Parameterized Algorithmics for Network Analysis: Clustering & Querying

Vorgelegt von
Diplom-Bioinformatiker Christian Komusiewicz

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. Stephan Kreutzer
Gutachter:	Prof. Dr. Rolf Niedermeier
Gutachter:	Prof. Dr. Sebastian Böcker
Gutachter:	Prof. Dr. Guillaume Fertin

Tag der wissenschaftlichen Aussprache: 14. Juli 2011

Berlin 2011
D83

ISBN: 978-3-7983-2379-7 (Druckausgabe)
ISBN: 978-3-7983-2380-3 (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 summarizes some of my results on NP-hard graph problems that have applications in the areas of network clustering and querying. The research for obtaining these results was performed from June 2007 to April 2011. Until December 2010, I was with the Friedrich-Schiller-Universität Jena as a member of the chair for Theoretical Computer Science/Complexity Theory then held by Rolf Niedermeier. From June 2007 until May 2010, I was supported by a PhD fellowship of the Carl-Zeiss-Stiftung; from June 2010 until December 2010, I received funding from the Deutsche Forschungsgemeinschaft (DFG), as a researcher in the DFG project "Parameterized Algorithmics for Bioinformatics" (PABI, NI 369/7). In January 2011, I joined the chair for Algorithmics and Complexity Theory at TU Berlin held by Rolf Niedermeier, who, in the meantime, had moved from Friedrich-Schiller-Universität Jena to TU Berlin. Currently, I am supported by the TU Berlin.

I want to express my gratitude to Rolf Niedermeier for giving me the opportunity to work in his group and for his advice and support that eventually led to this thesis. Furthermore, I want to thank my colleagues and former colleagues Nadja Betzler, René van Bevern, Robert Bredereck, Jiehua Chen, Michael Dom, Jiong Guo, Sepp Hartung, Falk Hüffner, Hannes Moser, André Nichterlein, Rolf Niedermeier, Manuel Sorge, Johannes Uhlmann, and Mathias Weller for creating an enjoyable working atmosphere and for many inspiring and instructive discussions. Moreover, I owe sincere thanks to my coauthors Nadja Betzler, René van Bevern, Daniel Brügmann, Michael R. Fellows, Jiong Guo, Sepp Hartung, Falk Hüffner, Iyad A. Kanj, Hannes Moser, Rolf Niedermeier, Alexander Schäfer, Johannes Uhlmann, and Mathias Weller for the pleasant and productive cooperation. In particular, I would like to thank René van Bevern for his implementation of the algorithms in Chapter 6. Finally, I want to thank the anonymous referees of several journals and scientific conferences for many pieces of advice that helped improving this work.

The results in this thesis are partially contained in journal and conference publications that were created in close collaboration with coauthors. Below, I will describe which publications contributed to which chapters, and I will also my specify my contributions to these publications. Further work to which I have contributed but that is not part of this thesis is concerned with parameterized algorithmics for graph modification problems [van Bevern et al. 2010, Brügmann et al. 2009, Komusiewicz and Uhlmann 2008, Weller et al. 2011], computational mass spectrometry [Komusiewicz et al. 2011], and for further problems in in graph-based data clustering [Fellows et al. 2011b, Guo et al. 2008, 2010a, Hüffner et al. 2009, 2010, Komusiewicz et al. 2009, Schäfer et al. 2011]. The latter collection of publications deals with extensions of the classical clustering notion, for example with hierarchical clusterings. In this work, the focus is on "classical" clusterings, that is, partitions of a set of objects.

Part II: Clustering. Chapter 2 is based on parts of the publication "Alternative Parameterizations for Cluster Editing", which appeared in the proceedings of the *37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)* [Komusiewicz and Uhlmann 2011]; a full version of this publication is in preparation. I proposed to study the parameter "local modification bound", participated in the development of the reduction from 3-SAT to CLUSTER EDITING, and observed the connections between CLUSTER DELETION and PARTITION INTO TRIANGLES. Furthermore, I developed the kernelization algorithms.

Chapter 3 is based on the two publications "A More Relaxed Model for Graph-Based Data Clustering: s-Plex Cluster Editing", which appeared in *SIAM Journal on Discrete Mathematics* [Guo et al. 2010b], and "Editing Graphs Into Disjoint Unions of Dense Clusters", to appear in *Algorithmica* [Guo et al. 2011]. Preliminary versions of these publications appeared in the proceedings of the *5th International Conference on Algorithmic Aspects in Information and Management (AAIM '09)* [Guo et al. 2009b], and the *20th International Symposium on Algorithms and Computation (ISAAC '09)* [Guo et al. 2009a]. I devised the search tree algorithm for s-PLEX-CLUSTER EDITING, the data reduction approach for AVERAGE-s-PLEX-CLUSTER EDITING, and polished the W[1]-hardness proof for μ-CLIQUE-CLUSTER EDITING. The presented forbidden subgraph characterizations were developed jointly by all coauthors.

Chapter 4 is based on the publication "Average Parameterization and Partial Kernelization for Computing Medians", which appeared in the *Journal of Computer and System Sciences* [Betzler et al. 2011b]; a preliminary version appeared in the proceedings of the *9th Latin American Theoretical Informatics Symposium (LATIN '10)* [Betzler et al. 2010b]. Jiong Guo developed the framework for obtaining fixed-parameter algorithms for the parameter "average distance" and proposed to study CONSENSUS CLUSTERING in this context; I developed the presented structural kernelizations. The definition of structural kernelization was developed jointly by all coauthors.

Part III: Querying. Chapters 5 and 7 are based on so far unpublished results that were all obtained autonomously. Chapter 6 is based on the publication "Parameterized Algorithmics for Finding Connected Motifs in Biological Networks", which appeared in the *IEEE/ACM Transactions on Computational Biology and Bioinformatics* [Betzler et al. 2011a]; a preliminary version appeared in the proceedings of the *19th Annual Symposium on Combinatorial Pattern Matching (CPM '08)* [Betzler et al. 2008]. I developed most of the parameterized algorithms and hardness results for LIST-COLORED GRAPH MOTIF and supervised the algorithm implementation. Furthermore, I simplified and extended the initial W[1]-hardness proof for BICONNECTED GRAPH MOTIF.

Berlin, September 2011 Christian Komusiewicz

Contents

Part I

Introduction

Networks are universal models for describing the behavior of complex systems. The analysis of networks has, for this reason, become a ubiquitous tool in the modern sciences [Barabási 2002]. Many of the systems which are described by such network models emerge from the interaction of thousands of protagonists: the world wide web has billions of pages, social networks have up to hundreds of millions of members, protein interaction networks have thousands of proteins and tens of thousands of interactions. The analysis of these networks therefore has to make use of algorithms. In computer science and mathematics, networks are commonly referred to as *graphs*. Researchers have been engaged in the study of algorithms for graphs long before the emergence of computer science, and the field of graph algorithms continues to be an important area of computer science. As graphs or networks are models for complex systems it comes as no surprise that many computational problems arising in graphs turn out to be computationally hard, in particular NP-hard. This means that there is good evidence that, in general, one cannot solve these problems efficiently because the running time of any algorithm that solves this problem grows super-polynomially with the size of the input [Garey and Johnson 1979]. For instances with thousands of nodes, algorithms whose running times are super-polynomial running in the input size are definitely infeasible. The importance of network analysis is therefore contrasted by the large amount of computational resources that is needed for performing this analysis.

Several approaches have been proposed to cope with the NP-hardness of computational problems: heuristics (cf. [Michalewicz and Fogel 2004]) either drop the demand of finding an exact solution or abandon the aim of providing a bound on the worst-case running time; approximation algorithms (cf. [Vazirani 2001]) drop the demand of finding an optimal solution but are able to find in polynomial time a provably good solution. In this work, we follow the approach of *parameterized algorithmics* [Downey and Fellows 1999, Flum and Grohe 2006, Niedermeier 2006]. The aim of parameterized algorithmics is to identify structural properties of input instances that strongly influence the complexity of a computational problem at hand. Whereas in classical complexity theory one measures the complexity of a problem with respect to the overall size of an input instance, parameterized algorithmics makes use of an additional *parameter*. The hope is that for instances in which this parameter is small we can devise efficient algorithms that exactly solve the problem. The parameterized approach to NP-hard problems is not only theoretically interesting but it can also lead to practical algorithms for solving NP-hard problems. The reason is that the input instances that one faces usually *do* exhibit a certain structure. Social networks, road networks, and protein interaction networks for example are all *sparse*, that is, the overall number of pairwise interactions or connections in such networks is much smaller than the maximum possible number of interactions or connections. A running time analysis that only uses the overall input size as measure is not fully capable of reflecting how much the sparseness of a graph may help in solving a

problem. In contrast, in parameterized algorithmics one can elegantly measure the running time of algorithms as a function of the input size *and* some sparseness *parameter*. Such a sparseness parameter is for example the maximum degree of a network. In ideal cases, one may find for a particular application scenario a parameter that is small in all the instances that we may encounter *and* an algorithm that is fast in case this parameter is small.

We study the parameterized complexity of computational problems that have applications in two areas of network analysis: *clustering* and *querying*. As network analysis is a universal paradigm for understanding complex systems, the problems and algorithms considered in this work are not limited to a particular application scenario. As a running example, however, we will often refer to applications in the analysis of protein interaction networks. Protein interaction networks are graphs whose vertices represent the proteins of a cell. Two vertices of the graph are connected by an edge if a physical interaction between them has been experimentally detected [Barabási and Oltvai 2004]. Clustering and querying are natural approaches to organize and explore the information that is contained in a protein interaction network.

Network clustering is the task of assigning the vertices of the network into groups such that the groups contain network vertices that are similar to each other. These groups are often referred to as *communities* [Palla et al. 2005] or *modules* [Przulj et al. 2004, Sharan et al. 2007] of networks. In the case of protein interaction networks, modules are sets of proteins that perform a common biological task, for example, proteins that play a role in the regulation of the cell cycle. The clustering of a protein interaction network then aims at partitioning the protein set into modules. These modules can then be used for instance to predict the function of proteins [Sharan et al. 2007].

Network clustering is a *global* analysis task that aims at uncovering the structure of a network. In contrast, the querying of network searches for *local* structures in a network that have certain properties. One example for a querying problem in protein interaction networks is the identification of signaling pathways that are similar to other known signaling pathways [Scott et al. 2006, Shlomi et al. 2006, Hüffner et al. 2007]. The input of such a problem consists of two parts. One part is a path, that is, a linear sequence of interacting proteins. The other part is a complete network , and the task is to find, if it exists, a path in the network that is similar to the given path.

The aim of this work is to make progress toward solving NP-hard computational problems from the areas of network clustering and querying. In particular, we aim at identifying interesting parameters that may be useful for a wide range of computational problems in network analysis, not only restricted to the problems considered in this work. The results in this work are somehow asymmetric in the sense that for clustering our results are so far mostly of theoretical interest, whereas for querying we also provide implementations for some of our algorithms. In our opinion,

one reason for this "gap" is that, because querying is a local analysis task, for querying problems it is easier to algorithmically exploit local—and thus small—parameters.

In the remainder of this introductory part, we briefly overview the algorithmic and graph-theoretic concepts that are employed in this work.

Chapter 1

Preliminaries

In this chapter, we give an overview of the central concepts and techniques of parameterized algorithmics that we make use of. We also summarize the graph-theoretic notions that are featured throughout this work.

1.1 NP-hard Problems

Most of the computational problems in this work are formulated as decision problems. Formally, a decision problem is defined by a language $L \subseteq \Sigma^*$ where Σ is a finite alphabet. The task is to decide whether for a given input $x \in \Sigma^*$ we have $x \in L$ or $x \notin L$. We refer to instances with $x \in L$ as *yes-instances* of L. In this work, we are only concerned with decidable problems, that is, problems for which there is an algorithm that always terminates. A central property of any decidable problem is the amount of computational resources that is needed to solve this problem. These resources are usually referred to as *complexity measures*. In this work, we are concerned with the standard complexity measure of running time. In standard complexity theory, the running time of an algorithm that decides a problem L is measured as a function of the input size.

The two most important classes of decidable decision problems are P and NP. The class P contains the problems that can be decided in polynomial time by a *deterministic* Turing machine. The class NP contains the problems that can be decided in polynomial time by a *nondeterministic* Turing machine. A widely believed conjecture in computer science is that there are problems in NP that can *not* be solved in polynomial time by a deterministic algorithm. This is known as the famous P≠NP conjecture. The most important class of these problems is the class of NP-*hard* problems which is is defined by *polynomial-time reductions* between problems. A problem $A \subseteq \Sigma^*$ reduces to a problem $B \subseteq \Sigma^*$ (abbreviated as $A \leq_p B$) if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in A$ if and only if $f(x) \in B$. This means that problem B is at least as hard as problem A with respect to the

notion of polynomial-time solvability. A problem B is NP-hard if for all problems $A \in$ NP we have $A \leq_p B$. An NP-hard problem is thus at least as hard as any problem in NP, again with respect to polynomial-time solvability. An NP-hard problem that is also contained in NP is called NP-*complete*. For an introduction to the theory of NP-completeness and a still very comprehensive overview of NP-complete decision problems, we refer the reader to the book of Garey and Johnson [1979]. While most of the problems in this work are defined as decision problems, our algorithms are usually also capable of solving the natural optimization problem that corresponds to the decision problem (sometimes with a small running-time overhead).

Different ways to cope with NP-hard problems have been proposed. One approach that has been intensively studied in the field of theoretical computer science are *approximation algorithms* which are defined for optimization problems. Here, the aim is not to decide a problem but to find a solution that minimizes (maximizes) some objective function. A polynomial-time factor-c approximation is an algorithm that finds in polynomial time a solution whose objective function value is at most c times the optimum (at least c times the optimum). There are problems for which arbitrarily small approximation factors can be obtained. However, most of the problems considered in this work are hard for the classes APX or MaxSNP which means that, under plausible complexity theoretic assumptions, there is a constant c for which a factor-c approximation can *not* be obtained in polynomial time. For an introduction to approximation algorithms, we refer to Vazirani [2001].

1.2 Parameterized Algorithmics

The algorithmic approach to NP-hard problems that is followed in this work is known as *parameterized algorithmics*. In the following, we introduce the notions and techniques of parameterized algorithmics and the algorithmic techniques that we make use of. A more comprehensive introduction to the topic can be found in the monographs of Downey and Fellows [1999], Flum and Grohe [2006], and Niedermeier [2006].

Fixed-Parameter Tractability. The basic idea of fixed-parameter algorithms is to accept a super-polynomial running time, but to confine the running time "explosion" to a *parameter* k. To this end, parameterized decision problems consist of two components.

Definition 1.1. *A* parameterized problem *is a language* $L \subseteq \Sigma^* \times \Sigma^*$. *The second component is the* parameter.

The aim for a parameterized problem is to achieve a good running time in case the parameter is small, or "fixed". This aim is captured by the following central definition of parameterized algorithmics.

Definition 1.2. *A parameterized problem L is* fixed-parameter tractable *if there is a deterministic algorithm that decides for every input instance (I,k) of L in $f(k)\cdot\text{poly}(|I|)$ time whether $(I,k)\in L$, where f is a computable function only depending on k. The corresponding complexity class is called FPT.*

We call an algorithm with the running time $f(k)\cdot\text{poly}(|I|)$ a *fixed-parameter algorithm* for L. The notion of fixed-parameter tractability is sometimes confused with "polynomial time for constant k". Fixed-parameter tractability, however, makes a stronger demand: the degree of the polynomial function of $|I|$ must be completely independent from k. The class of fixed-parameter tractable problems is actually *contained* in the class of problems that are polynomial-time solvable for constant k. This class is called XP.

Definition 1.3. *A parameterized problem L is contained in the class XP if there is a deterministic algorithm that decides for every input instance (I,k) of L whether $(I,k)\in L$ in $f(k)\cdot|I|^{g(k)}$ time, where f and g are computable functions only depending on k.*

W[1]-hardness. Assuming P≠NP, it is clear that not all parameterized problems are fixed-parameter tractable, since otherwise parameterizing an NP-hard problem by a constant parameter would immediately lead to a deterministic polynomial-time algorithm for this problem. There are, however, also many problems in XP that have resisted all attempts at developing fixed-parameter algorithms for them. To show that a parameterized problem in XP is unlikely to admit fixed-parameter algorithms, Downey and Fellows [1995a,b, 1999] developed a theory of parameterized intractability by means of devising a completeness program with complexity classes. A basic level of (presumable) parameterized intractability is captured by the complexity class W[1]. There is good reason to believe that problems that are hard for W[1] are not fixed-parameter tractable. To show this stronger form of hardness, a reduction concept was introduced.

Definition 1.4. *A* parameterized reduction *reduces a problem instance (I,k) in $f(k)\cdot\text{poly}(|I|)$ time to an instance (I',k') such that (I,k) is a yes-instance if and only if (I',k') is a yes-instance and $k'=g(k)$, where g is a function only depending on k.*

If for a given parameterized problem L there is a W[1]-hard parameterized problem L' such that there is a parameterized reduction from L' to L, then L is also W[1]-hard. A problem is W[1]-complete if it is W[1]-hard and also contained in W[1]. Recent surveys discussing parameterized hardness were presented by Chen and Meng [2008] and Downey and Thilikos [2011].

After defining the basic concepts of parameterized algorithmics, we will describe a central algorithmic approach for obtaining fixed-parameter tractability results.

Data Reduction and Kernelization. Polynomial-time preprocessing is a practical approach to attack NP-hard problems. The hope is that the preprocessing reduces the original instance to a smaller one, which can then be solved by a deterministic algorithm that has super-polynomial running time. In the one-dimensional classical complexity theory there is no way to measure the effect of preprocessing: a polynomial-time algorithm that provably reduces an instance I of size $|I|$ of an NP-hard problem to an instance I' with $|I| > |I'|$ of the same problem implies a deterministic polynomial-time algorithm that solves an NP-hard problem: repeating this algorithm roughly $|I|$ times yields an instance of constant size.

For parameterized problems, one can circumvent this problem, since one has the parameter as secondary measurement. The idea is to show that if the instance size is much larger than some function of k, then one can apply a preprocessing that shrinks the instance. This is captured by the notion of *reduction to a problem kernel*, also called *problem kernelization*.

Definition 1.5. *Let $L \subseteq \Sigma^* \times \Sigma^*$ be a parameterized problem. A* reduction to a problem kernel *for L is a polynomial-time computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^* \times \Sigma^*$ that reduces an instance (I,k) of L to an instance (I',k') with the following properties:*

- *$(I,k) \in L \Leftrightarrow (I',k') \in L$,*
- *$k' \leq k$, and*
- *$|I'| \leq g(k)$ for a computable function g that depends only on k.*

The instance that is produced by the kernelization algorithm is called *problem kernel*. If g is a polynomial function, then the problem kernel is called *polynomial problem kernel*. While every fixed-parameter tractable problem admits a kernelization [Downey and Fellows 1999], there are problems that, under reasonable complexity-theoretic assumptions, do not admit kernelizations that produce polynomial problem kernels [Bodlaender et al. 2009, Bodlaender 2009, Dom et al. 2009, Fortnow and Santhanam 2011].

Kernelization algorithms are often represented by a set of *data reduction rules*. We define data reduction rules as a reduction from an instance of (I,k) of a parameterized problem L to an instance (I',k') of a parameterized problem. We say that a data reduction rule is *correct* if $(I,k) \in L$ if and only if $(I',k') \in L$. We say that a data reduction rule has been *exhaustively applied* if any further application of this rule does not modify the instance. An instance is called *reduced* with respect to a set of data reduction rules if each data reduction rule in the set has been exhaustively applied.

For more on the topic of problem kernelization, we refer to surveys by Fellows [2006], Guo and Niedermeier [2007], and Bodlaender [2009].

Multivariate Algorithmics. Recently, the notion of "multivariate algorithmics" has been a new motif in the scientific discourse about parameterized algorithmics [Fel-

lows 2009, Niedermeier 2010]. The concept of multivariate algorithmics aims at reflecting the fact that many of the results and questions in parameterized algorithmics have become so detailed and multi-faceted that it appears they are not fully absorbed by the standard notions of parameterized algorithmics. We give some examples for how multivariate algorithmics extends parameterized algorithmics.

First, an increasing number of parameters for NP-hard problems are so-called *combined parameters*. A combined parameter is a parameter that consists of two or more parts that are independent from each other. We define combined parameters as tuples of single parameters. For example, the combined parameter (k_1, k_2) consists of the independent parts k_1 and k_2. Formally, one does not need to extend the definition of parameterized problems to show fixed-parameter tractability or W[1]-hardness with respect to a combined parameter, since the independent parts can be encoded into one parameter. When parameterizing by combined parameters, several questions emerge, for example: Can one of the parts of the combined parameters be removed without giving up fixed-parameter tractability? What are the relations between the parts of the combined parameter?

Second, for many problems, attempts are made to increase the number of parameters for which the parameterized complexity of the problem is known. The aim is to obtain a comprehensive picture of the computational complexity of the problem. For example, when a problem has been shown to be W[1]-hard when parameterized by the parameter solution size k, it is natural to ask whether it is fixed-parameter tractable parameterized by $|I| - k$, where $|I|$ is often referred to as "dual parameter". This comprehensive overview of fixed-parameter tractability and intractability results for a problem naturally applies also to combined parameters, which further increases the number of possible results for a problem, and thus yields an even more detailed analysis of problem. In order to justify the study of certain parameterizations of a problem, it is useful to compare parameters with each other. In this work, we say that a parameter k is *stronger than* a parameter l for a problem L if there is a function f such that in every possible input instance of problem L it holds that $k \leq f(l)$, and for some fixed value for parameter k there are instances of L in which l is arbitrarily large. This means that fixed-parameter tractability for the stronger parameter k implies fixed-parameter tractability for parameter l, while fixed-parameter tractability for l does not necessarily imply fixed-parameter tractability for k.

1.3 Graph Theory

We briefly define the main concepts of graph theory that are used throughout this work.

An undirected graph G is a tuple (V, E), where V is the *vertex set* and $E \subseteq \{\{u, v\} \mid \{u, v\} \subseteq V\}$ is the *edge set*. Unless stated otherwise, we use n to denote the number

of vertices of G which we also refer to as *order* of G. The number of edges is usually denoted by m and referred to as *size* of G. Two vertices $u, v \in V$ are *adjacent* if $\{u, v\} \in E$. A vertex v is *incident* with an edge e if $v \in e$, that is, v is an *endpoint* of e. For a vertex v, we denote with $N(v) := \{u \in V \mid \{u, v\} \in E\}$ the *neighborhood* of v. The *closed neighborhood* is defined as $N[v] := N(v) \cup \{v\}$. We use $\deg(v) = |N(v)|$ to denote the *degree* of the vertex v. We say that a graph G has *maximum degree* Δ if for every vertex v in G it holds that $\deg(v) \leq \Delta$.

For a set $V' \subseteq V$, the *induced subgraph* $G[V']$ is the graph over the vertex set V' with the edge set $\{\{v, w\} \in E \mid v, w \in V'\}$. For a vertex set $S \subseteq V$ we use $G - S$ as an abbreviation for $G[V \setminus S]$, and for a vertex $v \in V$, we use $G - v$ as an abbreviation for $G[V \setminus \{v\}]$. An *isomorphism* of two graphs $G = (V, E)$ and $H = (W, F)$ is a bijection $f : V \to W$, such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H. A *matching* in a graph $G = (V, E)$ is a set M of edges of E such that no two edges in M have a common endpoint, that is, for each pair of edges $e_1 = \{u, v\}$ and $e_2 = \{x, y\}$ in M we either have $e_1 \cap e_2 = \emptyset$ or $e_1 = e_2$. A path is a graph $P = (V, E)$ with vertex set $V := \{v_1, \ldots, v_n\}$ and edge set $E := \{\{v_1, v_2\}, \ldots, \{v_{n-1}, v_n\}\}$. The vertices v_1 and v_n are the *endpoints* of P. The path on n vertices is denoted as P_n.

Connectivity and Cut-vertices. Two vertices u and v in an undirected graph G are called *connected* if G contains as subgraph a path with the endpoints u and v. An undirected graph is called connected if every pair of vertices is connected. The maximal connected subgraphs of a graph are its *connected components*. A vertex u in an undirected graph G is called a *cut-vertex* if G contains two vertices v, w with $v \neq u$ and $w \neq u$ such that every path from v to w contains u. If an undirected graph is connected and has no cut-vertex, then it is *biconnected*. In general, if a graph cannot be disconnected by deletion of any set of $p - 1$ vertices, it is called *p-connected*. A *bridge* in an undirected graph G is an edge $\{u, v\}$ such that in G every path between u and v contains $\{u, v\}$. If a graph is connected and has no bridge, then it is *bridge-connected*. A graph is called *p-edge-connected* if it cannot be disconnected by deletion of any set of $p - 1$ edges.

Graph Properties. A graph property is a nonempty proper subset Π of the set of all graphs. We say that a graph has property Π if it is isomorphic to a graph in Π. A graph property that is closed under the operation of deleting vertices (and hence, taking induced subgraphs) is called *hereditary*. Hereditary graph properties can be described by forbidden induced subgraphs [Greenwell et al. 1973]. This means that for each hereditary graph property Π there exists a set $\mathcal{F}$ of graphs such that a given graph G fulfills Π if and only if G is *$\mathcal{F}$-free*, that is, G does not contain any graph from $\mathcal{F}$ as induced subgraph.

Graph Modification Problems. Many of the problems considered in this work are graph modification problems. In these problems the aim is to obtain a graph with a prespecified graph property by applying modifications to this graph. Commonly considered modification types are *vertex deletions*, *edge deletions*, or *edge modifications* (that is, edge insertions and deletions). A generic formulation of vertex deletion problems is as follows.

> Π-VERTEX DELETION
> **Input:** An undirected graph G and a nonnegative integer k.
> **Question:** Is there a set S of at most k vertices such that $G - S$ has property Π?

The Π-VERTEX DELETION problem is NP-hard for hereditary graph properties Π [Lewis and Yannakakis 1980]. Edge modification problems can be formulated as follows.

> Π-EDITING
> **Input:** An undirected graph G and a nonnegative integer k.
> **Question:** Can G be transformed by up to k edge deletions and insertions into a graph that has property Π?

Edge deletion problems can be formulated analogously. For edge modification problems, there is no general NP-hardness result for hereditary graph properties. For example, the problem of deleting a minimum number of edges to obtain a forest can be solved in polynomial time, since it is equivalent to computing a spanning tree of each connected component. For a given graph modification problem, we refer to a set of allowed modifications whose application yields the desired graph property as a *solution* of the problem.

Part II

Clustering

(a) The input graph G.

(b) By deleting the three dashed edges and inserting the bold edge, G is transformed into a cluster graph.

Figure 1.1: An example of CLUSTER EDITING.

An important task in the analysis of networks is the partition of the network vertices into cohesive groups, *clusters*. In the example of protein interaction networks, these clusters should correspond to functional modules [Barabási and Oltvai 2004, Przulj et al. 2004, Sharan et al. 2007]. The common theme of many clustering algorithms that are used for clustering protein interaction networks is that the clustering solution depends crucially on the edges of the network, and to a lesser extent on other knowledge about the proteins such as for example sequence similarity. In other words, many clustering algorithms for protein interaction networks are graph-based. The intuitive idea behind this approach (and, in fact, behind graph-based data clustering in general) is that the input network is considered to be a similarity graph, that is, there is an edge between two vertices if and only if these two vertices are similar.

There is an abundance of clustering algorithms that have been suggested for clustering protein interaction networks, and the algorithmic approaches range from spectral analysis [Inoue et al. 2010] to Markov-clustering [Krogan et al. 2006] to min-cut-based clustering algorithms [Hartuv and Shamir 2000, Przulj et al. 2004]. So far, the contribution of parameterized algorithmics to the particular task of clustering protein interaction networks is negligible since the commonly proposed approaches do not involve solving an NP-hard optimization problem. In the field of parameterized algorithmics, the most extensively studied graph clustering approach is the NP-hard CLUSTER EDITING problem [Böcker et al. 2009, 2011, Chen and Meng 2010, Dehne et al. 2006, Fellows et al. 2007, Guo 2009, Gramm et al. 2005]:

CLUSTER EDITING
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Can G be transformed by up to k edge deletions and insertions into a cluster graph?

Herein, a cluster graph is a vertex-disjoint union of cliques or, equivalently, a graph that does not contain an induced path on three vertices (a P_3). The connected components of a cluster graph are the clusters of the clustering solution. Figure 1.1 shows an example of CLUSTER EDITING. Informally, the idea behind CLUSTER EDITING can be described as follows. A clustering is an equivalence relation since it partitions

the objects into equivalence classes, the clusters. A good clustering is one in which the elements of a cluster are similar to each other. Furthermore, elements from different clusters should be different from each other. Hence, a good clustering of a similarity graph partitions the vertex set into clusters such that there are many edges inside the clusters and few edges between clusters. This aim can be reformulated as inserting few edges inside clusters and deleting few edges between clusters. The CLUSTER EDITING problem thus corresponds to finding a clustering that is closest to the observed input similarities.

While the CLUSTER EDITING model has been successfully applied to cluster real-world biological data [Böcker et al. 2009, Rahmann et al. 2007, Wittkop et al. 2010, 2011], there have been so far no attempts to apply this model to protein interaction networks. The reason lies in

- the large size of the protein interaction networks, which prohibits using the current state-of-the-art fixed-parameter algorithms for CLUSTER EDITING, and in
- the structure of these networks which makes it unlikely that the clustering produced by CLUSTER EDITING is reasonable.[1]

Our main interest thus lies in making progress toward answering the following two questions:

- Can we identify new interesting parameterizations for graph-based data-clustering problems that increase the range of instances for which we can find optimal solutions to NP-hard graph-based data clustering problems?
- Can the CLUSTER EDITING model be "tweaked" to yield better clustering models for protein interaction networks? In particular, do the fixed-parameter tractability results that have been established for CLUSTER EDITING carry over to these new clustering problems?

In the next three chapters, we study these questions. We use CLUSTER EDITING as the "origin" of our exploration and study generalizations of and other problems related to CLUSTER EDITING as well as new parameterizations of CLUSTER EDITING.

Related work. The NP-hardness of CLUSTER EDITING, also known as CORRELATION CLUSTERING ON COMPLETE GRAPHS, has been shown several times [Křivánek and Morávek 1986, Shamir et al. 2004, Bansal et al. 2004]. The problem remains NP-hard when the solution may contain at most two clusters [Shamir et al. 2004]. The parameterized complexity of CLUSTER EDITING with respect to the parameter number k of edge modifications has been extensively studied. After a series of improvements [Gramm et al. 2005, Fellows et al. 2007, Protti et al. 2009, Guo 2009, Böcker

[1] See Chapter 3 for a more detailed discussion.

et al. 2009, Chen and Meng 2010], the currently fastest fixed-parameter algorithm for parameter k has running time $O(1.62^k + m)$ [Böcker 2011], and the currently smallest problem kernel contains at most $2k$ vertices [Chen and Meng 2010]. Several experimental studies demonstrate that fixed-parameter algorithms can be applied to solve real-world instances of CLUSTER EDITING [Dehne et al. 2006, Böcker et al. 2009, 2011]. A particularly promising approach seems to be the combination of fixed-parameter algorithms with a parameter-dependent data reduction strategy [Böcker et al. 2011]. So far, however, the currently fastest implementation for solving CLUSTER EDITING uses a combination of an ILP-formulation and a cutting-plane algorithm of Grötschel and Wakabayashi [1989] with a parameter-dependent data reduction strategy [Böcker et al. 2011]. As to approximability, there is a polynomial-time-approximation scheme for maximizing $|V| \cdot (|V| - 1)/2 - k$, that is, the sum of vertex pairs whose relation is *not* changed by the edge modification set S [Bansal et al. 2004]. In contrast, minimizing k is APX-hard [Charikar et al. 2005]; the currently best approximation ratio is 2.5 [Ailon et al. 2008, van Zuylen and Williamson 2009]. A closely related NP-hard clustering problem is CONSENSUS CLUSTERING which can be seen as a special case of edge-weighted CLUSTER EDITING [Ailon et al. 2008, van Zuylen and Williamson 2009].

Several generalizations of CLUSTER EDITING have been studied with respect to their algorithmic properties. For example, CLUSTER EDITING is a special case case of HIERARCHICAL-TREE CLUSTERING [Křivánek and Morávek 1986, Guo et al. 2010a]. The variant of CLUSTER EDITING that allows the input graph to contain "uncertain edges" is referred to as FUZZY CLUSTER EDITING [Bodlaender et al. 2010] or CORRELATION CLUSTERING ON GENERAL GRAPHS [Charikar et al. 2005]. FUZZY CLUSTER EDITING parameterized by the "number of modified certain edges" has received some attention attention [Bodlaender et al. 2010, Bousquet et al. 2011, Marx and Razgon 2011] since it is—in terms of fixed-parameter tractability—equivalent to MULTICUT parameterized by the size of the cut set. Very recently, the fixed-parameter tractability of both problems was shown [Bousquet et al. 2011, Marx and Razgon 2011]. At the moment this is however a mere classification result since the running times of the corresponding fixed-parameter algorithms for FUZZY CLUSTER EDITING are huge. It is fixed-parameter tractable to enumerate all inclusion-minimal solutions to CLUSTER EDITING [Damaschke 2010]. Further theoretical studies present fixed-parameter tractability results for generalizations of CLUSTER EDITING that allow overlap between the clusters of the cluster graph [Damaschke 2010, Fellows et al. 2011b].

There are further combinatorial models for graph-based data clustering. For example, Hartuv and Shamir [2000] proposed a network clustering algorithm that repeatedly removes the edges of a minimum-cardinality cut until each connected component (that is, cluster) of the graph is "highly connected". Recently, a further graph clustering model was proposed in which each cluster is allowed to have at most

q outgoing edges and should fulfill further density constraints for example "missing at most p edges to be a clique" [Heggernes et al. 2010, Lokshtanov and Marx 2011]. Note that in both of these clustering models the overall number of edges (or missing edges) that disagree with the clustering is not minimized; this is a difference from the clustering approaches in this work, which demand a clustering that minimizes this number.

Overview of Part II. In Chapter 2, we study how "local degree bounds" influence the complexity of CLUSTER EDITING and of its edge-deletion version CLUSTER DELETION. We show that even for graphs with constant maximum degree CLUSTER EDITING and CLUSTER DELETION are NP-hard and that this implies NP-hardness even if every vertex is incident with only a constant number of edge modifications. We furthermore obtain lower bounds for running times of fixed-parameter algorithms with the parameter number k of edge modifications. Finally, we show that CLUSTER EDITING becomes easier in case the number of clusters is fixed by presenting a problem kernelization for the parameter "number d of clusters and local modification bound t".

In Chapter 3, we discuss drawbacks of CLUSTER EDITING in certain application scenarios and propose several generalizations of CLUSTER EDITING that could be applied to circumvent these drawbacks. We then study the parameterized complexity of these generalizations with respect to the parameter "number of edge modifications". The aim of this study is two-fold. First, we believe that the proposed generalizations are more appropriate clustering models for protein interaction networks than CLUSTER EDITING. Second, we further explore the applicability of clique relaxations in the area of fixed-parameter algorithms. Roughly speaking, our result is that it is possible to replace clique models for clusters with clique relaxations as long as there is a second combinatorial handle, the "slack" parameter s which measures how much the clique requirement is relaxed.

In Chapter 4 we study the CONSENSUS CLUSTERING problem, which is closely related to edge-weighted CLUSTER EDITING. The aim of CONSENSUS CLUSTERING is the integration of disagreeing clusterings into a new clustering that has a minimum amount of disagreement with the input clusterings. We show that CONSENSUS CLUSTERING is fixed-parameter tractable with respect to the parameter average distance of the input cluster. In other words, we demonstrate that the problem becomes easy in case the average disagreement between the input clusters is small. We also identify further related parameters for CONSENSUS CLUSTERING and present a new kernelization concept: structural kernelization.

Chapter 2

Cluster Editing with Locally Bounded Modifications

In this chapter, we study the effect of "bounding the local amount of edge modifications" in instances of CLUSTER EDITING which we defined as follows.

> CLUSTER EDITING
> **Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Can G be transformed by up to k edge deletions and insertions into a cluster graph?

Our studies are motivated by the observation that, so far, the proposed fixed-parameter algorithms for CLUSTER EDITING almost exclusively examine the parameter number k of edge modifications [Böcker et al. 2009, 2011, Chen and Meng 2010, Gramm et al. 2005, Guo 2009]. This focus on the parameter k is contrasted by the observation that k is often not really small for real-world instances. For example in a protein similarity data set that has been frequently used for evaluating CLUSTER EDITING algorithms, the instances with $n \geq 30$, n being the number of vertices, have an average number k of edge modifications that is between $2n$ and $4n$ [Böcker et al. 2009]. Still, the fixed-parameter algorithms can solve many of these instances [Böcker et al. 2009, 2011], which raises the question whether there are "hidden parameters" that are implicitly exploited by these algorithms. We therefore aim at identifying promising new parameterizations for CLUSTER EDITING that help to separate easy from hard instances.

One of the parameters under consideration in this chapter is a *stronger* parameter than the number k of edge modifications. We call this parameter *local modification bound* t. In the following, we refer to a set of edge deletions and insertions as *edge modification set*.

Definition 2.1. *Let $G = (V, E)$ be an undirected graph, and let S be an edge modification*

set for G. We say that S is locally-t-bounded *if for every vertex $v \in V$ it holds that*

$$|S \cap \{\{u, v\} \mid u \in V \setminus \{v\}\}| \leq t.$$

Informally, this means that a locally-t-bounded edge modification set performs at most t edge modifications on each vertex of the input graph. Another intuitive way of looking at locally-t-bounded edge modification sets is to visualize the graph that has vertex set V and edge set S. If S is locally-t-bounded, then this graph has maximum degree t.

The local modification bound t relates to the overall number k of edge modifications in the following way: First, any edge modification set S is clearly locally-$|S|$-bounded. Second, the local modification bound t can be arbitrarily small compared to the overall number of edge modifications. Hence, the local modification bound t is indeed a stronger parameter than the overall number of edge modifications. We expect that in most practically relevant instances the local modification bound t is much smaller than the overall number of edge modifications. As we observe here, the local modification bound is upper-bounded by the maximum degree Δ of the input graph which is the second parameter that we consider.

Unfortunately, as we show in this chapter, it turns out that Cluster Editing is NP-hard already for constant Δ and also for constant t. To contrast these NP-hardness results, we show that parameterizing by the combined parameter "upper bound d on the number of clusters and local modification bound t" yields fixed-parameter tractability.

In addition, we also consider the Cluster Deletion problem in which only edge deletions are allowed.

> Cluster Deletion
> **Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Can G be transformed by up to k edge deletions into a cluster graph?

Our results for Cluster Deletion are roughly the same as for Cluster Editing, although for Cluster Deletion we obtain, somewhat surprisingly, hardness results for even more restricted cases than for Cluster Editing.

Related Work. As discussed in the introduction to Part II, there are many results for Cluster Editing parameterized by the number k of edge modifications. Other parameterizations have played a marginal role so far. To the best of our knowledge, the only other parameter that has been considered is the so-called cluster vertex deletion number which is the number of vertices one needs to delete in order to obtain a cluster graph. Cluster Editing and Cluster Deletion are both fixed-parameter tractable with respect to the cluster vertex deletion number of the input graph [Komusiewicz

and Uhlmann 2011, Uhlmann 2011]. However, the running times of the algorithms for this parameter seem to be impractical so far.

A variant of CLUSTER EDITING in which the number of clusters is fixed (instead of upper-bounded as we consider in Section 2.2) has been previously studied: For every $d \geq 2$ it is NP-hard to decide whether the input graph can be transformed by at most k edge modifications into a graph with *exactly* d clusters [Shamir et al. 2004]. Guo [2009] showed that this variant of CLUSTER EDITING admits a problem kernel consisting of at most $(d+2) \cdot k + d$ vertices.

While not as extensively studied as CLUSTER EDITING, some results have been obtained for CLUSTER DELETION as well: CLUSTER DELETION is NP-hard in general and when one demands that the cluster graph has exactly $d \geq 3$ clusters but polynomial-time solvable when one demands that the cluster graph has exactly two clusters [Shamir et al. 2004]. CLUSTER DELETION can be solved in $O(1.415^k + n^3)$ time by a search tree algorithm [Böcker and Damaschke 2011].

Our Results. Table 2.1 summarizes our findings which are as follows. We present a reduction from 3-SAT to CLUSTER EDITING which yields several hardness results.[1] First, it shows that CLUSTER EDITING is NP-hard even on input graphs with maximum degree six. Second, it shows that CLUSTER EDITING is NP-hard even when every edge modification set of size at most k is locally-4-bounded. Hence, the local modification bound itself is not a suitable parameter for CLUSTER EDITING. Finally, the reduction from 3-SAT shows that CLUSTER EDITING does not admit an algorithm with running time $2^{o(k)} \cdot \text{poly}(|V|)$ time unless the so-called exponential time hypothesis fails. The exponential time hypothesis states that k-SAT, $k \geq 3$, cannot be solved within a running time of $2^{o(n)}$ or $2^{o(m)}$, where n is the number of variables and m is the number of clauses in the input k-CNF formula. This approach for showing super-polynomial lower bounds for running times goes back to work of Impagliazzo et al. [2001]; a survey by Woeginger [2003] discusses, among other things, some aspects of the exponential time hypothesis. In this context, algorithms with running time $2^{o(p)}$ for some parameter p are called *subexponential-time* algorithms. Our result on the nonexistence of such a subexponential-time algorithm for the parameter k negatively answers a recent conjecture by Cao and Chen [2010].

For CLUSTER DELETION, we can show hardness for even more restricted cases by observing close connections to PARTITION INTO TRIANGLES. We show that CLUSTER DELETION is NP-hard even when the input graph has maximum degree four, and that it is NP-hard even when every solution of size at most k is locally-2-bounded. Again, we also observe that our results imply that CLUSTER DELETION does not admit an algorithm with running time $2^{o(k)} \cdot \text{poly}(|V|)$ unless the exponential time

[1] Previous NP-hardness results were obtained for example by reductions from 3-DIMENSIONAL MATCHING [Křivánek and Morávek 1986] or EXACT COVER BY 3-SETS [Shamir et al. 2004].

Parameter	CLUSTER EDITING	CLUSTER DELETION
Δ	NP-hard for $\Delta = 6$	NP-hard for $\Delta \geq 4$, $\in P$ for $\Delta \leq 3$
t	NP-hard for $t = 4$	NP-hard for $t = 2$
k	No $2^{o(k)} \cdot \text{poly}(n)$ algorithm	No $2^{o(k)} \cdot \text{poly}(n)$ algorithm
(d,t)	$4 \cdot dt$-vertex kernel	$2 \cdot dt$-vertex kernel

Table 2.1: Summary of our results for CLUSTER EDITING and CLUSTER DELETION and the parameters maximum degree Δ, local modification bound t, number k of edge modifications, and the combined parameter (d,t). The results for parameter k hold unless the exponential time hypothesis fails.

hypothesis fails. We also show that CLUSTER DELETION is polynomial-time solvable on graphs with maximum degree three, thus achieving a dichotomy with respect to the maximum degree of the input graph.

We complement the negative results for CLUSTER EDITING and CLUSTER DELETION by showing that both problems are fixed-parameter tractable with respect to the combined parameter (d,t), where d is an upper bound on the number of clusters in the cluster graph and t is the local modification bound. More precisely, we consider a constrained version of both problems that might be of independent interest. Our algorithms for these problems are based on simple data reduction rules that produce in $O(|V|^3)$ time a problem kernel consisting of at most $4 \cdot dt$ vertices (in the case of CLUSTER EDITING) and $2 \cdot dt$ vertices (in the case of CLUSTER DELETION).

2.1 Constant Maximum Degree and Constant Local Modification Bound

We show that CLUSTER EDITING is NP-hard even when restricted to graphs with maximum degree six. To the best of our knowledge the previous NP-hardness proofs require an unbounded degree [Křivánek and Morávek 1986, Bansal et al. 2004, Shamir et al. 2004]. As an immediate consequence of our NP-hardness proof, CLUSTER EDITING is NP-hard even for a constant local modification bound. The following structural lemma will be used in our proof of NP-hardness.

Lemma 2.1. *Let $G = (V,E)$ be an undirected graph. There is a minimum-cardinality solution S producing a cluster graph G' such that for all vertices $u, v \in V$ with $|N(u) \cap N(v)| \leq 1$ and $\{u,v\} \notin E$ it holds that u and v are in different clusters of G'.*

Proof. Assume that there is a minimum-cardinality solution S that yields a cluster graph G' such that there is a pair of vertices $u, v \in V$ with $|N(u) \cap N(v)| \leq 1$ and $\{u,v\} \notin E$ that are in the same cluster K of G'. We show that one can construct from S a

solution S' with $|S'| \leq |S|$ that yields a cluster graph G'' in which either u or v is a singleton cluster.

Let $X := N(u) \cap N(v)$ be the common neighborhood of u and v in G, let $K_v := K \cap N(v) \setminus X$, and let $K_u := K \cap N(u) \setminus X$. Note that $|X| \leq 1$. Without loss of generality, assume that $|K_v| \geq |K_u|$. Then, u is in G adjacent to at most $\lfloor (|K|-1)/2 \rfloor$ vertices in K since $|K_u| \leq \lfloor (|K|-3)/2 \rfloor$ and since u has in G at most one further neighbor in K (because $|X| \leq 1$). Therefore, cutting u from K yields a solution S' with $|S'| \leq |S|$ since this operation "undoes" at least $\lceil (|K|-1)/2 \rceil$ edge insertions and causes at most $\lfloor (|K|-1)/2 \rfloor$ additional edge deletions.

Exhaustively applying the modification above for each such pair of vertices results in a minimum-cardinality solution with the desired property. Since each application of this modification produces at least one singleton cluster, there can be at most n iterations of this procedure. Hence, a solution with the desired property does indeed exist. □

In the CLUSTER EDITING instances created by the reduction, all nonadjacent vertices have at most one vertex in common. Hence, the lemma above implies that in every one of these instances there is an optimal solution that only deletes edges.

For the NP-hardness proof we present a reduction from 3-SAT, which has as input a boolean formula ϕ in 3-CNF and asks whether there is an assignment to the variables of ϕ that fulfills all clauses of ϕ.[2] The basic idea of the reduction is as follows. For each variable x_i of a given 3-CNF formula ϕ, we construct a *variable cycle* of length $4 \cdot \#(i)$, where $\#(i)$ denotes the number of clauses that contain x_i. It is easy to verify that only deleting every second edge yields a minimum-cardinality edge modification set for transforming an even-length cycle into a cluster graph. The corresponding two possibilities are used to represent the two choices for the value of x_i. Moreover, for each clause C_j containing the variables x_p, x_q, and x_r, we connect the three corresponding variable cycles by a clause gadget. In doing so, the goal is to ensure that if the solutions for the variable gadgets correspond to an assignment that satisfies C_j, then one needs only four edge modifications for the clause gadget and otherwise one needs at least five edge modifications. Let m be the number of clauses in ϕ and observe that, since ϕ is a 3-CNF formula, the overall number of vertices in the variable cycles is $12m$. Our construction guarantees that there is a satisfying assignment for ϕ if and only if the constructed graph can be transformed into a cluster graph by exactly $6m + 4m = 10m$ edge modifications, where $6m$ modifications are used for the variable cycles and $4m$ modifications are used for the clause gadgets. The details follow.

Given a 3-CNF formula ϕ consisting of the clauses $C_0, \ldots, C_{m-1}$ over the variables $\{x_0, \ldots, x_{n-1}\}$, construct a CLUSTER EDITING-instance $(G = (V, E), k)$ as follows.

[2] A similar reduction was previously used to show NP-hardness of the TRANSITIVITY EDITING problem which is defined on directed graphs [Weller et al. 2011].

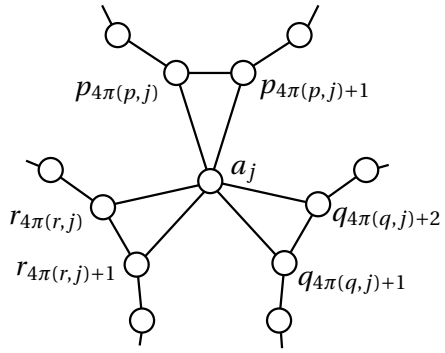


Figure 2.1: Illustration of the clause gadget for a clause $C_j = (x_p \vee \overline{x_q} \vee x_r)$.

