 3624 of *LNCS*, pages 123–133. Springer, 2005. Cited on pp. 84 and 86.

[105] J. Hartigan. Representation of similarity matrices by trees. *Journal of the American Statistical Assoication*, 62(320):1140–1158, 1967. Cited on p. 82.

[106] J. Hartigan. Statistical theory in clustering. *Journal of Classification*, 2(1):63–76, 1985. Cited on pp. 21 and 81.

[107] P. Heggernes, D. Lokshtanov, J. Nederlof, C. Paul, and J. A. Telle. Generalized graph clustering: recognizing (p, q)-cluster graphs. In *Proceedings of the 36th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '10)*, volume 6410 of *LNCS*, pages 171–183. Springer, 2010. Cited on pp. 36, 52, and 54.

[108] W. Hsu and T. Ma. Substitution decomposition on chordal graphs and applications. In *Proceedings of the 2nd International Symposium on Algorithms (ISA'91)*, volume 557 of *LNCS*, pages 52–60. Springer, 1991. Cited on pp. 16 and 29.

[109] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010. Cited on pp. 35, 49, and 79.

[110] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 51(1):7–25, 2008. Cited on p. 13.

[111] L. van Iersel, J. Keijsper, S. Kelk, and L. Stougie. Shorelines of islands of tractability: Algorithms for parsimony and minimum perfect phylogeny haplotyping problems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(2):301–312, 2008. Cited on pp. 158, 159, and 165.

[112] B. Jansen, Febuary 2011. Personal Communication. Cited on pp. 52 and 172.

[113] B. M. P. Jansen and H. L. Bodlaender. Vertex cover kernelization revisited: Upper and lower bounds for a refined parameter. In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS '11)*, Leibniz International Proceedings in Informatics (LIPIcs). IBFI Dagstuhl, Germany, 2011. To appear. Cited on pp. 12 and 172.

[114] C. J. Jardine, N. Jardine, and R. Sibson. The structure and construction of taxonomic hierarchies. *Mathematical Biosciences*, 1(2):173–179, 1967. Cited on p. 82.

[115] N. Jardine and R. Sibson. *Mathematical Taxonomy*. Wiley, 1971. Cited on p. 106.

[116] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967. Cited on p. 82.

[117] D. Jungnickel. *Graphs, Networks and Algorithms*. Springer, 3rd edition, 2008. Cited on p. 15.

[118] T. Köhler. Studienarbeit, 2011. University of Jena. Cited on p. 79.

[119] C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Deconstructing intractability—a case study for interval constrained coloring. In *Proceedings of the 20th Annual Symposium on Combinatorial Pattern Matching (CPM '09)*, volume 5577 of *LNCS*, pages 207–220. Springer, 2009. Cited on p. iv.

[120] C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Deconstructing intractability—a multivariate complexity analysis of interval constrained coloring. *Journal of Discrete Algorithms*, 9(1):137–151, 2011. Cited on pp. iv, 6, and 131.

[121] C. Komusiewicz and J. Uhlmann. A cubic-vertex kernel for flip consensus tree. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008)*, volume 2 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 280–291. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2008. Cited on pp. iv, 6, and 25.

[122] C. Komusiewicz and J. Uhlmann. Alternative parameterizations for cluster editing. In *Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*, volume 6543 of *LNCS*, pages 344–355. Springer, 2011. Cited on pp. iv, 5, 50, and 51.

[123] C. Komusiewicz and J. Uhlmann. A cubic-vertex kernel for flip consensus tree. Manuscript. Submitted to *Algorithmica*, 2011. Cited on p. iv.

[124] S. Kratsch and M. Wahlström. Two edge modification problems without polynomial kernels. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *LNCS*, pages 264–275. Springer, 2009. Cited on pp. 14, 25, 79, 110, and 170.

[125] M. Křivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986. Cited on pp. 21, 34, 50, 60, 81, 82, and 84.

[126] G. Lancia, M. C. Pinotti, and R. Rizzi. Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. *INFORMS Journal on Computing*, 16(4):348–359, 2004. Cited on p. 158.

[127] G. Lancia and R. Rizzi. A polynomial case of the parsimony haplotyping problem. *Operations Research Letters*, 34:289–295, 2006. Cited on pp. 158 and 165.

[128] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983. Cited on pp. 137 and 155.

[129] D. Lokshtanov and D. Marx. Clustering with local restrictions. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP '11)*, volume 6755 of *LNCS*, pages 785–797. Springer, 2011. Cited on p. 36.

[130] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004. Cited on p. 80.

[131] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT '04)*, volume 3111 of *LNCS*, pages 260–272. Springer, 2004. Cited on p. 56.

[132] D. Marx and I. Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC '11)*. ACM Press. To appear. Cited on pp. 36, 52, and 130.

[133] R. M. McConnell and J. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 536–545. ACM/SIAM, 1994. Cited on pp. 29 and 118.

[134] S. Micali and V. V. Vazirani. An $O(\sqrt{(|V|)}|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS '80)*, pages 17–27. IEEE, 1980. Cited on p. 69.