For each variable x_i, $0 \leq i < n$, G contains a *variable cycle* that consists of the vertices $V_i^v := \{i_0, \ldots, i_{4\#(i)-1}\}$ and the edges $E_i^v := \{\{i_k, i_{k+1}\} \mid 0 \leq k < 4\#(i)\}$ (for ease of presentation let $i_{4\#(i)} = i_0$). We use the following notation for the edges of variable cycles: the edges $\{i_0, i_1\}, \{i_2, i_3\}, \ldots, \{i_{4\#(i)-2}, i_{4\#(i)-1}\}$ of the variable cycle of x_i are called *even*, all other edges are called *odd*. So far, the constructed graph consists of a disjoint union of cycles. Next, we add a *clause gadget* to G for each clause of ϕ.

In the construction of the clause gadgets, we need for each clause C in the variable cycles of C's variables a fixed set of vertices that are "reserved" for C. To this end, suppose that for each variable x_i an arbitrary but fixed ordering of the clauses that contain x_i is given, and let $\pi(i, j)$ denote the position of a clause C_j that contains x_i in this ordering. We now give the details of the construction of the clause gadgets. Let C_j be a clause containing the variables x_p, x_q, and x_r (either negated or nonnegated). We construct a clause gadget connecting the variable gadgets of x_p, x_q, and x_r. First, we add a new vertex a_j. Furthermore, let E_j^c denote the edge set of the clause gadget and let E_j^c contain for each $i \in \{p, q, r\}$ the edges $\{a_j, i_{4\pi(i,j)}\}$ and $\{a_j, i_{4\pi(i,j)+1}\}$ if x_i occurs nonnegated in C_j or the edges $\{a_j, i_{4\pi(i,j)+1}\}$ and $\{a_j, i_{4\pi(i,j)+2}\}$, otherwise. See Figure 2.1 for an illustration. Finally, let $V := \bigcup_{i=0}^{n-1} V_i^v \cup \bigcup_{j=0}^{m-1} \{a_j\}$ and $E := \bigcup_{i=0}^{n-1} E_i^v \cup \bigcup_{j=0}^{m-1} E_j^c$. This completes the construction of $G = (V, E)$.

Theorem 2.1. CLUSTER EDITING *is NP-hard even when restricted to graphs with maximum vertex degree six.*

Proof. Let ϕ be a 3-SAT formula and let G be constructed from ϕ as described above. We show the correctness of the reduction by showing the following claim.

> ϕ is satisfiable $\Leftrightarrow$ G can be transformed into a cluster graph by at most $k := 10m$ edge modifications

In the following, we use the characterization of cluster graphs as the graphs that do not contain an induced P_3, that is, an induced path on three vertices.

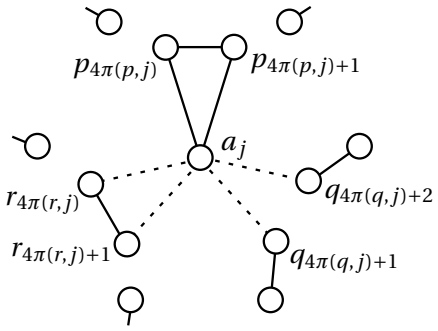


Figure 2.2: Illustration that all induced P_3s that contain a_j can be destroyed by four additional edge deletions if all odd edges in the variable cycle of x_p are deleted (observe that x_p occurs nonnegated in C_j, since a_j is adjacent to $p_{4\pi(p,j)}$ and $p_{4\pi(p,j)+1}$). The four edge deletions incident with a_j are marked by dotted lines.

$\Rightarrow$: Given a satisfying assignment β for ϕ we can transform G into a cluster graph as follows. For each variable x_i delete the odd edges of the variable cycle of x_i if $\beta(x_i)$ = true and the even edges otherwise. Moreover, for each clause C_j proceed as follows. Assume that C_j contains the variables x_p, x_q, and x_r. Without loss of generality assume that the literal that corresponds to x_p is true. All induced P_3s that contain a_j can be destroyed by the deletion of the four edges with one endpoint being a_j and the other endpoints from $V_q^v \cup V_r^v$ (see Figure 2.2). For the variable cycles, we perform altogether $\sum_{0\leq i<n} \#(i)/2 = 6m$ edge modifications, and for each clause gadget four edges are deleted. Hence, $10m$ edge modifications are performed overall. By construction, every induced P_3 contains either three vertices of the same variable cycle or at least one of the a_j's. Hence, all induced P_3s are destroyed and the resulting graph is a cluster graph.

$\Leftarrow$: Let S denote an optimal solution for G of size at most $k := 10m$. To show that ϕ is satisfiable, we need some observations about the structure of G and S.

First, we show that $10m$ is a lower bound on any solution for G. By the construction of G, for every nonadjacent pair of vertices u, v in G, it holds that $|N(u) \cap N(v)| \leq 1$. Therefore, we can assume, by Lemma 2.1, that S performs only edge deletions (since no nonadjacent vertices end up in the same cluster). Furthermore, note that for each variable x_i the variable cycle contains $\#(i)/2$ edge-disjoint induced P_3s with all three vertices on the cycle and that deleting either all even or all odd edges are the only two optimal ways to destroy these induced P_3s. Hence, G contains $6m$ edge-disjoint induced P_3s such that all three vertices of the induced P_3 are in the same variable cycle. Clearly, at least $6m$ edge deletions are needed for these induced P_3s. For each clause a_j, $0 \leq j < m$, at least four edge deletions are needed to destroy all induced P_3s that contain a_j. Since at least $6m$ edges are deleted in the variable cycle, this means that for each clause C_j *exactly* four edges incident with a_j are deleted by S, and that

for each variable cycle either all even or all odd edges are deleted.

Consider the assignment β for ϕ that, for each x_i, $0 \leq i < n$, sets $\beta(x_i) :=$ true if all odd edges of V_i^v are deleted and sets $\beta(x_i) :=$ false if all even edges of V_i^v are deleted. We show that β is a satisfying assignment. Consider an arbitrary clause C_j containing the variables x_p, x_q, and x_r. Since exactly four edge deletions are incident with a_j, the edges that are incident with the vertices of the variable cycle of one variable of C_j, say x_p, are not deleted by S. Without loss of generality, assume that x_p appears nonnegated in C_j. Then the two vertices of V_p^v that are adjacent to a_j are $p_{4\pi(p,j)}$ and $p_{4\pi(p,j)+1}$. Since S is a solution, the edge $\{p_{4\pi(p,j)}, p_{4\pi(p,j)+1}\}$ is not deleted by S. Hence, all odd edges of V_p^v are deleted, and therefore the assignment β fulfills clause C_j. □

We can use the presented reduction to obtain further hardness results for CLUSTER EDITING. Since the constructed graph has maximum degree six, every optimal solution is locally-6-bounded. This is due to the fact that if a vertex v is incident with more than 6 edge modifications, then one can obtain a better solution by undoing these edge modifications and deleting all edges that are incident with v in G.

This observation can be strengthened even further by observing that, by the construction of G, we either need more than $10m$ edge modifications or that the maximum number of edge modifications per vertex is four. The latter can be seen as follows. As described in the proof of Theorem 2.1, if there is a solution of size at most $10m$, then there is also a solution that only performs edge deletions and that has the following further properties. It performs $6m$ edge deletions in the variable cycles, and on each vertex in the variable cycle at most one of the deleted edges is incident. Note that each of the vertices in the variable cycle has at most one neighbor in a clause gadget. Hence, for each vertex of the variable cycle at most two edge deletions are performed on incident edges. Furthermore, for each clause gadget exactly four edge deletions are performed. Hence, we can assume that there is a solution that is locally-4-bounded.

Corollary 2.1. CLUSTER EDITING *is NP-hard even when for every yes-instance there is a solution that is locally-*4*-bounded.*

Our final hardness result for CLUSTER EDITING can be drawn from the observation that the solution size is ten times the number of clauses in the 3-CNF formula. By our reduction, a subexponential-time algorithm for CLUSTER EDITING parameterized by k would imply an algorithm for solving 3-SAT that has running time subexponential in the number m of clauses. The same can be observed for the number $|V|$ of vertices in the CLUSTER EDITING instance. Hence, we arrive at the following.

Theorem 2.2. CLUSTER EDITING *cannot be solved in* $2^{o(k)} \cdot \text{poly}(|V|)$ *time or in* $O(2^{o(|V|)})$ *time unless the exponential time hypothesis fails. This holds even when the input graph has maximum degree six.*

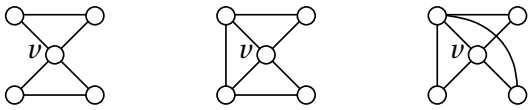


Figure 2.3: The three different neighborhoods in a 4-regular neighborhood restricted PARTITION INTO TRIANGLES instance. None of these graphs contains a clique of order four.

For CLUSTER DELETION, we can obtain hardness for even more restricted input graphs by observing close connections to PARTITION INTO TRIANGLES on graphs with maximum degree four. As recently shown by van Rooij et al. [2011], PARTITION INTO TRIANGLES is NP-hard even when the input graph $G = (V, E)$ is 4-regular. Moreover, NP-hardness persists even when for each vertex $v \in V$ the graph $G[N[v]]$ is isomorphic to one of the three graphs shown in Figure 2.3 [van Rooij et al. 2011]. In the following, we refer to such graphs as 4-*regular neighborhood restricted graphs*. The following easy observation is useful for establishing the connection to CLUSTER DELETION.

Observation 2.1. *Let $G = (V, E)$ be a 4-regular neighborhood-restricted graph. Then G does not contain any clique of order four or more.*

The observation says that if we use a 4-regular neighborhood-restricted graph as input graph for CLUSTER DELETION, then the largest clusters in the resulting cluster graph are triangles. In the next lemma, we show that the case in which every cluster is a triangle is optimal. Let $n := |V|$ in what follows.

Lemma 2.2. *Let $G = (V, E)$ be an instance of* PARTITION INTO TRIANGLES *such that G is a 4-regular neighborhood-restricted graph. Then, G is a yes-instance of* PARTITION INTO TRIANGLES $\Leftrightarrow$ *$(G, k := n)$ is a yes-instance of* CLUSTER DELETION.

Proof. We show both directions separately.

$\Rightarrow$: Let I be a yes-instance of PARTITION INTO TRIANGLES, and let $G_1, \ldots, G_{n/3}$ denote a set of triangles into which the input graph can be partitioned. Note that each G_i contains three edges and three vertices. Since G is 4-regular, it has $2n$ edges. Hence, there are exactly n edges that are not contained in any G_i. Deleting these edges from G yields a cluster graph, since each component is a triangle.

$\Leftarrow$: Let $S \subseteq E$ be an edge set of size at most $k := n$ such that deleting S from G yields a cluster graph G'. By Observation 2.1, every cluster contains at most three vertices. Each cluster on three vertices has exactly three edges and clusters with one or two vertices have less edges than vertices. Consequently, G' has at most n edges. Since $|S| \leq n$ and $|E| = 2 \cdot n$, G' has exactly n edges. Hence, every cluster is a triangle. Consequently, the clusters are a set of vertex-disjoint triangles, and I is thus a yes-instance of PARTITION INTO TRIANGLES. □

The above lemma directly implies a polynomial-time many-to-one reduction from PARTITION INTO TRIANGLES on 4-regular neighborhood-restricted graphs

to CLUSTER DELETION on 4-regular neighborhood-restricted graphs: all that needs to be done is to set $k := n$. Our main result that can be obtained by using this reduction is as follows.

Theorem 2.3. CLUSTER DELETION *is NP-hard even on 4-regular graphs.*

Note that since the input graph of the CLUSTER DELETION instance is 4-regular, and since every cluster must be a triangle we can assume that the edge deletion set is locally-2-bounded.

Corollary 2.2. CLUSTER DELETION *is NP-hard even when for yes-instances every solution is locally-2-bounded.*

Finally, we can also obtain lower bounds for the running time of CLUSTER DELETION with respect to parameter k. PARTITION INTO TRIANGLES does not admit a subexponential-time algorithm even on 4-regular neighborhood restricted graphs unless the exponential time hypothesis fails [van Rooij et al. 2011]. Since we can reduce such instances of PARTITION INTO TRIANGLES to CLUSTER DELETION instances on the same graph with $k = n$, we arrive at the following.

Theorem 2.4. CLUSTER DELETION *cannot be solved in* $2^{o(k)} \cdot \text{poly}(n)$ *time or in* $2^{o(n)}$ *time unless the exponential time hypothesis fails. This holds even when the input graph is 4-regular.*

In the following, we present a polynomial-time algorithm for CLUSTER DELETION in case the input graph has maximum degree three. Hence, we obtain the following dichotomy: CLUSTER DELETION is polynomial-time solvable on graphs with maximum degree three, and NP-hard, otherwise. The main idea of the presented algorithm is as follows. The algorithm starts by exhaustively applying two data reduction rules. One rule deals with all isolated clusters and, as we show, hence with all clusters of size four. The other rule deals with a certain type of triangles. We then show that after these reduction rules have been exhaustively applied, we can reduce our instance to a weighted version of CLUSTER DELETION whose input graph is triangle-free. Finally, we show that this instance can be solved by computing a maximum-weight matching.

Next, we present the two reduction rules in detail. The aim of the first reduction rule is to deal with all clusters of size four. Suppose that the final cluster graph contains such a cluster of size four. Then, since the input graph G has maximum degree three, this cluster must be a connected component and thus an isolated clique of G. Hence, we can remove all vertices that are part of these clusters in $O(n)$ time with the following trivial reduction rule.

Reduction Rule 2.1. *Remove from G all connected components that are cliques.*

Clearly, Rule 2.1 is correct and can be exhaustively applied in $O(n)$ time. We now present the second data reduction rule.

Reduction Rule 2.2. *If G contains three vertices u, v, and w such that*

- *$\{u, v, w\}$ induces a triangle in G, and*
- *there is no vertex $x \in V \setminus \{u, v, w\}$ that has at least two neighbors in $\{u, v, w\}$,*

then delete all edges between $\{u, v, w\}$ and $V \setminus \{u, v, w\}$, decrease k by the number of performed edge deletions, and remove $\{u, v, w\}$ from G.

Lemma 2.3. *Rule 2.2 is correct and can be exhaustively performed in $O(n)$ time.*

Proof. We first prove the correctness of the rule, and then show its running time. To show the correctness of the rule, we show that there is an optimal solution that yields a cluster graph in which $\{u, v, w\}$ is a cluster. Let $S \subseteq E$ be an optimal solution, let $G' := (V, E \setminus S)$ be the resulting cluster graph, and assume that $\{u, v, w\}$ does not form a cluster of G'. Then, either three or two edges between u, v, and w are deleted (if only one edge is deleted then u, v, and w induce a P_3). In the first case, we can obtain a solution S' by undoing all three edge deletions between u, v, and w and instead deleting the at most three edges between $\{u, v, w\}$ and $V \setminus \{u, v, w\}$. Clearly $|S'| \leq |S|$. In the second case, suppose that $\{u, v\}$ is not deleted by S. Then, $\{u, v\}$ is a cluster of G'. We can obtain a solution S' from S by undoing the deletion of $\{u, w\}$ and $\{v, w\}$ and instead deleting at most one edge between w and $V \setminus \{u, v, w\}$. Since $|S'| < |S|$, S is not an optimal solution, a contradiction.

The running time can be seen as follows. First, we can label in $O(n)$ time the edges of all triangles to which Rule 2.2 applies by checking for each vertex $v \in V$ whether $N[v]$ contains a triangle that fulfills the condition of the rule. Then, we can delete in $O(n)$ time all unlabeled edges that have a common endpoint with a labeled edge, since these are precisely the "outgoing" edges of a triangle that fulfills the condition of the rule. After the deletion of these edges, the rule has been exhaustively applied since the application of the rule does not create "new" triangles to which the rule can be applied. This can be seen as follows. Observe that the endpoints of an edge e that is deleted by Rule 2.2 do not have any common neighbors, since one of e's endpoints is in a triangle in which no two vertices have a common neighbor outside the triangle and G has maximum degree three. Now suppose that the deletion of an edge e produces a triangle $T = \{u, v, w\}$ to which Rule 2.2 applies. Clearly, e must be incident with one vertex from T. Hence, assume without loss of generality that $e = \{u, x\}$. Since the triangle T did not fulfill the condition of the rule before the deletion of $\{u, x\}$, the vertex x must have another neighbor in T, say w. This contradicts the observation that the endpoints of a deleted edge do not have any common neighbors. Hence, Rule 2.2 can be exhaustively applied in one pass which can be performed in $O(n)$ time. □

A graph with maximum degree three to which neither Rule 2.1 nor Rule 2.2 applies has a special structure: For every triangle $\{u, v, w\}$, there is at least one other vertex x

that has two neighbors, say u and v, in the triangle. Then, u and v have two common neighbors. Since the graph has maximum degree three and since they are adjacent, it holds that $N[u] = N[v]$. Note that since the graph does not contain cliques of size four after Rule 2.1 has been applied, there is also no further vertex y that is adjacent to two vertices in $\{u, v, w\}$. Altogether this leads to the following observation.

Observation 2.2. *Let G be a graph with maximum degree three that is reduced with respect to Rules 2.1 and 2.2. Then, every triangle contains exactly two degree-three vertices u and v with $N[u] = N[v]$.*

The above observation can be used in the following way: the vertices u and v are part of exactly two triangles, and they can be in at most one of those triangles in a cluster graph. Furthermore, the two vertices that are neighbors of u and v are part of exactly one triangle since they have at most one further neighbor. Hence, all triangles come in isolated pairs of which at most one is a cluster of the cluster graph. We can show that in this case two vertices in the intersection of two triangles end up in the same cluster. We can therefore "get rid" of these triangles by reducing the problem to a weighted version of Cluster Deletion by merging the two vertices. The resulting instance of this weighted version is triangle-free which makes it possible to compute an optimal solution by computing a maximum-weight matching.

Lemma 2.4. *Let (G, k) be an instance of* Cluster Deletion *such that G has maximum degree three and G is reduced with respect to Rules 2.1 and 2.2. Then, (G, k) can be solved in $O(\sqrt{n} \cdot n)$ time.*

Proof. Let (G, k) be as described in the lemma. We describe a polynomial-time algorithm for (G, k) that consists of two main steps. First, we reduce (G, k) to a triangle-free instance of the following edge-weighted version of Cluster Deletion:

> Weighted Cluster Deletion
> **Input:** An undirected graph $G = (V, E)$, an edge-weight function $\omega : E \to \mathbb{N} \setminus \{0\}$, and an integer $k \geq 0$.
> **Question:** Is there an edge set $S \subseteq E$ such that deleting S from G results in a cluster graph and $\sum_{e \in S} \omega(e) \leq k$?

Afterwards, we show that triangle-free instances of Weighted Cluster Deletion can be solved in polynomial time by computing a maximum-weight matching.

The reduction from Cluster Deletion to Weighted Cluster Deletion works as follows. First, we set $\omega(e) = 1$ for each $e \in E$ and thus obtain an instance of Weighted Cluster Deletion. Clearly, this instance is equivalent to the original instance. Then, we further apply the following reduction rule to reduce this instance of Weighted Cluster Deletion into a triangle-free instance of Weighted Cluster

DELETION.[3] As long as G contains a triangle, do the following. Let u and v denote the degree-three vertices of the triangle with $N[u] = N[v]$ (by Observation 2.2 there is exactly one such pair of vertices). Furthermore, let w and x denote the other two neighbors of u and v. Remove u from G and set $\omega(v, w) := 2$ and $\omega(v, x) := 2$. Note that after u is removed from G, v has degree two and it is furthermore not contained in a triangle in G.

The correctness of the reduction rule described above can be seen as follows. Since $N[u] = N[v]$ and by Observation 2.2, u and v are a so-called critical clique, that is, a maximal vertex set in which all vertices have the same closed neighborhood. Furthermore, all edges incident with u and v have weight one since u and v are still part of a triangle. Every optimal solution puts u and v into the same cluster which can be seen as follows. Suppose that there is an optimal solution S that puts u and v into different clusters. Since S is optimal, there must be a vertex w such that one of u and v, say u is in a cluster with w: otherwise, undoing the deletion of $\{u, v\}$ yields a better clustering. Then, by undoing the deletions of $\{u, v\}$ and $\{v, w\}$ and deleting at most one other edge instead, we obtain a better solution. As a consequence, if $\{u, w\}$ is deleted by an optimal solution, then also $\{v, w\}$ is deleted by this solution. Hence, every optimal solution before the removal of u one-to-one corresponds to an optimal solution after the removal of u (and the subsequent increase of the edge weights).

After all triangles have been replaced by edges of weight two, we have a triangle-free instance of WEIGHTED CLUSTER DELETION. We now show that this instance can be solved in polynomial time. The basis of this algorithm is the following claim:

> Let $G = (V, E)$ be a triangle-free graph, let $S \subseteq E$ be an edge set, and let $M := E \setminus S$. Then, $(V, E \setminus S)$ is a cluster graph $\Leftrightarrow$ M is a matching.

This claim can be seen as follows. Since G is triangle-free, any cluster graph that can be obtained by edge deletions has clusters of size at most two. Hence, the edges of this cluster graph are a matching. The converse is also true, since any two edges of a matching do not have an endpoint in common. Therefore, the graph that contains these edges and all vertices of the input graph is a cluster graph. Furthermore, since

$$\sum_{e \in S} \omega(e) = \sum_{e \in E} \omega(e) - \sum_{e \in M} \omega(e)$$

for $S \subseteq E$ and $M := E \setminus S$, minimizing the sum of the weights of the deleted edges is the same as maximizing the weight of the matching. Hence, we can compute an optimal solution for the triangle-free WEIGHTED CLUSTER DELETION instance by computing a maximum-weight matching M of G. This computation can be performed in $O(\sqrt{n}m)$ time [Micali and Vazirani 1980]. The overall running time is thus $O(\sqrt{n}n)$ since the procedure of replacing the triangles can be performed in $O(n)$ time and $m \leq 2n$. □

[3] The presented reduction rule is similar to previous approaches for CLUSTER EDITING that replace an unweighted instance by a weighted instance that works on the so-called critical clique graph [Böcker et al. 2009]. For the sake of completeness we include a short proof of correctness.

Altogether, we arrive at the following.

Theorem 2.5. CLUSTER DELETION *can be solved in* $O(\sqrt{n}n)$ *time when the input graph has maximum degree three.*

Proof. Given an instance of maximum degree three, we first exhaustively apply Rule 2.2 in $O(n)$ time. Then, we exhaustively apply Rule 2.1, also in $O(n)$ time. Note that the application of Rule 2.1 does not produce any triangle to which Rule 2.2 applies. Hence, the instance is reduced with respect to both rules. Consequently, Lemma 2.4 can be applied; the overall running time follows. □

2.2 Fixed-Parameter Tractability for the Combined Parameter "Number of Clusters and Local Modification Bound"

In the hardness results of the previous section, the number of clusters in the final cluster graph is unbounded. A natural question thus is: how does the number of clusters affect the computational complexity for instances that have a fixed local modification bound t? We answer this question 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 local modification bound t". We choose the following formulation to incorporate d and t into the problem:

> (d,t)-CONSTRAINED-CLUSTER EDITING:
> **Input:** An undirected graph $G = (V,E)$, a function $\tau : V \to \{0,\ldots,t\}$, and nonnegative integers d and k.
> **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 with at most $\tau(v)$ modified edges?

We use τ during our algorithm to keep track of the number of modifications that each vertex has been incident with. We can initially set $\tau(v) := t$ for each $v \in V$ and directly obtain the constraints posed by the local modification bound t. We refer to the corresponding problem in which only edge deletions are allowed as (d,t)-CONSTRAINED-CLUSTER DELETION. Clearly, CLUSTER EDITING is the same as (n,n)-CONSTRAINED CLUSTER EDITING with $\tau(v) = n$ for each $v \in V$. We present a set of polynomial-time data reduction rules to show the fixed-parameter tractability of (d,t)-CONSTRAINED-CLUSTER EDITING and (d,t)-CONSTRAINED-CLUSTER DELETION with respect to the combined parameter (d,t). Before presenting these rules, we discuss several aspects of the problem formulation and parameterization.

Concerning the problem formulation, in many application scenarios a reasonable upper bound for the number of clusters d is given in advance. Furthermore, the local modification bound t 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 better clusterings.

In this sense, (d,t)-CONSTRAINED-CLUSTER EDITING directly corresponds to a multi-criteria optimization problem where there is a trade-off between finding solutions that have small values of d, t, or k.

Concerning the parameterization, 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. An optimal solution for this graph is to delete all $|V|/2$ edges between K_1 and K_2. Hence, the parameter k is very large for this instance, whereas $d = 2$ and $t = 1$. 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 at least one edge modification is incident on each clique in the final cluster graph. In most application scenarios, the connected components of the input graph are processed independently from each other. Hence, we usually have $d \leq k+1$ for real-world instances. In summary, the parameters d and t can be arbitrarily small compared to k, are bounded from above by a linear function of k when G does not contain isolated cliques, and are usually smaller than k for real-world instances.

We now show that (d,t)-CONSTRAINED-CLUSTER EDITING is fixed-parameter tractable with respect to (d,t). More precisely, we present four data reduction rules for (d,t)-CONSTRAINED-CLUSTER EDITING that produce a problem kernel consisting of at most $4 \cdot dt$ vertices. The first two rules 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.

Reduction Rule 2.3. *If G contains two adjacent vertices $u, v \in V$ such that $|N(u) \setminus N[v]| > 2t$, then remove $\{u,v\}$ from E and set $\tau(v) := \tau(v) - 1$, $\tau(u) := \tau(u) - 1$, and $k := k - 1$.*

Reduction Rule 2.4. *If G contains two nonadjacent vertices $u, v \in V$ such that $|N(u) \cap N(v)| > 2t$, then add $\{u,v\}$ to E and set $\tau(v) := \tau(v) - 1$, $\tau(u) := \tau(u) - 1$, $k := k - 1$.*

Lemma 2.5. *Rules 2.3 and 2.4 are correct and can be exhaustively performed in $O(n^3)$ time.*

Proof. Let $(G = (V,E), d, t, k)$ be an input instance of (d,t)-CONSTRAINED-CLUSTER EDITING. We show the correctness of each rule and then bound the running time of exhaustively applying both rules.

Let u and v be as described in Rule 2.3. We show that every locally-t-bounded solution deletes the edge $\{u,v\}$. Suppose that there is a locally-t-bounded solution S that does not delete $\{u,v\}$, let G' be the cluster graph that results from applying S to G, and let K be the cluster of G' such that $u,v \in K$. Clearly, $|K \cap N(u) \setminus N[v]| \leq t$ since at most t inserted edges are incident with v. Then, however, more than t deleted edges are incident with u. This contradicts that S is a solution.

Let u and v be as described in Rule 2.4. We show that every solution adds the edge $\{u,v\}$. Suppose that there is some solution S that does not add $\{u,v\}$, let G' be the cluster graph that results from applying S to G, and let K be the cluster of G' such that $u \in K$ and $v \notin K$. Since at most t deleted edges are incident with u, we have $|N(u) \cap N(v) \cap K| > t$. Then, however more than t deleted edges are incident with v. This contradicts that S is a solution.

To achieve a running time of $O(n^3)$ we proceed as follows. First, we initialize for each pair of vertices $u,v \in V$ three counters, one counter that counts $|N(u) \cap N(v)|$, one counting $|N(u) \setminus N[v]|$, and one counting $|N(v) \setminus N[u]|$. For each such pair, this is doable in $O(n)$ time when an adjacency matrix has been constructed in advance. Hence, the overall time for initializing the counters for all possible vertex pairs is $O(n^3)$. All counters that warrant an application of either Rule 2.3 or Rule 2.4 are stored in a list. We call these counters *active*. Next, we apply the reduction rules. Overall, since $k \leq n^2$ the rules can be applied at most n^2 times. As long as the list of active counters is nonempty, we perform the appropriate rule for the first active counter of the list. It remains to update all counters according to the edge modification applied by the rule. Suppose Rule 2.4 applies to u and v, that is, $\{u,v\}$ is added. Then, we have to update the counters for each pair containing v or u. For v, this can be done in $O(n)$ time, by checking for each $w \neq v$, whether u must be added to $N(v) \cap N(w)$ or added to $N(v) \setminus N[w]$ or removed from $N(w) \setminus N[v]$ (for each counter this can be done in $O(1)$ time by using the constructed adjacency matrix). For each updated counter, we also check in $O(1)$ time whether it needs to be added to/removed from the list of active counters. The case that Rule 2.3 applies to u and v can be shown analogously. Overall, we need $O(n^3)$ time to initialize the counters and $O(n^3)$ time for the exhaustive application of the rules. □

The following reduction rule simply checks whether the instance contains vertices to which already more than t modifications have been applied. Clearly, in this case the instance is a no-instance.

Reduction Rule 2.5. *If there is a vertex $v \in V$ with $\tau(v) < 0$, then output "no".*

The final rule simply identifies isolated cliques that cannot be merged or split, and whose removal thus does not destroy solutions of (d,t)-CONSTRAINED-CLUSTER EDITING.

Reduction Rule 2.6. *If there is an isolated clique K in G such that $|K| > 2t$, then remove K from G and set $d := d - 1$.*

Lemma 2.6. *Rule 2.6 is correct and can be exhaustively performed in $O(m)$ time.*

Proof. The running time of the rule is obvious; for the correctness we show that K is a cluster of any cluster graph that can be obtained by a locally-t-bounded solution.

Since $|K| > 2t$, there is at least one vertex that is adjacent to at least t vertices of K in any cluster graph that can be obtained by a locally-t-bounded solution. Hence, there is a cluster K' of size at least $t+1$ that contains only vertices from K. Since every vertex from K that is not part of K' is incident with at least $t+1$ edge deletions, we have $K \subseteq K'$. Furthermore, we have $K' = K$ since adding a vertex $v \in V \setminus K$ to K causes at least $2k$ edge insertions that are incident with v. □

We now show that applying Rules 2.3–2.6 yields a problem kernel.

Theorem 2.6. (d,t)-CONSTRAINED-CLUSTER EDITING *admits a $4 \cdot dt$-vertex problem kernel which can be found in $O(n^3)$ time. It is thus fixed-parameter tractable with respect to the parameter (d,t).*

Proof. We first show the problem kernel size and then bound the running time of the kernelization.

Let $(G = (V,E), d, t, k)$ be an input instance of (d,t)-CONSTRAINED-CLUSTER EDITING and let G be reduced with respect to Rules 2.3–2.6. We show the following:

(G, d, t, k) is a yes-instance $\Rightarrow$ G has at most $4 \cdot dt$ vertices.

Let S be a solution of the input instance and let G' be the cluster graph that results from applying S to G. We show that every cluster K_i of G' has at most $4t$ vertices. Assume toward a contradiction that there is some K_i in G' with $|K_i| > 4t$. Since G is reduced with respect to Rule 2.6, there must be either an edge $\{u,v\}$ in G such that $u \in K_i$ and $v \in V \setminus K_i$ or a pair of vertices $u, v \in K_i$ such that $\{u,v\}$ is not an edge in G.

Case 1: $u \in K_i$, $v \in V \setminus K_i$ and $\{u,v\} \in E$. Since at most $t-1$ edge insertions are incident with u, it has in G at least $3t+1$ neighbors in K_i. Furthermore, since at most t edge deletions are incident with v, it has in G at most t neighbors in K_i. Hence, there are at least $2t+1$ vertices in K_i that are neighbors of u but not neighbors of v. Therefore, Rule 2.3 applies in G, a contradiction to the fact that G is reduced with respect to this rule.

Case 2: $u, v \in K_i$ and $\{u,v\} \notin E$. Both u and v are in G adjacent to at least $|K_i| - (t-1)$ vertices of $K_i \setminus \{u,v\}$. Since $|K_i| > 4t$ they thus have in G at least $2t+1$ common neighbors. Therefore, Rule 2.4 applies in G, a contradiction to the fact that G is reduced with respect to this rule.

We have shown that $|K_i| \leq 4t$ for each cluster K_i of G'. Since G' has at most d clusters, the overall bound on the number of vertices follows.

It remains to bound the running time of exhaustively applying Rules 2.3–2.6. By Lemma 2.5, the exhaustive application of Rules 2.3 and 2.4 runs in $O(n^3)$ time. After these two rules have been exhaustively applied, Rules 2.5 and 2.6 can be exhaustively applied in $O(m)$ time. □

The data reduction rules can be adapted to the case that only edge deletions are allowed. Indeed, we can show a $2 \cdot dt$-vertex problem kernel for (d,t)-CONSTRAINED-CLUSTER DELETION by replacing $2t$ by t in Rule 2.3 (note that Rule 2.4 is not suitable for CLUSTER DELETION since it adds an edge). More precisely, we have the following two reduction rules specifically for CLUSTER DELETION.

Reduction Rule 2.7. *If G contains two adjacent vertices $u, v \in V$ such that $|N(u) \setminus N[v]| > t$, then remove $\{u,v\}$ from E and set $\tau(v) := \tau(v) - 1$, $\tau(u) := \tau(u) - 1$, and $k := k - 1$.*

Lemma 2.7. *Rule 2.7 is correct and can be exhaustively applied in $O(n^3)$ time.*

Proof. The running time was already shown in the proof of Lemma 2.5. Hence, we only show the correctness of the rule.

Every locally-t-bounded solution deletes at most t edges incident with u. Hence, in the cluster graph that results from applying such a solution, u has at least one neighbor $w \notin N[v]$. Hence, the solution must also delete $\{u,v\}$. Otherwise the graph is not a cluster graph. □

The second rule deals with isolated clusters in G.

Reduction Rule 2.8. *If there is an isolated clique K in G, then remove K from G and set $d := d - 1$.*

The correctness of the rule follows from the simple observation that this isolated clique produces at least one cluster. Finally, we also apply Rule 2.5 in order to find vertices to which too many edge modifications have been applied. Altogether, the exhaustive application of these rules yields a $2 \cdot dt$-vertex problem kernel, as we show in the following.

Theorem 2.7. (d,t)-CONSTRAINED-CLUSTER DELETION *admits a $2 \cdot dt$-vertex problem kernel which can be found in $O(n^3)$ time. It is thus fixed-parameter tractable with respect to the parameter (d,t).*

Proof. The proof works in complete analogy to the proof of Theorem 2.6, the only difference is that we can show that every cluster of the cluster graph has at most $2t$ vertices instead of $4t$ vertices.

Let G be a graph that is reduced with respect to Rules 2.7, 2.8, and 2.5. We show that each cluster of every cluster graph that can be obtained by a locally-t-bounded

has size at most $2t$. Assume toward a contradiction that there is such a cluster graph that contains a cluster K that has more than $2t$ vertices. Since G is reduced with respect to Rule 2.8, there must be a pair of vertices $u \in K$ and $v \in V \setminus K$ such that $\{u, v\}$ is an edge in G. Since the solution is locally-t-bounded, v has in G at most t neighbors in K. Hence, u has in G more than t neighbors that are not neighbors of v. Therefore, Rule 2.7 applies, a contradiction to the assumption that G is reduced. □

2.3 Concluding Remarks

The presented hardness and tractability results provide a more detailed view on the computational complexity of CLUSTER EDITING and CLUSTER DELETION. Several open questions and research tasks concerning CLUSTER EDITING arise immediately from these results. In addition, we identify a number of further more methodological research questions that apply to graph modification problems in general.

- Concerning the NP-hardness of CLUSTER EDITING for graphs with bounded degree, achieving a complexity-dichotomy, as we now have for CLUSTER DELETION, would be desirable. We conjecture that CLUSTER EDITING on graphs with maximum degree three is solvable in polynomial time. For graphs with maximum degree four or five, we have no conjecture at the moment.

- Concerning the polynomial-time solvability of CLUSTER DELETION in graphs with maximum degree three, it would be interesting to extend this tractability result to the edge-weighted version. So far, we have only shown that this edge-weighted version is polynomial-time solvable on triangle-free graphs (in the proof of Theorem 2.5). While graphs with maximum degree three seem to be rather artificial input instances, this result could still be practically relevant since such graphs could be produced in the course of recursive search tree algorithms.

- Concerning the parameter "local modification bound t" several questions arise. For example, is CLUSTER EDITING polynomial-time solvable when the solution is locally-1-bounded? Another question is whether there are other graph modification problems for which this parameter yields fixed-parameter tractability? A good candidate seems to be the FEEDBACK ARC SET IN TOURNAMENTS problem, which appears to be "easier" than CLUSTER EDITING.[4]

- Concerning the combined parameter "number d of clusters and local modification bound t", developing a search tree algorithm would complement our problem kernelization results. Moreover, experimental studies should be performed to analyze what typical values of d and t are in real-world instances,

[4] For example, FEEDBACK ARC SET IN TOURNAMENTS can be solved in time that is subexponential in the size of the solution [Alon et al. 2009, Karpinski and Schudy 2010].

and to determine whether adding our data reduction rules provides a speed-up for some instances.

– Finally, further suitable parameterizations of CLUSTER EDITING should be explored. These could be structural graph parameters but also parameters that are related to the solution such as for example the parameter "number of edge deletions performed by the solution". This parameter could be considerably smaller than the parameter number of edge modifications.

Chapter 3

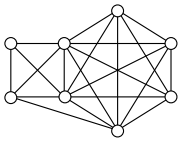
Clique Relaxation-Based Generalizations of Cluster Editing

In this chapter, we study the parameterized complexity of four generalizations of CLUSTER EDITING. The common feature of the problems under consideration is that the demand for the clusters to be cliques is replaced by other models for dense graphs. Formally, the problems can be seen as instances of the following class of problems:

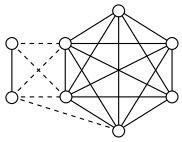
> Π-CLUSTER EDITING
> **Input:** An undirected graph $G = (V, E)$, a density property Π, and a nonnegative integer k.
> **Question:** Can G be transformed by up to k edge deletions and insertions into a Π-cluster graph, that is, a graph in which every connected component satisfies Π?

Formally, the density property Π can be any graph property. Setting Π :="the set of all cliques" we arrive exactly at CLUSTER EDITING. In our study, we consider four other graph properties for Π: *s-defective cliques*, *s-plexes*, *average-s-plexes*, and *μ-cliques*. These density properties are sometimes called "clique relaxations" since they pose weaker demands on the density of the graph than the clique model. Three of the considered clique relaxations have a "slack" parameter $s \geq 0$ that measures how much the clique demand is relaxed. For example, a graph $G = (V, E)$ is an s-plex if it has minimum degree $|V| - s$. For $s = 1$, this definition is equivalent to the clique definition; for $s > 1$, the density requirement is "relaxed".

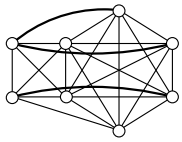
The remainder of this chapter is organized as follows. First, we give some motivation for the study of these relaxed variants of CLUSTER EDITING. Then, we present the considered problem variants, point to related work for the considered density properties, and summarize our findings. Afterwards, we present our results in detail for each of the studied variants.



(a) The input graph.



(b) An optimal solution for CLUSTER EDITING: two clusters arise.



(c) An optimal solution for 3-PLEX CLUSTER EDITING: one cluster arises.

Figure 3.1: An example of optimal solutions for two different cases of Π-CLUSTER EDITING. Dashed edges are deleted edges; edge insertions are bold.

Motivation. Although CLUSTER EDITING has been shown to yield high-quality clusterings, for example for protein similarity data [Wittkop et al. 2007, Rahmann et al. 2007] data, there are scenarios in which CLUSTER EDITING has drawbacks. In particular, applying CLUSTER EDITING to the simple undirected graph that underlies a protein interaction network typically does not result in a useful clustering. The problem is that CLUSTER EDITING deals poorly with the many low-degree vertices in scale-free protein interaction networks: In a minimum-cardinality solution of CLUSTER EDITING, each vertex v of the input graph ends up in a cluster of order at most $2 \cdot \deg(v) + 1$ (putting v in a larger cluster is always worse than putting it in a singleton cluster). For example, degree-one vertices end up in clusters of order at most three. Hence, many modules of the protein interaction network are "split" by a minimum-cardinality solution of CLUSTER EDITING, and the clustering produced by CLUSTER EDITING contains a core consisting of the vertices of high degree and many very small clusters that only contain vertices of low degree. The behavior of cutting away low-degree proteins from their biological modules has been identified as a weakness of several clustering algorithms for protein interaction networks [Inoue et al. 2010]. By choosing a density property that is less strict, one counteracts this behavior by allowing the clusters to contain some elements that have lower degree (see Figure 3.1). For example, in a minimum-cardinality solution for s-PLEX-CLUSTER EDITING a vertex may end up in clusters of order up to $2 \cdot \deg(v) + 2 - s$. There is clearly a trade-off between the "amount of relaxation" and clustering quality: if the density requirement is relaxed too much, then the complete protein-interaction network fulfills the requirement and thus the clustering algorithm outputs only one cluster containing all proteins in this case.

Another advantage of choosing clique relaxations as demand for the clusters lies in the observation that the number k of edge modifications usually decreases compared to CLUSTER EDITING.[1] This observation entails two positive aspects. First, it reflects

[1] More precisely, for each of the considered Π-CLUSTER EDITING problems, the minimum-cardinality solution is always at most as large as the minimum-cardinality solution for CLUSTER EDITING, since each of the considered clique relaxations is less strict than cliques.

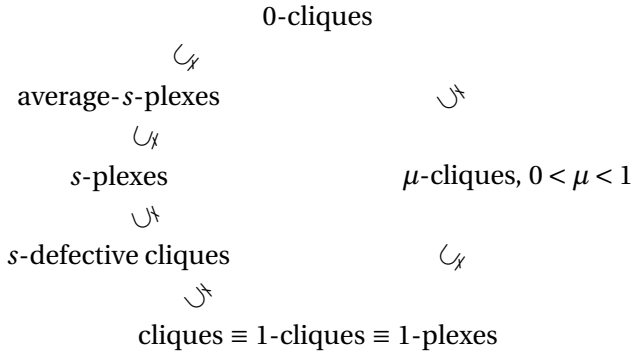


Figure 3.2: Inclusion relations between the considered density properties for fixed $s > 1$.

the premise that fewer edge modifications mean that we introduce fewer "errors" into our final cluster solution, because then the computed Π-cluster graph is closer to the original input graph. This is in accordance with the maximum parsimony principle: the natural hypothesis that the less one perturbs the input graph the more robust and plausible the achieved clustering is (also see Böcker et al. [2009] for making this point for CLUSTER EDITING). Second, it means that the number k of edge modifications becomes a better parameter since it is possibly much smaller. Next, we define each of the four density properties under consideration in this chapter.

Definitions of the Clique Relaxations. The first clique relaxation that we consider is to allow a fixed number s of missing edges in each cluster:

Definition 3.1. *A graph $G = (V, E)$ is an* s-defective clique *if G is connected and $|E| \geq |V| \cdot (|V| - 1)/2 - s$.*

We call $\{u, v\} \notin E$ a *missing edge* of G. A 0-defective clique is nothing but a clique. By increasing s, the clique model can be more and more relaxed. The concept of defective cliques has been used in biological networks to represent a clique with exactly one edge missing [Yu et al. 2006]. We generalize this notion[2] by allowing up to s missing edges.

The second considered clique relaxation is the s-plex notion which restricts the number of missing edges per vertex or, equivalently, the *minimum degree* for each vertex.

Definition 3.2. *A graph $G = (V, E)$ is an* s-plex *if the minimum vertex degree in G is at least $|V| - s$.*

[2] Yu et al. [2006] introduced a different generalization of defective cliques that is more restrictive than the one considered here.

Cliques are 1-plexes and vice versa. As for s-defective cliques, the density requirement can be more and more relaxed by increasing s. The s-plex concept has applications, for example, in social network analysis [Seidman and Foster 1978, Cook et al. 2007, Memon et al. 2007].

The third clique relaxation that we consider for the clusters is a variation of the s-plex notion that we call average-s-plex. In contrast to the s-plexnotion, which demands that the *minimum* degree of a graph $G = (V, E)$ be $|V| - s$, average-s-plexes restrict the *average* degree of the graph $G = (V, E)$.

Definition 3.3. *The* average degree *of a graph* $G = (V, E)$ *is defined as* $\bar{d} = 2|E|/|V|$. *A graph* $G = (V, E)$ *is an* average-s-plex *if its average degree* $\bar{d}$ G *is at least* $|V| - s$.

This density property is thus a relaxation of the s-plex notion. We are not aware of previous studies on average-s-plexes.

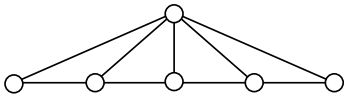
Finally, we consider a fourth density property, called μ-clique. Herein, the density parameter μ captures the ratio of edges in the graph versus the number of edges in a complete graph with the same number of vertices.

Definition 3.4. *The* density *of a graph* $G = (V, E)$ *is defined as* $2|E|/(|V|(|V| - 1))$. *A graph* $G = (V, E)$ *is a* μ-clique *for a rational constant* $0 \leq \mu \leq 1$ *if the density of* G *is at least* μ.

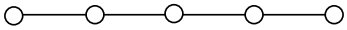
We assume that μ is represented by two constant integers a and b such that $\mu = a/b$. Observe that for $\mu = 0$ every graph is a μ-clique, and that a graph is a 1-clique if and only if it is a clique. The μ-clique concept was studied, for example, by Abello et al. [2002] and is also referred to as μ-dense graph [Kosub 2004].

For any fixed s, there is the following relationship between these density properties: An s-defective clique is also an s-plex, but not vice versa. An s-plex is also an average-s-plex, but not vice versa. The μ-clique definition does not directly compare to the other three density properties, since the density parameter μ does not, in contrast to s, measure the density in relation to the overall number of vertices. An overview of the inclusion relations between these four density properties is given in Figure 3.2.

Our Results. We show that for s-defective clique-cluster graphs, s-plexes, and average-s-plexes, the Π-CLUSTER EDITING problem is fixed-parameter tractable for the combined parameter (s, k). Informally, this means that we can extend the fixed-parameter tractability result of CLUSTER EDITING with respect to the parameter k as long as we also include the parameter s, which measures the "amount of relaxation" that is allowed. For the first two considered clique relaxations, s-defective cliques and s-plexes, it is possible to use the well-known technique of characterizing the respective target graphs via forbidden induced subgraphs, which then implies fixed-parameter tractability with respect to k via a general result due to Cai [1996]. In



(a) This graph is an average-4-plex since it has average degree more than 2.



(b) Deleting the degree-5 vertex results in a graph that is not an average-4-plex since its average degree is less than 1.

Figure 3.3: The property of being an average-s-plex is not hereditary.

contrast, for average-s-plexes this approach fails, since the property of being an average-s-plex is not hereditary, as shown in Figure 3.3. Hence, we follow a different approach here: we reduce the problem to a weighted version and then show that this weighted version can be reduced in polynomial time to an equivalent instance with at most $4k^2 + 8sk$ vertices. Finally, we show that for μ-cliques, the Π-CLUSTER EDITING problem is W[1]-hard with respect to the parameter k.

3.1 s-Defective Clique-Cluster Editing

In this section, we focus on the s-DEFECTIVE CLIQUE-CLUSTER EDITING problem: given a graph $G = (V, E)$, decide whether G can be transformed by at most k edge modifications into an *s-defective clique-cluster graph*, that is, a graph in which every connected component is an s-defective clique. The NP-hardness of s-DEFECTIVE CLIQUE-CLUSTER EDITING can be shown by reduction from CLUSTER EDITING [Guo et al. 2009a]. We show that s-defective clique-cluster graphs are characterized by forbidden induced subgraphs with at most $2(s+1)$ vertices. This characterization directly leads to fixed-parameter tractability of s-DEFECTIVE CLIQUE EDITING with respect to the parameter (s, k) by means of a search tree algorithm.

We start with some preliminaries. If we delete an arbitrary vertex of an s-defective clique-cluster graph, then the resulting graph is still an s-defective clique-cluster graph. Hence, the property of being an s-defective clique-cluster graph is hereditary. It can thus be described by a set of forbidden induced subgraphs. A graph H is a *minimal forbidden induced subgraph* for s-defective clique-cluster graphs if H is not an s-defective clique-cluster graph but every induced proper subgraph of H is an s-defective clique-cluster graph. Clearly, a graph is an s-defective clique-cluster graph if and only if it contains no minimal forbidden induced subgraph. Next, we show that, for $s \geq 1$, every minimal forbidden induced subgraph of s-defective clique-cluster graphs contains at most $2(s+1)$ vertices. Recall that for $s = 0$ the only forbidden induced subgraph is the path on three vertices [Gramm et al. 2005, Shamir et al. 2004].

Theorem 3.1. *For $s \geq 1$, every minimal forbidden induced subgraph of s-defective*

clique-cluster graphs contains at most $2(s+1)$ *vertices. Given a graph that is not an* s*-defective clique-cluster graph, a minimal forbidden induced subgraph can be found in* $O(nm)$ *time.*

Proof. Assume toward a contradiction that there exists a minimal forbidden induced subgraph $H = (W, F)$ with $|W| > 2(s+1)$. Clearly, we can assume that H is connected, since otherwise we can keep one connected component that is not an s-defective clique and delete all other connected components.