[135] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113:109–128, 2001. Cited on p. 25.

[136] A. Nichterlein, R. Niedermeier, J. Uhlmann, and M. Weller. On tractable cases of target set selection. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC '10)*, volume 6507 of *LNCS*, pages 378–389. Springer, 2010. Cited on pp. iii and 172.

[137] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006. Cited on pp. 3, 10, 11, 12, 14, and 88.

[138] R. Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS '10)*, volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17–32. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010. Cited on pp. 4, 11, 12, 23, 130, and 170.

[139] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2000. Cited on p. 15.

[140] R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003. Cited on p. 25.

[141] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005. Cited on pp. 20, 53, and 78.

[142] C. M. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994. Cited on p. 9.

[143] I. Pe'er, T. Pupko, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *SIAM Journal on Computing*, 33(3):590–607, 2004. Cited on pp. 107, 111, and 120.

[144] R. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003. Cited on p. 60.

[145] F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, 44(1):91–104, 2009. Cited on pp. 20, 23, 25, 27, 29, 35, 80, 84, 105, 110, and 115.

[146] S. Rahmann, T. Wittkop, J. Baumbach, M. Martin, A. Truß, and S. Böcker. Exact and heuristic algorithms for weighted cluster editing. In *Proceedings of the 6th Annual Conference on Computational Systems Bioinformatics (CSB'07)*, pages 391–401. Imperial College Press, 2007. Cited on pp. 84 and 103.

[147] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1–2):131–147, 1981. Cited on p. 130.

[148] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. Cited on p. 20.

[149] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6:139–154, 1978. Cited on p. 36.

[150] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1–2):173–182, 2004. Cited on pp. 20, 27, 34, 37, 50, 60, and 84.

[151] R. Sharan, B. V. Halldórsson, and S. Istrail. Islands of tractability for parsimony haplotyping. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(3):303–311, 2006. Cited on pp. 7, 158, 159, 160, and 165.

[152] R. Sharan, A. Maron-Katz, and R. Shamir. CLICK and EXPANDER: a system for clustering and visualizing gene expression data. *Bioinformatics*, 19(14):1787–1799, 2003. Cited on pp. 20 and 33.

[153] M. Talmaciu and E. Nechita. Recognition algorithm for diamond-free graphs. *Informatica*, 18(3):457–462, 2007. Cited on p. 58.

[154] A. Tanay, R. Sharan, and R. Shamir. Biclustering algorithms: A survey. In S. Aluru, editor, *Handbook of Computational Molecular Biology*, pages 26.1–26.17. Chapman Hall/CRC Press, 2006. Cited on p. 80.

[155] J. Uhlmann and M. Weller. Two-layer planarization parameterized by feedback edge set. In *Proceedings of the 7th Annual Conference on Theory and Applications of Models of Computation (TAMC '10)*, volume 6108 of *LNCS*, pages 431–442. Springer, 2010. Cited on pp. iii, 12, 25, and 172.

[156] M. Weller, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. On making directed graphs transitive. In *Proceedings of the 11th International Symposium on Algorithms and Data Structures (WADS '09)*, volume 5664 of *LNCS*, pages 542–553. Springer, 2009. Long version to appear in the *Journal of Computer and System Sciences*. Cited on pp. iii, 25, and 50.

[157] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, 2001. Cited on p. 15.

[158] T. Wittkop, J. Baumbach, F. P. Lobo, and S. Rahmann. Large scale clustering of protein sequences with FORCE – a layout based heuristic for weighted cluster editing. *BMC Bioinformatics*, 8(1):396, 2007. Cited on pp. 33 and 34.

[159] T. Wittkop, D. Emig, S. Lange, S. Rahmann, M. Albrecht, J. H. Morris, S. Böcker, J. Stoye, and J. Baumbach. Partitioning biological data with transitivity clustering. *Nature Methods*, 7(6):419–420, 2010. Cited on p. 34.

[160] T. Wittkop, D. Emig, A. Truss, M. Albrecht, S. Böcker, and J. Baumbach. Comprehensive cluster analysis with transitivity clustering. *Nature Protocols*, 6:285–295, 2011. Cited on p. 34.

[161] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993. Cited on p. 20.

[162] R. Xu and D. Wunsch II. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005. Cited on p. 20.

[163] M. Yannakakis. Edge-deletion problems. *SIAM Journal on Computing*, 10(2):297–309, 1981. Cited on pp. 25, 110, and 111.

[164] C. T. Zahn, Jr. Approximating symmetric relations by equivalence relations. *Journal of the Society for Industrial and Applied Mathematics*, 12(4):840–847, 1964. Cited on pp. 33 and 34.

[165] H. Zha, X. He, C. H. Q. Ding, M. Gu, and H. D. Simon. Bipartite graph partitioning and data clustering. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management (CIKM'01)*, pages 25–32. ACM, 2001. Cited on p. 80.

[166] A. van Zuylen and D. P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research*, 34:594–620, 2009. Cited on pp. 34 and 84.