First, we consider the case that H contains a cut-vertex v. Let U denote a set of $s+2$ vertices which together with v induce a connected graph $H' := H[U \cup \{v\}]$. Clearly, v remains a cut-vertex in H'. We show that H' is not an s-defective clique-cluster graph, a contradiction to the minimality of H. Let $U_1, \ldots, U_\ell$ denote the connected components of $H' - v$. At least $(\sum_{i=1}^{\ell} |U_i| \cdot (|U \setminus U_i|))/2$ edges are missing in H'. This sum is minimal for $\ell = 2$ and $|U_1| = 1$, in which case it is $s+1$. Hence, it is always larger than s and thus H' is not an s-defective clique-cluster graph. Since $s+3 \leq 2(s+1)$ for $s \geq 1$, H' is a proper subgraph of H which contradicts the minimality of H.

In the following, we assume that H does not contain any cut-vertex. Moreover, we can assume that no vertex v of H is adjacent to all other vertices of H, since otherwise we can delete v to obtain a connected graph that has the same number of missing edges as H, a contradiction to the minimality of H. Hence, there are at least $s+2$ missing edges in H, since every vertex is incident with at least one missing edge. Let v be an arbitrary vertex of H and let $A := W \setminus N[v]$. Since the deletion of v results in an s-defective clique-cluster graph, it follows that in $H - v$ there are at most s missing edges. Hence, there exists a vertex u that is adjacent to all vertices of $H - v$. Clearly, $u \in A$, since, otherwise, u would be adjacent to all vertices in G. Then, the deletion of u reduces the number of missing edges by one. Hence, $H - u$ is connected and has at least $s+1$ missing edges. Consequently, $H - u$ is not an s-defective clique-cluster graph, a contradiction to the minimality of H.

To find a minimal forbidden induced subgraph proceed as follows. Given a graph $H = (W, F)$ that is not an s-defective clique-cluster graph, check for every $v \in W$ whether $H - v$ is an s-defective clique-cluster graph in $O(n+m)$ time and delete v if this is not the case. Observe that we have to consider every vertex at most once, since if $H - v$ is an s-defective clique-cluster graph, then $H' - v$ is an s-defective clique-cluster graph for every induced subgraph H' of H containing v. Hence, the overall running time is $O(nm)$. □

The forbidden subgraph characterization given in Theorem 3.1 directly leads to fixed-parameter tractability with respect to the parameter (s, k) [Cai 1996]. More precisely, we can use the following search tree algorithm to find a solution of size at most k. As long as the given graph is not an s-defective clique-cluster graph find a minimal forbidden induced subgraph and branch into all cases (at most $\binom{2s+1}{2}$) to destroy this subgraph by inserting or deleting an edge between two of its vertices.

Since in each case we can decrease the parameter k by one, the size of the search tree is $O\left(\binom{2s+1}{2}^k\right)$. Putting all together, we arrive at the following.

Theorem 3.2. s-DEFECTIVE CLIQUE-CLUSTER EDITING *can be solved in* $O\left(\binom{2s+1}{2}^k \cdot nm\right)$ *time and hence is fixed-parameter tractable with respect to the parameter* (s,k).

The bound given in Theorem 3.2 should be seen as a first step: it shows fixed-parameter tractability but it can certainly be improved. For the next problem, s-PLEX-CLUSTER EDITING, we actually show how such an improvement can be performed.

3.2 s-Plex-Cluster Editing

In this section, we study s-PLEX-CLUSTER EDITING: given a graph $G = (V,E)$, decide whether G can be transformed by at most k edge modifications into an *s-plex cluster graph*, that is, a graph in which every connected component is an s-plex. The NP-hardness of s-PLEX-CLUSTER EDITING can be shown by slightly modifying an NP-hardness proof for CLUSTER EDITING [Guo et al. 2010b]. First, in Section 3.2.1, we show that s-plex cluster graphs can be characterized by minimal forbidden induced subgraphs with $O(s)$ vertices. Then, in Section 3.2.2, we show that a forbidden induced subgraph can be found in $O(n \cdot m)$ time. Altogether, this means that s-PLEX-CLUSTER EDITING is fixed-parameter tractable with respect to (s,k) by applying Cai's theorem [Cai 1996]. We then demonstrate two improvements of this first classification result. First, we show how to improve the running time for finding an induced forbidden subgraph to $O(s \cdot (n+m))$. Next, in Section 3.2.3, we present a—compared to the brute-force branching that stands behind Cai's theorem [Cai 1996]—more efficient branching strategy for s-PLEX-CLUSTER EDITING that leads to a search tree of size $O((2s+\lfloor\sqrt{s}\rfloor)^k)$.

3.2.1 Description of the Forbidden Induced Subgraphs

Analogously to s-defective clique-cluster graphs, we say that a graph H is a *minimal forbidden induced subgraph* if it is not an s-plex cluster graph but every induced proper subgraph of H is an s-plex cluster graph. In the following, our goal is to identify and describe the set $\mathcal{F}_{s,\min}$ of all minimal forbidden induced subgraphs for s-plex cluster graphs. More precisely, we first give a graph-theoretic description of the minimal forbidden induced subgraphs. Then, we show that the number of vertices in every minimal forbidden induced subgraph is upper-bounded by $s + t_s + 1$, where

$$t_s := \lfloor -0.5 + \sqrt{0.25 + s} \rfloor.$$

Note that for a nonnegative integer i it holds that $i \cdot (i+1) \leq s$ if and only if $i \leq t_s$.

Finally, we show that the upper bound of $s + t_s + 1$ on the number of vertices of a minimal forbidden induced subgraph is tight. That is, we show that for every $s \geq 2$ there exist minimal forbidden induced subgraphs with exactly this number of vertices.

We begin with a graph-theoretic description of the minimal forbidden induced subgraphs. The starting point are the connected graphs that contain a vertex that is not adjacent to exactly s other vertices. These graphs clearly are not s-plex cluster graphs. Let $\mathcal{C}$ denote the set of connected graphs. Define

$$\mathcal{C}(s,i) := \{H = (W,F) \in \mathcal{C} : (|W| = s + i + 1) \wedge (\exists w \in W : \deg_H(w) = i)\}$$

and

$$\mathcal{F}(s,j) := \bigcup_{i=1}^{j} \mathcal{C}(s,i).$$

Next, motivating the definitions of $\mathcal{C}(s,i)$ and $\mathcal{F}(s,j)$, we show that all minimal forbidden induced subgraphs are contained in $\mathcal{F}(s,n-s-1)$. Then we refine this characterization by showing that a graph from $\mathcal{C}(s,i)$ is minimal if and only if its minimum vertex degree is i and all neighbors of a degree-i vertex are cut-vertices.

Lemma 3.1. *G is an s-plex cluster graph $\Leftrightarrow$ G is $\mathcal{F}(s,n-s-1)$-free.*

Proof. $\Rightarrow$: Since the property of being an s-plex cluster graph is hereditary, all induced subgraphs of G are s-plex cluster graphs. Hence, G is $\mathcal{F}(s,n-s-1)$-free since the graphs in $\mathcal{F}(s,n-s-1)$ are not s-plex cluster graphs.

$\Leftarrow$: We show the contraposition. If G is not an s-plexcluster graph, then G contains a connected component $C = (W,F)$ that is not an s-plex. Since every connected component with at most $s+1$ vertices is an s-plex, we have $|W| \geq s+2$. Consider a vertex $v \in W$ of minimum degree. Since W does not form an s-plex, it contains at least s vertices that are not adjacent to v. Hence, we can find an induced subgraph of G from $\mathcal{C}(s,\deg_G(v)) \subseteq \mathcal{F}(s,n-s-1)$ using breadth-first search starting at v. □

In the following lemma, we precisely describe the minimal forbidden induced subgraphs.

Lemma 3.2. *A graph $H \in \mathcal{C}(s,i)$ is a minimal forbidden induced subgraph for s-plex-cluster graphs $\Leftrightarrow$ the minimum vertex degree of H is i and for every $v \in V(H)$ with $\deg_H(v) = i$ the neighbors of v are cut-vertices.*

Proof. $\Rightarrow$: We show the contraposition. First, assume that H contains a vertex v with $\deg_H(v) < i$. Then, by breadth-first search starting at v, we can find a set $S \subset V(H)$ such that $H[S] \in \mathcal{C}(s,\deg(v))$. Hence, H is not minimal. Second, assume that there exists a degree-i vertex v with a neighbor u that is not a cut-vertex. Then, by deleting u we obtain a graph from $\mathcal{C}(s,i-1)$, that is, H is also not minimal.

$\Leftarrow$: Consider an arbitrary vertex $v \in V(H)$. We show that $H' := H - v$ is an s-plex cluster graph. Note that H' contains $s+i$ vertices and the minimum vertex degree

in H' is $i-1$. First, all vertices $w \in V(H')$ with $\deg_{H'}(w) \geq i$ are not adjacent to at most $s-1$ other vertices. Second, consider an arbitrary degree-$(i-1)$ vertex $u \in V(H')$. Note that u is a neighbor of v in H (since the minimum degree in H is i). Therefore, v is a cut-vertex in H and the deletion of v separates from u at least one of the s vertices nonadjacent to u in H. As a consequence, u is not adjacent to at most $s-1$ other vertices in the connected component of H' containing u. In summary, every vertex $w \in V(H')$ is not adjacent to at most $s-1$ other vertices in the connected component in H' in which it is contained. Hence, H' is an s-plex cluster graph. □

Based on the description of the minimal forbidden induced subgraphs given in Lemma 3.2, we show that the number of vertices in a minimal forbidden induced subgraph is bounded by $O(s)$. To this end, we show that the set $\mathcal{F}_{s,\min}$ of all minimal forbidden induced subgraphs is contained in $\mathcal{F}(s,t_s)$, that is, the number of vertices of every minimal forbidden induced subgraph is upper-bounded by $s+t_s+1$.

Theorem 3.3. *A graph G is an s-plex-cluster graph $\Leftrightarrow$ G is $\mathcal{F}(s,t_s)$-free.*

Proof. It is sufficient to show that all minimal forbidden induced subgraphs are contained in $\mathcal{F}(s,t_s)$. Assume toward a contradiction that there exists a minimal forbidden induced subgraph H not contained in $\mathcal{F}(s,t_s)$. By Lemma 3.1, $H \in \mathcal{F}(s, n-s-1)$. This implies that $H \in \mathcal{C}(s,i)$ for some i with $i \cdot (i+1) > s$ (or, equivalently, $i > t_s$). Since H is a minimal forbidden induced subgraph, according to Lemma 3.2, the minimum vertex degree of H is i. Let $v \in V(H)$ be a degree-i vertex. Note that, according to Lemma 3.2, all neighboring vertices $N_H(v) = \{u_1, u_2, \ldots, u_i\}$ of v are cut-vertices. For every neighboring vertex u_j of v let U_j denote the set of vertices in $V(H)$ that are not reachable from v in $H - u_j$. On the one hand, note that $|U_j| \geq i$ for every $1 \leq j \leq i$ since the minimum vertex degree of H is i and since for every vertex $w \in U_j$ it holds that $N_H(w) \subseteq (U_j \cup \{u_j\}) \setminus \{w\}$. On the other hand, since v has degree $i = |V(H)| - s - 1$ in H, we have $\sum_{j=1}^{i} |U_j| \leq s$. Hence, there must exist at least one r, $1 \leq r \leq i$, with $|U_r| \leq s/i < i \cdot (i+1)/i = i+1$. Therefore, $|U_r| = i$. Moreover, since the minimum vertex degree in H is i, $U_r \cup \{u_r\}$ forms a clique of size $i+1$ and thus by deleting all but one vertex of U_r we obtain a graph in $\mathcal{C}(s,1)$ which is by definition not an s-plex cluster graph. This contradicts the assumption that H is a minimal forbidden induced subgraph. □

So far, we have shown that the number of vertices of every minimal forbidden induced subgraph is at most $s+t_s+1$. Clearly, this implies that the number of minimal forbidden induced subgraphs is upper-bounded by a function of s. The number of minimal forbidden induced subgraphs may, however, be exponential in s. Note that the maximum number of vertices in a minimal forbidden induced subgraph is of greater algorithmic importance than the exact number of the minimal forbidden induced subgraphs: providing a smaller upper bound on the number of vertices of

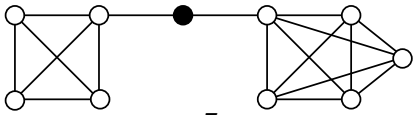


Figure 3.4: A minimal forbidden induced subgraph on 10 vertices for 7-plex cluster graphs. The black vertex is a vertex with minimum degree and all its neighbors are cut-vertices.

minimal forbidden induced subgraph would directly improve the exponential part of the running time of the search tree algorithms presented in Section 3.2.3. However, we show that the upper bound of $s + t_s + 1$ on the number of vertices in a minimal forbidden induced subgraph is tight. To this end, we show that for every $s \geq 2$ there is at least one minimal forbidden induced subgraph which is contained in $\mathcal{C}(s, t_s)$.

Consider the graph shown in Figure 3.4. It contains 10 vertices and the black vertex has degree two. Hence, this graph is contained in $\mathcal{C}(7,2)$ (note that $t_7 = 2$). Moreover, the black vertex is the only vertex with degree two and all its neighbors are cut-vertices. Hence, according to Lemma 3.2 the shown graph is a minimal forbidden induced subgraph for $s = 7$. This example can be generalized to every $s \geq 2$. That is, for every s we can construct a minimal forbidden induced subgraph contained in $\mathcal{C}(s, t_s)$ as follows. We start with a star with center vertex v and t_s leaves, say $u_1, \ldots, u_{t_s}$. Then, for every u_j we add a clique C_j in such a way that $\sum_{j=1}^{t_s} |C_j| = s$ and that all cliques are of same size plus/minus one. Finally, we make all vertices in C_j adjacent to u_j, $1 \leq j \leq t_s$. Since $\sum_{j=1}^{t_s} |C_j| = s$ (and $t_s \cdot (t_s + 1) \leq s$ by definition), we have $|C_j| \geq t_s + 1$, and, as a consequence, v is the only vertex of degree t_s and the degree of all other vertices is at least $t_s + 1$. Since the deletion of u_j separates the vertices in C_j from v, all neighbors of v are cut-vertices. Hence, by Lemma 3.2, the constructed graph is a minimal forbidden induced subgraph.

Summarizing, we arrive at the following.

Proposition 3.1. *For every $s \geq 2$ there exists a minimal forbidden induced subgraph contained in $\mathcal{C}(s, t_s)$.*

Consequently, the forbidden induced subgraph characterization achieved in Theorem 3.3 is tight.

3.2.2 Finding a Minimal Forbidden Induced Subgraph

In this subsection, we focus on efficiently finding a minimal forbidden induced subgraph. We show that, given a graph G that is not an s-plex cluster graph, a minimal forbidden induced subgraph can be found in $O(s \cdot (n + m))$ time.

We first present an algorithm (Algorithm A, see Figure 3.5) that, given a graph $H = (W, F) \in \mathcal{C}(s, i)$ (for some $i \geq 1$), checks in $O(|W| + |F|)$ time whether H is a minimal

Input: $H = (W, F)$ from $\mathcal{C}(s, i)$.
Output: "yes", if H is a minimal forbidden subgraph;
an induced subgraph $H' \in \mathcal{C}(s, i')$ of H with $i' < i$, otherwise.

1 $v := \arg\min_{w \in W}\{\deg_H(w)\}$
2 **if** $\deg_H(v) < i$ **then**
3 **return** a connected graph induced by $N_H[v] \cup S$
where S contains s vertices from $W \setminus N_H[v]$
4 Let Cutvertices be the set of cut-vertices of H
5 **if** $\exists v \in W : (N_H(v) \setminus \text{Cutvertices}) \neq \emptyset$ **then**
6 **return** graph $H - w$ for an arbitrary $w \in (N_H(v) \setminus \text{Cutvertices})$
7 **return** "yes"

Figure 3.5: Algorithm A checks whether a forbidden induced subgraph is minimal. If not, it computes a smaller forbidden induced subgraph.

forbidden induced subgraph. If this is the case, then it outputs "yes", otherwise it returns an induced subgraph of H contained in $\mathcal{C}(s, i')$ with $i' < i$.

Lemma 3.3. *Let $H = (W, F) \in \mathcal{C}(s, i)$. Algorithm A (Figure 3.5) computes in $O(|W| + |F|)$ time an induced subgraph $H' \in \mathcal{C}(s, i')$ of H, with $i' < i$, or outputs "yes" if H is a minimal forbidden induced subgraph.*

Proof. Consider lines 1 to 3 of the algorithm. If a vertex v in H has degree less than i, then we can clearly find a graph from $\mathcal{C}(s, \deg_H(v))$ by choosing $N_H[v]$ and a set $S \subseteq W \setminus N_H[v]$ of s further (arbitrary) vertices such that $H[N_H[v] \cup S]$ is connected. This is doable in linear time by breadth-first search starting at v.

Consider lines 4 to 6. If one of the neighboring vertices of v, say w, is no cut-vertex, then we can delete w from H obtaining a graph from $\mathcal{C}(s, i-1)$. Note that cut-vertices can be computed in linear time [Tarjan 1972].

Consider line 7. The minimum vertex degree of H is i and the neighbors of every degree-i vertex are cut-vertices. Hence, according to Lemma 3.2, H is a minimal forbidden induced subgraph. □

Algorithm A can be used iteratively to find a minimal forbidden induced subgraph. When starting with an arbitrary graph contained in $\mathcal{F}(s, n-s-1)$, Algorithm A has to be applied up to n times in the worst case, resulting in an overall running time of $O(n \cdot m)$ for finding a minimal forbidden induced subgraph. Combining this running time for finding a minimal forbidden induced graph with Cai's theorem [Cai 1996] leads to the following.

Proposition 3.2. *s-Plex-Cluster Editing can be solved in $O(\binom{s+t_s+1}{2}^k \cdot nm)$ time.*

Input: $H = (W, F)$ from $\mathcal{C}(s, i)$ with $i > s$
Output: An induced subgraph $H' \in \mathcal{C}(s, i')$ of H with $i' \leq s$

1 $u := \arg\min_{w \in V}\{\deg_H(w)\}$
2 **if** $\deg_H(u) \leq s$ **then**
3 **return** a connected graph induced by $N_H[u] \cup S$
where S contains s vertices from $W \setminus N_H[u]$
4 Let $v \in V$ be a vertex with $\deg_H(v) = i$
5 $N_H(v) := \{u_1, u_2, \ldots, u_i\}$
6 Let $K := \{K_1, K_2, \ldots, K_l\}$ with $l \leq s$
denote the connected components of $H - N_H[v]$
7 Construct an auxiliary bipartite graph $B = (X_N, X_K, R)$ with
8 $X_N := \{x_{u_j} \mid 1 \leq j \leq i\}$,
9 $X_K := \{x_{K_q} \mid 1 \leq q \leq l\}$, and
10 $R := \{\{x_{u_j}, x_{K_q}\} \mid \exists\{u_j, v'\} \in F$ with $v' \in K_q\}$
11 $r := \arg\min_{q \in \{1,\ldots,l\}}\{\deg_B(x_{K_q})\}$
12 $CC := \{u_j \mid x_{u_j} \in N_B(x_{K_r})\}$
13 $\hat{H} := H - (CC \setminus \{w\})$ for an arbitrary vertex $w \in CC$
14 $v' := \arg\min_{w \in V(\hat{H})}\{\deg_{\hat{H}}(w)\}$
15 **return** a connected graph induced by $N_{\hat{H}}[v'] \cup S$
where S contains s vertices from $V(\hat{H} \setminus N_{\hat{H}}[v']$

Figure 3.6: Algorithm B to compute a forbidden induced subgraph with $O(s)$ vertices.

This can be significantly improved as we show in the following. First, we show that the polynomial part of the running time can be improved. In particular, we show how to achieve, for constant s, linear running time for finding a minimal forbidden induced subgraph. To this end, we develop an algorithm (Algorithm B, see Figure 3.6) that, given a forbidden induced subgraph $H = (W, F)$ from $\mathcal{F}(s, n-s+1)$, finds in $O(s \cdot (|W| + |F|))$ time a forbidden induced subgraph from $\mathcal{F}(s, s)$. Since the number of vertices in such a subgraph is $O(s)$, we can then apply Algorithm A iteratively $O(s)$ times to obtain a minimal forbidden induced subgraph.

Lemma 3.4. *Let $H = (W, F) \in \mathcal{C}(s, i)$ with $i > s$. Algorithm B (Figure 3.6) computes in $O(s \cdot (|W| + |F|))$ time an induced subgraph $H' \in \mathcal{C}(s, i')$ of H with $i' \leq s$.*

Proof. Consider lines 1 to 3. If $\deg_H(u) \leq s$, then we can clearly find a set $S \subset W \setminus N_H[u]$ of s vertices such that $H[N_H[u] \cup S]$ is connected and $|S| = s$. This graph is in $\mathcal{C}(s, i')$ for some $i' \leq s$.

In the following, let v denote a vertex with degree i (line 4). We use the following observation:

> If one of the neighboring vertices of a degree-i vertex v is a cut-vertex, then there exists at least one vertex in H with degree at most s.

This can be seen as follows. Assume that $x \in N_H(v)$ is a cut-vertex and let $U \subseteq W$ denote the vertices not reachable from v in $H - x$. Since a vertex $w \in U$ can only be adjacent to vertices in $U \cup \{x\}$ and $|U| \leq s$ (by definition of $\mathcal{C}(s,i)$), we have $\deg_H(w) \leq s$.

According to this observation, when entering line 5 of Algorithm B, we know that none of the vertices in $N_H(v) = \{u_1, u_2, \ldots, u_i\}$ is a cut-vertex. To make use of the observation, the remaining part of the algorithm is devoted to finding a set of vertices from $N_H(v)$ whose removal leads to a connected graph in which one neighbor of v is a cut-vertex. To this end, one constructs an auxiliary bipartite graph $B = (X_N, X_K, R)$ (lines 5-10). Concerning the running time needed for the construction of B, note that the degree of a vertex in X_N is at most s since $H - N_H[v]$ contains exactly s vertices and, hence, X_K has size at most s. Thus, to define R, one iterates over the edge set F and, given an edge $\{u_j, v'\}$ with $v' \in K_q$, one can decide in $O(s)$ time whether the edge $\{x_{u_j}, x_{K_q}\}$ is contained in R. Thus, the bipartite auxiliary graph B can be constructed in $O(s \cdot (|W| + |F|))$ time.

Consider lines 11 to 13. By choosing a "component vertex" x_{K_r} of minimum degree, we ensure that the set CC is a minimum-cardinality set of vertices from $N_H(v)$ separating at least one connected component in K from v. That is, CC separates the vertices in K_r from v. Let w be an arbitrary vertex of CC. By the deletion of all but one vertex from CC (line 13), we ensure that the graph $\hat{H} = H - (CC \setminus \{w\})$ is still connected and contains at least one cut-vertex, namely w. Hence, according to the observation above, $\hat{H}$ contains a vertex of degree at most s. Let v' be a minimum-degree vertex of $\hat{H}$ (line 14). As a consequence, $\deg_{\hat{H}}(v') \leq s$ and we can clearly find a set $S \subseteq V(\hat{H})$ of s vertices such that $H' := \hat{H}[N_{\hat{H}}[v'] \cup S]$ is connected. Note that H' is contained in $\mathcal{C}(s, \deg_{\hat{H}}(v')) \subseteq \mathcal{F}(s,s)$. Altogether, the running time is $O(s \cdot (|W| + |F|))$. □

Summarizing, we obtain a linear-time algorithm for finding a minimal forbidden induced subgraph if s is a constant. In particular, this means we can find a forbidden induced subgraph comprising at most $s + t_s + 1$ vertices in linear time.

Theorem 3.4. *Let $G = (V,E)$ be a graph that is not an s-plex cluster graph. Then, a minimal forbidden induced subgraph can be found in $O(s \cdot (n+m))$ time.*

Proof. Let $C = (W,F)$ be a connected component of G that is not an s-plex. Let v be a vertex of minimum degree in C. Clearly, by breadth-first search starting at v we can find a set $S \subseteq W$ of s vertices such that $H := G[N_G[v] \cup S]$ is connected. Note that $H \in \mathcal{C}(s, \deg_H(v))$. If $\deg_H(v) > s$, then we can apply Algorithm B (Figure 3.6) once to find a forbidden induced subgraph H' from $\mathcal{F}(s,s)$. In order to find a minimal forbidden induced subgraph, we apply Algorithm A (Figure 3.5) at most $O(s)$ times. Hence, the overall running time is $O(s \cdot (n+m))$. □

3.2.3 A Refined Search Tree Algorithm

We present a search tree algorithm that is based on the forbidden subgraph characterization from Section 3.2.1 and yields an improved exponential running time part compared to Proposition 3.2. The idea is to find a minimal forbidden induced subgraph, and to consider all possibilities to destroy this subgraph. In comparison to the algorithm behind Proposition 3.2, we present a more detailed look at the forbidden induced subgraph and the possibilities to destroy this subgraph. This results in a smaller search tree size.

Theorem 3.5. *s-PLEX-CLUSTER EDITING can be solved in $O((2s + \lfloor\sqrt{s}\rfloor)^k \cdot s \cdot (n + m))$ time.*

Proof. Given an instance (G, k) of s-PLEX CLUSTER EDITING, we search in G for a minimal forbidden induced subgraph from $\mathcal{F}(s, t_s)$. By Theorem 3.4, this can be done in $O(s \cdot (n + m))$ time. If G does not contain an induced subgraph from $\mathcal{F}(s, t_s)$, then G already is an s-plex cluster graph and we are done. Otherwise let S be a set of vertices inducing a forbidden subgraph $G[S] \in \mathcal{C}(s, i) \subseteq \mathcal{F}(s, t_s)$, where $i \leq t_s$. In the following, let v denote a vertex with $\deg_{G[S]}(v) = i$. By the definition of $\mathcal{C}(s, i)$, such a vertex must exist. Branch into the different possibilities to destroy the forbidden induced subgraph $G[S]$ and then recursively solve the instances that are created in the respective search tree branches. The branching stops when $k \leq 0$.

For branching, either insert edges incident with v or delete edges in $G[S]$. It is sufficient to only consider these edge modifications since, if none of these is performed, then $G[S]$ remains connected and there are s vertices in $G[S]$ that are not adjacent to v, contradicting the s-plex-cluster graph definition.

First, consider edge insertions between v and vertices $u \in S \setminus N[v]$. Since $G[S] \in \mathcal{C}(s, i)$ and v has degree i in $G[S]$, we have $|S \setminus N[v]| = s$. Therefore, branch into s cases, inserting a different edge in each search tree branch. The parameter decreases by 1 in each branch.

Next, consider edge deletions. In each remaining branch, there is at least one vertex $u \in S$ such that u and v become disconnected, that is, they are in different connected components of the final s-plex cluster graph. We now show that for each $u \in S$ it is sufficient to create one search tree branch in which at least one edge deletion is performed for the case that u and v are not connected in the final cluster graph. Let $S_l \subset S$ denote the vertices that have distance exactly l to v in $G[S]$. We first consider the vertices in S_1 (the neighbors of v in $G[S]$), then the vertices in S_2, and so on.

For each $u \in S_1$, create a search tree branch in which one disconnects u and v. Clearly this means that one has to delete the edge $\{u, v\}$. To branch on the vertices in S_2, one can assume that the vertices from $N[v] = \{v\} \cup S_1$ end up in the same cluster, since we have already considered all possibilities of removing edges between v and

the vertices in S_1. Therefore, when considering the case that a vertex $u \in S_2$ and v are not connected in the final cluster graph, one must delete all edges between u and its neighbors in S_1. At least one such edge must exist because $u \in S_2$. Therefore, for each case, one creates a search tree branch in which the parameter is decreased by at least 1.

The case distinction is performed for increasing values of l, always assuming that v and the vertices in $S_1 \cup S_2 \cup \ldots \cup S_{l-1}$ end up in the same cluster of the final cluster graph. Hence, when considering the case that v and a vertex $u \in S_l$ end up in different clusters, one creates a search tree branch in which the edges between u and its neighbors in S_{l-1} are deleted, and at least one of these edges must exist. Hence, one creates $|S| - 1 = s + i \leq s + t_s$ branches in which edges are deleted. Together with the s cases in which edge insertions are performed, one branches into $2s + i$ cases, and in each branch, the parameter is decreased by at least 1. Branching is performed only as long as $k > 0$. Hence, the search tree has size $O((2s + t_s)^k) = O((2s + \lfloor\sqrt{s}\rfloor)^k)$, since $t_s = \lfloor -0.5 + \sqrt{0.25 + s}\rfloor$. Using breadth-first search, the steps at each search tree node can be performed in $O(s \cdot (n+m))$ time which results in the claimed running time bound. □

A further way to improve the algorithm for s-PLEX-CLUSTER EDITING is to develop a kernelization for s-PLEX-CLUSTER EDITING. This is indeed possible, as s-PLEX-CLUSTER EDITING admits a problem kernel of at most $(8s^2-6)\cdot k+8(s-1)^2$ vertices that can be computed in $O(n^4)$ time [Guo et al. 2010b]. Combining this problem kernel with the search tree of Theorem 3.5 and the technique of interleaving search tree and problem kernelization [Niedermeier and Rossmanith 2000] leads to the following running time.

Theorem 3.6. *s-PLEX-CLUSTER EDITING can be solved in $O((2s + \lfloor\sqrt{s}\rfloor)^k + n^4)$ time.*

The advantage of the running time of Theorem 3.6 is that the exponential running time part is now added (instead of multiplied) to the polynomial running time part. Theorem 3.5, however, gives linear running time for constant values of s and k.

3.3 Average-s-Plex-Cluster Editing

Here, we consider the AVERAGE-s-PLEX-CLUSTER EDITING problem, which is NP-hard for every constant $s \geq 1$ [Guo et al. 2009a]. As already shown in Figure 3.3, being an average-s-plex graph is not a hereditary graph property. Hence, the fixed-parameter tractability of AVERAGE-s-PLEX-CLUSTER EDITING cannot be shown by a forbidden subgraph characterization as in the case of s-DEFECTIVE CLIQUE-CLUSTER EDITING and s-PLEX-CLUSTER EDITING. Hence, we follow a different approach to show fixed-parameter tractability with respect to (s, k). Our algorithm consists of two main steps.

First, we reduce the original problem to a weighted version. Then, we show the fixed-parameter tractability of the weighted version by describing two polynomial-time data reduction rules that yield instances which contain at most $4k^2 + 8sk$ vertices. Note that formally these instances are not problem kernels since they can have unbounded weights.

We begin with describing a weighted version of AVERAGE-s-PLEX-CLUSTER EDITING. We introduce three types of weights: two vertex weights and one edge weight. The idea behind these weight types is the following: whenever there are two vertices in G that cannot be separated by at most k edge modifications, we can merge them into a new "super-vertex" since it is clear that they must end up in the same connected component of the solution. We say that a super-vertex v "comprises" a vertex u of the input graph if u is merged into v. When doing so, we must remember for each such super-vertex v

- how many vertices of the input graph v comprises,
- how many edges there are between the vertices that v comprises, and
- for each vertex w not comprised by v, how many vertices that v comprises are adjacent to w.

The first two aspects can be remembered by introducing two weights for v where

- $\sigma(v)$ keeps track of the number of vertices comprised by v, and
- $\delta(v)$ keeps track of the number of edges between these vertices.

The third aspect can be stored as the edge weight $\omega(e)$ for the edge $e = \{w, v\}$. Herein, we call a vertex pair having no edge between them a "nonedge". Then, edges have edge weight at least one and nonedges have edge weight zero. In the following, we will assume that the weight functions δ and ω have the following limits on their values:

- For every $v \in V$, it must hold that $\delta(v) \leq \sigma(v) \cdot (\sigma(v) - 1)/2$.
- For every edge $\{u, v\}$ it must hold that $\omega(\{u, v\}) \leq \sigma(u) \cdot \sigma(v)$.

We call this property *σ-limited*. Informally, it ensures that there is indeed an undirected graph from which the weighted graph can be obtained by merging vertices and it is also necessary for showing the size-bound on the reduced instance.

We introduce the following two notions for these weighted graphs. The "size" of a vertex set S is simply defined as $\sigma(S) := \sum_{v \in S} \sigma(v)$. The average degree $\bar{d}(V_i)$ of a connected component $G[V_i]$ can be computed as

$$\bar{d}(V_i) = \frac{2\sum_{v \in V_i} \delta(v) + \sum_{v \in V_i} \sum_{u \in N(v)} \omega(\{u, v\})}{\sigma(V_i)}.$$

The sum in the numerator of the fraction is simply the sum of all edges of the original graph that are "stored" in the weighted graph. The denominator is the number of all comprised vertices. Hence, this is the same sum as in Definition 3.3.

Similarly to the definition of average-s-plex cluster graphs, we say that a graph is a *weighted average-s-plex graph* if for each connected component $G[V_i]$ the average degree $\bar{d}(V_i)$ is at least $\sigma(V_i) - s$. For modifying the weighted graph, we allow the following modifications:

- increasing $\delta(u)$ by one for some $u \in V$,
- increasing $\omega(\{u,v\})$ by one for some $\{u,v\} \in E$,
- decreasing $\omega(\{u,v\})$ by one for some $\{u,v\} \in E$ with $\omega(\{u,v\}) > 1$,
- deleting some $\{u,v\} \in E$ with $\omega(\{u,v\}) = 1$, and
- inserting some edge $\{u,v\}$ to E and setting $\omega(\{u,v\}) := 1$.

Each of these operations has cost one, and the overall cost of a modification set S is thus exactly $|S|$. We assume that all these operations can only be applied when the weight function that is changed remains σ-limited after the modification.

Then, the weighted problem version is defined as follows.

WEIGHTED AVERAGE-s-PLEX-CLUSTER EDITING
Input: An undirected graph $G = (V,E)$, with a vertex-weight function $\sigma : V \to [1,n]$, a σ-limited vertex-weight function $\delta : V \to [0,n^2]$, a σ-limited edge-weight function $\omega : E \to [1,n^2]$, and a nonnegative integer k.
Question: Is there a set S of at most k edge modifications such that applying S to G yields a weighted average-s-plex graph?

One can easily reduce an instance $((V,E),k)$ of AVERAGE-s-PLEX-CLUSTER EDITING to an instance of WEIGHTED AVERAGE-s-PLEX-CLUSTER EDITING: set $\sigma(v) := 1$ and $\delta(v) := 0$ for each $v \in V$, and set $\omega(\{u,v\}) := 1$ if $\{u,v\} \in E$; otherwise, set $\omega(\{u,v\}) := 0$. Note that this reduction is parameter-preserving, that is, s and k are not changed.

In the following, we present two polynomial-time data reduction rules for WEIGHTED AVERAGE-s-PLEX-CLUSTER EDITING which (as we will show in Theorem 3.7) yield instances that contain at most $4k^2 + 8sk$ vertices.

Reduction Rule 3.1. *Remove from G all connected components that are weighted average-s-plexes.*

The rule is obviously correct since no optimal solution modifies any edges incident with vertices of such a connected component.

The second reduction rule identifies two vertices that have a large common neighborhood, or a "heavy" edge between them and "merges" these vertices into a new super-vertex.

Reduction Rule 3.2. *If G contains two vertices u and v such that $\omega(\{u,v\}) > k$ or u and v have more than k common neighbors, then remove u from G and set*

- $\sigma(v) := \sigma(u) + \sigma(v)$,

– $\delta(v) := \delta(u) + \delta(v) + \omega(\{u, v\})$, *and*

– $\omega(\{v, w\}) := \omega(\{v, w\}) + \omega(\{u, w\})$ *for each* $w \in V \setminus \{u, v\}$.

To see the correctness of the rule, consider the following: we cannot separate u and v using at most k edge modifications; thus, they must end up in the same connected component. Hence, we can remove one of them, and store the information about its adjacency in the vertex weights and edge weights of the other vertex. Note that Rule 3.2 preserves the property of being σ-limited for both δ and ω: because the weights of G are σ-limited, u and v "encode" vertex sets of an unweighted graph; the new merged vertex "encodes" the union of the two vertex sets.

With these two reduction rules we can show our main result of this section.

Theorem 3.7. (WEIGHTED) AVERAGE-s-PLEX-CLUSTER EDITING*is fixed-parameter tractable with respect to the parameter* (s, k).

Proof. We first show that a yes-instance I of WEIGHTED AVERAGE-s-PLEX-CLUSTER EDITING that is reduced with respect to Rules 3.1 and 3.2 contains at most $4k^2 + 8sk$ vertices. Let I be such a reduced instance, and let G be the input graph of I. Since I is a yes-instance, there is a weighted average-s-plex graph G' that can be obtained from G by applying at most k edge modifications. We now bound the size of G'. Herein, we call a vertex v "affected" if v is an endpoint of a modified edge or if $\delta(v)$ has been increased.

First, since G is reduced with respect to Rule 3.1, there is at least one affected vertex in each connected component of G'. Hence, there can be at most $2k$ connected components in G'.

Next, we show that each connected component of G' contains at most $2k + 4s$ vertices. Assume toward a contradiction that there is a connected component V_i of G' such that $|V_i| > 2k+4s$. Let $u \in V_i$ be a vertex that has a maximum number of neighbors in V_i. Since G' is a weighted average-s-plex graph, the average vertex degree $\bar{d}(V_i)$ of $G'[V_i]$ is at least $\sigma(V_i) - s$. Since $|V_i| \leq \sigma(V_i)$, u has at least $|V_i| - s \geq 2k+3s$ neighbors in $G'[V_i]$. We consider two cases for $\sigma(u)$.

Case 1: $\sigma(u) \geq \sigma(V_i)/2$**.** We show that the average degree $\bar{d}(V_i)$ of $G'[V_i]$ is less than $\sigma(V_i) - s$, contradicting the assumption that G' is a weighted average-s-plex graph. Since G is reduced with respect to Rule 3.2, each edge in G has weight at most k. Furthermore, the *overall* edge weight increase of edges incident with u is at most k. The average edge weight of edges incident with u in G' is thus at most $k+1$, since u has at least $2k + 3s$ neighbors in G'. However, with $\sigma(u) \geq \sigma(V_i \setminus \{u\}) \geq 2k + 3s$, this means that the average weight of edges incident with u is at most $\sigma(u)/2$. This leads to a low average degree. More precisely, we can bound the average degree of $G'[V_i]$ as

follows:

$$\begin{aligned}\bar{d}(V_i) &\overset{(*)}{<} \frac{\sigma(V_i)\cdot(\sigma(V_i)-1)-(\sigma(u)/2)\cdot\sigma(V_i\setminus\{u\})}{\sigma(V_i)}\\ &\overset{(**)}{<} \frac{\sigma(V_i)\cdot(\sigma(V_i)-1)-(\sigma(u)/2)\cdot 4s}{\sigma(V_i)}\\ &\overset{(***)}{\leq} \frac{\sigma(V_i)\cdot(\sigma(V_i)-1)-(\sigma(V_i)/4)\cdot 4s}{\sigma(V_i)}\\ &< \sigma(V_i)-s.\end{aligned}$$

Inequality (*) can be obtained from the following observations: The maximum value that can be achieved for the sum of δ and ω of $G'[V_i]$ is $\sigma(V_i)\cdot(\sigma(V_i)-1)$ because both δ and ω are σ-limited. From this we have to subtract the missing weight for the edges incident with u, which, as described above, have average weight at most $\sigma(u)/2$. Inequality (**) follows from $\sigma(V_i\setminus\{u\}) \geq |V_i|-1 > 4s$, and inequality (***) follows from $\sigma(u) \geq \sigma(V_i)/2$.

Case 2: $\sigma(u) < \sigma(V_i)/2$**.** First, we show that there must be at least one other vertex $w \in V_i$ that has at least $|V_i|-2s$ neighbors in $G'[V_i]$. Suppose that this is not the case. Then the average degree of $G'[V_i]$ can be bounded as follows

$$\begin{aligned}\bar{d} &\overset{(*)}{\leq} \frac{\sigma(V_i)\cdot(\sigma(V_i)-1)-2s\cdot(\sigma(V_i)-\sigma(u))}{\sigma(V_i)}\\ &\overset{(**)}{<} \frac{\sigma(V_i)\cdot(\sigma(V_i)-s)}{\sigma(V_i)}.\end{aligned}$$

Inequality (*) can be obtained from the following observations: The maximum possible value that can be achieved for the sum of δ and ω of $G'[V_i]$ is $\sigma(V_i)\cdot(\sigma(V_i)-1)$ because both δ and ω are σ-limited. From this we have to subtract the at least $2s$ edges that are missing for each vertex $v \in V_i\setminus\{u\}$. Inequality (**) follows from $\sigma(V_i)-\sigma(u) > \sigma(V_i)/2$. We have thus shown that there is at least one other vertex w that is adjacent to at least $|V_i|-2s$ vertices in G'. Since u has at least $|V_i|-s$ neighbors in $G'[V_i]$, there must be at least $|V_i|-3s > 2k+4s-3s = 2k+s$ vertices in $G'[V_i]$ that are common neighbors of u and w. Clearly, more than $k+s$ of those vertices are common neighbors of u and w in G. This contradicts the assumption that G is reduced with respect to Rule 3.2.

Altogether, we have shown that a reduced yes-instance contains at most $4k^2+8sk$ vertices. We obtain a fixed-parameter algorithm for Weighted Average-s-Plex-Cluster Editing as follows. First we exhaustively apply the reduction rules, which can clearly be done in polynomial time. If the reduced instance contains more than $4k^2+8sk$ vertices, then it is a no-instance. Otherwise we can solve the problem with running time only depending on s and k, for example by brute-force generation of all possible partitions of the graph. The fixed-parameter tractability of Average-s-Plex-Cluster Editing then directly follows from the described reduction to Weighted Average-s-Plex-Cluster Editing. □

It would be desirable to complement the data reduction result with a depth-bounded search tree. As already mentioned, devising such a search tree algorithm seems to be more difficult than it was the case for s-DEFECTIVE CLIQUE-CLUSTER EDITING and s-PLEX-CLUSTER EDITING: since being an average-s-plex is not hereditary, there is no forbidden local substructure that must be modified. Hence, new techniques are necessary for obtaining such a search tree algorithm.

3.4 μ-Clique-Cluster Editing

In this section, we show that μ-CLIQUE-CLUSTER EDITING is W[1]-hard with respect to the parameter "number k of allowed edge modifications". The W[1]-hardness as well as the NP-hardness are shown by a parameterized polynomial-time reduction from the following problem:

> MULTICOLORED CLIQUE
> **Input:** An undirected graph $G = (V, E)$ with a proper k-coloring $c : V \mapsto \{1, \ldots, k\}$ of the vertices.
> **Question:** Is there a k-vertex clique in G consisting of exactly one vertex from each color class?

Herein, proper means that the endpoints of every edge have different colors. MULTICOLORED CLIQUE parameterized by k is W[1]-hard [Fellows et al. 2009].

First, we briefly describe the basic idea of the reduction. Construct three types of dense connected components, all of which are μ-cliques. There are k connected components of the first type, called "color components", each corresponding to a color in the MULTICOLORED CLIQUE instance. The components of the second type correspond to pairs of colors, called "color pair components". The components of the third type are the "vertex components": for each vertex in G, there is a vertex component. Finally, we connect the vertex component for a vertex v to the color component corresponding to $c(v)$ by $2(k-1)$ "docking bridges" (the exact definition of docking bridges will be given in the following): For each color c' with $c' \neq c(v)$, we use two docking bridges between the vertex component for v and the color component for $c(v)$ to encode the edges in G between v and the vertices colored by c'. Each of these docking bridges is sparse, that is, a docking bridge alone is not a μ-clique. This is the rough construction of the μ-CLIQUE-CLUSTER EDITING instance.

The decisive trick of the reduction is the following: each color class of G corresponds to a connected component that contains a color component for this color and vertex components for the vertices with this color. This connected component is not a μ-clique due to the sparse vertex components and it is so large, compared to our new parameter k', that we cannot transform it into a μ-clique by inserting edges. However, cutting away *exactly one* vertex component corresponding to some

vertex v turns this connected component into a μ-clique. This means we have to cut $2(k-1)$ docking bridges connecting the vertex component for v to the color component. The construction of the docking bridges and the densities of the color and vertex components again force that cutting one docking bridge has to separate two small parts of the docking bridge, called "edge tails", from the color and vertex components and the docking bridge. An edge tail does not fulfill the condition of μ-cliques and corresponds to an edge between v and a vertex from the color class that the docking bridge represents. Finally, the separated edge tails are connected to the color pair components by edge insertions. The density of the color pair components allow them to be connected to exactly two edge tails which represent an edge between the two corresponding color classes. Altogether, the vertex components separated from the color components correspond to the vertices in the clique sought for in MULTICOLORED CLIQUE, and the edge tails separated from the vertex components and added to the color pair components correspond to the edges between the vertices in the clique.

In our reduction, we need to ensure that some of the constructed components are highly connected and have the property that adding a prespecified number of edges and vertices to these graphs results in a graph that has density *exactly* μ. In the following lemma, we show that the construction of these graphs is always possible and that it can be performed in polynomial time.

Lemma 3.5. *Given four positive integers a, b, c, and d, where $a < b$ and $d \leq c(c-1)/2$, we can construct in* $\mathrm{poly}(a,b,c,d)$ *time a graph G such that*
− G is $2(a-1)$-connected, and
− adding c vertices and d edges to G results in a graph that has density exactly a/b and has average degree more than a.

Proof. Without loss of generality, assume that $b-a>1$ and that $a>1$. Otherwise we can show the claim using $2a$ instead of a and $2b$ instead of b. We set the number of vertices of G to $n := (2b-1)c$ and the number of edges to $m := ac(2bc-1)-d$. First, since

$$\begin{aligned} 2m < 2ac(2bc-1) &\leq 2(b-2)c(2bc-1) \\ &= (2bc-4c)(2bc-1) \\ &< (n-3c)(n+c) \\ &< n(n-1) \end{aligned}$$

there is indeed a simple graph with the claimed number of edges. Moreover, since

$$\frac{m}{n} = \frac{ac(2bc-1)-d}{(2bc-1)c} = a - \frac{d}{(2bc-1)c} > a-1,$$

G has at least $(a-1)\cdot n$ edges. This means that we can construct G in polynomial time such that it is $2(a-1)$-connected [Harary 1962]. The density of the graph G' that results

from adding c vertices and d edges to G is

$$\frac{2(m+d)}{(n+c)(n+c-1)} = \frac{2(ac(2bc-1)-d+d)}{(2bc-c+c)(2bc-c-1+c)} = \frac{2ac(2bc-1)}{(2bc)(2bc-1)} = \frac{a}{b}.$$

The average degree of G' follows directly. □

With this "subroutine" at hand, we can show the following.

Theorem 3.8. *For all fixed* $0 < \mu < 1$, μ-CLIQUE-CLUSTER EDITING *is NP-complete and W[1]-hard with respect to the number k of allowed edge modifications.*

Proof. Let a and b be two fixed integers such that $\mu := a/b$ and assume without loss of generality that $a > n^6$ and $b > n^6$. Let C with $|C| = k$ be the set of colors in the MULTICOLORED CLIQUE instance $G = (V,E)$. Let $V_c := \{v \in V \mid c(v) = c\}$. Assume that $\forall c \neq c' : |V_c| = |V_{c'}|$ (in case $|V_c| < |V_{c'}|$ for some $c, c' \in C$ one can add isolated vertices of color c) and let $l := |V_c|$. In the following, we describe in detail the construction of the components of the μ-CLIQUE-CLUSTER EDITING instance. We first describe the construction of "docking bridges" which are used to connect vertex and color components and then the construction of the different types of components.

Construction of Docking Bridges. The docking bridges encode information about the adjacencies of G. This is done by adding tails of a specific length to the docking bridges. To this end, we assign to each ordered vertex pair (u,v) with $c(u) \neq c(v)$ an integer π_{uv} between 1 and x where $x = 12n^2 + 1$. Herein, the following rules should be obeyed:

1. No two pairs get the same number.
2. If there is an edge between u and v, then $|\pi_{uv} - \pi_{vu}| = 1$ and all numbers z with $|z - \pi_{uv}| \leq 2$ or $|z - \pi_{vu}| \leq 2$ should be reserved, that is, they should not be assigned to any other vertex pair.
3. If there is no edge between u and v, then $|\pi_{uv} - \pi_{vu}| = 2$ and all numbers z with $|z - \pi_{uv}| \leq 1$ or $|z - \pi_{vu}| \leq 1$ should be reserved.
4. If a number i is assigned to a vertex pair or reserved, then $x - i$ should not be assigned to any vertex pair.

Observe that, since $x > 12n^2$, such an assignment is always computable in polynomial time. These numbers will be needed to identify the edges in E.

For a vertex v with color $c(v)$, we connect the vertex component K_v of v to the color component $K_{c(v)}$ with $2(k-1)$ docking bridges. For each color $c' \neq c(v)$ we add a pair of docking bridges as illustrated in Figure 3.7. Each docking bridge consists of $4k(k-1)+1$ edge-disjoint paths all of length $2l$. A docking bridge is divided by the common vertices of the paths into l "intervals"; each interval corresponds to a vertex in $V_{c'}$ and consists of $4k(k-1)+1$ length-two paths. The vertices lying on these paths with the exception of the two endpoints are called "middle" vertices in this

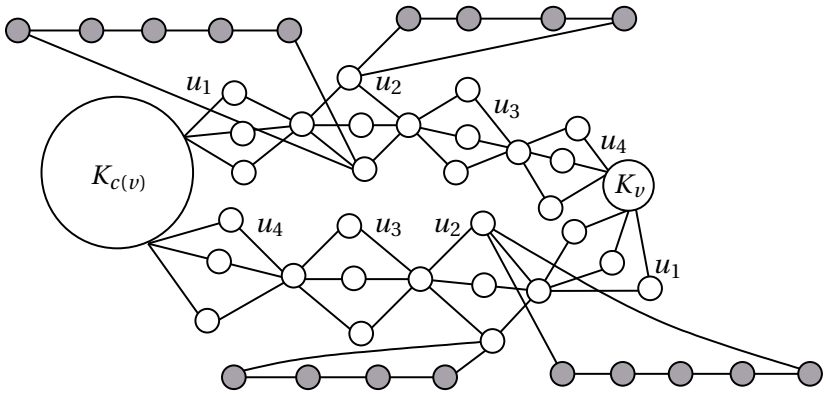


Figure 3.7: An example of a docking bridge pair between a vertex component K_v and a color component $K_{c(v)}$. These two docking bridges correspond to a color class c' with $c' \neq c(v)$ and $V_{c'} = \{u_1, u_2, u_3, u_4\}$. Thus, each docking bridge has four intervals. Here, for the the sake of simplicity, only three length-two paths are drawn in each interval (in the construction, there are $4k(k-1)+1$ such paths in each interval). To each interval of the docking bridges we add edge tails. Here, we show the edge tails for u_2 (the gray vertices are the vertices that are in the cycles of the edge tails) for $\pi_{vu_2} = 5$, $x = 11$, and $y = 0$ (we therefore create two cycles, one of length $y + \pi_{vu_2} = 5$ and one of length $y + x - \pi_{vu_2} = 6$).

interval. However, the orders of the vertices in $V_{c'}$ according to which they appear on the two corresponding docking bridges are reversed as shown in Figure 3.7. Note that all docking bridges have the same endpoints, one in K_v and the other in $K_{c(v)}$. Since the length of docking bridges can be extended to n^d for an arbitrary large constant d, we can assume that each of the docking bridges has density less than μ.

Next we add "edge tails" to the docking bridges. For each interval of a docking bridge which corresponds to a vertex w, we add two edge tails that are two cycles with length $y + \pi_{vw}$ and $y + x - \pi_{vw}$, respectively. Herein, y is a large integer such that each of the two cycles alone has density less than μ. This can be achieved for example by setting $y := ab$. See Figure 3.7 for an example of a color component and its docking bridges to a vertex component.

Now we describe the construction of the three types of components.

Color Pair Components. For each (unordered) pair of colors, add two color pair components K^1 and K^2, where K^1 should satisfy the following requirements:

- Adding two edges to connect two vertex-disjoint cycles that together have length *at most* $x + 2y + 1$ with K^1 results in a graph with density at least μ.
- Connecting two cycles with total length *more than* $x + 2y + 1$ will decrease the density below μ.

Lemma 3.5 shows that K^1 can be constructed in polynomial time such that adding two vertex-disjoint cycles that together have a length equal to $x+2y+1$ results in a graph that has density exactly μ. Note that by Lemma 3.5 and since $a > n^6$, the second part of the requirement also holds: adding a cycle of length more than $x+2y+1$ decreases—compared to adding a shorter cycle—the average degree of the connected component while further increasing the size of the connected component. Hence, such a component has density less than μ. The same requirement should also be fulfilled by K^2 with the length threshold of the cycles being $x+2y-1$.

Vertex Components. For each vertex v, add a component K_v satisfying the following requirements:

- K_v is $2(a-1)$-connected.
- The graph that contains K_v and $\alpha := (k-1)((l-1)(4k(k-1)+x+2y)+8k(k-1)-2+x+2y)$ vertices and $\beta := (k-1)((l-1)(8k(k-1)+x+2y+2)+8k(k-1)+x+2y)$ edges of the docking bridges incident with K_v has density exactly μ.
- The density will decrease below μ if more than α vertices of the docking bridges are included.

These requirements are needed to argue that the graph resulting from disconnecting K_v from the corresponding color component $K_{c(v)}$ has density μ. By Lemma 3.5, we can construct in polynomial time a graph that fulfills the first two requirements. The fulfillment of the third requirement again follows from Lemma 3.5 and the observation that adding more docking bridges decreases the average degree and increases the number of vertices in the connected component, resulting in a graph that has density less than μ.

Color Components. For each color c the color component K_c has to fulfill the following requirements. Herein, v is an arbitrary vertex from V_c, and we use $K_{c(v)}$ to denote K_c in order to emphasize this fact.

- $K_{c(v)}$ is $2(a-1)$-connected.
- The graph that consists of
  1. $K_{c(v)}$,
  2. the $l-1$ vertex components corresponding to the vertices of $V_{c(v)} \setminus \{v\}$ together with all docking bridges that connect these vertex components with $K_{c(v)}$, and
  3. further $\gamma := (k-1)(l-1)(4k(k-1)+x+2y)$ vertices and $\delta := (k-1)(l-1)(8k(k-1)+x+2y+2)$ edges from the docking bridges between $K_{c(v)}$ and K_v

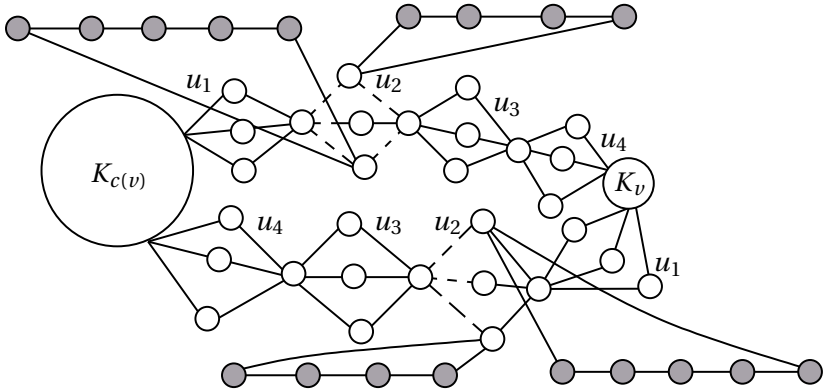


Figure 3.8: Illustration of the situation where the second requirement for the vertex components applies. The dashed lines represent edge deletions. Here, the two docking bridges between K_v and $K_{c(v)}$ are destroyed by deleting the edges in the intervals corresponding to u_2. Moreover, the middle vertices of these intervals belong to the connected component containing K_v with the only exception of two middle vertices of the first docking bridge. Observe that two edge tails corresponding to the vertex pair (v, u_2) are separated from K_v and $K_{c(v)}$.

has density *exactly* μ.

– Adding more vertices from the docking bridges between $K_{c(v)}$ and K_v to this graph results in a graph with density *less than* μ.

By Lemma 3.5, we can construct in polynomial time a graph that fulfills the first two requirements. The fulfillment of the third requirement again follows from the observation that adding more than γ vertices from the docking bridges between $K_{c(v)}$ and K_v decreases the average degree while increasing the number of vertices. Hence, the graph has density less than μ in this case.

We complete the construction of the μ-Clique-Cluster Editing instance by setting its parameter to $k' := 2k(k-1)(4k(k-1)+3)$. Let (H, k') denote the constructed instance of μ-Clique-Cluster Editing. We prove the theorem by showing the following claim:

> (G, k) is a yes-instance of Multicolored Clique $\Leftrightarrow$ (H, k') is a yes-instance of μ-Clique-Cluster Editing.

$\Rightarrow$: Given a size-k clique $X := \{v_1, v_2, \ldots, v_k\}$ of the Multicolored Clique instance, we disconnect the vertex components which correspond to the vertices in X from their color components.

Observe that, in order to disconnect a vertex component from a color component, one needs $2(k-1)(4k(k-1)+1)$ edge deletions, that is, the edge deletions to cut every docking bridge at some interval. More precisely, to disconnect the vertex

component K_{v_i} from $K_{c(v_i)}$, we cut the docking bridges incident with K_{v_i} at the intervals corresponding to the vertices in $X \setminus \{v_i\}$ as shown in Figure 3.8. Note that in Figure 3.8 we use two additional edge deletions to separate two edge tails from the vertex component and the color component. Altogether, we use $2(k-1)$ additional edge deletions to separate $2(k-1)$ edge tails which correspond to the edges between v_i and $X \setminus \{v_i\}$. By the construction of K_{v_i} and $K_{c(v_i)}$ we can conclude that the resulting two connected components, one containing K_{v_i} and the other containing $K_{c(v_i)}$, fulfill the condition of μ-cliques. Further, we add edges between the separated edge tails and the color pair components. According to the assignment of integers from the interval $[1,x]$ to the ordered vertex pairs, the edge tails can be pairwise put together such that each pair has a total length of either $x+2y-1$ or $x+2y+1$. Then, we connect each pair with a total length of $x+2y+1$ to one color pair component K^1 and each pair with a total length of $x+2y-1$ to one K^2. According to the construction of the color pair components, all resulting components are μ-cliques. Here, we need altogether $2k(k-1)$ edge insertions. Putting edge deletions and insertions together, we have made $2k(k-1)(4k(k-1)+1)+4k(k-1) = 2k(k-1)(4k(k-1)+3)$ edge modifications.

$\Leftarrow$: By construction, the connected components of the μ-Clique-Cluster Editing instance that contain l vertex components and one color component have density less than μ. We cannot transform these connected components into μ-cliques by splitting the vertex or color components, since these components are $2(a-1)$-connected and $a > n^6 > k^4$. By the same reason, we cannot transform such a connected component into a μ-clique with at most k' edge insertions. Therefore, at least one vertex component has to be disconnected from each color component.

Since there are $2(k-1)(4k(k-1)+1)$ edge-disjoint paths between a vertex component and a color component, we need at least $2(k-1)(4k(k-1)+1)$ edge deletions for each color. This means that from a solution for (H,k') only $4k(k-1)$ modifications remain to be specified. Therefore, for at least one color, we have at most $4(k-1)$ edge modifications.

From the construction of vertex and color components we know that, when separating a vertex component K_v from a color component $K_{c(v)}$, there have to be exactly $(k-1)(x+2y)$ vertices that are separated from both the connected component that contain K_v and the connected component that contains $K_{c(v)}$. Since K_v and $K_{c(v)}$ are inseparable, these vertices come from the docking bridges between K_v and $K_{c(v)}$ and edge tails attached to the docking bridges. Note that, since each docking bridge consists of $4k(k-1)+1$ edge-disjoint paths, we cannot separate enough vertices by deleting at most $4(k-1)$ edges of docking bridges. The only choice is to separate some edge tails from the docking bridges. From Figure 3.8 we can observe that by deleting two edges we can separate at most $x+2y$ vertices from the docking bridges, namely, separating the two edge tails, one having length $y+i$ and the other having length $x+y-i$, attached to the intervals whose length-two paths are destroyed while separating K_v from $K_{c(v)}$ (in Figure 3.8, this is the interval corresponding to u_2).

Separating other edge tails or separating only a part of an edge tail requires at least two edge deletions and leaves less than $x+2y$ vertices separated. Thus, all separated edge tails come from the intervals where K_v is separated from $K_{c(v)}$.

Observe that these separated edge tails are not μ-cliques and connecting them to each other by at most $2k(k-1)$ edge insertions cannot transform them into μ-cliques because their sizes are at least $y+1$ for a large integer y. Hence, the only possibility is to connect them to the $2k(k-1)$ color pair components. This means one edge insertion for each edge tail. From the construction of color pair components, vertex-disjoint cycles with a total length at most $x+2y+1$ can be attached to K^1's, while vertex-disjoint cycles with a total length at most $x+2y-1$ can be attached to K^2's. Thus, we are forced to group the edge tails into at most $k(k-1)$ groups such that the groups can be partitioned to two same-size subsets; in one, each group has a total length at most $x+2y+1$ and, in the other, each has a total length at most $x+2y-1$. According to the assignment for the ordered vertex pairs described above, this is only possible when each group consists of two edge tails, one corresponding to the vertex pair (u,v) and the other corresponding to the vertex pair (v,u), and there is an edge between u and v in the original instance. This means that the k vertices whose corresponding vertex components are separated from the color components form a clique. □

3.5 Concluding Remarks

We have introduced four generalizations of CLUSTER EDITING and presented fixed-parameter tractability and intractability results for each of them. In particular, we have shown the following:

- s-DEFECTIVE CLIQUE-CLUSTER EDITING and s-PLEX-CLUSTER EDITING can be handled by the technique of "finding and destroying" forbidden induced subgraphs that leads to search tree algorithms for both problems.
- AVERAGE-s-PLEX-CLUSTER EDITING demands a different approach since the graph property that shall be obtained is not hereditary. We can still obtain fixed-parameter tractability with respect to (s,k) by a combination of reduction to a weighted problem version and data reduction.
- μ-CLIQUE-CLUSTER EDITING is $W[1]$-hard with respect to the parameter k.

Future research tasks for these four problems could be as follows:

- For s-DEFECTIVE CLIQUE-CLUSTER EDITING and s-PLEX-CLUSTER EDITING first experimental implementations could be developed. Then, comparisons with CLUSTER EDITING concerning the running times and the quality of the produced clusterings should be performed. For protein-interaction networks, a measure of clustering quality could be functional coherence of the clusters (see Section 7.2 for more details on functional coherence of protein sets).

- For AVERAGE-s-PLEX-CLUSTER EDITING, a more efficient algorithm is desirable. In particular, can we describe an efficient branching strategy despite the fact that being an average-s-plex cluster graph is not hereditary?
- For μ-CLIQUE-CLUSTER EDITING, a completely different approach is needed. What are useful additional parameters for coping with this density property?
- It would be interesting to study for the first three density properties, how the parameter number k of edge modifications behaves with increasing s. Is there a range of s values for which k decreases particularly fast for increasing s? If yes, does this imply that an optimal s-value lies within this range?

For all four problems also the edge deletion variants of the clustering problems have been studied [Guo et al. 2009a]. For s-defective cliques, s-plexes, and average-s-plexes the results are roughly the same as for their editing versions: the edge deletion variants of the clustering problems are NP-hard but fixed-parameter tractable with respect to the parameter (s,k). For μ-cliques, the situation is different: so far, only the NP-hardness of the edge deletion version of μ-CLIQUE-CLUSTER EDITING is known, its fixed-parameter tractability with respect to the number k of edge modifications remains open, we conjecture that this problem is W[1]-hard as well. Summarizing, our results indicate that clique relaxation-based network clustering can be "handled" by the means of parameterized algorithmics, but that it is necessary to include the slack parameter s that measures the "distance" of the clique relaxation to the clique model into the parameterization.

Chapter 4

Average Parameterization and Structural Kernelization for Consensus Clustering

In this chapter, we present a parameterization approach for so-called *consensus problems* as well as a new kernelization concept that extends the standard notion of kernelization and is useful in case a small (for example, polynomial) standard problem kernel cannot be achieved. Informally, consensus problems are problems in which one is given a set of combinatorial objects, for example permutations, over a base set S and wants to find a *median object* that has minimum distance to the input objects.

We present both the parameterization and the kernelization concept by applying them to the NP-hard CONSENSUS CLUSTERING problem which is closely related to a special case of edge-weighted CLUSTER EDITING [Ailon et al. 2008] and arises in attempts to reconcile clustering information. CONSENSUS CLUSTERING has applications for example in the clustering of gene expression data [Monti et al. 2003] and in the identification of protein families [Nikolski and Sherman 2007].

The input of CONSENSUS CLUSTERING is a set of partitions over a base set S and the goal is to find a *median partition* that minimizes the sum of distances to the input partitions. Herein, the distance between two partitions is the sum of element pairs that are clustered differently in the two partitions. Following the notation introduced by Goder and Filkov [2008], we call two elements $a, b \in S$ *co-clustered* with respect to a partition C if a and b occur together in one of the sets in C and *anti-clustered* if a and b occur in different sets in C. Then, the distance $\mathrm{dist}(C_i, C_j)$ between two input partitions C_i and C_j is defined as the number of unordered pairs $\{a, b\}$ of elements from the base set S such that a and b are co-clustered in one of C_i and C_j and anti-clustered in the other. This distance is sometimes referred to as *Mirkin metric* or *Mirkin distance*; more precisely, the Mirkin distance is defined as $2 \cdot \mathrm{dist}(C_i, C_j)$ [Meilă

$C_1 = \{\{a,b\},\{c\},\{d\}\}$ $\quad$ $\mathrm{dist}(C_1,C_2) = 2$
$C_2 = \{\{a,b,c\},\{d\}\}$ $\quad$ $\mathrm{dist}(C_1,C_3) = 3$
$C_3 = \{\{a,c\},\{b,d\}\}$ $\quad$ $\mathrm{dist}(C_2,C_3) = 3$

Figure 4.1: An input instance of CONSENSUS CLUSTERING with base set $S = \{a,b,c,d\}$. For this instance, the parameter average distance d is $(2+3+3)/3 = 8/3$. Both C_1 and C_2 are median partitions of $\mathcal{C} = \{C_1, C_2, C_3\}$, that is, of all partitions of S they have minimum Mirkin distance to $\mathcal{C}$.

2005]. Altogether, this leads to the following problem definition.

CONSENSUS CLUSTERING
Input: A set $\mathcal{C} = \{C_1, \ldots, C_n\}$ of partitions over a base set S and a nonnegative integer k.
Question: Is there a partition C of S with $\sum_{C_i \in \mathcal{C}} \mathrm{dist}(C, C_i) \leq k$?

In the following, we refer to the elements of a partition of S as *clusters* of this partition. An example of a CONSENSUS CLUSTERING instance is shown in Figure 4.1. A reduction of CONSENSUS CLUSTERING to edge-weighted CLUSTER EDITING is roughly as follows: Build a graph whose vertices are the elements of S. For each vertex pair, introduce a "positive" and a "negative" edge whose value is the part of the overall Mirkin-distance that is caused by anti-clustering (or co-clustering) the endpoints of the edges in a partition of S. This graph with positive and negative edges can then be translated into an "equivalent" edge-weighted graph.

The standard way of parameterizing CONSENSUS CLUSTERING would be to use k as parameter. A closer inspection reveals that this parameter is very large unless the input partitions are almost identical: every pair that is co-clustered in one partition and anti-clustered in at least one other partition contributes a value of at least one to this parameter. Hence, we consider a parameter that is *stronger* than this parameter: the *average distance* d of the input partitions which we define as

$$d := \frac{\sum_{C_i, C_j \in \mathcal{C}} \mathrm{dist}(C_i, C_j)}{n \cdot (n-1)}.$$

The advantage of parameter d over the standard parameter k is especially pronounced in instances where the number of input partitions is large compared to the number of elements. This could be the case when the input partitions are defined by categorical features of the elements. Suppose, for instance, that the element set is a relatively small set of human diseases, and that for each disease and for a large set of human genes it is known whether or not this gene is expressed (that is, active) in patients that have this disease. Then, each gene defines a partition of the diseases

into two categories; one category contains the diseases where the gene is expressed and the other one contains the diseases where the gene is not expressed. In such an instance, the number of partitions is much larger than the number of elements.

At first sight, it is not clear that the parameter average distance of the input partitions is always smaller than the standard parameter; however, it can be seen by examining the relation between average distance of the input partitions and the "average distance $\bar{d}$ of the median partition to the input distances". Intuitively, the parameter $\bar{d}$ can be much smaller than the parameter d: since d is the average distance between the input partitions, there must be at least one input partition whose average distance to the other input partitions is at most d, and a median partition could even be "closer" to the input partitions. For the most part, this intuition is true, but $\bar{d}$ cannot be arbitrarily small compared to d, since one can show that $d \leq 2\bar{d}$ as follows.

Let $\mathcal{C}$ be a set of n input partitions of a CONSENSUS CLUSTERING instance, let d be the average distance between the input partitions, let P be a median partition of $\mathcal{C}$, and let $\bar{d} := \sum_{C \in \mathcal{C}} \operatorname{dist}(P, C)/n$ denote the average distance of P to $\mathcal{C}$. By the triangle inequality, the following is easy to see:

$$\begin{aligned} d &= \Big(\sum_{C \in \mathcal{C}} \sum_{C' \in \mathcal{C} \setminus \{C\}} \operatorname{dist}(C, C')\Big)/(n(n-1)) \\ &\leq \Big(\sum_{C \in \mathcal{C}} \sum_{C' \in \mathcal{C} \setminus \{C\}} \operatorname{dist}(C, P) + \operatorname{dist}(C', P)\Big)/(n(n-1)) \\ &= 2 \cdot \Big(\sum_{C \in \mathcal{C}} \sum_{C' \in \mathcal{C} \setminus \{C\}} \operatorname{dist}(C, P)\Big)/(n(n-1)) \\ &= 2 \cdot \Big(\sum_{C \in \mathcal{C}} \operatorname{dist}(C, P)\Big)/n = 2\bar{d}. \end{aligned}$$

Hence, we have $\bar{d} \leq d \leq 2\bar{d}$, which means that the parameters d and $\bar{d}$ are roughly "equivalent". Furthermore, this observation also implies that d is always smaller than the standard parameter k which can be seen as follows. By definition $\bar{d} = \sum_{C \in \mathcal{C}} \operatorname{dist}(P, C)/n$. Moreover, we can assume that $n > 2$, since for $n = 2$ CONSENSUS CLUSTERING is polynomial-time solvable [Filkov and Skiena 2004]. Then, it immediately follows that $d < \sum_{C \in \mathcal{C}} \operatorname{dist}(P, C) = k$. Note that the parameter d can also be *arbitrarily* small compared to the k. For instance, when the input has two elements a and b and contains $n/2$ times the partition $\{\{a, b\}\}$ and $n/2$ times the partition $\{\{a\}, \{b\}\}$. Then k is at least $n/2$ in a yes-instance whereas $d < 1$. Summarizing, the parameter "average distance d between the input partitions" is stronger than the parameter "overall distance k of the solution to the input partitions", and, up to a constant factor, equivalent to the parameter "overall distance k of the solution to the input partitions".

A further motivation for the parameter d can be drawn from a consensus clustering approach that applies to heterogeneous input partitions [Goder and Filkov 2008]. This approach tries to deal with the problem that a consensus between

clusterings that are too different is not likely to yield useful information. The idea is to first cluster the input partitions into groups that are similar and then to compute a median partition for each group of similar partitions. In these groups of similar partitions the parameter d should be much smaller than in the original input instance.

Related Work. The NP-hardness of CONSENSUS CLUSTERING was shown by Křivánek and Morávek [1986] and later also by Wakabayashi [1998]. In case the input consists of only two partitions, CONSENSUS CLUSTERING is polynomial-time solvable: each input partition is a median partition [Filkov and Skiena 2004]. In contrast, the minimization version of CONSENSUS CLUSTERING is APX-hard even if the input consists of only three partitions [Bonizzoni et al. 2008]; the current best polynomial-time approximation algorithms achieve an approximation factor of 4/3 [Ailon et al. 2008, van Zuylen and Williamson 2009]. For the maximization version of CONSENSUS CLUSTERING, in which wants to maximize $\binom{|S|}{2} \cdot n - k$, a polynomial-time approximation scheme (PTAS) can be achieved [Bonizzoni et al. 2008]. Various heuristics and approximation algorithms for CONSENSUS CLUSTERING have been experimentally evaluated [Bertolacci and Wirth 2007, Goder and Filkov 2008]. The general task of integrating a set of given partitions into a new median partition is also known as *cluster ensemble* problem [Strehl and Ghosh 2002]; other approaches for cluster ensemble problems are for example based on hypergraph partitioning [Strehl and Ghosh 2002] or bipartite graph partitioning [Fern and Brodley 2004].

Our Results. We initiate the study of CONSENSUS CLUSTERING in the context of parameterized algorithmics. We show that CONSENSUS CLUSTERING is fixed-parameter tractable with respect to the parameter d by presenting a data reduction rule that leads to an instance with $16d/3$ elements. We furthermore show fixed-parameter tractability of CONSENSUS CLUSTERING with respect to another parameter, the "number of dirty elements" which can be much smaller than the parameter d. These results are embedded into a general algorithmic framework that is applicable to other consensus problems as well. Moreover, we present a new kernelization concept, *structural kernelization*.[1] The idea behind structural kernelization is that we shrink some "dimension" or "property" of the input instance in order to obtain fixed-parameter tractability. In the standard problem kernel definition, this dimension is the overall input size; in the presented algorithms for CONSENSUS CLUSTERING it is the number $|S|$ of elements. Structural kernelization is a general concept completely independent of average parameterization and consensus problems, and we believe that it can be useful for a wide range of problems.

[1] In previous work [Betzler et al. 2011b], we used the term "partial kernelization". We now find structural kernelization to be the more appropriate term.

4.1 The General Framework and Structural Kernelization

In the first part of this section, we present a general framework for achieving fixed-parameter tractability with respect to the parameter "average distance between the input objects" for consensus problems such as CONSENSUS CLUSTERING. So far, the presented framework has been also applied to KEMENY SCORE and SWAP MEDIAN PARTITION [Betzler et al. 2010a, 2011b], and further applications are conceivable. Subsequently, we define the concept of structural kernelization that is employed in our general framework and discuss the relationship between structural kernelization and other recently proposed extensions of the kernelization definition.

The framework consists of the following four main steps.

Step 1. Define a "tidiness/dirtiness" concept that divides the base set S into a "tidy" part and a "dirty" part. Subsequently prove that an instance of the underlying consensus problem can be solved in polynomial-time when the complete input is tidy.

Step 2. Show that the size of the dirty part of S is upper-bounded by a polynomial only depending on the average distance d between the given combinatorial objects.

Step 3. Develop polynomial-time data reduction rules which shrink the size of the tidy part of S, generating an equivalent problem instance of smaller size. Then show that in the reduced instance the size of the tidy part of S can be upper-bounded by a polynomial only depending on the size of the dirty part of S and, thus, also the average distance d.

Step 4. Show that the desired median object can be found in a running time whose exponential running time part only depends on the number of elements in S, and not on the number of input objects.

When applicable, this framework yields fixed-parameter tractability with respect to the parameter "average distance" but also with respect to the parameter "size of the dirty part of S". In general, fixed-parameter tractability also follows for nonpolynomial functions in Steps 2 and 3, but so far in all applications of the framework polynomial bounds were obtained. A special feature of our framework is that in Step 3 we perform a data reduction that does not lead to a problem kernel but that is used to obtain fixed-parameter tractability via Step 4. Hence, this data reduction does not lead to a bound on the overall input size but it shrinks an important part of the input *structure*. We therefore refer to this type of kernelization as *structural kernelization*. In the example of CONSENSUS CLUSTERING, we shrink the size of the tidy part of the input, and in

a reduced instance the overall number of elements is bounded by a function of the average distance d. The term "structural" refers to the fact that only the number of elements is bounded, but not the number of input partitions. We formalize the ideas behind this kernelization concept as follows.

Definition 4.1. *Let (I,k) be an instance of a parameterized problem L, where $I \in \Sigma^*$ denotes the input instance and $k \in \Sigma^*$ a parameter. Let $s : \Sigma^* \to \Sigma^*$ (the* structure*) be a computable function such that L is fixed-parameter tractable with respect to $s(I)$. The problem L admits a* structural problem kernel *if there is a polynomial-time algorithm that computes an instance (I',k') of the same problem such that:*

- *(I,k) is a yes-instance if and only if (I',k') is a yes-instance,*
- *$k' \leq f(k)$, and*
- *$s(I') \leq g(k)$*

for computable functions f and g.

For I, k, and s meeting the above conditions, the existence of a structural problem kernel directly implies fixed-parameter tractability with respect to the parameter k. Note that structural kernelization can be seen as a generalization of the standard problem kernelization that reduces an instance of a problem to an instance whose size is bounded by a function of the parameter: choosing $s(I) := |I|$ directly leads to the standard problem kernel definition. In the following, we discuss the relation between structural kernelization and two other "new" kernelization concepts.

One of the recently proposed extensions of the standard problem kernelization definition are *weak kernels* [Jiang and Zhu 2010]: weak kernels are defined for a class of search problems, and the idea of weak kernels is to reduce the size of the search space. Consider for example a vertex deletion problem parameterized by the number of vertex deletions. A data reduction that produces an instance in which there is a set S of at most $f(k)$ vertices such that every vertex that will be deleted by an optimal solution is contained in S is a weak kernelization: the overall size of the instance is not necessarily reduced, but one has computed the "small" set of vertices S on which a brute-force algorithm directly yields fixed-parameter tractability. The structural kernelization concept is more general in the sense that the part of the input that is reduced could be any structural property of the input such as the treewidth or branchwidth of an input graph. For example, in a fixed-parameter algorithm for the k-CYCLE problem on planar graphs parameterized by the cycle length k, Dorn et al. [2010] make use of the fact that a planar graph with branchwidth at least $4\sqrt{k+1}-3$ always contains a cycle of length at least k. Consequently, instances whose branchwidth is larger than $4\sqrt{k+1}-3$ can be decided in polynomial time.[2] Since k-CYCLE is fixed-parameter tractable with respect to the branchwidth

[2] The branchwidth of a planar graph is computable in polynomial time [Seymour and Thomas 1994].

of the graph [Bodlaender 1993] this approach of dealing with instances with large branchwidth can be seen as a structural kernelization where the branchwidth is the structure s.

Another recently proposed extension of the standard kernelization definition are *Turing kernels* [Fernau et al. 2009, Lokshtanov 2009]. Here, the idea is to allow that the reduction is not a many-to-one reduction (as in the standard definition) but a Turing reduction. That is, in a Turing kernelization one may create polynomially many problem kernels and somehow "combine" the results for these problem kernels. For example, one could create $|I|$ problem kernels such that the original instance is a yes-instance if and only if at least one of the created problem kernels is a yes-instance. Turing kernelizations have been proposed for parameterized problems that do not admit a standard problem kernel of polynomial size [Fernau et al. 2009, Lokshtanov 2009, Schäfer et al. 2011]. The differences between Turing kernelization and structural kernelization are as follows. In Turing kernelizations, the aim is, as for standard problem kernelization, to reduce the overall size of the created instances; in structural kernelization the aim is only to reduce a certain part of the input structure. In Turing kernelizations, one is allowed to produce many kernels and combine them; structural kernelizations as we define them are many-to-one reductions.

In the next section, we begin to describe how the presented framework can be applied to CONSENSUS CLUSTERING. In particular, we describe how to define the dirty and tidy part of S, as demanded by Step 1.

4.2 Structural Properties of Tidy and Dirty Pairs

In this section, we present a definition of tidy and dirty pairs and then study two special cases of this definition. We will then show that for both special cases an instance is solvable in polynomial-time when the input contains only tidy pairs. This corresponds to Step 1 of the framework. Then, we will show that in both cases, the number of dirty pairs is upper-bounded by a polynomial function of the average distance d. This corresponds to Step 2 of the framework. Moreover, for both cases, we observe structural properties of the instance that can be inferred from the existence of tidy pairs and which are then exploited by the data reduction rules. These structural properties will then also explain why we choose these two particular special cases of our definition.

Our general definition of tidiness and dirtiness is motivated by the following observations. An input instance with $d = 0$ is clearly trivial since all input partitions are equal: the median partition is identical to all input partitions. In such an input instance each element pair is either co-clustered in all partitions or anti-clustered in all partitions. Such element pairs are prototypical tidy pairs, since for such pairs it is clear how a median partition places them: If two elements a and b are co-clustered in

all n partitions, then they are co-clustered in a median partition as well. If they are anti-clustered in all n partitions, then they are anti-clustered in a median partition. This property of such pairs also holds if there are element pairs in the instance that are not tidy in this sense (this can be shown by an easy "improvement argument"; we omit the details). Informally, the tidiness/dirtiness concept tries to separate the tidy element pairs from element pairs that are especially "untidy". More precisely, the decision whether an element pair is considered to be dirty or tidy is based on the fraction of input partitions in which this element pair is co-clustered and anti-clustered. To this end we introduce the following notation. Given a set $\mathcal{C}$ of partitions of a base set S and two elements $a, b \in S$, we denote with $\text{co}(a,b)$ the number of partitions in $\mathcal{C}$ in which a and b are co-clustered and with $\text{anti}(a,b)$ the number of partitions in $\mathcal{C}$ in which a and b are anti-clustered. A generic definition of tidy and dirty element pairs is as follows.

Definition 4.2. *Let $\mathcal{C}$ be a set of n partitions of a base set S, and let t be a rational number with $1/2 \leq t \leq 1$.*

- *A pair of elements $a, b \in S$ is called a* t-tidy pair *of $\mathcal{C}$ if $\text{co}(a,b) > t \cdot n$ or $\text{anti}(a,b) > t \cdot n$.*
- *The predicate $(ab)_t$ is true if $\text{co}(a,b) > t \cdot n$.*
- *The predicate $(a \nleftrightarrow b)_t$ is true if $\text{anti}(a,b) > t \cdot n$.*
- *A pair of elements $a, b \in S$ is called a* t-dirty pair *$(a\#b)_t$ of $\mathcal{C}$ if $\text{co}(a,b) \leq t \cdot n$ and $\text{anti}(a,b) \leq t \cdot n$.*

Note that it is only interesting to consider $t \geq 1/2$, since for $t < 1/2$ every element pair is a t-tidy pair, since $\text{co}(a,b) + \text{anti}(a,b) = n$. We use two different thresholds t for considering element pairs as dirty. In the first and more strict case we set $t := 3/4$. In the second case we set $t := 2/3$.[3] As we show in the following, both thresholds imply interesting structural properties of dirty and tidy element pairs in the input instance. Roughly speaking, in both cases we observe that the presence of tidy pairs implies properties on other element pairs in their "neighborhood".

First, we examine the effect of tidy pairs for $t = 3/4$. The main observation to exploit is that for a $3/4$-tidy pair of elements a and b the values of $\text{co}(a,c)$ and $\text{co}(b,c)$ for some third element c differ by less than $n/4$.

Lemma 4.1. *Let $\{a,b,c\}$ be a set of three elements. Then,*

- $(ab)_{3/4} \wedge (ac)_{3/4} \Rightarrow \text{co}(b,c) > n/2$, *and*
- $(ab)_{3/4} \wedge (a \nleftrightarrow c)_{3/4} \Rightarrow \text{anti}(b,c) > n/2$.

[3] The case $t = 2/3$ was proposed simultaneously for Consensus Clustering and Kemeny Score. Later it was observed that for Kemeny Score setting $t = 3/4$ leads to better kernelization results [Betzler et al. 2010a]. Consequently, we also consider $t = 3/4$ for Consensus Clustering.

Proof. We show both statements separately.

First, let $(ab)_{3/4}$ and $(ac)_{3/4}$ hold. By definition, there are less than $n/4$ partitions in which a and b are not co-clustered, that is, $\text{anti}(a,b) < n/4$. Hence, of the more than $3n/4$ partitions in which a and c are co-clustered, there are less than $n/4$ partitions in which a and b are not co-clustered. Therefore, a, b, and c are co-clustered in more than $n/2$ partitions which also implies that $\text{co}(b,c) > n/2$.

Second, let $(ab)_{3/4}$ and $(a \leftrightarrow c)_{3/4}$ hold. By definition, there are less than $n/4$ partitions in which a and c are co-clustered, that is, $\text{co}(a,c) < n/4$. Hence, of the more than $3n/4$ partitions in which a and b are co-clustered, there are less than $n/4$ partitions in which a and c are co-clustered. Therefore, in more than $n/2$ partitions it holds that a and b are co-clustered and a and c are anti-clustered. In these partitions also b and c are anti-clustered which implies $\text{anti}(b,c) > n/2$. □

For $t < 3/4$ we cannot make a similar inference on the values of $\text{co}(b,c)$ and $\text{anti}(b,c)$ for an element pair b,c that forms t-tidy pairs with a third element a. This can be seen from the following counterexample. Let the input consist of the following partitions:

$$\begin{aligned} C_i &:= \{\{a,b,c\}\}, 1 \le i \le n/2-1, \\ C_i &:= \{\{a,b\},\{c\}\}, n/2 \le i \le 3n/4-1, \\ C_i &:= \{\{a,c\},\{b\}\}, 3n/4 \le i \le n-1, \\ C_n &:= \{\{a\},\{b\},\{c\}\}. \end{aligned}$$

Then, $\text{co}(a,b) = 3n/4-1$ and $\text{co}(a,c) = 3n/4-1$. With increasing n these values are arbitrarily close to $3n/4$. Hence, for any $t < 3/4$ and sufficiently large n both $(ab)_t$ and $(ac)_t$ hold. The value of $\text{co}(b,c)$, however, is only $n/2-1$. Hence, $3/4$ is the smallest value for which a statement similar to Lemma 4.1 can be obtained. For $2/3$-tidy pairs, the conclusions that can be drawn from the existence of tidy pairs are therefore less strict. Informally, the difference is that for $2/3$-tidy pairs we only make observations about other $2/3$-*tidy* pairs, whereas for $3/4$-tidy pairs we could make observations about all other pairs.

Lemma 4.2. *Let $\{a,b,c\}$ be a set of three elements where b and c form a $2/3$-tidy pair. Then,*

- $(ab)_{2/3} \wedge (ac)_{2/3} \Rightarrow (bc)_{2/3}$, *and*
- $(ab)_{2/3} \wedge (a \leftrightarrow c)_{2/3} \Rightarrow (b \leftrightarrow c)_{2/3}$.

Proof. We prove both statements separately.

First, let $(ab)_{2/3} \wedge (ac)_{2/3}$ hold. Then, there are more than $n/3$ partitions in which a, b, and c are co-clustered. Hence, $\text{co}(b,c) > n/3$ which implies (bc) since b and c form a $2/3$-tidy pair.

Second, let $(ab)_{2/3}$ and $(a \leftrightarrow c)_{2/3}$ hold. Then, there are more than $n/3$ partitions in which a and b are co-clustered and a and c are anti-clustered. Hence, $\text{anti}(b,c) > n/3$ which implies $(b \leftrightarrow c)_{2/3}$ since b and c form a 2/3-tidy pair. □

As in the case of Lemma 4.1 and 3/4-tidy pairs, Lemma 4.2 is tight in the sense that for any $t < 2/3$ we cannot obtain a similar result. This can be seen from the following counterexample. Let the input consist of the following partitions:

$$C_i := \{\{a,b,c\}\}, 1 \leq i \leq n/3,$$
$$C_i := \{\{a,b\},\{c\}\}, n/3+1 \leq i \leq 2n/3-1,$$
$$C_i := \{\{a,c\},\{b\}\}, 2n/3 \leq i \leq n-2,$$
$$C_i := \{\{a\},\{b\},\{c\}\}, n-1 \leq i \leq n.$$

In this instance, $\text{co}(a,b) = 2n/3-1$ and $\text{co}(a,c) = 2n/3-1$. Hence, for any $t < 2/3$ and sufficiently large n both $(ab)_t$ and $(ac)_t$ hold. The value of $\text{anti}(b,c)$, however, is $2n/3$. Consequently, $(b \leftrightarrow c)_t$ also holds. Therefore, a statement similar to Lemma 4.2 does not hold for any $t < 2/3$.

The fact that $t = 3/4$ and $t = 2/3$ lead to the most general definition of tidy pairs for which statements like Lemma 4.1 and Lemma 4.2, respectively, can be obtained is the reason for considering these particular values for t. As we will see in Sections 4.3 and 4.4, the difference between Lemma 4.1 and Lemma 4.2, that is, the circumstance that for 2/3-tidy pairs, we can only observe properties for other 2/3-tidy pairs, results in very different data reduction rules for the two tidiness definitions. In particular, the data reduction rule for 3/4-tidy pairs is much simpler than the one for 2/3-tidy pairs.

Next, we show that Consensus Clustering is polynomial-time solvable in case the input contains only 2/3-tidy pairs. This serves as an indication that the definitions of dirtiness and tidiness may lead to fixed-parameter tractability results. Note that, since every 3/4-tidy pair is also a 2/3-tidy pair, the following proposition also shows the polynomial-time solvability in case the input only contains 3/4-tidy pairs.

Proposition 4.1. Consensus Clustering *is solvable in polynomial time if the input contains only* 2/3*-tidy pairs.*

Proof. Let C be a median partition. We show that the following two statements are true.

1. If $(ab)_{2/3}$, then a and b are co-clustered in C.
2. If $(a \leftrightarrow b)_{2/3}$, then a and b are anti-clustered in C.

Clearly, since there are only 2/3-tidy pairs, any pair $a, b \in S$ must fulfill either $(ab)_{2/3}$ or $(a \leftrightarrow b)_{2/3}$. Hence, the two statements specify for each pair of elements whether they are co-clustered or not.

To prove the first statement, suppose that there is a median partition C not fulfilling it. Then, there must exist two clusters S_a and S_b of C with $a \in S_a$ and $b \in S_b$. One can further partition both S_a and S_b into each time two subsets. More specifically, let $S_a^1 := \{x \in S_a : (ax)_{2/3}\}$ and $S_a^2 := S_a \setminus S_a^1$. The sets S_b^1 and S_b^2 are defined analogously with respect to b. In this way, by replacing S_a and S_b with $S_a^1 \cup S_b^1$, S_a^2, and S_b^2, one obtains a modified partition C'. Consider any $x \in S_a^1$ and any $y \in S_a^2$. Then, $(x \leftrightarrow y)_{2/3}$ follows from $(ax)_{2/3}$, $(a \leftrightarrow y)_{2/3}$, and Lemma 4.2. The same is true with respect to S_b^1 and S_b^2. Moreover, if $x \in S_a^1$ and $y \in S_b^2$, this means that $(ax)_{2/3}$ and $(b \leftrightarrow y)_{2/3}$, implying by Lemma 4.2 and using $(ab)_{2/3}$ that $(x \leftrightarrow y)_{2/3}$. It remains to consider $x \in S_a^1$ and $y \in S_b^1$. Then, again the application of Lemma 4.2 yields $(xy)_{2/3}$. Thus, C' is a better partition than C is because in C' now $(ab)_{2/3}$ holds for all elements $a, b \in S_a^1 \cup S_b^1$ (without causing any increased cost elsewhere). This contradicts C being a median partition, proving the first statement. The second statement is proved analogously.

For an input instance that only contains 2/3-tidy pairs, a polynomial-time algorithm for computing a median partition P thus is as follows: Initially set $P = \emptyset$. Pick an arbitrary element $a \in S$. Then, create a set S_a that contains a and all elements $b \in S$ for which $(ab)_{2/3}$ holds. Add S_a as one element to P, and remove S_a from S and all input partitions. Repeat this process until $S = \emptyset$. □

We have thus completed Step 1 of the framework. In Step 2 we show that the number of dirty pairs is bounded from above by a polynomial of the average distance of the input partitions.

Lemma 4.3. *For an input instance of* Consensus Clustering *with average distance d*

- *the number of $3/4$-dirty pairs is less than $8d/3$, and*
- *the number of $2/3$-dirty pairs is less than $9d/4$.*

Proof. We first show the statement for 3/4-dirty pairs, and then show how the bound for 2/3-dirty pairs can be achieved.

We claim that every dirty pair $(a\#b)_{3/4}$ contributes at least $3n^2/8$ to $\sum_{C_i,C_j \in \mathcal{C}} \mathrm{dist}(C_i, C_j)$, which is the overall distance between the input partitions. Given that, the statement of Lemma 4.3 follows by observing that $\sum_{C_i,C_j \in \mathcal{C}} \mathrm{dist}(C_i, C_j) = d \cdot n \cdot (n-1)$. Hence, it remains to prove the claim. First, recall that for every dirty pair $\mathrm{co}(a,b) \geq n/4$ and $\mathrm{anti}(a,b) \geq 3n/4$. Clearly, $\mathrm{co}(a,b) + \mathrm{anti}(a,b) = n$. To show that a dirty pair $(a\#b)_{3/4}$ contributes at least $3n^2/8$ to the overall distance, note that each pair makes the contribution

$$\mathrm{co}(a,b) \cdot (n - \mathrm{co}(a,b)) + \mathrm{anti}(a,b) \cdot (n - \mathrm{anti}(a,b)) = 2 \cdot \mathrm{co}(a,b) \cdot \mathrm{anti}(a,b). \tag{4.1}$$

The product of two rational numbers x and y with $x + y = n$ reaches a maximum for $x = y = n/2$ and decreases with decreasing $x \leq n/2$. Hence, for $x \geq n/4$, the product is at least $(n/4) \cdot (3n/4)$. The minimum contribution is thus at least $2 \cdot (n/4) \cdot (3n/4) = 3n^2/8$.

For 2/3-dirty pairs, we claim that every dirty pair $(a\#b)_{2/3}$ contributes at least $4n^2/9$ to the overall distance $\sum_{C_i,C_j\in\mathcal{C}} \text{dist}(C_i,C_j)$. Given that, the statement of Lemma 4.3 follows as in the proof for 3/4-dirty pairs. Recall that for every dirty pair $\text{co}(a,b) \geq n/3$ and $\text{anti}(a,b) \geq n/3$. Again, $\text{co}(a,b)+\text{anti}(a,b) = n$ and any pair makes the contribution $2\cdot\text{co}(a,b)\cdot\text{anti}(a,b)$. Using the same arguments as for 3/4-dirty pairs, it follows that the minimum contribution is at least $2\cdot(n/3)\cdot(2n/3) = 4n^2/9$. □

In the next two sections, we show how Steps 3 and 4 can be carried out for 3/4-dirtiness (Section 4.3) and 2/3-dirtiness (Section 4.4) and present our main fixed-parameter tractability results for CONSENSUS CLUSTERING.

4.3 Data Reduction for 3/4-Tidy Elements

Step 3 of our framework now calls for a polynomial-time data reduction that reduces the number of elements that only appear in tidy pairs or, equivalently, that do not appear in any dirty pair. For 3/4-tidy pairs, we call these elements 3/4-*tidy elements* and all other elements 3/4-*dirty elements*. For 3/4-tidy elements, we devise a very simple data reduction rule that is mainly based on Lemma 4.1. This rule exploits the observation that for a 3/4-tidy element the set of elements with which it is co-clustered in more than $3n/4$ input partitions forms a cluster of every median partition.

Reduction Rule 4.1. *Let $a \in S$ be a 3/4-tidy element, and let $K := \{a\} \cup \{b \in S \mid (ab)_{3/4}\}$. Then, remove the elements of K from all input partitions, and set*

$$k := k - \left(\sum_{\{a,b\}\subseteq K} \text{anti}(a,b)\right) - \sum_{a\in K}\sum_{b\in S\setminus K} \text{co}(a,b).$$

Lemma 4.4. *Rule 4.1 is correct and can be exhaustively performed in $O(n\cdot|S|^2)$ time.*

Proof. Let a and K be as defined in Rule 4.1. By Lemma 4.1, for each pair of elements $b,c \in K$ it holds that $\text{co}(b,c) > n/2 > \text{anti}(a,b)$. Furthermore, for each pair of elements $b \in K$ and $c \in S\setminus K$ it holds that $\text{anti}(a,b) > n/2 > \text{co}(a,b)$. We show that a partition C that does not contain K as one of its clusters is not a median partition.

Let C be a partition that does not contain K as one of its clusters, and let $\text{dist}(C) := \sum_{C_i\in\mathcal{C}} \text{dist}(C,C_i)$ denote the distance of this median partition to the input partitions. Then, either there is a cluster K' that contains elements from K and elements from $S\setminus K$ or there are at least two clusters K_1 and K_2 that are subsets of K.

In the first case, we can obtain a better partition by replacing K' with the two clusters $K_1 := K'\setminus K$ and $K_2 := K\cup K'$. Clearly, the distance from this new partition to the input partitions is

$$\text{dist}(C) + \sum_{a\in K_1}\sum_{b\in K_2} \text{co}(a,b) - \text{anti}(a,b).$$

Since $\text{anti}(a,b) > \text{co}(a,b)$ for all pairs in the sum, this is smaller than $\text{dist}(C)$. Hence, C is not a median partition.

In the second case, we can obtain a better partition by replacing K_1 and K_2 with the cluster $K' = K_1 \cup K_2$. The distance from this new partition to the input partitions is

$$\text{dist}(C) + \sum_{a \in K_1} \sum_{b \in K_2} \text{anti}(a,b) - \text{co}(a,b).$$

Since $\text{co}(a,b) > \text{anti}(a,b)$ for all pairs in the sum, this is smaller than $\text{dist}(C)$. Hence, C is not a median partition. The decrement in k is precisely the distance that is caused by the elements of K in a median partition of the original input instance.

It remains to show the running time. In $O(n \cdot |S|^2)$ time we can compute $\text{co}(a,b)$ and $\text{anti}(a,b)$ for each pair of elements and for each element the number of dirty pairs that it is contained in. Then, we build in $O(n \cdot |S|^2)$ time a complete graph G that contains all the elements of S, and additionally store for each vertex a the number of dirty pairs that its corresponding element is contained in and for each edge $\{a,b\}$ whether $(ab)_{3/4}$ or $(a \leftrightarrow b)_{3/4}$ holds. As long as the graph contains a vertex that does not appear in a dirty pair, we perform Rule 4.1 (K can be computed by scanning through the adjacency list of the vertex). Then we remove K from G and simultaneously update the number of dirty pairs that every vertex is contained in. This can be done within the same running time as removing K from G since we check for every removed edge $\{a,b\}$ whether $(ab)_{3/4}$ or $(a \leftrightarrow b)_{3/4}$ hold, and update the number of dirty pairs if neither is the case. When every element appears in at least one dirty pair, the rule has been exhaustively applied. Since every edge is visited only a constant number of times using this approach, the overall running time for the application of the rule after the graph G has been constructed is $O(n \cdot |S|^2)$ also. The overall running time follows. □

After the exhaustive application of Rule 4.1, every element appears in at least one 3/4-dirty pair which leads to our main result for the parameter average distance.

Theorem 4.1. *1.* Consensus Clustering *can be reduced in $O(n \cdot |S|^2)$ time to a structural problem kernel with less than $16d/3$ elements in S, all of which are 3/4-dirty.*

2. Consensus Clustering *is fixed-parameter tractable with respect to the average distance d between the input partitions as well as with respect to the number of 3/4-dirty elements.*

Proof. Consider an instance I that is reduced with respect to Rule 4.1. Clearly, every element in I is 3/4-dirty. By Lemma 4.3, the number of 3/4-dirty pairs in I is less than $8d/3$. Hence, the overall number of elements in a reduced instance is less than $2 \cdot 8d/3 = 16d/3$. Hence, Consensus Clustering admits a structural problem kernel with less than $16d/3$ elements which can be computed in $O(n \cdot |S|^2)$ time.

The second part of the theorem follows from the observation that we can solve CONSENSUS CLUSTERING by trying all possible partitions (whose number is clearly a function of d in the structural problem kernel), computing their distance to the input partitions in polynomial time, and then outputting "yes" when a partition with distance at most k has been found. □

4.4 Data Reduction for 2/3-Tidy Elements

In this section, we show that we can also obtain fixed-parameter tractability results by considering 2/3-tidy/dirty pairs instead of 3/4-tidy/dirty pairs. While this does not lead to a better bound on the number of elements with respect to the average distance d, we obtain fixed-parameter tractability with respect to the parameter "number of 2/3-dirty elements", that is, the number of elements that are contained in at least one 2/3-dirty pair. This parameter can be much smaller than the average distance d. Consider for example the following instance:

$$C_i := \{\{s_1, s_2, \ldots, s_m\}\}, 1 \leq i \leq 2n/3+1,$$
$$C_i := \{\{\{s_1\}, \{s_2\}, \ldots, \{s_m\}\}, 2n/3+2 \leq i \leq n.$$

The instance does not contain any 2/3-dirty pairs and, consequently, also no 2/3-dirty elements. The average distance d of the input partitions, however, increases with increasing m since for constant n each pair of elements makes a constant contribution to the overall distance among the input partitions. Therefore, the average distance d can be arbitrarily large even in instances that do not contain any 2/3-dirty pairs or any 2/3-dirty elements. Note that by Lemma 4.3 the number of 2/3-dirty pairs is less than $9d/4$. Hence, the parameters "number of 2/3-dirty pairs" and "number of 2/3-dirty elements" are stronger parameters than the average distance d. This also holds for the parameters "number of 3/4-dirty pairs" and "number of 3/4-dirty elements" which can be shown by adapting the example above. These two parameters, however, are also always at least as large as the number of "number of 2/3-dirty pairs" and "number of 2/3-dirty elements". Furthermore, as the example above shows, they can also be arbitrarily large compared to the "number of 2/3-dirty pairs" and "number of 2/3-dirty elements".

For notational simplicity, we refer to 2/3-dirty pairs/elements and 2/3-tidy pairs/elements simply as dirty pairs/elements and tidy pairs/elements in the following. Also, we use (ab), $(a \leftrightarrow b)$, and $(a\#b)$ instead of $(ab)_{2/3}$, $(a \leftrightarrow b)_{2/3}$, and $(a\#b)_{2/3}$.

Roughly speaking, the aim of our data reduction rule is to find subsets of S that contain many tidy elements a, b for which (ab) holds. If these subsets are too large, then we can reduce the instance. In order to find such subsets, we describe a partition of S that is based on its tidy elements. In the following, let S_1 denote the tidy elements of S, and S_2 the dirty elements. First, we describe a partition $P_1 = \{S_1^1, \ldots, S_1^l\}$ of S_1

into equivalence classes according to the tidy pairs in S_1. Then, we show that these equivalence classes also induce a partition of S_2.

For each equivalence class $S_1^i \in P_1$, we demand that

- $\forall a \in S_1^i \, \forall b \in S_1^i : (ab)$ and
- $\forall a \in S_1^i \, \forall b \in S \setminus S_1^i : (a \leftrightarrow b)$.

Observe that, by Lemma 4.2, the partition P_1 of S_1 fulfilling these requirements is well-defined, since the predicate (ab) describes a transitive relation over S_1. Using P_1, we define the subsets S_2^i of S_2 as follows:

$$S_2^i := \{a \in S_2 \mid \exists b \in S_1^i : (ab)\}.$$

Informally, each S_2^i is the set of elements $a \in S_2$ that are often co-clustered with at least one element $b \in S_1^i$. We also define one additional set S_2^0 that contains all elements $a \in S_2$ such that there is no $b \in S_1$ for which (ab) holds.

Finally, we obtain a set of subsets $P = \{S^0, S^1, \ldots, S^l\}$ of S by setting $S^i := S_1^i \cup S_2^i$ for $1 \leq i \leq l$ and $S^0 := S_2^0$. We call this set of subsets *tidiness-based*. The following lemma shows that P is indeed a partition of S, and also provides some further structural properties of P.

Lemma 4.5. *Let $P = \{S^0, S^1, \ldots, S^l\}$ be a tidiness-based set of subsets of S constructed as described above. Then, P is a partition of S, and for each $S^i \in P$ it holds that*

- $\forall a \in S^i \, \forall b \in S : (ab) \Rightarrow b \in S^i$ *and*
- $\forall a, b \in S^i, i \geq 1 : (ab) \vee (a\#b)$.

Proof. First, we show that P is a partition. By Lemma 4.2, it is easy to verify that the claim holds for the partition P_1 of S_1. By definition, $\bigcup_{i=0}^{l} S_2^i = S_2$. We now show that for each $a \in S_2$ there is exactly one set S_2^i that contains a, and, thus, that P is a partition of S. By definition, S^0 does not overlap with any other set S^i, $i \geq 1$. Now, suppose that there are two sets S_2^i, $i \geq 1$, and S_2^j, $j \geq 1$, $j \neq i$, containing a. Then there are two elements $b \in S_1^i$ and $c \in S_1^j$ such that (ab) and (ac) holds. Since P_1 is a partition of S_1, we have $c \notin S_1^i$ and thus also $b(\leftrightarrow c)$. But then it follows from Lemma 4.2 that $(b \leftrightarrow a)$ holds (since we have $(b \leftrightarrow c)$ and (ca)). This clearly contradicts (ab). We have thus shown that P is a partition of S.

We now show that for each $S^i \in P$ it holds that $\forall a \in S^i \forall b \in S : (ab) \Rightarrow b \in S^i$. Suppose that there is a pair of elements $a \in S^i$ and $b \in S^j$, $j \neq i$, for which (ab) holds. By definition, this can be only the case if $a \in S_2$ and $b \in S_2$. Without loss of generality, assume that $i \geq 1$. This means that there is some element $c \in S_1^i$ with (ac). However, by Lemma 4.2, then also (cb) must hold. This contradicts $b \notin S^i$.

Finally, we show that for each $S^i \in P$, $i \geq 1$, it holds that $\forall a, b \in S^i : (ab) \vee (a\#b)$. Suppose that there is some S^i containing two elements a and b for which $(a \leftrightarrow b)$

holds. By definition of S_1^i, one of a and b must be from S_2^i, say $a \in S_2^i$, and there must be some $c \in S_1^i$ such that (ac) holds. By Lemma 4.2, we have $(c \leftrightarrow b)$. This means, however, that, also by Lemma 4.2, we have $(b \leftrightarrow d)$ for all $d \in S_1^i$. This contradicts $b \in S^i$. □

Informally, Lemma 4.5 says that inside any $S^i \in P$ we have only pairs that are *co-clustered* in more than $2n/3$ input partitions or dirty pairs; between two subsets $S^i \in P$ and $S^j \in P$ we have only dirty pairs or pairs that are *anti-clustered* in more than $2n/3$ input partitions. Clearly, the elements in S_1^i then are co-clustered in more than $2n/3$ partitions with *all* elements in S^i and are anti-clustered in more than $2n/3$ partitions with *all* elements in $S \setminus S^i$. This means that an S^i with too many elements in S_1^i is forced to become a set of a median partition. With the subsequent data reduction rule, we remove these sets from the input.

We introduce the following notation for subsets of S.

Definition 4.3. *Let E and F be subsets of S. We denote with*

- $\mathrm{dp}(E) := \{\{a,b\} \mid a,b \in E \wedge (a\#b)\}$ *the set of dirty pairs among the elements of E, and with*
- $\mathrm{dp}(E,F) := \{\{a,b\} \mid a \in E, b \in F \wedge (a\#b)\}$ *the set of dirty pairs between E and F.*

Using this notation, our data reduction rule reads as follows.

Reduction Rule 4.2. *Let P be a tidy-based partition of S. If there is an $S^i \in P$ such that*

$$|S_1^i| > |\mathrm{dp}(S^i)| + |\mathrm{dp}(S^i, S \setminus S^i)|.$$

Then, remove the elements of S^i from all input partitions, and set

$$k := k - \left(\sum_{a,b \in S^i} \mathrm{anti}(a,b) \right) - \sum_{a \in S^i} \sum_{b \in S \setminus S^i} \mathrm{co}(a,b)$$

Lemma 4.6. *Rule 4.2 is correct and can be exhaustively performed in $O(n \cdot |S|^2)$ time.*

Proof. Let S^i be as described in Rule 4.2. The correctness proof is based on the following.

> *Claim:* Every median partition C contains one cluster C^j such that $S^i = C^j$.

For our proof, we only consider clusters C^j that contain at least one element of S^i. In what follows, we partition each such C^j into four subsets. Figure 4.2 shows these sets and their relation to S^i.

- $C_1^j := \{a \in C^j \cap S^i \mid \forall b \in C^j \setminus S^i : (a \leftrightarrow b)\}$ contains those elements from S^i that do not appear in dirty pairs with elements from $C^j \setminus S^i$.

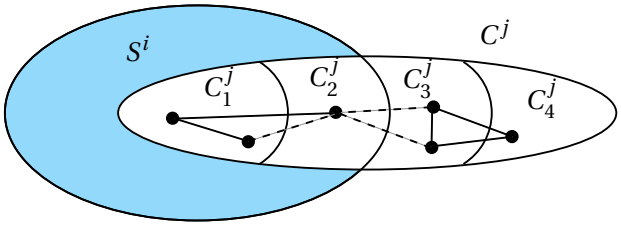


Figure 4.2: The subsets of a cluster C^j with respect to the set S^i as defined in the proof of Lemma 4.6. Solid edges are between elements a and b for which (ab) holds; dashed edges are between elements that form a dirty pair; elements a and b for which $(a \leftrightarrow b)$ holds have no edge between them.

- $C_2^j := \{a \in C^j \cap S^i \mid \exists b \in C^j \setminus S^i : (a\#b)\}$ contains the (dirty) elements from S^i that form a dirty pair with some element from $C^j \setminus S^i$.
- $C_3^j := \{a \in C^j \cap (S \setminus S^i) \mid \exists b \in S^i \cap C^j : (a\#b)\}$ contains the elements of $C^j \setminus S^i$ that form a dirty pair with some element from $S^i \cap C^j$.
- $C_4^j := C^j \setminus (C_1^j \cup C_2^j \cup C_3^j)$ contains all other elements.

We prove the claim in three steps. First, we show that $|C_1^j| \geq |C_2^j|$ implies $C^j = (C_1^j \cup C_2^j)$. Then, we show that there is exactly one C^j with $C^j = (C_1^j \cup C_2^j)$. Finally, we show that in a median partition, there is no C^j with $|C_1^j| < |C_2^j|$. The first two of these statements imply that there is exactly one cluster C^j with $C^j \subseteq S^i$. The third statement implies that there can be no other clusters that have nonempty intersection with S^i. Altogether, this means that in a median partition there is exactly one cluster C_j with $C_j \cap S^i \neq \emptyset$, which proves the correctness of Rule 4.2.

Now, we show that in a median partition C, there is no C^j such that $|C_1^j| \geq |C_2^j|$ and $C^j \neq (C_1^j \cup C_2^j)$. More precisely, we show that for any partition C which contains such a cluster C^j there is an alternative partition C' that has lower cost. This partition C' is constructed as follows: replace the cluster C^j by two new clusters $C_1^j \cup C_2^j$ and $C_3^j \cup C_4^j$. We now show that C' has lower cost than C. Let $d(C)$ denote the cost of the partition C, and let $d(C')$ be the cost of the partition C'. Clearly, the costs of C and C' differ only in the costs for the pairs that contain one element from $C_1^j \cup C_2^j$ and one from $C_3^j \cup C_4^j$. For each pair of elements $a \in C_1^j \cup C_2^j$ and $b \in C_3^j \cup C_4^j$, partition C' saves the cost of $\text{anti}(a,b)$ compared to C, since these two elements now appear in different clusters. However, this means that C' has an additional cost of $\text{co}(a,b)$ for each such pair. Note that, by definition, the following holds:

- $(ab) \Rightarrow (\text{co}(a,b) - \text{anti}(a,b) > n/3)$,
- $(a \leftrightarrow b) \Rightarrow (\text{anti}(a,b) - \text{co}(a,b) > n/3)$, and
- $(a\#b) \Rightarrow (|\text{anti}(a,b) - \text{co}(a,b)| \leq n/3)$.

Overall, the cost difference between C and C' is then

$$\begin{aligned} d(C) - d(C') &= \sum_{a \in C_1^j \cup C_2^j} \sum_{b \in C_3^j \cup C_4^j} \text{anti}(a,b) - \text{co}(a,b) \\ &\overset{(*)}{>} \sum_{a \in C_1^j} \sum_{b \in C_3^j} \frac{n}{3} - \sum_{a \in C_2^j} \sum_{b \in C_3^j} \frac{n}{3} \\ &\overset{(**)}{\geq} 0. \end{aligned}$$

Inequality $(*)$ follows from the following four facts:

1. $\forall a \in C_1^j \cup C_2^j \forall b \in C_4^j : (a \leftrightarrow b)$. Hence, $\text{anti}(a,b) - \text{co}(a,b) > 0$ for $b \in C_4^j$.

2. $\forall a \in C_1^j \forall b \in C_3^j : (a \leftrightarrow b)$. Hence, $\text{anti}(a,b) - \text{co}(a,b) > n/3$ for $b \in C_4^j$ for these pairs.

3. $\forall a \in C_2^j \forall b \in C_3^j : (a \leftrightarrow b) \vee (a \# b)$. Hence, $\text{anti}(a,b) - \text{co}(a,b) \geq -n/3$ for these pairs.

4. $C_3^j \cup C_4^j \neq \emptyset$. Hence, the first sum is indeed strictly larger than the term in the second row.

Inequality $(**)$ follows from the fact that $|C_1^j| \geq |C_2^j|$. Thus, we have shown that in a median partition there can be no clusters C^j with $|C_1^j| \geq |C_2^j|$ and $C^j \neq (C_1^j \cup C_2^j)$. Hence, we can have clusters C^j of two types, those with $C^j = (C_1^j \cup C_2^j)$ and those with $|C_1^j| < |C_2^j|$.

Next, we show that in a median partition, there is exactly one cluster with $C^j = (C_1^j \cup C_2^j)$. Let C_{iso} be the set of clusters C^j with $C^j = (C_1^j \cup C_2^j)$, that is, the clusters that are "isolated" from $S \setminus S^i$. Let C be a partition that creates more than one cluster in C_{iso}. We show that there is an alternative partition C' that merges two clusters of C_{iso} to a new cluster and that has lower cost than C. First, there must be two clusters $C^j \in C_{\text{iso}}$ and $C^l \in C_{\text{iso}}$ such that $|(C^j \cup C^l) \cap S_1| > \text{dp}(C^j \cup C^l)$, because, otherwise, the union of all clusters in C_{iso} has more dirty pairs than tidy elements. However, this is also the case for all other clusters C^h, since for these clusters we have $|C_1^h| < |C_2^h|$, which means that then S^i has more dirty pairs than tidy elements, contradicting the precondition of the reduction rule. Our alternative partition C' merges C^j and C^l into a new cluster $C^j \cup C^l$. All other clusters are the same as in C. The costs of C and C' differ only with respect to pairs that contain one element $a \in C^j$ and one element $b \in C^l$. For each pair, putting the elements in the same cluster instead of two different clusters saves $\text{co}(a,b)$ and

costs anti(a,b). The cost difference between C and C' is thus

$$\begin{aligned} d(C) - d(C') &= \sum_{a \in C^j} \sum_{b \in C^l} \mathrm{co}(a,b) - \mathrm{anti}(a,b) \\ &\overset{(*)}{\geq} \sum_{a \in (C^j \cup C^l) \cap S_1} \frac{n}{3} - |\mathrm{dp}(C^j, C^l)| \cdot \frac{n}{3} \\ &\overset{(**)}{>} 0. \end{aligned}$$

Inequality $(*)$ follows from the two facts

1. $\forall a \in C^j \forall b \in C^l : (ab) \vee (a\#b)$. Hence, all pairs $a \in C^j, b \in C^l$ either form a dirty pair (then $\mathrm{co}(a,b) - \mathrm{anti}(a,b) \geq -n/3$) or it holds that $\mathrm{co}(a,b) - \mathrm{anti}(a,b) > n/3$.

2. $\forall a \in (C^j \cup C^l) \cap S_1 \forall b \in C^j \cup C^l : (ab)$. Hence, for these pairs $\mathrm{co}(a,b) - \mathrm{anti}(a,b) > n/3$.

Inequality $(**)$ follows from the fact that $|(C^j \cup C^l) \cap S_1| > \mathrm{dp}(C^j \cup C^l)$. Hence, partition C is clearly not a median partition. We have thus shown that in a median partition there is at most one cluster C^j with $C^j = (C_1^j \cup C_2^j)$, and possibly some other clusters C^l with $|C_1^l| < |C_2^l|$. Furthermore, by the precondition of Rule 4.2, this means that there must be *exactly* one cluster C^j with $C^j = (C_1^j \cup C_2^j)$ in a median partition C.

We complete the proof of the correctness of Rule 4.2 by showing that in a median partition there is no cluster C^l with $|C_1^l| < |C_2^l|$. Let C^j be the cluster with $C^j = (C_1^j \cup C_2^j)$. We show that a median partition C never contains a cluster C^l with $|C_1^l| < |C_2^l|$, since then we can obtain a better partition C' by removing $C_1^l \cup C_2^l$ from C^l and merging $C_1^l \cup C_2^l$ and C^j into a new cluster $C^j \cup C_1^l \cup C_2^l$. First, observe that, by the precondition of Rule 4.2, we have $|(C^j \cup C^l) \cap S_1| > \mathrm{dp}(C^j \cup C^l) + \mathrm{dp}(C^j \cup C^l, S \setminus (C^j \cup C^l))$. Otherwise we would have $|S^i \cap S_1| < \mathrm{dp}(S^i) + \mathrm{dp}(S^i, S \setminus S^i)$, since already $C^j \cup C_1^l \cup C_2^l$ has less tidy elements than dirty pairs, and for each other cluster C^h from D there are more dirty pairs than tidy elements (since $|C_1^h| < |C_2^h|$). We now compare the cost of C with the cost of C'. First, the costs have changed for pairs with $a \in C^j$ and $b \in C_1^l \cup C_2^l$, where in C' we have—compared to C—an additional cost of anti(a,b) and save a cost of co(a,b), since a and b are now in the same clusters. Second, the costs have changed for pairs $a \in C_1^l \cup C_2^l$ and $b \in C_3^l \cup C_4^l$, where in C' we have—compared to C—an additional cost of co(a,b) and save a cost of anti(a,b). Overall, the cost difference

is

$$\begin{aligned}
d(C) - d(C') &= \sum_{a \in C^j} \sum_{b \in C_1^l \cup C_2^l} \operatorname{co}(a,b) - \operatorname{anti}(a,b) \\
&\quad + \sum_{a \in C_1^l \cup C_2^l} \sum_{b \in C_3^l \cup C_4^l} \operatorname{anti}(a,b) - \operatorname{co}(a,b) \\
&\overset{(*)}{>} -|\operatorname{dp}(C^j, C^l \cap S^i)| \cdot \frac{n}{3} + \sum_{a \in C^j \cap S_1} \sum_{b \in C_1^l \cup C_2^l} \frac{n}{3} \\
&\quad - |\operatorname{dp}(C_2^j, C_3^j)| \cdot \frac{n}{3} + \sum_{a \in C_1^l} \sum_{b \in C_3^l \cup C_4^l} \frac{n}{3} \\
&\overset{(**)}{\geq} \frac{n}{3} \cdot (|C^j \cap S_1| \cdot |C_1^l \cup C_2^l| + |C_1^l| \cdot |C_3^l \cup C_4^l|) \\
&\quad - \frac{n}{3} \cdot (|\operatorname{dp}(C^j, C^l \cap S^i)| + |\operatorname{dp}(C_2^l, C_3^l)|) \\
&\overset{(***)}{>} 0.
\end{aligned}$$

Inequality $(*)$ follows from the following facts:

1. $\forall a \in C^j \forall b \in C_1^l \cup C_2^l : (ab) \vee (a \# b)$,
2. $\forall a \in C_1^l \forall b \in C_3^l \cup C_4^l : (a \nleftrightarrow b)$, and
3. $\forall a \in C_2^l \forall b \in C_3^l \cup C_4^l : (a \nleftrightarrow b) \vee (a \# b)$.

Inequality $(**)$ is straightforward, and inequality $(***)$ follows from the fact that $|(C^j \cup C_1^l) \cap S_1| > \operatorname{dp}(C^j \cup C_1^l \cup C_2^l) + \operatorname{dp}(C^j \cup C_1^l \cup C_2^l, S \setminus (C^j \cup C_1^l \cup C_2^l))$. A median partition thus does not contain a cluster C^l with $|C_1^l| < |C_2^l|$. Therefore, a median partition contains exactly one cluster C^j that contains all the elements from S_i and no other elements. The decrement in k is precisely the distance that is caused by the elements of S_1 in a median partition of the original input instance. Altogether, this proves the correctness of Rule 4.2.

It remains to show the running time of Rule 4.2. In $O(n \cdot |S|^2)$ time we can compute $\operatorname{co}(a,b)$ and $\operatorname{anti}(a,b)$ for each pair of elements and for each element the number of dirty pairs that it is contained in. Then, we build in $O(n \cdot |S|^2)$ time a complete graph G that contains all the elements of S, and furthermore store for each edge $\{a,b\}$ whether (ab) or $(a \nleftrightarrow b)$ hold. We then compute the subgraph of G that only contains the tidy elements and edges $\{a,b\}$ for which (ab) holds. The connected components of this graph are precisely the sets of the partition P_1 of S_1. Then, we label the vertices of S_2 by the (at most one) set S_1^i of P_1 to which they are connected by an edge $\{a,b\}$ for which (ab) holds. Then, we compute for each S_i the values of $|\operatorname{dp}(S^i)|$ and $|\operatorname{dp}(S^i, S \setminus S^i)|$. Each of these steps can be performed in $O(n \cdot |S|^2)$ time by traversing the edge set once. Then, as long as P_1 contains a set S_1^i to which Rule 4.2 applies, we

perform the rule and simultaneously update the value of $|\mathrm{dp}(S^j, S \setminus S^j)|$ for all other sets S^j. This can be done within the same running time as removing S^i from G since we check for every removed edge a, b whether it corresponds to a dirty pair between S^i and some other S^j, and update $|\mathrm{dp}(S^j, S \setminus S^j)|$ if this is the case. Since every edge is visited only a constant number of times using this approach, the overall running time for the exhaustive application of the rule is $O(n \cdot |S|^2)$. □

In the following theorem, we combine Steps 3 and 4 of our framework: we show that exhaustively applying the reduction rule yields an equivalent instance whose number of elements is less than $9d$. Observe that this bound on $|S|$ is worse than the bound of $16d/3$ achieved in Theorem 4.1. However, we obtain fixed-parameter tractability with respect to the parameter "number of 2/3-dirty elements" which—as we have shown—is possibly much smaller than d.

Theorem 4.2. *1. An exhaustive application of Rule 4.2 produces in $O(n \cdot |S|^2)$ time a structural problem kernel for* Consensus Clustering *with less than $9d$ elements.*

2. Consensus Clustering *is fixed-parameter tractable with respect to the parameter "number of 2/3-dirty elements in S".*

Proof. By Lemma 4.6, Rule 4.2 can be performed exhaustively in $O(n \cdot |S|^2)$ time. Therefore, consider an instance $I = (\mathcal{C}, S, k)$ that is reduced with respect to the reduction rule. With S_1 we denote the tidy elements of I, and with S_2 we denote the elements of S that appear in dirty pairs. By Lemma 4.3, the number of dirty pairs in I is less than $9d/4$. Hence, the size of the set S_2 containing the elements appearing in dirty pairs is less than $9d/2$. It remains to bound the number of tidy elements. For this, consider the tidy-based partition P of S. Since the reduction rule cannot be applied, the number of tidy elements of each set $S^i \in P$ is bounded by the number of dirty pairs that contain at least one element from S^i. The overall size of the set S_1 containing the non-dirty elements can thus be bounded by

$$|S_1| \leq \sum_{S^i \in P} |\mathrm{dp}(S^i) + \mathrm{dp}(S^i, V \setminus S^i)| < 9d/2. \tag{4.2}$$

The second inequality follows from the fact that we have at most $9d/4$ dirty pairs and that the dirty pairs between different sets $S^i, S^j \in P$ have to be counted twice. Hence, a reduced instance contains at most $|S_1| + |S_2| < 2 \cdot (9d/2) < 9d$ elements.

The second part of the theorem follows from the observation that by Inequality 4.2, the number of tidy elements is upper-bounded by the number of dirty pairs. Since the number of dirty pairs is at most quadratic in the number of dirty elements, the number of tidy elements is upper-bounded by a polynomial of the number of dirty elements. Hence, one can solve Consensus Clustering by trying all possible partitions (whose number is a function of the number of dirty elements in the structural problem

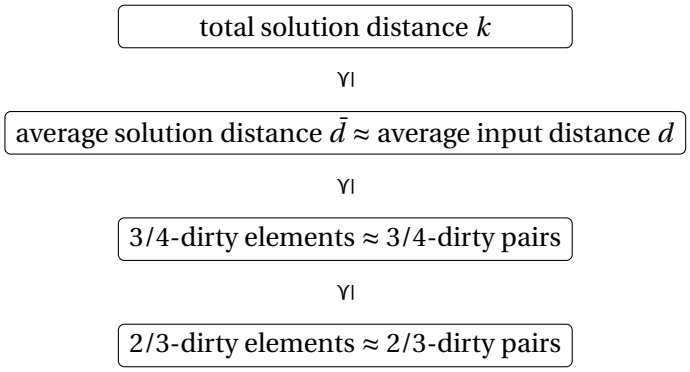


Figure 4.3: The relationship between the discussed parameters for CONSENSUS CLUSTERING: a downward '$\preceq$' means that the parameter on the bottom is *stronger* than the one on top, parameters in the same box are "equivalent" since one is upper-bounded by a function of the other and vice versa.

kernel), computing their distance to the input partitions in polynomial time, and then outputting "yes" when a partition with distance at most k has been found. □

4.5 Concluding Remarks

We have presented fixed-parameter tractability results for three different parameterizations of CONSENSUS CLUSTERING: average distance of the input partitions, number of 3/4-dirty elements, and number of 2/3-dirty elements. In Figure 4.3 we provide a condensed view of the relationships between the parameters considered for CONSENSUS CLUSTERING in this work. Our results rely on the new technique of structural kernelization. Since our work initiates the study of CONSENSUS CLUSTERING in the field of parameterized algorithmics and also introduces structural kernelizations, there are numerous research tasks worth pursuing.

- An efficient solving algorithm would complement our kernelization results. For the parameter d, an algorithm with running time $O(n|S|^2 + 4.24^d)$ has been recently shown [Dörnfelder et al. 2011]. Can we also obtain efficient algorithms for the stronger parameters "number of 2/3-dirty pairs" and "number of 2/3-dirty elements"?

- For KEMENY SCORE, recent experimental studies confirmed the usefulness of dirtiness-based data reduction rules [Betzler et al. 2010a]. Similar studies should be also performed for CONSENSUS CLUSTERING. In particular, it seems interesting to compare the two dirtiness definitions with each other, since one yields better results for the parameter average distance d whereas the other

yields fixed-parameter tractability for the stronger parameter "number of 2/3-dirty pairs".

– In CONSENSUS CLUSTERING the input consists only of partitions of S. This assumption is often too strict for realistic input settings. For example, one benchmark data set for CONSENSUS CLUSTERING consists of 22 clusterings of 8124 mushrooms (this data set belongs to the UCI machine learning repository [Frank and Asuncion 2010]). Each clustering is defined by an attribute, for example the color of the mushroom cap, but some values are missing from the data. To cope with these missing values van Zuylen and Williamson [2009] suggested to allow that the input consists of partitions of subsets of S and to measure the distance between the median partition and each input partition only for the subset S' of S that is partitioned by this input partition. The resulting problem is called PARTIAL CONSENSUS CLUSTERING. Can our approach also be applied to PARTIAL CONSENSUS CLUSTERING?

– The presented kernelization results are closely related to kernelization results for the KEMENY SCORE problem [Betzler et al. 2010a, 2011b] for which similar definitions of 3/4-dirtiness and 2/3-dirtiness were proposed. Interestingly, for CONSENSUS CLUSTERING we can obtain with both dirtiness definitions a structural problem kernel that is linear in d whereas for KEMENY SCORE the 3/4-dirtiness definition leads to a structural problem kernel with at most $11d$ elements and the 2/3-dirtiness so far only leads to a structural problem kernel with at most $162d^2 + 9d$ elements. This leads to the question whether the kernelization results—either for CONSENSUS CLUSTERING and 3/4-dirtiness or for KEMENY SCORE and 2/3-dirtiness—can be significantly improved or whether there are significant differences between the two problems in this regard.

– The usefulness and limits of the structural kernelization concept should be further explored. In the presented example, the "structure" of the input that was reduced was the number of elements which is somehow also related to the overall input size. In Section 4.1, we briefly described a reduction approach due to Dorn et al. [2010] that can be seen as a structural kernelization for branchwidth. Are there further nontrivial applications of structural kernelization where the dimension that is reduced is a structural parameter? Like Turing kernelization (cf. [Fernau et al. 2009, Lokshtanov 2009, Schäfer et al. 2011]), this could be a promising way to attack problems that do not admit a polynomial-size standard problem kernel (see [Bodlaender et al. 2009, Bodlaender 2009, Dom et al. 2009, Fortnow and Santhanam 2011] for more on lower bounrds for problem kernel sizes).

Part III

Querying

The second main part of this work deals with computational problems that arise in the context of the so-called *querying* of protein interaction networks. The input of such a querying problem consists of two main parts, the *query* which is either a small set of proteins or a small protein interaction network, and the *host* is a large protein interaction network. Usually the query and the host are from different species, and the task is to identify subnetworks of the host that are similar to the query [Sharan and Ideker 2006]. If such a subnetwork is found, then it can be conjectured that this subnetwork is a functional module also in the species of the host. In this work, we call subnetworks of the host that are similar to the query "occurrences of the query".

This rather informal description of the network querying problem gives rise to different formalizations, usually depending on the biological function of the query. When the query is for example a signaling path, then the querying problem is modeled as finding a high-scoring path in the host network whose proteins fulfill additional similarity constraints [Shlomi et al. 2006]. In most querying scenarios it is imperative to make assumptions about the interactions among the vertices of the host that are similar to the query. For the querying of signaling paths this is obvious, since a similar pathway is expected to have a similar succession of acting proteins. For protein complexes, van Dam and Snel [2008] showed that the interaction patterns are evolutionary conserved. Hence, the occurrence should have an interaction pattern that is similar to the interaction pattern of the query. Consequently, most mathematical formalizations of the task of network querying are thus related to the NP-hard SUBGRAPH ISOMORPHISM problem [Garey and Johnson 1979].

In this work, we consider several formulations of the querying problem. Unless stated otherwise we use $G = (V, E)$ to denote the query and $H = (W, F)$ to denote the host. All of these formulations have the following aspects in common.

Injectivity. We only consider formulations where each protein of the query is mapped to exactly one protein of the host. Furthermore, two proteins of the query should be mapped to different proteins of the host. Hence, we seek a function f that injectively maps from the protein set V of the query to the protein set W of the host. In other words, we look for a bijective mapping from V to a subset S of W. Note that in one case, the study of LIST-COLORED GRAPH MOTIF in Chapter 6, we also allow deletion of proteins from V, that is, we look for a maximum-cardinality subset V' of V such that we can find an injective mapping from V' to W that fulfills the further constraints given by the problem formulation.

Orthology Constraints. A common approach to ensure that the occurrence is indeed biologically similar to the query is to add *orthology constraints*. This means that for each protein in the query we only allow that it is mapped to proteins that are orthologs of this protein, that is, proteins that are evolutionary closely related to this protein. Formally, the input contains for every protein $p \in V$ a set of proteins $\mathcal{L}(p)$

such that for the mapping m it must hold that $m(p) \in \mathcal{L}(p)$. We refer to $\mathcal{L}$ as *ortholog list* of p and to the elements of $\mathcal{L}(p)$ as *orthologs* of p. If a graph G is associated with such a family of ortholog lists, then we say that $\mathcal{L}$ is a *list-coloring* of V and that G is a *list-colored* graph. In the formulations under consideration in this work, the list-coloring is given either for the query graph or the host graph.

Prohibition of "Gap Proteins". In order to keep the problems under consideration conceptually simple, we do not consider the addition of further proteins that are not explicitly matched to any protein of the query. Such proteins might be needed, for example, to connect different subgraphs of the occurrence.

The differences between the formulations of the different querying problems that we consider lie in what we call the "mapping criterion". Herein, the mapping criterion is simply a function that, given a mapping m from V to a subset S of W, decides whether this mapping is *valid*. For example, one can demand that the mapping is a graph isomorphism from G to $H[S]$. Another example is to ask for a mapping m that maps to a connected subgraph.[4] Hence, in this case the mapping m is valid if $H[S]$ is connected. In the following, we give a mathematical definition of mapping criteria c. Let $\mathcal{M}(G = (V,E), H = (W,F)) := \{f \mid f : V \to S, S \subseteq W\}$ be the set of all mappings from the vertex set of G to the vertex set of H. Then, the mapping criterion $c_{G,H}$ for two graphs G and H is a boolean function $c_{G,H} : \mathcal{M}(G,H) \to \{0,1\}$. We say that a mapping $m : V \to S \subseteq W$ is *valid under* $c_{G,H}$ if $c_{G,H}(m) = 1$; otherwise, we say that m is *not valid under* $c_{G,H}$.

Altogether, the aspects above lead to the following generic definition of occurrence.

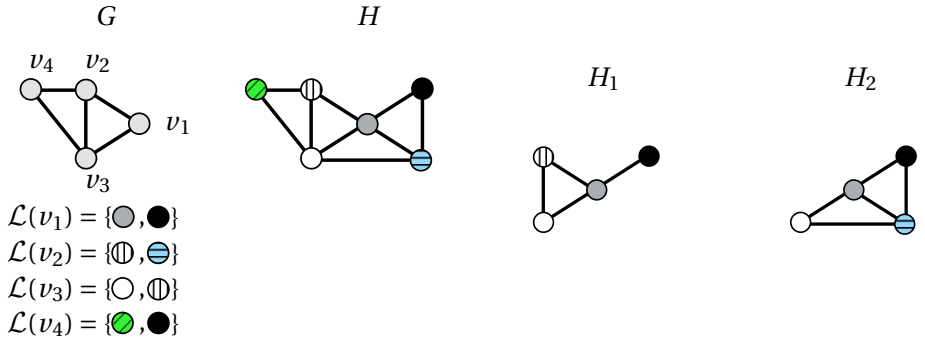
Definition 4.4. *Let $G = (V,E)$ and $H = (W,F)$ be two undirected graphs, let $\mathcal{L} : V \to 2^W$ be a list-coloring of V, and let $c_{G,H}$ be a mapping criterion for G and H. We then call an induced subgraph $H[S]$ of H an* occurrence of G in H *when there is a mapping $m : V \to S$ such that*

- $\forall v \in V : m(v) \in \mathcal{L}(v)$, *and*
- m *is valid under* $c_{G,H}$.

A generic formalization of the network querying problem that we consider then reads as follows.

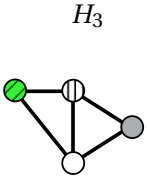
NETWORK QUERYING:
Input: A query graph $G = (V,E)$, a host $H = (W,F)$, a list-coloring $\mathcal{L} : V \to 2^W$ of V and a mapping criterion c.
Task: Find an occurrence $H[S]$ of G.

[4] This is precisely the criterion of the LIST-COLORED GRAPH MOTIF problem studied in Chapter 6.



(a) The list-colored input query and the host.

(b) Two induced subgraphs of H that are *not* occurrences of G: H_1 is not isomorphic to G, and in H_2 the vertices v_3 and v_4 must be mapped to vertices that are not adjacent (because of the list-coloring).



(c) The induced subgraph H_3 of H is an occurrence of G: mapping each vertex of G to the first vertex/color of its ortholog list is an isomorphism from G to H_3.

Figure 4.4: An example of NETWORK QUERYING in which one demands that the mapping from query to host is an isomorphism.

The interaction pattern of the query is often referred to as *topology* of the query. So far, many of the available implementations for variants of NETWORK QUERYING are limited to sparse topologies such as paths or trees [Sharan and Ideker 2006]. We believe that there are two reasons for these limitations. First, in many biologically relevant application scenarios, the queries are intrinsically sparse. For example, this is the case for queries that are signaling pathways or collections of such pathways [Shlomi et al. 2006, Dost et al. 2008]. A second reason for this limitation lies in the fact that sparse topologies appear to be algorithmically easier to handle than arbitrary topologies. In fact, almost all of the presented querying algorithms for these sparse topologies are color-coding based fixed-parameter algorithms for the parameter "query size". However, for arbitrary topologies, parameterization by the query size is often not applicable because SUBGRAPH ISOMORPHISM parameterized by solution size is W[1]-hard. Hence, for arbitrary query topologies other parameters have to be considered.

In this work, we present fixed-parameter tractability and intractability results on NETWORK QUERYING for topology-free querying, dense topologies, and arbitrary

topologies. We also perform computational experiments concerning the performance of some of the presented algorithms with respect to running time as well as biological quality of the reported occurrences.

Related Work. We quickly review the work that has been presented in the area of network querying for formulations of the querying problems that differ from the ones considered in this work. To our knowledge, the notion of "network querying" was coined by Kelley et al. [2003] who developed a network alignment algorithm PATHBLAST and used PATHBLAST for aligning complete protein interaction networks with each other as well as single pathways (the queries) with complete networks. The PATHBLAST algorithm finds high-scoring paths in a so-called alignment graph, which is a graph in which each vertex corresponds to a pair of proteins, one from the query and one from the host. Several algorithms have been proposed for the special cases when the query is a path [Shlomi et al. 2006, Hüffner et al. 2007, 2008] or tree-like [Blin et al. 2010b, Dost et al. 2008]. In work that is also related to network querying, Sharan et al. [2005] presented an algorithm that, given two interaction networks, finds subnetworks of both networks that are dense, and whose proteins have high sequence-similarity. These subnetworks are supposed to form conserved protein complexes. Since they look simultaneously for similar subnetworks in both networks and not only one, this approach can be seen as a generalization of the querying problem.

Overview over Part III. In Chapter 5, we present a case study for the querying problem when the mapping criterion can be informally described as "the occurrence has to be dense". We give a biological motivation for this criterion and consider two different density demands. We show that in case one demands that the occurrence is a clique, then one can obtain very efficient fixed-parameter algorithms for protein interaction networks by using parameters related to the so-called degeneracy of the host network and the query size. Afterwards, we show that if one demands that the occurrence is a so-called μ-clique, which is a relaxation of the clique definition, then the problem becomes much harder, since it is W[1]-hard with respect to the so-called dual parameter.

In Chapter 6, we study the List-Colored Graph Motif problem, a variant of the querying problem that demands that the occurrence is a connected subgraph of the host. More precisely, the List-Colored Graph Motif problem asks for a maximum-cardinality subset of the query that has an occurrence in the host. This querying problem has applications in the identification of protein complexes in protein interaction networks [Bruckner et al. 2010] and in the identification of reaction motifs in metabolic networks [Lacroix et al. 2006]. We consider three different parameters for this problem: query size k, the solution size $k' \leq k$, and the parameter $n - k$, where n is the number of vertices in the host. We present fixed-parameter algorithms

for parameters k and k', and W[1]-hardness for the parameter $n-k$. Moreover, we also perform computational experiments on protein interaction networks concerning the speed of our algorithms as well as the biological relevance of the computed occurrences. Additionally, we show that demanding stronger types of connectivity such as biconnectivity or two-edge connectivity leads to W[1]-hardness with respect to the parameter "query size".

In Chapter 7 we study the querying problem for arbitrary query topologies when we demand that the mapping from query to occurrence is an injective homomorphism. We present an improved algorithm for this querying problem parameterized by the combined parameter "maximum number of orthologs $\max_{v \in V} |\mathcal{L}(v)|$ and number of query proteins with at least three orthologs". We also show that we can apply our algorithm when one demands that the query is isomorphic to the occurrence.

Furthermore, we show that, for some query-host combinations, homomorphism-based querying yields better results in terms of biological quality than isomorphism-based querying.

Chapter 5

Parameterization by Degeneracy and Dual—A Case Study for Dense Queries

In this chapter, we study network querying problems in which we demand that the occurrence of the query is a dense graph. The two types of dense graphs that we consider in this context are cliques and μ-cliques. The aim of this chapter is mainly to give an easy-to-follow example of how the identification of small parameters can lead to very efficient algorithms for problems that are very hard in general. Nevertheless, we also believe that the problems under consideration might be of interest in biologically motivated querying scenarios. In the following, we discuss why demanding that the occurrence of a query has to be dense is reasonable in some application scenarios.

As discussed for example by Sharan et al. [2005], many protein complexes form dense subgraphs of protein interaction networks. In fact, for certain types of protein complexes, cliques can be seen as a perfect model of protein complexes [Sharan et al. 2005]. Hence, for queries that are protein complexes, it is likely that the interaction pattern represents a dense graph. Since there is evidence that interactions in evolutionary related complexes are conserved [van Dam and Snel 2008] the occurrence of such a dense query should also be dense. Moreover, even in case the topology of the protein complex is completely unknown, a dense subgraph of the host that also fulfills the orthology-constraints given by the list-coloring of the query is a good candidate for a complex. Querying problems for dense query topologies are therefore a relevant research topic from a theoretical as well as from an applied point of view.

5.1 The Querying Problem for Cliques

As argued above, it is not uncommon to have queries that are cliques. In case the host contains a perfect match of the query, that is, in case all interactions are conserved by evolution, the occurrence should be a clique as well. We therefore define the problem of finding a perfect match for a query that is a clique as follows:

> LIST-COLORED CLIQUE QUERYING:
> **Input:** A clique $G = (V, E)$, a graph $H = (W, F)$, and a list-coloring $\mathcal{L} : V \to 2^W$ of V.
> **Task:** Find, if it exists, a vertex set $S \subseteq W$ such that $H[S]$ is a clique, and there is a one-to-one mapping f from V to S such that $\forall v \in V : f(v) \in \mathcal{L}(v)$.

Obviously, a special case of LIST-COLORED CLIQUE QUERYING is CLIQUE, the problem of finding a clique of size k. Therefore, the negative computational results for CLIQUE hold for LIST-COLORED CLIQUE QUERYING as well. This means, for example, that LIST-COLORED CLIQUE QUERYING is W[1]-hard with respect to $|V|$, or that it is NP-hard to approximate the problem of finding a sub-occurrence of G in H that has maximum order within a factor of $|V|^{1-\varepsilon}$ for any $\varepsilon > 0$ [Zuckerman 2007].

These negative results indicate that solving LIST-COLORED CLIQUE QUERYING is very hard and might lead to the conclusion that solving LIST-COLORED CLIQUE QUERYING is not feasible for protein interaction networks, considering that they contain several thousand vertices. In the following, we demonstrate that such a conclusion is premature. Moreover, we show that an efficient algorithm for LIST-COLORED CLIQUE QUERYING can be derived by simple combinatorial observations and algorithms.

First of all, consider the case that the host H is a clique. Then, an efficient polynomial-time algorithm for checking whether H contains an occurrence of query G works as follows. Since H is a clique, $H[S]$ is a clique for every $S \subseteq W$ as well. Hence, all that needs to be done is to check whether there is an injective mapping from V to W that respects the list-coloring of V. This can be achieved as follows. Create an auxiliary bipartite graph M where the two parts are V and W, respectively. Insert an edge between a vertex $v \in V$ and a vertex $w \in W$ if and only if $w \in \mathcal{L}(v)$. Then, an injective mapping m from the vertices in V to the vertices in W such that for each $v \in V$ it holds that $m(v) \in \mathcal{L}(v)$ is a matching in M. Hence, after constructing M in $O(|V| \cdot |W|)$ time, such a mapping can be found in $O(|V| \cdot |W| \cdot \sqrt{|V| + |W|})$ time by using the matching algorithm of Hopcroft and Karp [1971].

Proposition 5.1. *Let $(G, H, \mathcal{L})$ be an instance of* LIST-COLORED CLIQUE QUERYING *such that H is a clique. Then,* LIST-COLORED CLIQUE QUERYING *can be solved in $O(|V| \cdot |W| \cdot \sqrt{|V| + |W|})$ time.*

Consequently, LIST-COLORED CLIQUE QUERYING is NP-hard only in case H is not a clique, and the actual combinatorial "challenge" for solving LIST-COLORED CLIQUE QUERYING lies in finding a clique in H that contains an occurrence of G. In the following, we gradually develop a strategy for finding this clique by using increasingly "refined" parameterizations.

A trivial observation is that any clique in H is completely contained in the neighborhood of some vertex $w \in W$. Hence, one can find an occurrence of G by the following simple algorithm: For each vertex $w \in W$, enumerate all maximal cliques K in $N[w]$. For each such clique K, check whether $H[K]$ contains an occurrence of G. By using a modified version of the Bron-Kerbosch algorithm for enumerating maximal cliques [Tomita et al. 2006] and Proposition 5.1 we can achieve the following.

Proposition 5.2. LIST-COLORED CLIQUE QUERYING *can be solved in* $O(|W| \cdot 3^{\Delta(H)/3} \cdot |V| \cdot |W| \cdot \sqrt{|V|+|W|})$ *time, where* $\Delta(H)$ *is the maximum degree of* H.

The running time of Proposition 5.2 is not really useful for protein interaction networks, since these networks usually contain so-called "hubs" which are proteins that participate in many interactions. For example, the yeast protein interaction networks that was used in the experiments of Chapter 6 and Chapter 7 has maximum degree 971. For this instance, the running time of the algorithm is clearly infeasible if it is anywhere close to its worst-case estimate. Hence, the question is whether we can avoid the parameter maximum degree $\Delta(H)$ in our algorithm. Clearly, it would be desirable to obtain fixed-parameter tractability with respect to the parameter "average degree of H". This, however, is not possible since one could simply add a sufficiently large (but polynomial in $|W|$) set of degree-1 vertices to H and obtain a graph with constant average degree. A fixed-parameter algorithm for the parameter "average degree" would thus imply P=NP. Nevertheless, it is possible to improve on Proposition 5.2 by using the parameter "degeneracy d of H" instead of the maximum degree.

Definition 5.1. *The* degeneracy *of a graph G is the smallest integer d such that every induced subgraph of G has at least one vertex with degree at most d.*

Degeneracy is a measure for the sparseness of a graph and was originally introduced as coloring number of a graph [Erdös and Hajnal 1966]. Trees, for example, have degeneracy one since every induced subgraph of a tree has at least one leaf. Notably, the degeneracy of protein interaction networks is much smaller than its maximum degree. For example, the above-mentioned yeast network that has maximum degree 971 has degeneracy 23. In the following, we roughly describe a simple enumeration algorithm for this parameter.

First, find a vertex v of minimum degree in H and enumerate all maximal cliques in $N[v]$. Since H has degeneracy d, it follows that $N[v] \leq d+1$. Hence, the enumeration takes only $f(d)$ time. Then, for each enumerated maximal clique $K \subseteq$

$N[v]$, check whether $H[K]$ contains an occurrence of G using Proposition 5.1. If none of the enumerated maximal cliques contains an occurrence of G, then one does not need to consider v anymore. Therefore, remove v from H and consider again a vertex of minimum degree (which is again at most d by the definition of degeneracy). Continue until either an occurrence of G has been found, or H is the empty graph. The running time of this algorithm can be bounded as follows. The enumeration of the maximal cliques in H can be performed in $O(d \cdot |W| \cdot 3^{d/3})$ time by using an algorithm of Eppstein et al. [2010]. This algorithm roughly follows the approach presented above, namely, looking for a vertex of minimum degree and then enumerating all maximal cliques that contain this vertex, and uses some further speed-up techniques. For the time that is needed for checking for all the enumerated cliques whether they contain an occurrence of G observe the following.

First, we can build in $O(|V| \cdot |W|)$ time a matrix L with a row for each $v \in V$ and a column for each $w \in W$ such that $L(v,w) = 1$ if $w \in \mathcal{L}(v)$, and $L(v,w) = 0$, otherwise. We can then use this matrix to create for each enumerated maximal clique K and each vertex $v \in V$ a new set of ortholog lists $\mathcal{L}_K(v)$ that contains only vertices of K. This can be done in $O(d^2)$ time since $|V| \leq d$ and $|K| \leq d$, and by using the matrix L to check whether $w \in \mathcal{L}(v)$ for each pair of $v \in V$ and $w \in K$. After this, an occurrence of G in $H[K]$ can be found in $O(d^2 \cdot \sqrt{d})$ by using Proposition 5.1. The overall running time can then be bounded as follows.

Proposition 5.3. List-Colored Clique Querying *can be solved in* $O(|V| \cdot |W| + |W| \cdot 3^{d/3} \cdot d^3 \cdot \sqrt{d})$ *time, where* d *is the degeneracy of* H.

This running time is a significant improvement over the running time of Proposition 5.2, and for most real-world instances, it will lead to a very fast algorithm for solving List-Colored Clique Querying. Still, even this result can be improved in case $|V|$ is almost as large as d. The simple observation that we can exploit is that when enumerating maximal cliques in $N[v]$ for some vertex $v \in W$ that has degree at most d, we only need to consider cliques that have size at least $|V|$. These cliques have the property that all of their vertices have at most $d - (|V| - 1)$ neighbors in $N[v] \setminus K$. Using an algorithm that is based on the enumeration of minimal vertex covers in the complement graph, we can enumerate all maximal cliques $K \subseteq N[v]$ with this property in $O(2^{d-|V|} \cdot d^3)$ time [Komusiewicz et al. 2009]. Again, we use the algorithm behind Proposition 5.1 to check whether $H[K]$ contains an occurrence of G for each enumerated maximal clique K. The overall running time thus amounts to the following.

Proposition 5.4. List-Colored Clique Querying *can be solved in* $O(|V| \cdot |W| + |W| \cdot 2^{d-(|V|)} \cdot d^5 \cdot \sqrt{d})$ *time, where* d *is the degeneracy of* H.

Note that the degeneracy d, and therefore also $d - |V|$, can be determined in linear time [Batagelj and Zaversnik 2003]. Depending on the values of d and $d - |V|$ one could

then choose between the algorithms of Propositions 5.3 and 5.4.

Summarizing, we have given evidence that LIST-COLORED CLIQUE QUERYING can be solved very efficiently on protein interaction networks. In the next section, we explore the computational complexity of the querying problem when we search for a different type of dense graphs: μ-cliques.

5.2 On Finding μ-Cliques

The clique requirement is often too strict for modeling real-world data such as protein interaction networks since it requires that the input data is error-free. However, current protein interaction networks contain many errors. In particular, there are many *false negative* interactions, that is, interactions that take place in biology but are missing from the data [Yu et al. 2006]. For example, some interactions only act over a very short period of time and therefore cannot be detected by current experimental methods. Still, in case the query is known to exhibit a dense interaction pattern in the query graph, one aims to find an occurrence of the query that also has many interactions. One of the most natural ways to model density is to demand that a certain ratio of edges is present in a graph. This is precisely the notion of μ-cliques which we already considered in Chapter 3.

Definition 5.2. *The* density *of a graph* $G = (V, E)$ *is defined as* $2|E|/(|V|(|V|-1))$. *A graph* $G = (V, E)$ *is called* μ-clique *for a rational constant* $0 \leq \mu \leq 1$ *if the density of* G *is at least* μ.

Hence, it is interesting to study the querying problem when we look for μ-cliques instead of cliques. Unfortunately, we will demonstrate that the problem of querying for such μ-cliques is much harder than querying cliques. More precisely, we demonstrate that already deciding whether a host graph H contains any μ-clique of a given size is much harder than the querying problem for cliques, and that many of the approaches presented for maximal cliques fail. The problem that we consider is formalized as follows:

> μ-CLIQUE:
> **Input:** A graph $G = (V, E)$, and a nonnegative integer k.
> **Question:** Is there a vertex set $S \subseteq V$ of size at least k such that $G[S]$ is a μ-clique?

The strongest parameterization presented for LIST-COLORED CLIQUE QUERYING was $d-k$, where d is the degeneracy of H and k is the number of vertices in the query. For μ-CLIQUE we show that parameterization by $|V|-k$ leads to W[1]-hardness. Note that $|V|-k$ is always larger than $d-k$ and can be arbitrarily large compared to $d-k$; it is therefore a *weaker* parameter than $d-k$.

Theorem 5.1. *For any fixed μ, $0 < \mu < 1$, μ-CLIQUE is W[1]-hard with respect to the parameter $n - k$, where n is the number of vertices in the input graph.*

Proof. We reduce from the W[1]-hard CLIQUE problem [Downey and Fellows 1995b, 1999]. Let $(G = (V, E), k')$ be an instance of CLIQUE, that is, G is an undirected graph, and we ask whether G contains a clique of order k'. Assume without loss of generality that $\mu = a/b$, and that $a > 2|V|$. In the following, we describe how to construct a graph G^* such that deleting k vertices from G^* yields a μ-clique if and only if G has a clique of size k.

The idea of the construction can be roughly described as follows. We add to G a large and dense graph $H = (W, F)$ that has minimum degree much larger than $|V|$. Then, we add edges between H and G such that in the resulting graph G^*, all vertices from V have degree $|V|$. Then we show that G^* is not a μ-clique and that, because of the way H is constructed, only by deleting exactly k' vertices from V that induce a clique in G we can obtain a subgraph of G^* that has density μ and at least $|W| + |V| - k'$ vertices. The crucial observation that helps proving that we must delete a clique is that

1. by deleting a clique, we remove $|V| + (|V| - 1) + \ldots + (|V| - k' + 1) = |V| \cdot k' - (k' - 1) \cdot k'/2$ edges from G^*, and
2. by deleting a set of vertices that is not a clique, we remove more edges from G^*.

Next, we describe the details of the construction. We construct $H = (W, F)$ such that it fulfills the following condition: A graph that contains H as induced subgraph and

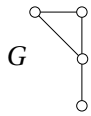
- $|V| - k'$ additional vertices and
- $|E| + \left(\sum_{v \in V}(|V| - \deg_G(v))\right) - (|V| \cdot k' - (k' - 1) \cdot k'/2)$ additional edges

has density exactly $\mu = a/b$. By Lemma 3.5 we can construct H in $\text{poly}(a, b, |V|)$ time such that it fulfills this condition and such that

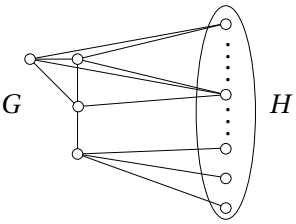
- H is $2(a-1)$-connected, and
- a graph that contains H and the above described additional number of vertices and edges has average degree more than a.

Let $G^* := (V \cup W, E \cup F)$ be the disjoint union of the graphs G and H. In G^*, we then add for each vertex $v \in V$ exactly $|V| - \deg_G(v)$ edges between v and H (the neighbors of v in H can be arbitrarily chosen). After adding these edges, every vertex $v \in V$ has degree exactly $|V|$ in G^*. A schematic illustration is presented in Figure 5.1.

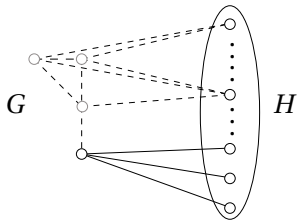
We complete the construction of the μ-CLIQUE instance by setting $k := n - k'$ (recall that n denotes the number of vertices in the μ-CLIQUE instance, that is, $n = |V| + |W|$). This described construction can be clearly performed in polynomial time. Note that the parameter of the CLIQUE instance is k' and the parameter of the μ-CLIQUE instance is $n - k = k'$, that is, the reduction is parameter-preserving. To prove the theorem, it thus remains to show the following claim:



(a) The input graph G of the CLIQUE Instance.



(b) The connections to the gadget graph H that is added to G. For each vertex v of G we add $4 - \deg(v)$ edges between v and H.



(c) Deleting in G^* a clique of size three from G deletes $3 \cdot 4 - 3 = 9$ edges from G^*. If a set of three deleted vertices is not a clique in G, more edges are deleted.

Figure 5.1: An illustration of the reduction presented in the proof of Theorem 5.1.

(G, k') is a yes-instance of CLIQUE $\Leftrightarrow$ (G^*, k) is a yes-instance of μ-CLIQUE.

$\Rightarrow$: Let $S := \{v_1, \ldots, v_{k'}\}$ be a size-k' clique in G. We show that $G^* - S$ is a μ-clique with k vertices. Clearly, the overall number of vertices in $G^* - S$ is $n - k' := k$. The overall number of edges in $G^* - S$ is $|F| + |E| + \left(\sum_{v \in V}(|V| - \deg_G(v))\right) - (|V| \cdot k' - (k'-1) \cdot k'/2)$ which can be seen as follows. First, note that, by construction, G^* has $|F| + |E| + \left(\sum_{v \in V}(|V| - \deg_G(v))\right)$ edges. Hence, it remains to show that the number of edges that have at least one endpoint in S is $|V| \cdot k' - (k'-1) \cdot k'/2$. Since S is a size-k' clique and since each vertex $v \in S$ has degree exactly $|V|$ in G^*, each vertex $v \in S$ has exactly $|V| - (k'-1)$ neighbors in $(W \cup V) \setminus S$. Furthermore, there are exactly $\binom{k}{2} = (k'-1) \cdot k'/2$ edges in $G[S]$. Hence, the overall number of edges with at least one endpoint in S is

$$k' \cdot (|V| - (k'-1)) + (k'-1) \cdot k'/2) = |V| \cdot k' - (k'-1) \cdot k'/2.$$

The claimed overall number of edges in $G^* - S$ follows. Consequently, $G^* - S$ is a μ-clique.

$\Leftarrow$: Let $S \subseteq W \cup V$ be a vertex set such that $G^* - S$ is a μ-clique of size at least k. Clearly, $|S| \leq k$. We show that G contains a size-k clique by showing the following claims. First, we show that it can be assumed that $S \subseteq V$. Second, we show that $|S| = k'$. Finally, we show that $G[S]$ is a clique.

First, we show that it can be assumed that $S \subseteq V$. Suppose that S contains some vertex $w \in W$. Note that w has in G^* at least $4|V| - k' > |V|$ neighbors in W since H (which is a subgraph of G^*) is $4|V|$-connected (because $2(a-1) \geq 4|V|$). Let v be some arbitrary vertex in $V \setminus S$. Clearly, v has degree at most $|V|$ in $G^* - S$. Therefore, deleting the set $S' := \{w\} \cup S \setminus \{v\}$ from G^* yields a graph that has more edges than $G^* - S$ and, obviously, the same number of vertices. Since $G^* - S$ is a μ-clique, so is $G^* - S'$. This "replacement procedure" can be applied as long as S contains a vertex from W. Hence, we can assume without loss of generality that $S \subseteq V$.

Second, we show that $|S| = k'$. More precisely, we show that for all vertex sets $S := \{v_1, \ldots, v_i\}$ of size $i < k$ it holds that $G^* - S$ has density less than μ. Let $S' := \{v_1, \ldots, v_i, \ldots v_k\}$ be an arbitrary size-k superset of S. Note that $G^* - S'$ has at most

$$|F| + |E| + \left(\sum_{v \in V} (|V| - \deg_G(v)) \right) - (|V| \cdot k' - (k'-1) \cdot k'/2)$$

edges which can be seen as follows. The number of edges in G^* is $m := |F| + |E| + \left(\sum_{v \in V}(|V| - \deg_G(v))\right)$. The graph $G_1^* := G^* - v_1$ has at most $m - |V|$ edges, since the minimum degree in G^* is $|V|$. Consequently, the graph $G_2^* := G_1^* - v_2$ has at most $m - |V| - (|V| - 1)$ edges, the graph $G_3^* := G_2^* - v_3$ has at most $m - |V| - (|V| - 1) - (|V| - 2)$ edges, and so on. Hence, the number of edges in $G^* - S'$ is at most

$$m - |V| - (|V|-1) - \ldots - (|V| - k' + 1) = |F| + |E| + \left(\sum_{v \in V} (|V| - \deg_G(v)) \right) - (|V| \cdot k' - (k'-1) \cdot k'/2).$$

This means, by construction of H, that $G^* - S'$ has density *at most* μ. For illustrative purposes, suppose that $G^* - S$ is obtained from $G^* - S'$ by adding, one by one, the vertices from $S' \setminus S$. Furthermore, note that $G^* - S'$ has, again by construction of H, average degree more than $a > 2|V|$. Finally note that the vertices that are added to $G^* - S'$ have degree at most $|V|$ in the final graph $G^* - S$. This means that $G^* - S$ must have density *less than* μ since $G^* - S'$ has average degree at least $2|V|$ and adding a vertex to a graph whose degree is (after it has been added to the graph) less than the average degree of the graph before the vertex has been added reduces the density (note that by adding at most $k' < |V|$ vertices to $G^* - S'$ we always produce a graph with average degree more than $|V|$). Summarizing, the graph $G^* - S$ has density less than μ if $|S| < k'$.

Finally, we show that $G[S]$ is a clique. Note that, as argued above, $G^* - S$ has at most $F| + |E| + \left(\sum_{v \in V}(|V| - \deg_G(v))\right) - (|V| \cdot k' - (k'-1) \cdot k'/2)$ edges. Hence, by construction of H and by the fact that $G^* - S$ is a μ-clique, it also holds that $G^* - S$ has *exactly* this many edges. Let m_S denote the number of edges in $G[S]$. The number of edges that are removed from G^* by the deletion of S is $\sum_{v \in S} |V| - m_S$, since every vertex has degree exactly $|V|$ in G^*. It follows that $m_S = (k'-1) \cdot k'/2)$ and, therefore, that $G[S]$ is a clique. □

Somehow counterintuitively, the reduction used in Theorem 5.1 suggests that in order to obtain a graph with density μ it might be of advantage to delete a clique from the input graph. Hence, one cannot expect that a set of vertices that is removed from the input graph in order to obtain a μ-clique induces a sparse graph.

5.3 Concluding Remarks

The two problems studied in this chapter are very different from each other concerning the status of their tractability in protein interaction networks. For LIST-COLORED

CLIQUE QUERYING, it is clear that we can develop an efficient implementation whereas for μ-CLIQUE, which does not even contain orthology-constraints, there are so far only negative results. Therefore, the next steps that could be taken for each problem are quite different:

- For LIST-COLORED CLIQUE QUERYING an implementation could be developed. As argued above, the restriction that the occurrence must be a clique is probably too strict. Still, the results for some querying scenarios could possibly be improved if we first check whether there is indeed such a perfect match, which as shown here is not too costly in terms of running time.
- Further extensions of LIST-COLORED CLIQUE QUERYING should be considered such as finding an occurrence of maximum-cardinality in case there is no occurrence of order $|V|$. These extensions should increase the number of instances in which a match is indeed found.
- Other clique relaxations, for example s-plexes, that is, graphs in which each vertex has at most $s-1$ "missing" edges, could be considered in order to cope with noise in the data. For constant s the problem of finding such s-plexes should be fixed-parameter tractable with respect to the degeneracy of the host graph. For unbounded s, we conjecture that the problem is W[1]-hard for the parameter (d,s). In general, the hope is to find a clique relaxation that represents the "middle ground" between the computational hardness of finding μ-cliques and the inflexibility of using cliques.
- For μ-CLIQUE, a thorough computational complexity analysis of μ-CLIQUE considering further parameters such as minimum degree or degeneracy should be undertaken. Because μ-CLIQUE appears to be much harder than CLIQUE, it is probably necessary to consider the combination of several parameters in order to obtain fixed-parameter tractability results.

Chapter 6

Parameterizing Topology-free Querying by Query and Solution Size

The task of finding so-called graph motifs is a variant of the querying problem where the mapping criterion demands that the occurrence is connected. The rationale behind this model is that a set of proteins that forms a connected subgraph is more likely to form a protein complex than a disconnected set of proteins. Since the connectedness of the occurrence only depends on the topology of the *host* network and not on the topology of the *query*, we can view the query simply as a set of colors, where each color represents a protein or, in different application scenarios, some other functional unit, such as a reaction in a metabolic network. In fact, we mainly follow a formal definition given by Lacroix et al. [2006]:

> LIST-COLORED GRAPH MOTIF:
> **Input:** A multiset of colors M (the query or motif), an undirected graph $H = (W, F)$ (the host), and a list-coloring $\mathcal{L} : W \to 2^M$ of W.
> **Task:** Find maximum-cardinality sets $S \subseteq W$ and $M' \subseteq M$ such that the induced subgraph $H[S]$ is connected and there is a one-to-one mapping f from S to M' such that $\forall v \in S : f(v) \in \mathcal{L}(v)$.

An *occurrence* of a motif M in H is a set of vertices $S \subseteq W$ such that $|S| = |M|$, $H[S]$ is connected, and there are x vertices of color c in S if and only if M contains c exactly x times. For each vertex of the host, we are given a set of query colors that it can be "matched" to. Note that this is equivalent to specifying for each query color a vertex set $S \subseteq W$ that it can be matched to. Also note that in this formulation, the query can be a multiset instead of a set, this is motivated by the application in metabolic networks that we describe below. An example input instance is shown in Figure 6.1.

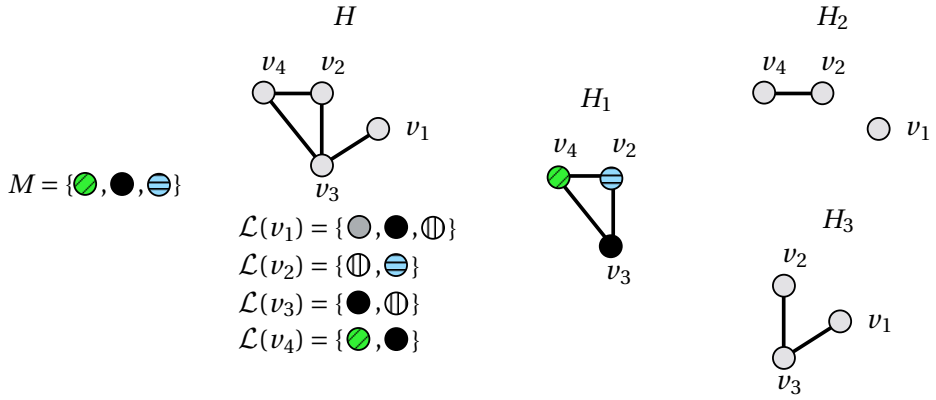


Figure 6.1: An input instance of LIST-COLORED GRAPH MOTIF with a colorful motif M, host graph H, and a list-coloring $\mathcal{L}$. The subgraph H_1 is an occurrence of M since it is connected and the shown coloring is a bijective mapping from M to the vertices of H_1 that uses for each vertex v_i only colors from $\mathcal{L}(v_i)$. In contrast, H_2 is not an occurrence since it is not connected and H_3 is not an occurrence, since the "first" color of the motif is not contained in the color lists of v_1, v_2, and v_3.

Fellows et al. [2011a] studied a restricted variant of LIST-COLORED GRAPH MOTIF, called GRAPH MOTIF, which is the special case of LIST-COLORED GRAPH MOTIF where each vertex is associated with a single color. These types of graphs will be called *vertex-colored* (instead of list-colored) in this chapter. For vertex-colored graphs we denote by $\mathcal{L}(v)$ the (uniquely determined) color of v. An even further restricted variant of GRAPH MOTIF demands that the motif is a set instead of a multiset. We call motifs that are sets *colorful*.

Next, we briefly describe two applications for LIST-COLORED GRAPH MOTIF in the area of biological network analysis.

Reaction Motifs in Metabolic Networks

The LIST-COLORED GRAPH MOTIF problem was originally defined in the context of finding motifs in metabolic networks [Lacroix et al. 2006]. Here, the motif is a multiset of reaction types and the given network is a reaction graph: vertices correspond to reactions, and two vertices are connected if the corresponding reactions can occur successively, that is, the products of one reaction are the inputs of the other reaction or vice versa. The task is to find a subgraph of the reaction graph that is connected and has exactly the reaction types specified by the motif. Since a specific reaction may be classified as an instance of more than one reaction type, the vertices of the reaction graph may be colored by more than one color resulting in a list-colored graph.

Topology-free Querying of Protein interaction Networks

Bruckner et al. [2010] proposed the following approach for querying protein interaction networks in case the topology of the query is unknown. Since the topology of the query is unknown, it is simply a set of proteins that is known to form a protein complex in some species A and the task is to identify similar complexes in a different species B. To this end, every protein of the query is identified by one color and a protein in the interaction network receives the colors of all query proteins with similar sequence (that is, the BLAST[1] score exceeds a predefined threshold). The main point is that, since the topology of the query is unknown, no assumptions about the topology of the subnetwork that corresponds to this protein complex are made, with one exception: the subnetwork should be connected or almost connected, that is, it should be possible to make it connected with few insertions of nonorthologs. Furthermore, of all occurrences of the query meeting these criteria, one with a maximum-weight spanning tree should be reported: The edge weights in the network correspond to interaction probabilities. Accordingly, a spanning tree with large weight corresponds to a high interaction probability among the proteins of the putative complex. This leads to a weighted version of LIST-COLORED GRAPH MOTIF with the further restriction that the motif is a set because each protein of the query is identified with a unique color.

Known Results. We first summarize the current state of the art concerning the algorithmic complexity of GRAPH MOTIF, that is, the special case where H is vertex-colored. Formulated as a decision problem, GRAPH MOTIF is NP-complete even if the input graph is a tree [Lacroix et al. 2006]. This NP-completeness result was later strengthened by showing that the problem is NP-hard even when the input multiset M actually is colorful (that is, a set) and the input graph is a tree with maximum vertex degree three [Fellows et al. 2011a]. Moreover, NP-completeness has also been shown for the case that M consists of only two colors and the input graph is restricted to be bipartite with maximum degree four [Fellows et al. 2011a]. Given the apparent hardness of GRAPH MOTIF, Fellows et al. [2011a] initiated a parameterized complexity analysis showing that GRAPH MOTIF, parameterized by the motif size $|M|$ is fixed-parameter tractable. The algorithm with the asymptotically fastest running time for GRAPH MOTIF is due to Guillemot and Sikora [2010], who use algebraic techniques to solve both the problem of finding an occurrence of a motif, and the problem of counting the number of occurrences. Dondi et al. [2011] extended the investigations for GRAPH MOTIF by studying the case where the subgraph induced by the chosen motif vertices does not need to be connected. Furthermore, they showed that GRAPH MOTIF is hard in terms of polynomial-time approximability and presented a first fixed-parameter algorithm for the parameter "solution size" $|S|$ [Dondi et al. 2011]. The

[1] http://blast.ncbi.nlm.nih.gov/

Table 6.1: An overview of the complexity results for the different variants of the LIST-COLORED GRAPH MOTIF problem.

coloring of graph and motif	running time bounds/complexity
vertex-colored graph and colorful motif	$2^{\|S\|} \cdot \text{poly}(n)$ [Guillemot and Sikora 2010] $2^{n-\|M\|} \cdot \text{poly}(n)$ (Theorem 6.3)
vertex-colored graph and multiset motif	$4^{\|S\|} \cdot \text{poly}(n)$ [Guillemot and Sikora 2010] W[1]-hard for $n-\|M\|$ (Theorem 6.4)
list-colored graph and multiset motif	$10.88^{\|M\|} \cdot \text{poly}(n)$ (Theorem 6.1) $29.6^{\|S\|} \cdot \text{poly}(n)$ (Theorem 6.2) W[1]-hard for $n-\|M\|$ (Theorem 6.4)

algorithm with the best asymptotic running time for GRAPH MOTIF and parameter $|S|$ runs in $O(4^{|S|} \cdot |S|^2 \cdot m)$ time on m-edge graphs [Guillemot and Sikora 2010].

Concerning LIST-COLORED GRAPH MOTIF, there are fewer results, although the hardness results for GRAPH MOTIF clearly hold as well. Bruckner et al. [2010] presented a fixed-parameter algorithm for LIST-COLORED GRAPH MOTIF with a worst-case running time of $O(|M|! \cdot 3^{|M|} \cdot m)$. They also showed that on many real-world instances the actual running time is much smaller than this worst-case estimate, and successfully applied their algorithm to protein interaction networks. Furthermore, a web-server for solving LIST-COLORED GRAPH MOTIF (relying on fixed-parameter algorithms and an integer linear programming formulation) is available [Bruckner et al. 2009]. Blin et al. [2010a] presented a further algorithm for LIST-COLORED GRAPH MOTIF, based on a 0/1-integer programming formulation.

Our Results. We present fixed-parameter algorithms as well as parameterized hardness results for several variants of LIST-COLORED GRAPH MOTIF. In Section 6.1, we consider different parameterizations for LIST-COLORED GRAPH MOTIF.

First, considering the parameter $|M|$, we present an algorithm that solves LIST-COLORED GRAPH MOTIF in $O(10.88^{|M|} \cdot m)$ time on host graphs with m-edges. Second, we consider the parameter solution size $|S|$. Adapting an approach of Dondi et al. [2011] for GRAPH MOTIF, we present a randomized algorithm for LIST-COLORED GRAPH MOTIF that runs in $O(29.6^{|S|} \cdot |S| \cdot m)$ time. For both parameterizations, we thus improve on the currently best worst-case running time bounds.

Third, we consider the parameter $n-|M|$. Note that this parameter is always at most as large as the dual parameter $n-|S|$, that is, the number of vertices that are *not* in a solution. We show that in general LIST-COLORED GRAPH MOTIF becomes W[1]-hard for the parameter $n-|M|$, and that for the special case where M is colorful and H is vertex-colored, LIST-COLORED GRAPH MOTIF can be solved in $O(2^{n-|M|} \cdot m)$ time. An overview of our results and further state-of-the art results for special cases of LIST-COLORED GRAPH MOTIF is given in Table 6.1.

We implemented our two fixed-parameter algorithms for LIST-COLORED GRAPH MOTIF parameterizing in the one case by $|M|$ and in the other case by $|S|$. We developed and implemented a number of further heuristic speed-ups for both algorithms, and we report here on their performance for real-world instances.[2] As one example of such a heuristic speed-up, we identify cases in which applying a different color-coding procedure helps in decreasing the running time. Furthermore, we identify cases in which the application of a simple (randomized) brute-force algorithm yields a speed-up.

We applied our algorithms in the context of querying of protein interaction networks, showing that both the algorithm with parameter $|M|$ and the algorithm with parameter $|S|$ can solve almost all of the considered input instances within 10 minutes, and that in most cases the algorithm with parameter $|M|$ outperforms the algorithm with parameter $|S|$. Furthermore, we examined the quality of the solutions by testing them for enrichment of functional annotation terms: for each solution, which is a set of proteins, we retrieved functional annotation terms from public databases [Barrell et al. 2009, SGD project, Tweedie et al. 2009] and applied the GO::TermFinder tool [Boyle et al. 2004] to find annotation terms that have a statistically significant overrepresentation compared to random protein sets. We found that about 79% have a significant enrichment in at least one functional annotation and that 100% of the solutions of size at least four show this enrichment. Furthermore, the percentage of solutions that have this functional enrichment is roughly the same for solutions with $|S|/|M| \geq 40\%$, and is lower when $|S|/|M| < 40\%$.

Finally, we further chart the range of (theoretical) tractability of LIST-COLORED GRAPH MOTIF by exploring what happens if we demand "more" than simple connectivity. We show that if one requires that the found motif shall not only be connected but biconnected or bridge-connected (see Section 1.3 for formal definitions), then, in both cases, the corresponding LIST-COLORED GRAPH MOTIF problem becomes W[1]-complete with respect to the parameter motif size. Since these are the two simplest ways of demanding more than connectivity, this shows that the request for connected motifs is already a topology demand close to the border between tractability and intractability. These W[1]-hardness results also generalize to higher connectivity demands such as p-connectivity and p-edge connectivity (see Section 1.3 for formal definitions). Even further, W[1]-hardness also holds for uncolored graphs, where one searches for a p-(edge)-connected subgraph and the parameter is the number of subgraph vertices.

[2] Source code available at http://fpt.akt.tu-berlin.de/graph-motif/

6.1 Searching for Connected Motifs

In this section, we provide and analyze randomized fixed-parameter algorithms for LIST-COLORED GRAPH MOTIF and the parameters motif size $|M|$ and solution size $|S|$. Note that, since $|S| \leq |M|$, fixed-parameter tractability with respect to $|S|$ implies fixed-parameter tractability with respect to $|M|$. On the contrary, the inverse is not true since $|S|$ can be *much* smaller than $|M|$. We provide fixed-parameter algorithms for both single parameters because for the parameter $|M|$ a better worst-case running time bound can be achieved. Furthermore, we study the parameterized complexity of LIST-COLORED GRAPH MOTIF for the dual parameter $n - |M|$.

6.1.1 Parameter Motif Size

We present a color-coding algorithm that partially resembles the algorithm for GRAPH MOTIF by Fellows et al. [2011a]. This algorithm employs the *color-coding* technique which was introduced for special cases of the SUBGRAPH ISOMORPHISM problem by Alon et al. [Alon et al. 1995]. The main idea is to randomly color the vertices of the graph, and then to solve the corresponding problem under the assumption that the subgraph that is searched for obtains a *colorful* coloring, that is, all of the vertices of the subgraph have pairwise different colors. This assumption often leads to an easier problem. The whole procedure of coloring and then solving the subsequent problem on the colored graph is repeated until the subgraph that is searched for has obtained with high probability a colorful coloring in at least one of the repetitions. We say that a randomized algorithm solves a problem with *error probability* ε if the probability that it fails to return the correct answer is at most ε. Note that the randomized algorithms in this work do not make false positive errors, that is, if the algorithm returns that a graph has an occurrence of a motif, then such an occurrence does indeed exist.

Our color-coding algorithm follows the main scheme presented above. First, randomly assign to each vertex one out of $|M|$ *labels*. Then, in case all vertices of an occurrence have received pairwise different labels (we call this event a *good labeling*), we can use dynamic programming to find this occurrence.

Theorem 6.1. LIST-COLORED GRAPH MOTIF *can be solved with error probability* ε *in* $O(|\ln(\varepsilon)| \cdot 10.88^{|M|} \cdot m)$ *time.*

Proof. Without loss of generality, we can assume that M is colorful. Otherwise we can transform M and H as follows: For each color c that occurs $\mathrm{occ}(c)$ times, we add $\mathrm{occ}(c)$ new colors to M and completely remove c from H. Furthermore, for every vertex v in H with $c \in \mathcal{L}(v)$, we remove c from $\mathcal{L}(v)$ and add the $\mathrm{occ}(c)$ new colors to $\mathcal{L}(v)$. Let M' and H' be the thus modified motif and graph, respectively. We now solve the problem of finding an occurrence of M' in H'. Each such occurrence clearly corresponds to an occurrence of M in H.

Let $L = \{l_1, l_2, \ldots, l_{|M|}\}$ denote a set of $|M|$ distinct labels. We randomly assign the labels of L to the vertices of the graph and solve the problem of finding an occurrence of the motif M under the assumption that all vertices of the occurrence have received a different label. The problem of finding a colorful occurrence of M that has the labels of L is solved by dynamic programming. First, we extend our notion of occurrence. Let $J \subseteq (L \cup M)$ be a set that contains labels as well as colors. An *occurrence of* J is defined as a set of vertices S such that the vertices of S have exactly the labels of $J \cap L$, and there is an injection $f : S \to J \cap M$ such that $f(v) \in \mathcal{L}(v)$ for each vertex v. The idea is that with the set J we store both the set of labels $L \cap J$ that we have "used" so far, and the colors that the vertices with the labels in $L \cap J$ are assigned. This way, we can find for each combination of sets $M' \subseteq M$ and $L' \subseteq L$, $|L'| = |M'|$, an occurrence of M' that has the labels of L'.

We use two dynamic programming tables D and T. The table D has entries for each vertex, and sets $J \subseteq L \cup M$. The aim of the dynamic programming procedure is to compute the values of D such that $D_v(J) = 0$ if there exists an occurrence of J that contains v, and $D_v(J) > 0$, otherwise. The table T has entries for each vertex v, each color $c \in \mathcal{L}(v)$ and sets $J \subseteq L \cup M$. The aim is to compute the values of T such that $T_{v,c}(J) = 0$ if there is an occurrence of $J \uplus \{\text{label}(v), c\}$ in which v receives the color c, and $T_{v,c}(J) > 0$, otherwise. Clearly, for each v, we have to create $T_{v,c}(J)$, only for c with $c \in \mathcal{L}(v)$ and for J such that $c \notin J$ and $v \notin J$.

We initialize the table T with $T_{v,c}(\emptyset) = 0$. In the recursion, we fill in the values for increasing sizes of J for both T and D. First, we use table T to calculate values of D:

$$D_v(J) = \min_{c \in \mathcal{L}(v)} \{T_{v,c}(J \setminus \{\text{label}(v), c\}), 1\}.$$

The correctness of this recurrence follows from the definition of the table entries. Then, we calculate the value for $T_{v,c}(J)$ branching into two cases. The first case is that there is a neighbor u of v such that there is an occurrence of J that contains u. The second case is that J can be partitioned into two sets J' and $J \setminus J'$ such that there is an occurrence of $J' \cup \{\text{label}(v), c\}$ and of $(J \setminus J') \cup \{\text{label}(v), c\}$ such that in both occurrences v is contained and has color c. The "score" for these occurrences can be found in the table T. The recurrence reads as follows:

$$T_{v,c}(J) = \min_{\substack{u \in N(v), \\ J' \subset J}} \left\{ \begin{array}{l} D_u(J), \\ T_{v,c}(J') + T_{v,c}(J \setminus J') \end{array} \right\}.$$

If there is a $v \in W$ such that $D_v(L \cup M) = 0$, then there is an occurrence of $L \cup M$ in H. A maximum-cardinality occurrence can be found by finding a vertex v and a maximum-cardinality set J such that $D_v(J) = 0$. The vertex set that corresponds to the occurrence can be computed by doing a simple traceback.

For the running time consider the following. Clearly, $|L \cup M| = 2|M|$. The recurrence for table D and the first part of the recurrence for table T can be computed

in $O(2^{2|M|} \cdot |M| \cdot m)$ time overall. For the second part of the recurrence of table T, Björklund et al. [Björklund et al. 2007] showed that recurrences of this type can be solved in $2^x \cdot \text{poly}(x)$ time, when the base set over which the table is defined has size x. Here, this base set is $L \cup M$ and it has size $2 \cdot |M|$. This results in a running time of $2^{2 \cdot |M|} \cdot \text{poly}(2|M|) = 4^{|M|} \cdot \text{poly}(|M|)$ for the dynamic programming procedure for each table $T_{v,c}$, and thus in a running time of $4^{|M|} \cdot \text{poly}(|M|) \cdot n$ for all such tables. For the random labeling, each vertex of W is labeled with one of $|M|$ labels under uniform distribution. Then, the probability of a good labeling is $|M|!/|M|^{|M|} > e^{-|M|}$. Therefore, the number of trials needed to obtain a good labeling with probability $1-\varepsilon$ is $O(|\ln(\varepsilon)| \cdot e^{|M|})$. The total running time thus amounts to $O(|\ln(\varepsilon)| \cdot e^{|M|} \cdot 4^{|M|} \cdot \text{poly}(|M|) \cdot m) = O(|\ln(\varepsilon)| \cdot 10.88^{|M|} \cdot m)$. □

6.1.2 Parameter Solution Size

In this section, we present an algorithm for the parameter solution size $|S|$, applying an idea of Dondi et al. [2011] to our algorithm from Section 6.1.1. The approach can be roughly described as follows. Let $S \subset W$ be a solution of LIST-COLORED GRAPH MOTIF, and let $M' \subseteq M$ be a submotif such that there is an injection f from S to M' with $f(v) \in \mathcal{L}(v)$ for each $v \in S$. Clearly, to distinguish the vertices of S using color-coding, one only needs to use $|S|$ many labels. However, we do not know in advance what colors the color set M' comprises. Hence, the idea is to use a further color-coding step, this time mapping the colors of M to the colors of a newly created set M_k of size $k = |S|$ in order to obtain an equivalent instance that has a motif of size $|S|$. The problem of finding an occurrence of M_k is fixed-parameter tractable with respect to the parameter $|S|$ since we can apply our algorithm from Section 6.1.1. This extends the fixed-parameter tractability result of Dondi et al. [2011] for GRAPH MOTIF with respect to the parameter $|S|$ to LIST-COLORED GRAPH MOTIF. In the following, we present the details of the algorithm and bound its running time.

Theorem 6.2. LIST-COLORED GRAPH MOTIF *can be solved with error probability* ε *in* $O(|\ln(\varepsilon)| \cdot 29.6^{|S|} \cdot |S| \cdot m)$ *time.*

Proof. The algorithm proceeds as follows. Starting with $k = 1$, it finds an occurrence of size k if such an occurrence exists. The value k will be incremented by one as long as a solution has been found. If for some value of k the algorithm fails to find a solution, then with high probability no size-k solution exists and the algorithm reports the solution of size $k-1$ that was found previously. We now describe in detail how the algorithm works for a fixed value of k. First, create a new set of k colors $M_k := \{c_1, \ldots, c_k\}$. Then, construct a mapping $\phi : M \to M_k$ by mapping each color $c \in M$ uniformly at random to a color in M_k. Create a new graph H' from H as follows. For each vertex v and each $c \in \mathcal{L}(v)$, remove c from $\mathcal{L}(v)$ and add $\phi(c)$ to $\mathcal{L}(v)$. We now use the algorithm from Section 6.1.1 to find an occurrence of M_k in G'. If

no occurrence was found, then repeat the procedure above without changing k until either an occurrence was found, or we can, with sufficiently low error probability, conclude that H contains no occurrence of a size-k subset of M.

First, we show that if there is a size-k set $M' \subseteq M$ such that there is an occurrence S of M' in H, then with probability at least e^{-k} we create an instance that has an occurrence of M_k in H': Since each color of M' is mapped uniformly at random to one of the colors of M_k, the probability that all colors of M' are mapped to pairwise different colors of M_k is at least $k!/k^k > e^{-k}$. In this case, S is an occurrence of M_k in H'.

Second, we show that if the algorithm finds an occurrence S of M_k in H', then there is also a size-k set $M' \subseteq M$ such that S is an occurrence of M' in H. Let S be an occurrence of M_k in H', and let f be an injection from S to M_k (which must exist since S is an occurrence of M_k). Since f is an injection, $f(v) \neq f(u)$ for each pair of vertices $u, v \in S$, $u \neq v$. Hence, by choosing for each vertex v of S an arbitrary color c such that $\phi(c) = f(v)$, we obtain a set M' of k pairwise different colors such that S is an occurrence of M' in H.

We now bound the running time of the algorithm for some fixed k. Suppose there is a size-k set $M' \subseteq M$ such that there is an occurrence S of M' in H. The probability, that each color of M' was mapped to a different color in M_k is at least e^{-k}. The problem of finding an occurrence of M_k in H' can be solved with constant error probability in $O(10.88^k \cdot m)$ time by using the algorithm from Section 6.1.1. Hence, the probability that this algorithm finds an occurrence of M_k in H' (and consequently an occurrence of some size-k subset $M' \subseteq M$ in H) is at least $O(e^{-k})$. Repeating the procedure of coloring M and then applying the algorithm from Section 6.1.1 $O(e^k)$ times, a size-k occurrence of some $M' \subseteq M$, if such an occurrence exists, is found with constant probability. Hence, for fixed ε we can solve the problem of finding with constant error probability an occurrence of some size-k subset of M in $O(|\ln(\varepsilon)| \cdot e^k \cdot 10.88^k \cdot m) = O(|\ln(\varepsilon)| \cdot 29.6^k \cdot m)$ time. If no such occurrence has been found, we can conclude that with probability at least $1 - \varepsilon$ no such occurrence exists.

In the overall algorithm loop, we abort as soon as $k = |S| + 1$, since by definition of the solutions size $|S|$, no occurrence of a size-$(|S| + 1)$ subset of M exists. The overall running time bound follows. □

6.1.3 Dual Parameterization

We study the parameterized complexity of List-Colored Graph Motif for the so-called *dual* parameter $n - |M|$, where $n := |W|$. At first, this parameterization appears to be uninteresting since the motif is very small compared to the network. However, in some applications, there are many vertices of the input graph that can be removed by a simple data reduction since their color-lists do not contain any colors of the motif. After this data reduction, the remaining graph has modest size, and furthermore often

contains several connected components (this observation was also made by Bruckner et al. [2010]). For many of these components, the motif is relatively large compared to the order of the connected component. Then, only few vertices may be "removed" from this component to obtain the motif. Unfortunately, in general the LIST-COLORED GRAPH MOTIF problem becomes W[1]-hard as we show later in this section. However, for the simple case, when the graph is vertex-colored instead of list-colored and the motif is colorful, we obtain fixed-parameter tractability. The simple idea of the corresponding search tree algorithm is to branch on vertices in H that have the same color. Each occurrence of the motif contains at most one of these two vertices since the motif is colorful. In the following, we describe this algorithm in detail.

Theorem 6.3. *For colorful motifs and vertex-colored graphs* LIST-COLORED GRAPH MOTIF *can be solved in* $O(2^{n-|M|} \cdot m)$ *time.*

Proof. Given a vertex-colored n-vertex graph $H = (W, F)$ and a colorful motif M, the algorithm proceeds as follows. Initially, set $d := n - |M|$. Clearly, we can assume that every color of M appears in H, otherwise we can simply remove this color from M. Furthermore, let M' be a maximum-cardinality subset of M such that there is an occurrence of M' in H. For $d \leq 1$, the graph must contain two vertices u and v such that $\mathcal{L}(v) = \mathcal{L}(u)$. At most one of these vertices belongs to an occurrence of M'. Accordingly, the algorithm recursively finds occurrences of M' in $H[W \setminus \{u\}]$ and in $H[W \setminus \{v\}]$. In each recursive branch, set $d := d - 1$. In case $d = 0$, the graph H contains exactly the colors of M. Then, the largest occurrence of a subset of M is simply the largest connected component of H. There is at least one search tree leaf in which H has a connected component whose colors are M'. Hence, the overall solution is the largest occurrence that was found over all recursive calls in the search tree.

As to the running time, the search tree has size $O(2^d)$ since it has depth d and branches into two cases at each search tree node. Furthermore, the operations at each search tree node can be performed in $O(m)$ time. □

Unfortunately, Theorem 6.3 does not carry over to the general LIST-COLORED GRAPH MOTIF problem. On the contrary, we show that the problem becomes W[1]-hard for vertex-colored graphs and motifs that consist of two colors, and also for colorful motifs when the input graph is list-colored.

Theorem 6.4. LIST-COLORED GRAPH MOTIF *is W[1]-hard with respect to the parameter* $n-|M|$ *even if the input graph* H *is vertex-colored and the motif* M *consists of two colors.*

Proof. We reduce from the W[1]-complete INDEPENDENT SET [Downey and Fellows 1999] problem:

> **Input:** An undirected graph G and a nonnegative integer k.
> **Question:** Is there a size-k vertex set S such that $G[S]$ has no edges?

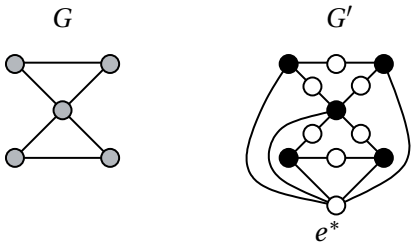


Figure 6.2: An example of the reduction from INDEPENDENT SET to GRAPH MOTIF with two colors. Black vertices in G' have color b, white vertices have color w. For $k = 3$, the motif contains two times the color b and seven times the color w. Hence, one must delete three black vertices from G' to obtain an occurrence. Since in G there is no independent set of size three, one must delete the two neighbors of at least one white vertex in G'.

Given an instance $(G = (V,E),k)$ of INDEPENDENT SET, we build an instance of LIST-COLORED GRAPH MOTIF as follows. The motif M is a multiset over two colors b and w such that M contains the color b exactly $|V| - k$ times and the color w exactly $|E| + 1$ times. Each vertex $v \in V$ is colored with color b. Furthermore, each edge $\{u,v\} \in E$ is replaced by a path of length two, that is, we remove $\{u,v\}$ from G, add a new vertex $e_{\{u,v\}}$ to G and insert edges between u and $e_{\{u,v\}}$ and v and $e_{\{u,v\}}$. Each new vertex receives the color w. Finally, we add one further vertex e^* with color w and add edges between e^* and every vertex that has color b. Let G' denote the resulting vertex-colored graph. The construction can be clearly performed in polynomial time. Note that $n - |M| = k$. An example of this reduction is shown in Figure 6.2. We complete the proof by showing that

$$G \text{ has a size-}k \text{ independent set} \Leftrightarrow G' \text{ has an occurrence of } M.$$

$\Rightarrow$: Let S be a size-k independent set in G. Consider the graph G'' that is obtained from G' by removing S. We show that G'' is an occurrence of M. Consider the colors of G''. Since all k vertices that have been removed from G' have color b there are $|V| - k$ vertices that have color b and $|E| + 1$ vertices that have color w in G''. It remains to show that G'' is connected. Since e^* is adjacent to all vertices of V, the subgraph that is induced by $V \cup \{e^*\}$ is connected. Furthermore, every other vertex is adjacent to at least one vertex of V. Suppose that this is not the case, then for some vertex $e_{\{u,v\}}$ both u and v are in S. This contradicts the fact that S is an independent set in G.

$\Leftarrow$: Let S be a vertex set such that removing S from G' results in an occurrence of M, that is, a graph that is connected and has the colors of M. By construction of G' and M, S has size k and contains only vertices that have color b. Hence, these vertices correspond to vertices of G. We show that S is an independent set in G. Suppose that this is not the case, then there must be two vertices $u,v \in S$ such that $\{u,v\} \in E$. Then, however, both neighbors of $e_{\{u,v\}}$ in G' are in S. This contradicts the fact that removing S from G' results in a connected graph. □

By slightly modifying the construction in the proof of Theorem 6.4, we can also transfer this result to the case that the motif is colorful. The only difference is that instead of using two colors b and w, we use two sets of colors $B = \{b_1, \dots b_{n-k}\}$ and $W = \{w_1, \dots w_{m+1}\}$. Every color that was colored by b now receives the color list B and every color that was colored by w receives the color list W. The motif is $B \cup M$. The correctness proof works analogously.

Theorem 6.5. LIST-COLORED GRAPH MOTIF *is W[1]-hard with respect to the parameter* $n - |M|$ *even if* M *is colorful.*

Theorems 6.4 and 6.5 exclude any hope for fixed-parameter algorithms for LIST-COLORED GRAPH MOTIF parameterized by the dual parameter $n - |M|$. However, as we show in Section 6.2, there are some cases in which even a brute-force algorithm for guessing the set of vertices to delete is faster than the color-coding fixed-parameter algorithms for parameters $|M|$ and $|S|$, respectively. Hence, it seems worthwhile to find further special cases in which LIST-COLORED GRAPH MOTIF is fixed-parameter tractable with respect to $n - |M|$ or to study $n - |M|$ in combination with other parameters.

6.2 Application to Querying of Protein Interaction Networks

We applied our algorithms for LIST-COLORED GRAPH MOTIF to topology-free querying of protein interaction networks. As described in the introduction, the input of the graph motif instance here is a colorful motif and the vertices in H are list-colored. Our program is written in the C++ programming language, uses the Boost Graph Library[3], and consists of ≈ 1000 lines of code. The source code is publicly available.[4]

6.2.1 Implementation Details

This section describes some details that distinguish the implemented algorithms from those specified in Section 6.1. Given a graph $H = (W, F)$ and a colorful motif M, our implemented algorithms not only find a maximum-cardinality subset $S \subseteq W$ such that $H[S]$ is an occurrence of M, but also compute S such that the weight of a spanning tree of $H[S]$ is maximized. Our implementation provides three algorithms: the first is a simple (randomized) brute-force approach. This algorithm is not used on its own, but as a subroutine of the other two algorithms. The second algorithm solves LIST-COLORED GRAPH MOTIF with the parameter motif size $|M|$ as described in Section 6.1.1. The third algorithm solves LIST-COLORED GRAPH MOTIF with the parameter solution size $|S|$ as described in Section 6.1.2. The algorithm

[3] http://www.boost.org/
[4] http://fpt.akt.tu-berlin.de/graph-motif

with parameter $|M|$ and the algorithm with parameter $|S|$ resort to the brute-force approach if it is expected to outperform the dynamic programming routines described in Section 6.1.1. This is, for example, the case if the dual parameter (Section 6.1.3) is small.

6.2.1.1 Randomized Brute-Force

Given a LIST-COLORED GRAPH MOTIF instance comprising a graph $H = (W, F)$, a colorful motif M, and a natural number k, this algorithm proceeds as follows: it chooses a size-k set $S \subseteq W$ uniformly at random and checks whether $H[S]$ is an occurrence of a subset of M. This verification step is carried out by computing a maximum-cardinality matching in the bipartite graph $B = (S \uplus M, F)$, where an edge $\{v, c\} \in F$ with $v \in S$ and $c \in M$ exists if and only if $c \in \mathcal{L}(v)$. If all vertices in S are matched and $H[S]$ is connected, then $H[S]$ is an occurrence of a size-k subset of M. The process of randomly choosing subsets of H is repeated a sufficient number of times so that an existing occurrence of a size-k subset of M is found with high probability. More precisely, the probability that a size-k occurrence is chosen is at least $1/\binom{|W|}{k}$. Hence, we repeat the procedure of guessing and verifying a solution $O(|\ln(\varepsilon)| \cdot \binom{|W|}{k})$ times to obtain an error probability of at most ε. For each motif occurrence found in the process, its maximum-weight spanning tree is computed. An occurrence with the maximum-weight spanning tree among all size-k occurrences is reported. Since we have to repeat the procedure $O(|\ln(\varepsilon)| \cdot \binom{|W|}{k})$ times, the algorithm works fast if k is small or close to $|W|$. In the latter case, the dual parameter (Section 6.1.3) is small. However, Randomized Brute-Force is not a fixed-parameter algorithm with respect to the parameter k or the parameter $|W| - k$.

6.2.1.2 Parameter Motif Size

In contrast to the algorithm described in Section 6.1.1, the implemented algorithm does not employ the (theoretical) result by Björklund et al. [2007] to quickly evaluate the given recurrences. Instead, they are straightforwardly evaluated by dynamic programming. However, we implemented heuristics to reduce the running time of the algorithm.

Overall Strategy

Each connected component of the input graph H is processed independently. For each connected component, the Randomized Brute-Force procedure described in Section 6.2.1.1 is applied if it is expected to outperform the color-coding algorithm from Section 6.1.1. Otherwise the color-coding algorithm is invoked. Next, we describe the implementation details of the color-coding algorithm.

Increasing the Success Probability of Color-Coding

The running time depends to a large extent on the number of trials needed to achieve a good labeling with sufficiently high probability. We implemented two approaches to increase the probability of a motif occurrence to receive a good labeling. This, in turn, reduces the number of trials needed to achieve a sufficiently low error probability. The two approaches are described in the following two paragraphs.

Separating Color Sets. This approach to increase the probability of a motif occurrence to receive a good labeling is due to Bruckner et al. [2010]. Assume that there is a *separating* color subset C such that the color list of each vertex either contains *no* colors from C or *exclusively* colors from C. In this case, we can improve on standard color-coding. Let W_1 be the vertices that contain only colors from C and let W_2 be the vertices that contain no colors from C. If the given motif contains k_1 colors from C and k_2 colors not from C, then the occurrence of the motif must contain k_1 vertices from W_1 and k_2 vertices from W_2. Thus, we may draw the labels for W_1 and W_2 from disjoint label sets L_1 and L_2, respectively. The probability for a good labeling is the probability that the vertices in W_1 and W_2 receive a good labeling. This results in a good labeling of vertices with a probability of $(k_1!/k_1^{k_1}) \cdot (k_2!/k_2^{k_2})$ instead of $(k_1+k_2)!/(k_1+k_2)^{k_1+k_2}$. Separating color subsets, if they exist, can be computed in $O(k \cdot m)$ time by finding connected components in an auxiliary graph [Bruckner et al. 2010].

Injective Color-Coding. Assume that C is a separating color set, as described above, such that the sought colorful motif M contains k colors from C. Moreover, let W_C be the vertices that contain only colors of C. Observe that the probability that W_C receives a good labeling is $k!/k^k$ while the probability of guessing a vertex subset of W_C that is part of a motif-occurrence is $1/\binom{|W_C|}{k}$. As a result, if the latter probability is higher than the former, then we avoid the standard color-coding technique. Instead, we choose a random subset of W_C and label its vertices injectively.

Our experiments (see Section 6.2.2) showed that the combination of these two heuristics reduces the number of color-coding trials tremendously.

6.2.1.3 Parameter Solution Size

As described in Section 6.1.2, the algorithm works by iterating over all possible solution sizes from $k := 1$ to $|S|+1$. Color-coding is applied repeatedly to obtain a colorful motif $M' \subseteq M$ with $|M'| = k$. In each repetition, the algorithm for the parameter motif size (Section 6.2.1.2) is applied to M' and H (with new color lists, as described in Section 6.1.2). We use the following two approaches to heuristically improve the running time of this algorithm:

Injective Color-Coding. Similar to Section 6.2.1.2, Injective Color-Coding may be applied to increase the probability that a size-k subset of M with an occurrence in H receives a good coloring. This applies if $k!/k^k$ is less than $1/\binom{|M|}{k}$. In this case, we reduce the number of trials needed to obtain good colorings for the subsets of M by injectively coloring a randomly chosen size-k subset of M.

Lower Bounds for Solution Size. The algorithm for the parameter solution size checks whether size-k subsets of M have occurrences in H, iterating over increasing values of k and starting with $k = 1$. A lower bound on the solution size allows us to skip many of these iterations by not increasing k by only one, but setting it to the currently best known lower bound. Such a lower bound on the solution size can be obtained using Random Occurrence Guessing (see Section 6.2.1.1). We also use Randomized Brute-Force to find size-$(|M| - k + 1)$ occurrences in H if this approach is expected to outperform the color-coding algorithm for finding size-k occurrences.

As for the algorithm for parameter $|M|$, these two heuristics reduce the number of color-coding trials needed before a solution is found (see Section 6.2). However, the overall number of trials needed is usually higher than for the algorithm with parameter $|M|$. The reason is that we have to perform color-coding twice: once for the motif colors and once for the vertex labels.

6.2.2 Experiments

Data Acquisition

We tested our algorithms on three weighted protein interaction networks (the weights denote interaction probabilities) from yeast (5430 proteins, 39936 interactions), fly (6650 proteins, 21275 interactions), and human (7915 proteins, 28972 interactions) that were assembled by Bruckner et al. [2010]; the queries were complexes from the same three species. We considered the following four combinations of network and complexes: we queried yeast complexes in the human network and in the fly network, human complexes in the yeast network, and fly complexes complexes in the yeast network.[5] For each complex of the query species we created an instance of LIST-COLORED GRAPH MOTIF as follows. Each query protein was identified with a unique color, the proteins of the networks received the colors of all query proteins whose sequence similarity to the network protein exceeded a predefined threshold. This threshold was set to a BLAST score of 10^{-7}. Table 6.2 shows the total number of instances, grouped into categories of small ($|M| \leq 7$), medium ($8 \leq |M| \leq 14$) , and large motif size ($|M| \geq 15$).

[5] This somewhat arbitrary choice of query–host combinations was based on the convenient availability of the sequence similarity data.

Table 6.2: Comparison of the algorithms with parameters $|M|$ and $|S|$. Herein, unsolved instances needed either more than 10 minutes of time or more than 900 MiB of space. All values shown are for the set of solved instances only. All instances were processed with a maximum error probability of 0.1%.

(a) Results for the algorithm with parameter $|M|$ (Section 6.2.1.2).

	# instances		avg. size		time (secs)		avg. # coloring trials	
$\|M\|$	solved	unsolved	$\|M\|$	$\|S\|$	avg.	max.	standard	improved
1–7	1089	0	2.20	1.27	0.00	0.07	36.43	1.07
8–14	47	2	10.55	4.09	2.62	59.08	$9.7 \cdot 10^4$	22.23
≥ 15	18	3	21.28	5.33	0.43	7.59	$4.1 \cdot 10^{12}$	8.28

(b) Results for the algorithm with parameter $|S|$ (Section 6.2.1.3)

	# instances		avg. size		time (secs)		avg. # coloring trials	
$\|M\|$	solved	unsolved	$\|M\|$	$\|S\|$	avg.	max.	standard	improved
1–7	1089	0	2.20	1.27	0.00	0.08	416.35	0.82
8–14	43	6	10.55	3.63	1.50	55.05	$2.8 \cdot 10^8$	65.44
≥ 15	14	7	21.28	3.43	0.02	0.07	$1.1 \cdot 10^{13}$	41.72

Computational Setting

The experiments have been executed on a standard desktop PC with 2.2 GHz Athlon64 CPU and 1 GiB of RAM. For each network-motif pair, we aborted execution after 10 minutes or if the memory limit of 900 MiB was exceeded.

For each pair of complex (equivalently, motif) and network, we computed the largest subset of the motif that has an occurrence in the network. Of these, we output the occurrence with the spanning tree of maximum weight. This implies that we cannot stop the color-coding process after a motif occurrence has been found. Instead, we have to execute all color-coding trials in order to find the maximum-weight motif occurrence with the given error probability.

Experimental results

Table 6.2 summarizes the results. The average running time of the solved instances is lower for the algorithm with parameter $|S|$ (Section 6.2.1.3) than for the algorithm with parameter motif size $|M|$ (Section 6.2.1.2). However, the algorithm with parameter $|M|$ is able to solve more instances within the running time limit of 10 minutes. As one would expect, these are the instances where the motif occurrences were large. This is also the reason why the average size of the occurrences found by the algorithm with parameter $|S|$ is smaller. It is notable that, while we set an upper limit of 10

Table 6.3: Results for the algorithms with parameters $|M|$ and $|S$ when Randomized Brute Force was not used as a subroutine. Herein, unsolved instances needed either more than 10 minutes of time or more than 900 MiB of space. All values shown are for the set of solved instances only. All instances were processed with a maximum error probability of 0.1%.

(a) Results for the algorithm with parameter $|M|$ without Randomized Brute-Force

	# instances		avg. size		time (secs)	
$\|M\|$	solved	unsolved	$\|M\|$	$\|S\|$	avg.	max.
1–7	1089	0	2.20	1.27	0.00	0.44
8–14	42	7	10.45	3.43	8.12	113.82
≥ 15	14	7	21.71	3.43	0.10	0.72

(b) Results for the algorithm with parameter $|S|$ without Randomized Brute Force

	# instances		avg. size		time (secs)	
$\|M\|$	solved	unsolved	$\|M\|$	$\|S\|$	avg.	max.
1–7	1089	0	2.20	1.27	0.00	0.60
8–14	41	8	10.39	3.27	6.71	159.17
≥ 15	14	7	21.71	3.43	0.18	1.75

minutes to solve an instance, all instances solved within this time limit were solved in a few seconds. Most instances could be solved within milliseconds. We conclude that usually the algorithm with parameter $|M|$ should be applied instead of the algorithm with parameter $|S|$, since it was able to solve more instances and is outperformed by the algorithm for parameter $|S|$ only for "easier" instances.

The effect of the heuristics for reducing the number of color-coding trials (see Sections 6.2.1.2 and 6.2.1.3) is also shown in Table 6.2. For both algorithms, there is a difference of several orders of magnitude between the number of color-coding trials that are needed with and without heuristics, and this difference is especially pronounced for the more difficult instances with large motifs. Furthermore, we examined how the inclusion of the Randomized Brute-Force algorithm helps in reducing the overall running time for both algorithms. The running times of the algorithms without the Randomized Brute-Force subprocedure are shown in Table 6.3 a) and b), respectively. For both algorithms there is a decrease in the number of solved instances and an increase in the average running times, demonstrating the usefulness of Randomized Brute-Force for some instances. Furthermore, since the average occurrence size of the solved instances is smaller when Randomized Brute-Force is not used, we conclude that Randomized Brute-Force is useful when $|S|$ is

relatively large.

6.2.3 Related implementations

Bruckner et al. [2009, 2010] developed a web-service tool (TORQUE) to solve an extension of our problem allowing for insertions of vertices that are not part of the motif. They combine several algorithms: a color-coding procedure for small motifs, an Integer Linear Programming formulation for large motifs, and a shortest-path based heuristic. The main difference of our implementation compared to TORQUE is that TORQUE allows for insertion of additional/uncolored vertices in order to establish the connectivity of the solution set. That is, TORQUE reports solutions that contain uncolored vertices that connect different connected components of colored vertices if this results in a larger overall occurrence size. Furthermore, the number of deletions, that is, the number of motif colors that are *not* in an occurrence, is limited for TORQUE (for example, for motifs/queries of size 10, the occurrence has to contain at least six vertices that are colored). In summary, the solutions of TORQUE are usually larger than our solutions and TORQUE may not remove all of the uncolored vertices by data reduction, since some of them might be needed to connect colored vertices. Hence, we solve a different problem, and our running times do not compare directly to the ones of Bruckner et al. [2010]. Blin et al. [2010a] provide a Cytoscape plug-in (based on a Linear Pseudo-Boolean optimization solver) for LIST-COLORED GRAPH MOTIF with insertions. Again, this makes a comparison of running times difficult, since our algorithm solves a more restricted problem.

Our color-coding algorithm can handle larger motifs than the color-coding algorithm of Bruckner et al. [2010] who use Integer Linear Programming to handle queries/motifs of size > 10. Furthermore, our algorithm almost always terminates within few seconds which is not the case for the algorithm of Bruckner et al. [2010]. This encourages both the application of our algorithm in case insertions are explicitly forbidden and makes desirable the extension of our algorithm to the more general problem that allows for insertions. One advantage of our approach is that the overall number of solutions is much larger than for TORQUE since we always report the largest occurrence that was found, that is, we allow an arbitrary number of deletions. Hence, it can be applied in case TORQUE does not yield a solution.

6.2.4 Functional Enrichment of Predicted Complexes

We examine the quality of the solutions that were found by our algorithm by assessing the enrichment of functional terms in the protein sets of the solutions. For each solution, we retrieved the functional annotation terms from the SGD database [SGD project] (for the yeast network), the GOA database [Barrell et al. 2009] (for the human network), or FlyBase [Tweedie et al. 2009] (for the fly network). Then,

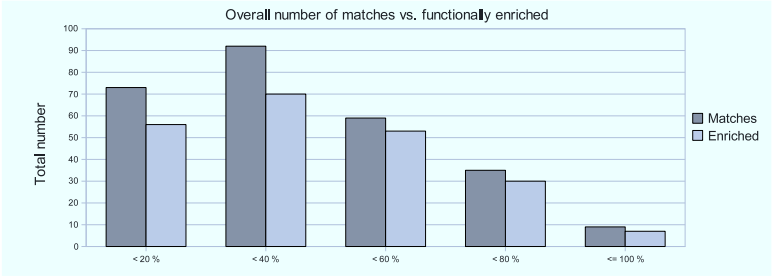


Figure 6.3: Comparison of the number of solutions with the number of solutions that showed a significant enrichment of at least one functional annotation term. Along the x-axis, the solutions are grouped into five categories, according to the value of solution size/query size.

we used the GO::TermFinder tool [Boyle et al. 2004] to find functional annotation terms that have a statistically significant overrepresentation compared to random protein sets. This is done by computing for each functional annotation term the p-value (that is, probability) of its abundance in the solution under the hypothesis that the solution does not show an overrepresentation of this functional annotation term. The reported p-values are corrected for multiple hypotheses testing using the Bonferroni method and the threshold for considering an enrichment as significant was set to $p < 0.05$. Since the solutions should be complexes, they are expected to have a common function. Hence, the percentage of solutions that have a common function is a measure of the solution quality.

First, we examined how the percentage of functionally enriched solutions correlates with solution sizes. We found that of the solutions with size two or three, which make up roughly 85% of the solutions, more than 78% have a significant enrichment of at least one functional annotation term. Of the solutions of size at least four, 100% had a significant enrichment of at least one functional annotation term. We therefore examined how the relation between query size and solution size influences the solution quality. Our results are shown in Figure 6.3. For solutions whose size is more than 40% of the query, the ratio of enriched vs nonenriched is roughly the same. For solutions whose size is less than 40% of the query, the percentage of enriched solutions drops. We thus conjecture that the maximum number of allowed deletions should be set to roughly $|M|/2$. For some instances, this is more than the number of deletions that are allowed by TORQUE, and possibly the percentage of instances that have a solution in TORQUE could be increased without a drop-off in solution quality by allowing more deletions. However, our results also show that by excluding insertions, the number of vertices in the solution is often very small. Hence, a limited number of insertions should be permitted. So far, however, the precise number of insertions that should be allowed seems to be unexplored.

6.3 On Finding More Robust Motifs

Lacroix et al. [2006] motivated the study of (variants) of the GRAPH MOTIF problem by considerations comparing "topological motifs" with "functional motifs". The GRAPH MOTIF problem only poses a minimal demand on the motif topology by requiring connectedness. The question arises what happens if we ask for somewhat "more robust" motifs, replacing the connectedness demand by standard graph-theoretic demands for biconnectivity, bridge-connectivity, and the like. Surprisingly, as we will show in this section, these seemingly small steps toward topologically more constrained motifs already lead to W[1]-hardness results, destroying the hope for fixed-parameter algorithms for these GRAPH MOTIF variants. Indeed, we will prove even stronger results, perhaps of independent interest, by showing that the problems of deciding on the existence of fixed-size biconnected or bridge-connected subgraphs, parameterized by the size of the subgraphs, are W[1]-hard. Moreover, we extend these results to higher connectivity demands. Furthermore, we answer an open question of Dondi et al. [2007] by showing that the parameter "number of connected components" in a graph motif leads to a W[1]-hard problem.

6.3.1 Biconnected Subgraphs of Size Exactly k

Originally, the GRAPH MOTIF problem was suggested, because it imposes the least possible restriction on the topology of the occurrence of the motif [Lacroix et al. 2006]. One way of extending the GRAPH MOTIF problem in this spirit is to search for *biconnected* occurrences instead of connected occurrences of the motif. Recall that a graph is biconnected if it has no cut-vertex. For example in the scenario of protein interaction network querying, demanding biconnectivity could be used to demand occurrences with higher interaction probabilities, without restricting the actual topology of the occurrence too much. The decision version of the problem can be formulated as follows:

> BICONNECTED GRAPH MOTIF
> **Input:** A multiset of colors M (the motif), and a vertex-colored undirected graph $H = (W, F)$.
> **Question:** Does there exist an $S \subseteq W$ such that the induced subgraph $H[S]$ is biconnected and there is a bijection between the colors of the vertices in S and M?

We will show that BICONNECTED GRAPH MOTIF is W[1]-complete when parameterized by the motif size M. In fact, we prove an even stronger result. Consider the special case that M contains only one color c, $|M| = k$, and all vertices in G have color c. Then, the resulting problem is to find a biconnected subgraph of size *exactly* k:

Figure 6.4: An example of the transformation of a CLIQUE instance with $k = 3$ into a BICONNECTED SUBGRAPH instance with $k' = 15$. White vertices in G' belong to V_1, black vertices to V_2.

BICONNECTED SUBGRAPH:
Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.
Question: Does there exist an $S \subseteq V$ of size k such that the induced subgraph $G[S]$ is biconnected?

Note that looking for a biconnected subgraph of order *at least* k is solvable in polynomial time by removing all cut-vertices of the graph and then finding the largest component. However, restricting the order of the biconnected subgraph to exactly k makes the problem surprisingly hard. We prove the parameterized hardness by reduction from the CLIQUE problem, which is known to be W[1]-complete [Downey and Fellows 1999] with respect to the order of the clique searched for.

CLIQUE
Input: An undirected graph G and a nonnegative integer k.
Question: Is there a complete subgraph of order k in G?

Theorem 6.6. BICONNECTED SUBGRAPH *is W[1]-complete with respect to the parameter* k.

Proof. To show the W[1]-hardness, we give a parameterized many-one reduction from CLIQUE to BICONNECTED SUBGRAPH.

Let (G, k) be a CLIQUE instance. We construct a graph G' from G by replacing every edge e of G with a simple path p_e that has $\binom{k}{2} + 1$ internal new vertices. The vertex set of G' can be partitioned into two vertex sets V_1 and V_2, where V_1 contains the vertices that correspond to vertices of the original graph G and V_2 contains the new internal path vertices. An example of this transformation is shown in Figure 6.4. Note that the reduction works for arbitrary path lengths greater than $\binom{k}{2}$. For reasons of simplicity, we choose the path length to be $\binom{k}{2} + 1$.

We prove in the following that G has a clique of order k if and only if G' has a biconnected subgraph of order $k' = k + \binom{k}{2} \cdot (\binom{k}{2} + 1)$. If G has a clique C of order k, then the subgraph that is induced by the k vertices of C and by the vertices on the $\binom{k}{2}$ paths that were created from the $\binom{k}{2}$ clique edges of C in G has order exactly $k + \binom{k}{2} \cdot (\binom{k}{2} + 1)$. Clearly, this subgraph is also biconnected.

It remains to show that if G' has a biconnected subgraph of order $k' = k + \binom{k}{2} \cdot (\binom{k}{2} + 1)$, then G has a clique of order k. Let G' have a biconnected subgraph $G'[S]$ of order k. If S contains one vertex of a path p_e, then it must contain all vertices from p_e, because otherwise $G'[S]$ would not be biconnected. Hence, the number of vertices k' in S can be expressed as $k' = a + b \cdot (\binom{k}{2} + 1)$, where $a = |S \cap V_1|$ and b denotes the number of paths in G' that correspond to edges of G.

We show that b must be $\binom{k}{2}$. First, if $b > \binom{k}{2}$, then, since we can assume without loss of generality that $k \geq 3$, $k' > (\binom{k}{2}+1) \cdot (\binom{k}{2}+1) > k + \binom{k}{2} \cdot (\binom{k}{2}+1)$, a contradiction. Second, we consider the case that $b < \binom{k}{2}$. Since it must hold that $k + \binom{k}{2} \cdot (\binom{k}{2}+1) = a + b \cdot (\binom{k}{2}+1)$, in this case one must have that $a > \binom{k}{2}$. This means that there are more than $\binom{k}{2}$ vertices in V_1 which must form a biconnected graph by inserting less than $\binom{k}{2}$ paths from V_2. Clearly, this it not possible. Hence, we have shown that $b = \binom{k}{2}$ and thus also $a = k$.

Since $G'[S]$ contains exactly $\binom{k}{2}$ paths consisting of vertices from V_2 and each path must connect two vertices of A, all vertices of A are pairwise connected via a path of length $\binom{k}{2}$. Hence, the subgraph $G[A]$ must be a clique of order k since it contains exactly k vertices and exactly $\binom{k}{2}$ edges.

Using a characterization of W[1] by Chen et al. [2005], the containment of BICONNECTED SUBGRAPH in W[1] can be shown in complete analogy to Guo et al. [2007, Theorem 12]. □

6.3.2 Bridge-connected Motifs and Motifs of Higher Connectivity

Another way to augment the connectivity demands is to search for bridge-connected motifs. We define BRIDGE-CONNECTED SUBGRAPH in complete analogy to BICONNECTED SUBGRAPH, simply replacing the demand for biconnectivity by the demand for bridge-connectivity. Recall that a graph is bridge-connected when it has no bridge, that is, an edge $\{u, v\}$ such that every path between u and v contains $\{u, v\}$. The reduction from CLIQUE as used in the proof of Theorem 6.6 works also for bridge-connected subgraphs. Since W[1]-membership also follows in complete analogy to Guo et al. [2007, Theorem 12] for bridge-connected motifs as well, we can state the following theorem.

Theorem 6.7. BRIDGE-CONNECTED SUBGRAPH *is W[1]-complete with respect to the parameter "number of subgraph vertices".*

In addition, we can generalize the hardness results to higher-connected graph motifs. To this end, consider the following problem.

> p-CONNECTED SUBGRAPH:
> **Input:** An undirected graph $G = (V, E)$ and a nonnegative integer k.
> **Question:** Does there exist an $S \subseteq V$ of size k such that the induced subgraph $G[S]$ is p-connected?

Observe that p-CONNECTED SUBGRAPH is nontrivially posed only if $p \leq k$. Otherwise the answer is clearly always "No".

Theorem 6.8. *p-CONNECTED SUBGRAPH is W[1]-complete with respect to the parameter k.*

Proof. We further extend the construction used for the proof of Theorem 6.6 as follows: We add a set A of $p-2$ additional vertices to $G' = (V', E')$, that is, we have $V' := V_1 \cup V_2 \cup A$. Furthermore, for every vertex $a \in A$ we have an edge from a to every vertex of $V' \setminus \{a\}$. The desired motif size is increased by $p-2$, that is, we set $k' := k + \binom{k}{2} \cdot (\binom{k}{2} + 1) + p - 2$. As a p-connected component must consist of at least p vertices, we have $p \leq k$ and, thus, the new parameter k' can be expressed as a function of k.

In the following, we prove that G contains a clique of order k if and only if there is a p-connected subgraph of order k' in G'.

Given a clique of order k in G, as argued in the proof of Theorem 6.6, we can find a biconnected subgraph of order $k + \binom{k}{2} \cdot (\binom{k}{2} + 1)$ that contains only vertices of $V_1 \cup V_2$. Adding the vertices of A to this subgraph obviously results in a p-connected subgraph of order k'.

Given a p-connected subgraph $G'[S]$ of order $k' := k + \binom{k}{2} \cdot (\binom{k}{2} + 1) + p - 2$, we show that the vertex set $S \cap V_1$ corresponds to vertices that form a clique in G. We start by proving that A must be a subset of S. Assume that there is an $a \in A$ with $a \notin S$. Then, in $G'[V' \setminus \{a\}]$ all vertices $v_j \in V_2$ have degree $p-1$ and, hence, cannot be part of a p-connected subgraph. In addition, since $G[V' \setminus (\{a\} \cup V_2)]$ is not p-connected, it cannot contain a p-connected subgraph. Thus, we know that $A \subseteq S$ and, hence, all motif vertices are exactly $(p-2)$-connected via vertices of A. Hence, we have to choose $k + \binom{k}{2} \cdot (\binom{k}{2} + 1)$ vertices of $V_1 \cup V_2$ that increase the connectivity by two. As argued in the proof of Theorem 6.6, this can only be achieved by choosing vertices of V_1 that correspond to a clique.

W[1]-membership can be shown in complete analogy to Guo et al. [2007, Theorem 12], and is therefore omitted. □

In complete analogy, we obtain the following.

Theorem 6.9. *p-EDGE CONNECTED SUBGRAPH is W[1]-complete with respect to the parameter k.*

6.4 Concluding Remarks

We have proposed and implemented new fixed-parameter algorithms for solving the LIST-COLORED GRAPH MOTIF problem. We also applied these algorithms for the topology-free querying of protein interaction networks. Our experiments show that realistic LIST-COLORED GRAPH MOTIF instances can be solved efficiently by

our implementations. However, we also encountered some instances where our color-coding based algorithm fails. Furthermore, our implementations are at the moment not able to deal with the insertion of gap proteins. Nevertheless, we believe that our current algorithm can serve as a base implementation that—with further improvements—should be able to cope with all relevant inputs in the scenario of topology-free querying of protein interaction networks. Hence, further work on LIST-COLORED GRAPH MOTIF and variants thereof is a worthwhile task. We point out three directions whose investigation seems promising.

- As already mentioned, our algorithms should be extended to deal with the insertion of "gap proteins" as proposed by Lacroix et al. [2006] and Bruckner et al. [2010]. In addition, it should be investigated how a variant of LIST-COLORED GRAPH MOTIF that finds a largest occurrence with a bounded number of *edge insertions* compares to TORQUE [Bruckner et al. 2010] and to our results. This could be an appropriate way to deal with networks in which many edges are missing, without adding vertices to the solution that are not similar to any query proteins.

- The enumeration variant of LIST-COLORED GRAPH MOTIF that reports all occurrences of a motif is also of practical relevance. Unfortunately, the problem of enumerating all occurrences of a motif is not fixed-parameter tractable with respect to $|M|$ since there can be $\Omega(n^{|M|})$ occurrences of a motif M. Also, it is not possible to obtain an output-sensitive enumeration algorithm that enumerates k occurrences of the motif in $f(k) \cdot \text{poly}(n)$ time, since already reporting one occurrence is NP-hard. For fixed values of $|M|$, the situation could be different, however. Hence, an interesting question is whether there is an enumeration algorithm that can enumerate k solutions in $f(k) \cdot g(|M|) \cdot \text{poly}(n)$ time, where f and g are functions only depending on k and $|M|$ respectively. Such an algorithm was presented for example for finding a fixed number x of high-scoring pathways in protein interaction networks [Lu et al. 2007]; it is not clear, however, whether this is also possible for LIST-COLORED GRAPH MOTIF.

- As shown in Section 6.1.3, LIST-COLORED GRAPH MOTIF is in general W[1]-hard with respect to the parameter $k := n - |M|$. In our experiments, we observed that the Randomized Brute-Force algorithm works well in case k is small, since it has running time $n^k \cdot \text{poly}(n)$. It would therefore be interesting to consider k in combination with other parameters, for example the size of the color lists in the host graph H. This might lead to fixed-parameter tractability for more general cases than vertex-colored graphs (as presented in Section 6.1.3).

Chapter 7

Querying with Arbitrary Topologies and Bounded Number of Orthologs

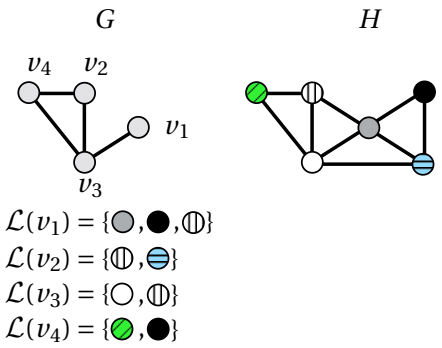
After having studied the querying problem for dense queries (in Chapter 5) and topology-free querying (in Chapter 6), we now consider querying problems for arbitrary query topologies. So far, the implementations for querying problems are restricted to path- and tree-like query topologies [Blin et al. 2010a,b, Dost et al. 2008, Hüffner et al. 2007, Sharan and Ideker 2006, Scott et al. 2006]. While these topologies are of great biological importance since they model for example signaling pathways [Scott et al. 2006], the query may not always have such a sparse topology. Protein complexes, for example, are often expected to have a dense interaction pattern in protein interaction networks [Sharan et al. 2005]. For protein complexes it has also been observed that interactions are evolutionary conserved [van Dam and Snel 2008]. Hence, if the query is a protein complex whose interactions are known, then it is reasonable to search for an occurrence in which the interactions of the query are also present. In other words, interactions should be "preserved" by the function that maps the query proteins to the proteins of the host network.

Fagnot et al. [2008] proposed to model this "preservation of interactions" by demanding that the mapping is an *injective homomorphism*.

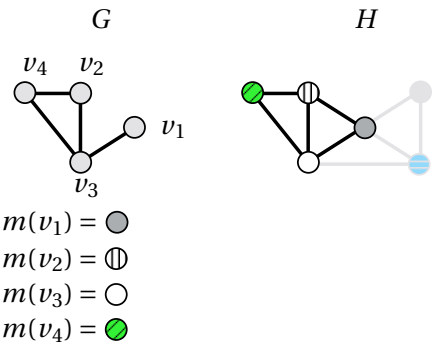
Definition 7.1. *Let $G = (V, E)$ and $H = (W, F)$ be two undirected graphs. An* injective homomorphism *from G to H is an injective mapping $m : V \to W$ such that if $\{u, v\} \in E$ then $\{m(u), m(v)\} \in F$.*

In graph-theoretic terms, the existence of an injective homomorphism from G to H means that H has a (not necessarily induced) subgraph H' that is isomorphic to G; the isomorphism from G to H' is an injective homomorphism from G to H.

Since finding an injective homomorphism is a computationally hard task—CLIQUE for example is a special case of this problem—Fagnot et al. [2008] proposed to also consider the additional structural information that is given by the orthology



(a) An input instance with $\mu_G = 3$ (since v_1 has three orthologs) and $\mu_H = 3$ (since v_1, v_2, and v_3 have a common ortholog).



(b) Mapping each vertex of G to the first vertex/color of its ortholog list is an injective homomorphism from G to H. Note that the graph induced by $\bigcup_{v \in V} \{m(v)\}$ is a supergraph of G.

Figure 7.1: A μ_G-INJECTIVE-HOMOMORPHISM instance with query graph G and host graph H.

constraints $\mathcal{L}$ which are also input of network querying problems. More precisely, they called ortholog lists (μ_G, μ_H)-*bounded* if each protein of the query G has at most μ_G orthologs in the host H and each protein in the host H has at most μ_H orthologs in the query G. An example of a query graph G with $(3,2)$-bounded ortholog lists is shown in Figure 7.1 (a). The additional information about the list structures allows for a more fine-grained analysis of the computational complexity of the querying problem. For example, if $\mu_G \leq 2$, then an injective homomorphism from G to H can be computed in polynomial time [Fagnot et al. 2008]. As defined in the introduction to Part III, we refer to the collection of ortholog lists as *list-coloring* of the query graph.

Combining the ortholog list bounds with requiring that the mapping should be an injective homomorphism, Fagnot et al. [2008] arrived at the following problem:

> (μ_G, μ_H)-INJECTIVE HOMOMORPHISM:
> **Input:** A query graph $G = (V, E)$ and a host graph $H = (W, F)$, and a (μ_G, μ_H)-bounded list coloring $\mathcal{L} : V \to 2^W$ of V.
> **Question:** Is there an injective homomorphism $m : V \to W$ such that for all $v \in V$ it holds that $m(v) \in \mathcal{L}(v)$?

We refer to an injective homomorphism from G to H for which it holds that $m(v) \in \mathcal{L}(v)$ for all $v \in V$ as an *orthology-respecting injective homomorphism* (*orih* for short), and to any mapping for which the latter holds as *orthology-respecting* mapping.

We present an easy-to-implement fixed-parameter algorithm for (μ_G, μ_H)-INJECTIVE HOMOMORPHISM. Since the running time of our algorithm does not depend on μ_H, we refer to the problem as μ_G-INJECTIVE-HOMOMORPHISM which is

equivalent to (μ_G, μ_H)-INJECTIVE HOMOMORPHISM with unbounded μ_H. An example of an instance of μ_G-INJECTIVE-HOMOMORPHISM is presented in Figure 7.1.

We also apply our algorithm to the problem that is obtained when subgraph isomorphism instead of injective homomorphism is used as mapping criterion.

μ_G-SUBGRAPH-ISOMORPHISM
Input: A query graph $G = (V, E)$ and a host graph $H = (W, F)$, and a (μ_G, μ_H)-bounded list coloring $\mathcal{L} : V \to 2^W$ of V.
Question: Is there a subgraph isomorphism $m : V \to W$ such that for all $v \in V$ it holds that $m(v) \in \mathcal{L}(v)$?

The difference between the two problems is that μ_G-INJECTIVE-HOMOMORPHISM demands finding a subgraph of H that is isomorphic to G whereas μ_G-SUBGRAPH-ISOMORPHISM demands finding an *induced* subgraph of H that is isomorphic to G. In the case of perfect data, μ_G-SUBGRAPH-ISOMORPHISM would correspond to demanding that the occurrence perfectly resembles the query. Protein interaction networks, however, are noisy. For example, there are many *false negative* interactions, that is, interactions that take place in biology but are missing from the data [Yu et al. 2006]. One could thus expect that μ_G-INJECTIVE-HOMOMORPHISM handles noisy data better than μ_G-SUBGRAPH-ISOMORPHISM. In fact, the proposal to use injective homomorphisms instead of subgraph isomorphisms was made in particular to deal with missing interactions [Fagnot et al. 2008].

We implemented our algorithms for μ_G-INJECTIVE-HOMOMORPHISM and for μ_G-SUBGRAPH-ISOMORPHISM and applied them for the querying of protein complexes in protein interaction networks. Our results indicate that injective homomorphism may indeed be a more appropriate model for querying of protein complexes than subgraph isomorphism.

In the following, we give a survey of the known results for the problems under consideration and summarize our findings.

Previous Work. μ_G-INJECTIVE-HOMOMORPHISM was introduced by Fagnot et al. [2008], who showed that the problem is NP-hard for $\mu_G > 2$ and polynomial-time solvable, otherwise. They also described an algorithm that solves μ_G-INJECTIVE-HOMOMORPHISM in $O((\mu_G)^k \cdot k \cdot (|V|+|E|))$ time, where k is the number of vertices in G that have at least two orthologs in H. The μ_G-INJECTIVE-HOMOMORPHISM problem is NP-hard even in very restricted cases, for example even in case G and H are bipartite with maximum degree one and two [Fertin et al. 2009], and a maximization variant of the problem is hard to approximate even if both G and H are linear forests [Brevier et al. 2010]. In case the query G is a path, tree, or has tree-like structure (for example by having bounded tree-width), the problem can be solved by the color-coding technique in $c^{|V|} \cdot \text{poly}(|V| + |W|)$ time where c is a constant [Blin et al. 2010b, Dost et al. 2008, Hüffner et al. 2008, Scott et al. 2006].

Our results. We present an algorithm for μ_G-INJECTIVE-HOMOMORPHISM with running time $O((\mu_G - 1)^{k'} \cdot (|V|^2 + |V||E|) + |W|^2)$, where k' is the number of vertices in G that have at least *three* orthologs in H. Note that the parameter k' can be much smaller than k since it does not count vertices with exactly two orthologs. Compared to the $O(((\mu_G)^k \cdot k \cdot (|V| + |E|)))$-time algorithm of Fagnot et al. [2008], our algorithm improves on the exponential part of the worst-case running time, but it has worse polynomial running time. We believe that the latter is acceptable, since in many instances k is almost as large as $|V|$ (see also Section 7.2 for more on typical parameter values). Furthermore, the running time in practice is dominated by the instances for which the parameter is large. Hence, tuning the running times for these instances leads to algorithms with considerably better average running times.

We then present an improved analysis of our algorithm, showing that the base in the exponential function can be expressed by using the geometric mean of the numbers in the multiset that contains the number $|\mathcal{L}(v)| - 1$ for each $v \in V$ (we will describe in Section 7.1 how this number is computed and how it relates exactly to μ_G). This bound is better in case there are only few vertices with many orthologs. We also apply our algorithm for solving μ_G-SUBGRAPH-ISOMORPHISM, achieving a running time of $O((\mu_G - 1)^{k'} \cdot |V|^3 + |W|^2)$.

We then evaluate our algorithms for μ_G-INJECTIVE-HOMOMORPHISM and for μ_G-SUBGRAPH-ISOMORPHISM in the context of querying protein complexes in protein interaction networks, showing that the theoretical speed-up achieved by our algorithm carries over into practice.[1] Furthermore, we examine the differences in using injective homomorphisms or subgraph isomorphisms as mapping criteria, showing that more putative complexes are reported when using injective homomorphisms instead of subgraph isomorphisms, while the functional coherence of the reported matches remains roughly the same. We see this is as evidence that for currently available protein interaction networks, using injective homomorphism as mapping criterion is preferable to using subgraph isomorphism.

7.1 A New Algorithm for μ_G-INJECTIVE-HOMOMORPHISM

In this section, we describe our new algorithm for μ_G-INJECTIVE-HOMOMORPHISM in detail and analyze its running time. Before doing so, we briefly outline the algorithm of Fagnot et al. [2008], and then describe the main idea that we exploit in our algorithm in order to achieve an improved running time.

The algorithm of Fagnot et al. [2008] works as follows. Recall that we are given the query graph $G = (V, E)$, the host graph $H = (W, F)$ and the list-coloring $\mathcal{L}$ of G. First, a

[1] We do not perform comparisons with the algorithm of [Fagnot et al. 2008] (which has not been implemented), but rather with a search tree algorithm that we believe actually outperforms the algorithm of Fagnot et al. [2008].

bipartite graph is built, where one part of the vertex set is V and the other part is the set of orthologs of V, that is, $\bigcup_{v \in V} \mathcal{L}(v)$. For each vertex $v \in V$ an edge to each of its orthologs is inserted. Then, there is a one-to-one correspondence between matchings of cardinality $|V|$ in this bipartite graph and orthology-respecting injective mappings from V to W. The algorithm then finds an orih from G to H by enumerating all matchings of cardinality $|V|$ in this bipartite graph and checking for each enumerated matching whether the corresponding orthology-respecting injective mapping fulfills the homomorphism condition. This is basically a brute-force approach that—in the analysis—makes use of the fact that the number of enumerated matchings is at most $(\mu_G)^k$, where k is the number of vertices in the query that have at least two orthologs in the host.

A very similar algorithm that has the same exponential running time bound is a recursive search tree algorithm which works as follows. Start by choosing some vertex v in G. Then, for each $w \in \mathcal{L}(v)$ branch into the case that there is an orih that maps v to w. Since $|\mathcal{L}(v)| \leq \mu_G$, this creates at most μ_G search tree branches. For each branch, recursively solve the problem of finding an injective homomorphism from G to H with the further restriction that in the recursive search tree branch, we can assume that the injective homomorphism f from G to H maps v to a fixed vertex $w \in \mathcal{L}$, that is, $f(v) = w$. Note that for a vertex v with only one ortholog one actually does not need to branch, one can simply set $f(v) = w$, where w is the single vertex of $\mathcal{L}(v)$. Using standard search tree analysis, the size of the search tree is $O((\mu_G)^k)$: We branch for each of the k vertices with two or more orthologs; the tree thus has depth k. The overall search tree size follows from the fact that the maximum degree of the search tree is μ_G. Observe that it is also possible that the search tree has size *exactly* $(\mu_G)^k$. This happens when $|\mathcal{L}(v)| = \mu_G$ for each of the k vertices with at least two orthologs. Basically, with this approach there is no way to avoid branching into μ_G cases since we do not know in advance which of the cases leads to success. In other words, while in a node of the search tree algorithm we have already fixed the mapping f for a vertex set $V' \subseteq V$, we have no information at all about the vertices in $V \setminus V'$.

The main idea of our algorithm can be described as follows. By iteratively building up the query graph G, we can solve μ_G-INJECTIVE-HOMOMORPHISM by solving at most $\mu_G \cdot |V|$ instances of an "intermediate" problem. We show that this intermediate problem can be solved in $O(\mu_G - 1)^{k'} \cdot (|V| + |E|) + |W|^2)$ time, where k' is the number of vertices with at least *three* orthologs. We explain the motivation behind using an intermediate problem, and its main advantage by comparing it with the search tree algorithm described above.

Recall that in the search tree algorithm, at each search tree node there is a set of vertices $V' \subseteq V$ for which one has already fixed the mapping, that is, we are given a mapping $f : V' \to W$ and we want to find a mapping $f' : V \to W$ such that for each $v \in V'$ we have $f'(v) = f(v)$. We say that f' *extends* f.

Definition 7.2. *Let $f : X_1 \to Y$ and $g : X_2 \to Y$ be two functions where X_1, X_2, and Y are arbitrary sets. We say that g* extends *f if $X_1 \subseteq X_2$ and $\forall x \in X_1 : f(x) = g(x)$.*

In the search tree algorithm described above, we cannot really make use of the fact that we are only looking for an extension of f: In the worst case, for every vertex $v \in V \setminus V'$ there are μ_G vertices in $\mathcal{L}(v)$, and it might be that only for one choice there is an orih f' with $f'(v) = w$. However, the reason that w is the only feasible way to map v does not necessarily depend on f. The trick of the intermediate problem is that we always have *two* mappings. The first mapping $f : V' \to W$ is an orih from $G[V']$, and the aim is, as in the search tree algorithm described above, to find an orih that extends f'. The second mapping $m : (V \setminus V') \to W$ is an orih from $G[V \setminus V']$, that is, the graph that is induced by those vertices of V that are *not* mapped by f, to the host graph H. The advantage lies in the observation that we can either obtain an orih from G to H by combining f and m, or we can find at least one vertex in $V \setminus V'$ for which we only have to consider $\mu_G - 1$ possibilities (instead of μ_G). We thus can exclude at least one possibility from our exhaustive search. This is the basis of our running time improvement.

Next (in Section 7.1.1), we will specify the intermediate problem and show how to solve it within the claimed running time. Afterwards (in Section 7.1.2), we show how to solve μ_G-INJECTIVE-HOMOMORPHISM by solving at most $\mu_G \cdot |V|$ instances of the intermediate problem.

7.1.1 Solving Injective Homomorphism With Candidate Mappings

As described above, in our algorithm for μ_G-INJECTIVE-HOMOMORPHISM we will repeatedly create a situation in which we have two mappings f and m from disjoint domains whose union is V. The aim is to find an injective homomorphism from G to H that extends f. Formally, we want to solve the following problem.

> μ_G-INJECTIVE-HOMOMORPHISM-WITH-CANDIDATE MAPPING (HCM):
> **Input:** A query graph $G = (V, E)$ and a host graph $H = (W, F)$, a (μ_G, μ_H)-bounded list coloring $\mathcal{L} : V \to 2^W$ of V, an orih $f : V' \to W$ from $G[V']$ to W, an orih $m : (V \setminus V') \to W$ from $G[V \setminus V']$ to W.
> **Question:** Is there an orih $f' : V \to W$ from G to H that extends f?

We now present an algorithm that solves HCM in $O((\mu_G - 1)^k \cdot (|V| + |E|) + |W|^2)$ time, where k is the number of vertices in $V \setminus V'$ with at least three orthologs. More precisely, the algorithm does the following. If the instance is a yes-instance, that is, if an orih from G to H that extends f exists, then the algorithm returns such an orih. Otherwise it returns the empty set. We call this algorithm *HCM-Solve*, and its pseudo-code is shown in Figure 7.2. Again, the main idea of the algorithm is that either we combine the mappings f and m or there is at least one vertex $v \in V \setminus V'$, that is, a vertex that

HCM-Solve$(G=(V,E), H=(W,F), \mathcal{L}, f: V \to W, m: (V \setminus V') \to W)$
1 **if** $f \cup m$ is conflict-free **then: return** $f \cup m$
2 **else:**
3 find conflict $\{j,l\}$ in $f \cup m$ with $j \in V \setminus V'$ ▷ *At least one vertex must be from* $V \setminus V'$
4 **for each** $w \in \mathcal{L}(j) \setminus \{m(j)\}$: ▷ *At most* $\mu_G - 1$ *branches for* v_j
5 **if** $f \cup \{(j,w)\}$ is conflict-free **then:**
6 $f' \leftarrow$*HCM-Solve*$(G, H, \mathcal{L}, f \cup \{(j,w)\}, m \setminus \{(j,m(j))\})$
7 **if** $f' \neq \emptyset$ **then: return** f'
8 **return** $\emptyset$

Figure 7.2: Pseudo-code of *HCM-Solve*. The mappings f and m are represented as sets of ordered pairs. In case a conflict-free mapping has been found it is returned. Otherwise the algorithm branches into all possible ways to resolve the conflict without introducing conflicts in f.

is mapped by m, for which we can show that $f'(v) \neq m(v)$. For this vertex we must consider only $\mu_G - 1$ possibilities for branching. We now describe *HCM-Solve* in detail.

The procedure *HCM-Solve* first checks whether combining f and m yields an orih from G to H. Slightly abusing notation, we use $f \cup m$ to denote the mapping that results from combining f and m.[2] Since f and m are orthology-respecting, so is $f \cup m$. Hence, it remains to determine whether $f \cup m$ is an injective homomorphism from G to H. To this end, we introduce the notion of *conflicts*.

Definition 7.3. *Let $G=(V,E)$ and $H=(W,F)$ be two undirected graphs, and let $g: V \to W$ be a mapping from V to W. A pair $\{j,l\}$ of vertices $j,l \in V$, $j \neq l$, forms* a conflict in g *if*

- $\{j,l\} \in E$ *and* $\{g(j),g(l)\} \notin F$, *or*
- $g(j) = g(l)$.

Mappings not containing any conflict are called *conflict-free*.

Observation 7.1. *Let $G=(V,E)$ and $H=(W,F)$ be two undirected graphs. A mapping $g: V \to W$ is an injective homomorphism from G to H if and only if g is conflict-free.*

Observation 7.1 follows from the definition of conflicts: The first part of the conflict definition is fulfilled whenever the mapping is not a homomorphism; the second part of the definition is fulfilled whenever the mapping is not injective. Conversely, a mapping that does not contain any conflict is a homomorphism and it is injective.

[2] Since f and m have disjoint domains, this formulation is well-defined.

As a consequence of Observation 7.1, we can determine whether a mapping is an injective homomorphism by checking whether it contains any conflicts. The main observation that *HCM-Solve* uses is that if there is a conflict in the candidate mapping $f \cup m$, then one of the two vertices of the conflict is mapped by f, and the other vertex is mapped by m. This fact is expressed by the following observation.

Observation 7.2. *Let $G = (V, E)$ and $H = (W, F)$ be two undirected graphs, and let $f : V' \to W$ and $f : (V \setminus V') \to W$, be two mappings such that*

- *f is an injective homomorphism from $G[V']$ to H, and*
- *m is an injective homomorphism from $G[V \setminus V']$ to H.*

Then, $f \cup m$ does not contain any conflict $\{i, j\}$ such that $\{i, j\} \subseteq V'$ or $\{i, j\} \subseteq V \setminus V'$. If $f \cup m$ is conflict-free, then it is an injective homomorphism from G to H.

The first part of the observation follows from the fact that f and m are conflict-free within their respective domains. Hence, if there is a conflict, then this conflict must contain one vertex from V' and one vertex from $V \setminus V'$. The second part of the observation follows from the fact that the domain of $f \cup m$ is V. Since it is conflict-free, it is thus an injective homomorphism from G to H.

If $f \cup m$ is conflict-free, then it is according to Observation 7.2 an injective homomorphism from G to H. Clearly, $f \cup m$ also extends f. Hence, an orih from G to H that extends f has been found, and the procedure thus correctly returns it.

Otherwise there is at least one conflict in $f \cup m$. In order to resolve this conflict, *HCM-Solve* recursively branches into several cases. If one of the search tree branches is successful, then it returns an orih from G to H.

Let $\{j, l\}$ be an arbitrary conflict in $f \cup m$. By Observation 7.2, we can assume without loss of generality that $j \in V \setminus V'$ and $l \in V'$. Since the aim in HCM is to find an orih f' that extends f, we know that $f'(l) = f(l)$. This implies that $f'(j) \neq m(j)$ which can be seen as follows. Since f' is an orih from G to H it is conflict-free. In case $f'(j) = m(j)$, however, j and l still form a conflict. Hence, if there is an orih f' from G to H that extends f, then it holds that $f'(j) \in \mathcal{L}(j) \setminus \{m(j)\}$. The procedure *HCM-Solve* branches into all possible cases to map j. Let $w \in \mathcal{L}(j) \setminus \{m(j)\}$ be one such possibility. Clearly, if there is an orih from G to H that extends f and maps j to w, then $f \cup \{j, w\}$ is conflict-free. Hence, we only have to recurse if this is the case. For the recursion, one has to ensure that we create again an instance of HCM. This is done as follows.

First, $f \cup \{(j, w)\}$ becomes the mapping that shall be extended. Since $f \cup \{(j, w)\}$ is conflict-free it is an injective homomorphism from $G[V' \cup \{j\}]$ to H. Since f is orthology-respecting, and $w \in \mathcal{L}(j)$, $f \cup \{(j, w)\}$ is also orthology-respecting. Second, $m \setminus \{(j, m(j))\}$ becomes the other part of the candidate mapping. Since it is a subset of m, it is also conflict-free and orthology-respecting. Furthermore, the

domain of $m \setminus \{(j, m(j))\}$ is $V \setminus (V' \cup \{j\})$. The domains of the two mappings are therefore disjoint, their union is V, and they are orihs from their respective domains. Consequently $(G, H, \mathcal{L}, f \cup \{(j, w)\}, m \setminus \{(j, m(j))\})$ is an instance of HCM, which is solved by a recursive call to *HCM-Solve*.

After a recursive call is finished, the algorithm checks whether an orih was found by the recursive call. If this is the case, then it correctly returns it. Otherwise the algorithm proceeds with the next recursive call. If none of the recursive calls found an orih, then there is no orih that extends f. Hence, the algorithm correctly returns the empty set in this case.

In the following, we bound the running time for this algorithm.

Lemma 7.1. *HCM can be solved in $O((\mu_G - 1)^{k'} \cdot (|V| + |E|) + |W|^2)$ time, where k' is the number of vertices in $V \setminus V'$ that have at least three orthologs in H.*

Proof. The correctness of the algorithm was shown above; it remains to show the running time bound. Suppose that in a preprocessing, we have computed an adjacency matrix of H in $O(|W|^2)$ time.

First, consider the case that $f \cup m$ is conflict-free. Then, the running time of the algorithm is the time needed for checking whether $f \cup m$ is conflict-free. One can check in $O(|V|)$ time whether $f \cup m$ is injective as follows. Label for each vertex $v \in V$ its mapping "destination" with color c_f if v is mapped by f and color c_m if v is mapped by m. Then, return a conflict when there is a vertex in W that has been labeled with both colors. Checking whether the mapping is an injective homomorphism can be performed in $O(|E|)$ time by checking for each edge in G whether its endpoints are mapped to vertices that are adjacent in H (for each edge this check takes constant time, since we have precomputed the adjacency matrix). In case $f \cup m$ is conflict-free, *HCM-Solve* thus achieves the claimed running time bound. Note that in $O(|V| + |E|)$ time we can also *list* all conflicts (we will make use of this in the following).

In the case that $f \cup m$ is not conflict-free, we prove the running time bound by induction on k'. For the base case $k' = 0$, the algorithm does the following. First, it computes in $O(|V| + |E|)$ time a list of all conflicts in $f \cup m$. Since $f \cup m$ is not conflict-free, the algorithm has found a conflict $\{j, l\}$ with $j \in V'$. Since $k' = 0$, we have $|\mathcal{L}(j)| \leq 2$. In case $|\mathcal{L}(j)| = 1$ the algorithm simply returns $\emptyset$ because it does not recurse. In case $|\mathcal{L}(j)| = 2$, the algorithm "recurses" into one case since $|\mathcal{L}(j) \setminus m(j)| = 1$. Let w be the uniquely determined vertex in $\mathcal{L}(j) \setminus m(j)$. By keeping the labels of the labeling procedure described above, we can determine in constant time whether w is already used by some other vertex in f (in this case the algorithm can immediately return the empty set) or some other vertex in m (in this case the algorithm updates the list of conflicts). Furthermore, we can find in $O(\deg(j))$ time all neighbors of j that are mapped to a vertex that is not adjacent to w. Depending on whether this neighbor is mapped by f or m, we return the empty set or update the conflict list. This is done until either the mapping is conflict-free, or the empty set is returned.

The overall running time is $O(|V|+|E|)$ for computing the initial list of conflicts, and $O(\sum_{v\in V} O(1)+O(\deg(v))) = O(|V|+|E|)$ for updating the set of conflicts for each vertex with at most two orthologs that is contained in a conflict that the procedure "branches" on. Hence, the overall running time bound holds for $k'=0$.

For $k'>1$, the running time bound can be seen as follows. First, we can compute in $O(|V|+|E|)$ a list of all conflicts. Then, we arbitrarily choose one of these conflicts. In case the chosen conflict contains a vertex with three or more orthologs that is mapped by m, we branch into at most μ_G-1 cases. In each of the recursively created instances, the number of vertices in $V\setminus V'$ with at least three orthologs has decreased by one. The time needed for solving them is therefore $O((\mu_G-1)^{k'-1}\cdot(|V|+|E|))$. The overall running time needed for the recursion then is

$$O((\mu_G-1)\cdot(\mu_G-1)^{k'-1}\cdot(|V|+|E|)) = O((\mu_G-1)^{k'}\cdot(|V|+|E|)).$$

In case the conflict contains a vertex $j\in V\setminus V'$ with at most two orthologs, we can (as for $k'=0$) simply "move" this vertex from m to f and update the set of conflicts in $O(\deg(j))$ time. Hence, the overall time that is spent on vertices with at most two orthologs before recursing is $O(|V|+|E|)$. The overall running time bound follows. □

7.1.2 Solving μ_G-Injective-Homomorphism

We now describe how we can solve μ_G-Injective-Homomorphism by solving HCM. The main idea here is to iteratively build up the query graph G, starting with a graph consisting of only one vertex and adding vertices one by one, computing an injective homomorphism for the current induced subgraph of G in each step. The pseudo-code of this algorithm (called *IterativeMap*) is shown in Figure 7.3. It takes as input the query graph G, the host graph H, and the family of ortholog lists $\mathcal{L}$. Let v_i denote the i-th vertex added to the subgraph of G for which the intermediate solution is computed, and let $G_i := G[\{v_1,\dots,v_i\}]$. *IterativeMap* first computes an orih from G_1 to H. Then, it computes an orih from G_2 to H, from G_3 to H, and so on. If for some i, there is no orih from G_i to H, then the procedure is aborted, and the algorithm outputs that there is no solution. In the following, we describe how the orih from G_i to H is computed. Note that, with the exception of G_1, we have already computed an orih from G_{i-1} to H and this orih is stored in the variable m.

When computing the orih from G_i to H, the algorithm first branches into all possible cases to map v_i to an ortholog in $w\in W$. For each $w\in W$, $\{(v_i,w)\}$ is an orih from $G[\{v_i\}]$ to H. Furthermore, the variable m stores an orih from $G[\{v_1,\dots,v_{i-1}\}]$ to H (which was computed in the previous iteration of the main loop). Hence, $(G[\{v_1,\dots,v_i\}],H,\mathcal{L},\{(v_i,w)\},m)$ is an instance of HCM. This instance is solved by a call to *HCM-Solve*.

The procedure *HCM-Solve* then either returns an orih from $G[\{v_1,\dots,v_i\}]$ or the empty set. In the first case, the algorithm sets $m\leftarrow f'$ and continues with computing

```
IterativeMap(G = ({v_1, ..., v_n}, E), H, L)
1   V' ← ∅
2   m ← ∅
3   for i := 1 to n:
4       V' ← V' ∪ {v_i}
5       f' ← ∅
6       for each w ∈ L(v_i):                    ▷ Try all cases for mapping v_i
7           f' ← HCM-Solve(G[V'], H, L, {(v_i, w)}, m)
8           if f' ≠ ∅ then: break               ▷ I.h. from G[V'] to H is found
9       if f' = ∅ then: return "no solution"    ▷ There is no i.h. from G[V'] to H
10      m ← f'
11  return "m is a solution"
```

Figure 7.3: Pseudo-code of the procedure *IterativeMap* that solves μ_G-INJECTIVE-HOMOMORPHISM by solving HCM. It returns either an orih f or reports that no such injective homomorphism exists. Herein, 'i.h.' stands for injective homomorphism.

an orih from G_{i+1} to H. In the second case, there is no orih from G_i to H, since for each $w \in \mathcal{L}(v_i)$ there is no orih from G_i to H that extends $\{(v_i, w)\}$. In this case, the algorithm correctly returns that the instance does not have a solution.

After the n-th iteration of the main loop of *IterativeMap*, m stores an orih from $G_n = G$ to H and the algorithm thus correctly returns that the instance has a solution m.

The overall running time of this algorithm can be bounded as follows.

Proposition 7.1. μ_G-INJECTIVE-HOMOMORPHISM *can be solved in* $O((\mu_G - 1)^{k'} \cdot (|V|^2 + |V||E|) + |W|^2)$ *time, where* k' *is the number of vertices in* V' *that have at least three orthologs in* H.

Proof. The correctness of the algorithm was shown above. It thus remains to bound its running time.

Overall, there are at most $|V|$ iterations of the main loop. In each iteration, the algorithm adds one vertex (v_i) to V' and thus also to $G[V']$. The overall time needed for this is $O(|V| + |E|)$. All other steps can be either performed in constant time or they are part of the branching and the calls to *HCM-Solve* in Lines 6–8 of the algorithm. In the following, assume that we have chosen v_n to be one of the vertices that have at least three orthologs. Then, in each call to *HCM-Solve*, the mapping m contains at most $k' - 1$ vertices which have at least three orthologs in H. Furthermore, the graph $G[V']$ is a subgraph of H. Hence, by Lemma 7.1 each call of *HCM-Solve* can be solved in $O((\mu_G - 1)^{k'-1} \cdot (|V| + |E|) + |W|^2)$ time. The overall number of calls to *HCM-*

Solve is at most $|V|\mu_G$. Hence, the overall time that is spent on these calls is

$$|V| \cdot \mu_G \cdot O((\mu_G - 1)^{k'-1} \cdot (|V| + |E|) + |W|^2).$$

Note that the $O(|W|^2)$ term in this running time is due to the computation of the adjacency matrix of H. Since this computation needs to be done only once, we arrive at the claimed overall running time bound. □

Proposition 7.1 implies a direct combinatorial polynomial-time algorithm for $\mu_G = 2$; so far the polynomial-time solvability was shown by a reduction to 2-SAT [Fagnot et al. 2008]. Furthermore, the running time for this polynomial-time algorithm was $O(|V|^3)$ [Fagnot et al. 2008]. Our algorithm runs in $O(|V| \cdot (|V| + |E|) + |V|^2)$ time in this case, since $|W| \leq 2 \cdot |V|$. This is true because every vertex in G has at most two orthologs in H, and we can assume that every vertex in H is an ortholog of at least one vertex in G. Hence, we also obtain an improved running time for the case $\mu_G = 2$.

7.1.3 A More Detailed Running Time Analysis

In the following, we give a more detailed analysis of the search tree size which leads to a more precise running time bound of the iterative search tree algorithm. The motivation for this analysis is the fact that using μ_G as the basis of the exponential function that bounds the search tree size is rather crude since this is the maximum of $|\mathcal{L}(v)|$ over all $v \in V$. We found that in the application to querying of protein complexes often many proteins of the query have only few orthologs in the host, while there are few proteins of the query that have many orthologs in the host. In these instances μ_G was often more than twice as large as the arithmetic mean of ortholog list sizes. We show that for such instances we can achieve a bound that is much better than $(\mu_G - 1)^{k'}$. In the following, suppose without loss of generality that $\{v_1, \ldots, v_k\} \subseteq V$ is the set of vertices in G that have at least two orthologs in H. Furthermore, let $(l_1, l_2, \ldots, l_k), l_i = |\mathcal{L}(i)|$, denote the vector that contains for each vertex from $\{v_1, \ldots, v_k\}$ exactly one entry with the number of its orthologs. We call $(l_1, \ldots, l_k)$ the *orthology-list vector* of G, and $(l_1 - 1, l_2 - 1, \ldots, l_k - 1)$ the *reduced orthology-list vector*. The main idea of the improved running time bound is that the search trees produced by *HCM-Solve* have size at most

$$\prod_{i=1}^{k} (l_i - 1),$$

that is, the search tree size is upper-bounded by the product of the numbers in the reduced orthology-list vector. Note that since all 2's in the orthology-list vector become 1's in the reduced orthology-list vector, they do not contribute to the size of the search tree (as we also showed in the proof of Lemma 7.1).

This observation leads to the following running time bound.

Theorem 7.1. μ_G-INJECTIVE-HOMOMORPHISM *can be solved in* $O((\tilde{\mu}_G)^{k'} \cdot (|V|^2 + |V||E|) + |W|^2)$ *time, where* k' *is the number of vertices in* G *that have at least three orthologs in* H*, and* $\tilde{\mu}_G := (\prod_{i=1}^{k}(l_i - 1))^{1/k'}$ *is the geometric mean of the numbers in the reduced orthology-list vector of* G*.*

Proof. The algorithm is the same algorithm as in Proposition 7.1. Hence, we prove the theorem by bounding the running time of the algorithm. The polynomial part of the running time is bounded as in Proposition 7.1; in the following, we give a more fine-grained analysis of the number of search tree nodes created by the calls to *HCM-Solve* in one iteration of the main loop of *IterativeMap*.

Suppose that in *IterativeMap*, we branch on the vertex v_i, that is, we call *HCM-Solve* l_i times. For each of these l_i initial calls to *HCM-Solve*, we can bound the size of the search tree that is created as $\prod_{j=1, j\neq i}^{k}(l_j - 1))$ because for each vertex v_j, $j < i$, in the recursive branching in *HCM-Solve* we branch into at most $l_j - 1$ cases. Hence, the overall number of search tree nodes created in the l_i initial calls to *HCM-Solve* is

$$l_i \cdot \prod_{j=1, j\neq i}^{k} (l_j - 1) \leq 2 \cdot \prod_{j=1}^{k'} (l_j - 1).$$

The inequality above follows from the observation that either $l_i \leq 2$ or $l_i \leq 2 \cdot (l_i - 1)$.

The base of the search tree size follows simply from taking the k'-th root of the overall number of search tree nodes since the tree has depth at most k'. The overall running time bound follows as in the proof of Proposition 7.1. □

The geometric mean of a multiset of numbers is always at most as large as the arithmetic mean, and the two are only equal if all numbers of the set are the same. For our instances, this is the case when $|\mathcal{L}(v)| = \mu_G$ for all $v \in V$. In this case, we have $\tilde{\mu}_G = \mu_G - 1$, that is, we achieve precisely the running time stated by Proposition 7.1. In the case of protein complex querying, however, $\tilde{\mu}_G$ is usually much smaller than μ_G.

Of course, this better analysis also applies to the search tree algorithm described in the beginning of Section 7.1. Hence, it is not "exclusive" to our algorithm. Note that for the standard search tree algorithm the worst-case search tree size is the product of the numbers in the orthology-list vector (as opposed to the *reduced* orthology-list vector). Somehow counterintuitively, the geometric mean of the numbers in the orthology-list vector might be smaller than $\tilde{\mu}_G$, for example, in case one vertex in G has four orthologs and all other vertices in G have two orthologs. This is explained by the fact that we define $\tilde{\mu}_G$ by taking the k'-th root of the search tree size (instead of the k-th root). Hence, in the example above, k can be arbitrarily large whereas $k' = 1$, and thus the fact that $\tilde{\mu}_G$ is larger than the geometric mean of the orthology-list vector is not a contradiction to our claimed running time improvement.

7.1.4 Solving μ_G-SUBGRAPH-ISOMORPHISM

In this section, we briefly outline how the presented algorithm can be adapted to solve μ_G-SUBGRAPH-ISOMORPHISM. The iterative procedure, that is, the main loop of the algorithm remains exactly the same. This is correct, since its correctness relies mainly only on two facts that also apply to μ_G-SUBGRAPH-ISOMORPHISM. First, if there is an orthology-respecting subgraph-isomorphism from G to H, then there is also an orthology-respecting subgraph-isomorphism from every induced subgraph of G to H. Second, each vertex $v \in V$ can only be mapped to a vertex in $\mathcal{L}(v)$. Given these two facts, the correctness completely relies on the procedure that solves the problem of finding an orthology-respecting subgraph-isomorphism that extends one mapping f and is given a candidate mapping $f \cup m$. For μ_G-INJECTIVE-HOMOMORPHISM this problem is solved by *HCM-Solve*. For μ_G-SUBGRAPH-ISOMORPHISM, we can use the same algorithm, the only difference is that we need a new definition of conflicts.

Definition 7.4. *Let $G = (V, E)$ and $H = (W, F)$ be two undirected graphs, and let $g : V \to W$ be a mapping from V to W. A pair $\{j, l\}$ of vertices $j, l \in V$, $j \neq l$, forms* an isomorphism-conflict in g *if*

- $\{j, l\} \in E$ *and* $\{g(j), g(l)\} \notin F$, *or*
- $\{j, l\} \notin E$ *and* $\{g(j), g(l)\} \in F$, *or*
- $g(j) = g(l)$.

By using isomorphism-conflicts instead of conflicts, we obtain an algorithm with almost the same running time for solving the intermediate problem. The correctness of this algorithm can be shown in complete analogy to the correctness of *HCM-Solve*. We therefore omit the details. The difference in the running time follows from the fact that we need $O(|V|^2)$ time for finding and listing the conflicts in a mapping.

Theorem 7.2. μ_G-SUBGRAPH-ISOMORPHISM *can be solved in $O((\tilde{\mu}_G)^{k'} \cdot |V|^3 + |W|^2)$ time, where k' is the number of vertices in G that have at least three orthologs in H, and $\tilde{\mu}_G := (\prod_{i=1}^{k}(l_i - 1))^{1/k'}$ is the geometric mean of the numbers in the reduced orthology-list vector of G.*

7.2 Application to Querying of Protein Interaction Networks

In this section, we apply our algorithm to the problem of querying protein interaction networks for protein complexes. Our experiments show that our iterative algorithm not only has an improved theoretical worst-case running time, but that it also performs better on real-world data than the previously proposed search tree algorithm [Fagnot et al. 2008]. Furthermore, we compare the quality of the occurrences that are returned using injective homomorphism with the ones returned when using subgraph isomorphism as mapping criterion.

Experimental Setup. We used protein interaction networks from yeast (5430 proteins and 39936 interactions), fly (6650 proteins, 21275 interactions), and human (7915 proteins, 28972 interactions) that were taken from Bruckner et al. [2010]. The queries were complexes from the same three species. We considered the following four combinations of query and host: with yeast complexes we queried both human and fly networks; with human and fly complexes we queried the yeast network.[3]

For each protein of the query, we collected all proteins in the host network whose sequence similarity to the query protein exceeded a predefined threshold, and created the corresponding lists of orthologs. The similarity threshold was set to a BLAST score of 10^{-7}. We then removed all proteins from the query that did not have any orthologs in the host, and solved μ_G-INJECTIVE-HOMOMORPHISM and μ_G-SUBGRAPH-ISOMORPHISM for the remaining subcomplex. In the following, we refer to these subcomplexes as *queries*.

The program is written in the Java Programming Language and comprises approximately 1300 lines of code.[4] Graphs are represented with their adjacency matrices, allowing for fast testing of adjacency for two given vertices. Experiments were run on an Intel Core-i3-550 processor with 3.2 GHz, 512 KB L2 cache, and 4 GB main memory running under the Debian GNU/Linux 6.0 operating system with Java version 1.6.0_12.

We implemented our algorithm (to which we refer as iterative algorithm in the following) and a search tree algorithm that is basically like the search tree algorithm described in the beginning of Section 7.1. The iterative algorithm was implemented as described in Section 7.1; the search tree algorithm was implemented in the following way. For $G = (\{v_1, \ldots, v_n\}, E)$, the algorithm starts with branching into the at most μ_G possibilities to map v_1. Consider one such possibility, say v_1 is mapped to w_1. Then, the algorithm removes w_1 from the ortholog lists of each v_i, $i <$ 1, and recursively branches into all possibilities to map v_2. Herein, only those possibilities are considered that do not introduce any conflict between v_1 and v_2, that is, they are an injective homomorphism (or subgraph isomorphism) from $G[\{v_1, v_2\}]$ to H. Then, the algorithm branches into all valid possibilities to map v_3, and so on. Branching is performed until either an injective homomorphism (or subgraph isomorphism) from G to H has been found, or all recursive branchings return that no such injective homomorphism (or subgraph isomorphism) exists. Note that this search tree algorithm should outperform the algorithm that is based on the enumeration of matchings Fagnot et al. [2008], since we recurse only into branches that do not introduce any conflicts. Hence, we do not necessarily enumerate all injective mappings from G to H.

We also performed the following simple reduction rule before actually starting the

[3] This somewhat arbitrary choice of query–host combinations was based on the convenient availability of the sequence similarity data.

[4] Source code available at http://fpt.akt.tu-berlin.de/blm

Table 7.1: Running times of the iterative algorithm and a search tree algorithm for homomorphism and isomorphism queries and different sequence similarity thresholds t. Running times are given in milliseconds, n denotes the size of the queries, '#' denotes the number of instances for the respective size category, the number of asterisks ('*') denotes the number of instances that could not be solved within 30 minutes.

			Homomorphism		Isomorphism	
threshold	n	#	Iterative	Search Tree	Iterative	Search Tree
$t = 10^{-5}$	$3 \leq n \leq 5$	427	0.06	0.14	0.08	0.23
	$6 \leq n \leq 9$	194	0.07	0.14	0.07	0.19
	$10 \leq n \leq 19$	101	1.33	3436.49	0.08	4585.33
	$20 < n$	44	*2.86	****3351.87	0.17	**2812.03
$t = 10^{-7}$	$3 \leq n \leq 5$	409	0.03	0.09	0.03	0.13
	$6 \leq n \leq 9$	185	0.05	0.09	0.02	0.08
	$10 \leq n \leq 19$	91	0.05	4300.27	0.04	5673.72
	$20 < n$	45	0.72	***57.31	0.12	*6186.80
$t = 10^{-9}$	$3 \leq n \leq 5$	392	0.03	0.07	0.02	0.11
	$6 \leq n \leq 9$	175	0.03	0.05	0.03	0.09
	$10 \leq n \leq 19$	88	0.03	1019.85	0.04	957.06
	$20 < n$	40	*0.33	**24.49	*0.18	*1493.53

algorithms.

Reduction Rule 7.1. *If there is a vertex $v \in V$ with exactly one ortholog w, then remove w from the ortholog lists of all vertices in $V \setminus \{v\}$.*

The effect of this rule was not very pronounced, although for our instances, it appeared that the search tree algorithm benefited more from it.

Running Times. Table 7.1 shows the average running times of the instances with at least three proteins in the query for both algorithms and isomorphism and homomorphism querying. Our main observations are as follows. First, for queries of size at most 9, both algorithms are very fast; the average running time is less than one millisecond for these instances. For queries of size at least 10 the problem becomes much harder and the differences between the algorithms are more pronounced; for these instances, our algorithm is faster than the search tree algorithm by several orders of magnitude. This behavior can be observed for all three considered thresholds and for injective homomorphisms and isomorphisms. While the search tree algorithm exceeded the running time limit of 30 minutes in altogether 13 instances, the iterative search tree algorithm exceeded this limit in only three instances. For both algorithms,

Table 7.2: Parameter values and expected speed-up for the considered input instances: k_{avg} denotes the average number of vertices with at least two orthologs, k'_{avg} denotes the average number of vertices with at least three orthologs, $k_{\max}$ is the maximum number of vertices with at least two orthologs, $k'_{\max}$ is the maximum number of vertices with at least two orthologs, su_{avg} is the average speed-up predicted by computing the worst-case search tree sizes, $\text{su}_{\max}$ is the maximum of the speed-ups predicted by computing the worst-case search tree sizes.

threshold	n	k_{avg}	k'_{avg}	$k_{\max}$	$k'_{\max}$	su_{avg}	$\text{su}_{\max}$
$t = 10^{-5}$	$3 \leq n \leq 5$	2.34	1.79	5	5	1.35	9
	$6 \leq n \leq 9$	4.32	3.26	9	9	3.04	19.15
	$10 \leq n \leq 19$	8.34	6.14	19	17	42.20	2880
	$20 < n$	27.0	19.46	62	52	84,185,953	3,657,466,365
$t = 10^{-7}$	$3 \leq n \leq 5$	2.17	1.67	5	5	1.28	9
	$6 \leq n \leq 9$	4.07	3.10	9	9	2.79	27
	$10 \leq n \leq 19$	7.53	5.47	17	15	57.87	4096
	$20 < n$	23.8	16.78	49	43	17,313,034	669,570,338
$t = 10^{-9}$	$3 \leq n \leq 5$	2.09	1.60	5	5	1.27	9
	$6 \leq n \leq 9$	3.91	2.96	9	9	2.78	30.38
	$10 \leq n \leq 19$	7.05	5.11	16	13	91.63	4096
	$20 < n$	22.19	16	44	42	9,758,122	340,650,885

solving μ_G-INJECTIVE-HOMOMORPHISM takes longer than solving μ_G-SUBGRAPH-ISOMORPHISM. Furthermore, as one would expect, the instances in which the ortholog lists are longer because the similarity threshold is less strict ($t = 10^{-5}$) were harder for the algorithm than the instances with shorter ortholog lists ($t = 10^{-7}$ and $t = 10^{-9}$). Surprisingly, one query that could not be solved within 30 minutes by the iterative search tree algorithm for $t = 10^{-9}$ was solvable for $t = 10^{-7}$.

The running time improvement is much larger than expected by merely comparing the theoretical worst-case running times of the two algorithms. Hence, one could assume that it is due to a particularly bad implementation of the search tree algorithm. We thus examined the input instances more closely, to see whether the observed running time improvement really corresponds to the theoretical analysis of the worst-case running time.

First, we computed the parameter values for both algorithms, that is, the number k of vertices that have at least two orthologs (which is the parameter for the search tree), and the number k' of vertices that have at least three orthologs (which is the parameter for the iterative algorithm). Table 7.2 shows for each considered threshold t and for the different query size categories the average and maximum values of k and k'. As expected, k is on average larger than k'. However, the difference between the average

values of k and k' is not large enough to explain the observed speed-up. For example, for queries of size at least 20 and $t = 10^{-5}$ the average value of k was 27 and the average value of k' was 19.46, which does not explain the observed running time differences for this set of instances, especially since for the vertices with two orthologs we branch only into two cases. Of course, average parameter values can be deceiving since the instances with particularly high parameter values contribute much more to the overall running times than the instances with low parameter values. The difference between the maximum value of k and the maximum value of k', however, is also not large enough to explain the running time improvement. Hence, we directly computed the worst-case sizes of both search trees by multiplying the numbers in the orthology-list vector, yielding the worst-case search tree size T_{st} for the search tree algorithm, and in the reduced orthology-list vector, yielding the worst-case search tree size T_{iter} for the iterative algorithm. We then calculated an expected speed-up factor $T_{\text{st}}/T_{\text{iter}}$ for each instance. The average and maximum speed-up for each set of instances is also shown in Table 7.2. Both the average and the maximum expected speed-up factors are much larger than one would expect by merely considering the differences in the parameter values. Since T_{st} and T_{iter} are worst-case estimates, these predicted speed-up factors are not actually achieved, but they can help explaining the observed running time differences.

Functional Coherence. We studied the quality of the matches that were found by the algorithm with respect to functional similarities between matched proteins. One aim of our analysis is to assess the differences between isomorphism and injective homomorphism matching. To this end, each reported match was tested for the enrichment of gene ontology terms using the GO::Term Finder Software [Boyle et al. 2004]: For each solution, we retrieved functional annotation terms from the SGD database [SGD project] (for the yeast network), the GOA database [Barrell et al. 2009] (for the human network), or FlyBase [Tweedie et al. 2009] (for the fly network). Then, we used the GO::TermFinder tool [Boyle et al. 2004] to find functional annotation terms that have a statistically significant overrepresentation compared to random protein sets. This is done by computing for each functional annotation term the p-value of its abundance in the solution under the hypothesis that the solution is a random set of proteins. The reported p-values are corrected for multiple hypotheses testing using the Bonferroni method and the threshold for considering an enrichment as significant was set to $p < 0.05$. Since the solutions should be complexes, they are expected to have a common function, participate in the same biological process, or appear in the same cellular component.

Table 7.3 shows for each considered similarity threshold and for each combination of query and host the number of reported matches and the number of the reported matches that had a statistically significant enrichment of annotations for function, cellular component, or biological process.

Table 7.3: Functional coherence in the predicted complexes for homomorphism and isomorphism queries and different sequence similarity thresholds t. The total number of matches that were found is denoted by '#'; the columns with the number of matches with significant enrichment of GO annotations for cellular component, function, and biological process are denoted by C, F, and P, respectively.

			Homomorphism				Isomorphism			
threshold	query	host	#	C	F	P	#	C	F	P
$t = 10^{-5}$	human	yeast	177	148	153	162	101	72	83	85
	fly	yeast	41	32	36	38	20	11	15	17
	yeast	fly	8	6	5	5	8	6	5	5
	yeast	human	13	11	11	11	12	10	10	10
$t = 10^{-7}$	human	yeast	179	153	158	167	91	65	78	78
	fly	yeast	37	31	32	34	19	11	16	17
	yeast	fly	6	4	3	5	6	4	3	5
	yeast	human	14	12	13	12	13	11	12	11
$t = 10^{-9}$	human	yeast	169	148	148	158	81	59	66	70
	fly	yeast	37	32	32	35	19	11	15	17
	yeast	fly	4	3	3	4	4	3	3	4
	yeast	human	13	11	11	10	12	10	10	9

The most notable difference is that when the yeast protein interaction network is the host, the number of matches reported by injective homomorphism is much higher than for isomorphism. The percentage of reported matches for injective homomorphism that have an enrichment for at least one GO term is in most cases also higher than for isomorphism. When using yeast proteins as query, both approaches have basically the same outcome. These results indicate that, in some cases, injective homomorphisms might be preferable to isomorphisms. For instance, this could be the case when the percentage of false negative interactions is lower in the host than in the query.

7.3 Concluding Remarks

We have presented new algorithms for solving the querying problem with arbitrary topologies. We also contributed the, to our knowledge, first implementation for querying with arbitrary topologies. While our implementation is probably not as advanced as most biological applications would demand (for example, we have not touched at all on edge-weighted versions of querying), we were able to report on some results that indicate that injective homomorphisms are a useful querying model also

for arbitrary topologies.

- Further heuristic improvements of the implemented algorithms are conceivable. For the search tree algorithm this seems to be worthwhile for several reasons. First, in contrast to the iterative algorithm it can be used to enumerate all injective homomorphisms from the query to the host. Second, it is possible that with such heuristic improvements, the search tree algorithm could be improved so much that it is better than the iterative algorithm. This is due to the fact that in the search tree algorithm there is more freedom in choosing the vertex to branch on. In the iterative algorithm, there might be always only one conflict, which means that there is no freedom at all to choose the vertex to branch on.

- As we have shown, μ_G-INJECTIVE-HOMOMORPHISM and μ_G-SUBGRAPH-ISOMORPHISM are fixed-parameter tractable with respect to the combined parameter (μ_G, k) where k is the number of proteins in the query that have at least *three* orthologs. Is it possible to achieve a polynomial-size problem kernel for this combined parameter? In other words, can we develop data reduction rules that *remove vertices with at most two* orthologs?

- Studying the more difficult problem of finding a maximum-cardinality subset of the query that can be matched to the host (in the case of μ_G-SUBGRAPH-ISOMORPHISM this is related to the MAXIMUM COMMON SUBGRAPH problem which has applications for example in the analysis of chemical compounds [Cao et al. 2008]) would be interesting. From an applied standpoint, this would help in increasing the number of instances for which a match in the host is found, especially for larger queries. From an algorithmic standpoint it would be interesting to study whether our iterative algorithm is also useful for this more general problem or not.

- Finally, implementations for other problem variants that are able to cope with noise in the data such as the maximization variant proposed by Brevier et al. [2010] should be developed in order to increase the availability and flexibility of querying tools for arbitrary topologies.

Part IV

Conclusion

Chapter 8

Summary

In this work, we have reported on algorithmic progress for combinatorial problems in the areas of network clustering and querying. An overview over the specific results is given in the introductions of Part II and Part III. Instead of repeating the single findings for each problem, we attempt to bundle our results into cohesive groups, clusters if you will, that are defined by their algorithmic contributions.

Data Reduction. Data reduction and kernelization are perhaps the most important contributions of parameterized algorithmics to practical computing. In this thesis we have made the following contributions to this area of parameterized algorithmics.

In Chapter 4, we presented a new kernelization concept: structural kernelizations and discussed how this new kernelization concept relates to other recently introduced extensions of standard problem kernels. We demonstrated an application of structural kernelizations to CONSENSUS CLUSTERING. The idea of structural kernelizations is to reduce not necessarily the overall size of the input but the "size" of an important structural measure of the input. We believe that the definition of structural kernelization nicely extends the standard problem kernel definition, and that this concept can find applications for a wider range of problems. In fact, many claimed problem kernels are formally not "real" problem kernels. For example, some kernelizations for graph problems result in graphs that contain $f(k)$ vertices but arbitrarily large edge-weights. For these instances, the size of the instance is not bounded by the parameter, but the data reduction routine results in an instance in which an important part of the input—the number of vertices—is bounded. These kernelizations can be seen as structural kernelizations. An example for such a data reduction routine is the algorithm that was used to show the fixed-parameter tractability of WEIGHTED AVERAGE-s-PLEX-CLUSTER EDITING in Section 3.3.

This brings us to the second contribution that we have made in the area of data reduction. We have presented several data reduction algorithms that result in what could be called "multivariate problem kernels" because the "size" of the problem

kernel depends on several variables: In Section 3.3, our data reduction approach for WEIGHTED AVERAGE-s-PLEX-CLUSTER EDITING produces instances with $4k^2 + 8sk$ vertices, where k is the number of edge modifications, and s is the "slack" parameter in the definition of average-s-plexes. In Section 2.2, we presented a kernelization for (d,t)-CONSTRAINED-CLUSTER EDITING and (d,t)-CONSTRAINED-CLUSTER DELETION that produces problem kernels with at most $O(d \cdot t)$ vertices, where d is the number of clusters and t is the local modification bound. In Chapter 4, our structural kernelizations for CONSENSUS CLUSTERING are also standard problem kernels for the combined parameter "average distance d of the input partitions and number n of input partitions". We believe that, going forward, multivariate problem kernels will be an intriguing research area for achieving detailed measures of the effectiveness of data reduction algorithms. Measuring the "size" of problem kernels by more than one parameter is of course not only limited to the standard problem kernel definition, but could also be applied to structural kernelization, weak kernels [Jiang and Zhu 2010], or Turing kernelization [Fernau et al. 2009, Lokshtanov 2009].

Identification of New Parameters. As the running time of fixed-parameter algorithms depends exponentially on the value of the parameter, finding the "right" parameter is of utmost importance. We believe that the area of parameter identification is still somewhat underdeveloped in the field of parameterized algorithmics. We have contributed to this area in the following ways.

For average-degree based density definitions, we have introduced (in Chapter 3) the average-s-plex model, which naturally introduces a slack parameter s that can be used to obtain fixed-parameter tractability results. Given the apparent hardness of computational problems that involve average-degree based models for dense graphs, which can be witnessed for example in the W[1]-hardness results in Sections 3.4 and 5.2, defining such problems by using the average-s-plex model introduces the further structural parameter s which could then lead to fixed-parameter tractability results.

For edge modification problems we have introduced (in Chapter 2) the parameter "local modification bound t" which is an interesting alternative to the standard parameter "number k of edge modifications". While it appears that this parameter alone will usually not lead to fixed-parameter tractability results, we have demonstrated that in combination with other parameters we can obtain fixed-parameter tractability results that complement those for standard parameters.

For consensus problems, we have demonstrated (in Chapter 4) how fixed-parameter tractability can be achieved for the parameter "average distance of the input objects". We believe that parameterizing by this parameter can lead to fixed-parameter tractability results for a wide range of problems that contain as input a set of objects with a naturally defined pairwise distance function between them. This includes but is not limited to median problems.

For querying problems, we have presented several hardness and tractability results for parameters that describe the structure of the orthology constraints such as "the maximum number of orthologs" or "the number of colors in the query". With the help of these parameters one obtains a more detailed view of the structure of the instances of typical querying problems. Furthermore, we have presented (in Chapter 5) a parameterization that can be smaller than the degeneracy of the host graph. This parameterization can be seen as an easy example for "parameterizing below guarantee": we know that the query in this particular querying problem must be smaller than the degeneracy, and that the case in which it is equal to the degeneracy is "easy".

Multivariate Implementations & Their Experimental Analysis. We have developed implementations of parameterized algorithms for topology-free querying (in Chapter 6) and querying with arbitrary topologies (in Chapter 7). Both implementations use more than one parameter and, hence, are examples of "multivariate implementations".

For topology-free querying, we have considered three different parameters: the query size, the solution size, and the corresponding dual parameter. For the first two parameters we compared the fixed-parameter algorithms, showing that—for our algorithms—parameterization by query size is better than parameterization by solution size although the latter is usually the smaller parameter. For the third parameter, we demonstrated the usefulness of an XP-algorithm with running time $n^{k+O(1)}$. In particular, we showed that using the XP-algorithm to complement the fixed-parameter algorithms yielded a significant speed-up for many instances. In this sense, our implementation is a primitive example of a multivariate algorithm that exploits several parameters and works well when at least one of them is small.

In the case of querying with arbitrary topologies, we have also implemented and compared algorithms that use two different parameters. Our experimental analysis demonstrated (in particular the analysis of the parameter values and the expected search tree sizes) that even a seemingly incremental improvement in the worst-case running time may result in a large speed-up factor in real-world instances.

Chapter 9

Outlook

We conclude with a collection of open questions and suggestions for future research that are inspired by the findings in this work. We attempt to order these suggestions starting with the ones that are the most theoretical and address fundamental questions, gradually moving to the ones that are more applied. The recurrent theme of these suggestions could be phrased as "carving out the transformation from parameterized algorithmics to multivariate algorithmics".

Theoretical Foundations. Parameterized Algorithmics has led to a plethora of algorithm design and analysis tools, for example kernelization, that can be used to analyze algorithmic properties of combinatorial problems. These tools appear to be tailor-made for parameterizations with one parameter. The development of all these tools is more or less rooted in the basic question whether for instances (I,k) of a parameterized problem a running time of $f(k)\cdot\text{poly}(n)$ can be achieved or not. As combined parameterizations where the parameter consists of several independent "components" become increasingly important, a multitude of questions concerning possible running times for such a combined parameter arise. Consider for example a parameterized problem with a combined parameter (k,l) that consists of two independent parts k and l. Suppose that this problem is fixed-parameter tractable for the combined parameter (k,l), for example because there is an algorithm of running time $k^l\cdot\text{poly}(n)$. Assuming that k and l are really independent, there can be instances in which k is much smaller than l. For these instances, a running time of $l^k\cdot\text{poly}(n)$ would be preferable. How can one prove that such a running time cannot be achieved under plausible complexity-theoretic assumptions? A possibility would be to show that the problem is NP-hard for constant k, but what if this is not the case? Can one build a framework, for instance by adapting parameterized complexity theory, to answer this type of questions?

Similar questions can be posed for kernelizations of parameterized problems with a combined parameter. How can one show that while a kernel of size $f(k)\cdot\text{poly}(l)$ is

possible, it is not possible to achieve a kernel of size $f(l)\cdot\text{poly}(k)$? Or similarly, how can one show that while a kernel of size $O(k\cdot l)$ is achievable, one cannot achieve a kernel of size $O(k+l)$? Can such lower bounds for the kernel size for combined parameters be obtained by using the lower bound techniques for single parameters [Bodlaender et al. 2009, Bodlaender 2009, Dell and van Melkebeek 2010, Dom et al. 2009, Fortnow and Santhanam 2011]?

Identifying and Sorting Parameterizations. A lot of research in parameterized algorithmics focuses on the development of algorithmic techniques and frameworks for showing lower bounds on the computational complexity of parameterized problems. From an applied point of view, an equally important part of parameterized algorithmics is what has been coined "the art of parameterization" [Niedermeier 2006, 2010] since the first step of obtaining a useful fixed-parameter algorithm is to identify the parameter. Apart from considering the more traditional parameters such as "solution size" and "treewidth", it has been suggested for example to identify parameters by studying trivial special cases [Cai 2003, Guo et al. 2004] or by analyzing the structure of instances that are created by NP-hardness proofs [Komusiewicz et al. 2011]. We propose a further approach for parameter identification. The idea of this approach is to automatically analyze the structure of typical input instances for a problem under consideration. For example for graphs, there is an abundance of structural parameters that might be of interest for a particular problem. On the one side, a large number of potential parameters is beneficial since it increases the number of possibilities to attack a hard problem. On the other side, the larger the number of possible parameters, the harder it is to find the "best" parameter. A way out of this quandary could be to examine a set of typical instances, for example a set of graphs, and to estimate the value of all parameters that have been applied for NP-hard graph problems. Usually, the size of the input graphs, and the number of possible parameters prohibits a manual examination of the set of input instances. Hence, one should aim at implementing a tool for the automatic estimation of parameter values in such a set of input instances. Note that for such a tool one does not necessarily have to compute the actual parameter values (which for many structural parameters is an NP-hard problem itself). Instead, one can also use approximation algorithms and heuristics. After a promising set of input parameters has been identified in this way, one can focus on the development of fixed-parameter algorithms for these parameters.

Another related way of parameter identification would be to study the behavior of known exact algorithms for an NP-hard problem that outperform their theoretical worst-case running time for a certain set of input instances. Then, one could search for structural parameters that are small for this set of instances, and large for instances in which the algorithm does not perform well. Subsequently, one could show that the problem is fixed-parameter tractable for one of the found structural parameters. Once

a parameter has been identified in this way, one has also identified a target for further reducing the running time or increasing the "range" of tractable instances.

A further challenge that has been created by the increasing number of possible parameters is to keep track of the results that have been achieved for all the different parameterization of a particular problem. Also, it becomes increasingly difficult to understand the relationships between possible parameters. However, with an increasing number of parameters, the relationship between them becomes more complicated. We believe that a considerable effort should be made to "sort" the results on the relationship between different parameterizations for a problem. In Section 4.5, we have attempted to do so for the different parameters that we identified for CONSENSUS CLUSTERING, but of course this overview of the parameterizations for CONSENSUS CLUSTERING is very far from complete. We believe that for almost every parameterized problem there are many more interesting and practically relevant parameters that can be discovered. Providing for each particular problem a "condensed" view of the different parameterized complexity results and of the relationships between the parameters then becomes more and more important.

Adaptive Implementations of Fixed-Parameter Algorithms. Fixed-parameter algorithms are only one way for exactly solving computationally hard problems: they compete, for example, with running-time heuristics and integer linear programs. We believe that the impact of parameterized algorithmics to practical computing can and should be increased in the future.

One way of doing so is the continued improvement of fixed-parameter algorithms with respect to a known parameter for a problem, for example by improving the overall worst-case running time, or by developing better kernelization algorithms that produce smaller kernels. A further way of increasing the power of fixed-parameter algorithms for solving instances of a particular problem is to continually add implementations for different parameterizations of this problem. An ultimate goal would be to develop a meta-algorithm that—depending on the parameter values in a particular instance of a problem—switches between different parameterizations. This switching could take place even during the course of an algorithm. For example, if a recursive search tree algorithm for some parameter k creates an instance for which it seems advantageous to solve it by a fixed-parameter algorithm for parameter l, then such an algorithm could simply switch to the parameter l. This approach is not at all limited to "pure" fixed-parameter algorithms but it could also be used to augment the power of other approaches to hard problems such as integer linear programs.

Bibliography

J. Abello, M. G. C. Resende, and S. Sudarsky. Massive quasi-clique detection. In *Proceedings of the 5th Latin American Theoretical Informatics Symposium (LATIN '02)*, volume 2286 of *Lecture Notes in Computer Science*, pages 598–612. Springer, 2002. Cited on p. 44.

N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM*, 55(5), 2008. Article 23 (October 2008), 27 pages. Cited on pp. 19, 69, and 72.

N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. Cited on p. 116.

N. Alon, D. Lokshtanov, and S. Saurabh. Fast FAST. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP '09), Part 1*, volume 5555 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2009. Cited on p. 39.

N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1–3):89–113, 2004. Cited on pp. 18, 19, and 24.

A.-L. Barabási. *Linked: The New Science of Networks*. Perseus Books Group, 2002. Cited on p. 3.

A.-L. Barabási and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004. Cited on pp. 4 and 17.

D. Barrell, E. Dimmer, R. P. Huntley, D. Binns, C. O'Donovan, and R. Apweiler. The GOA database in 2009 - an integrated gene ontology annotation resource. *Nucleic Acids Research*, 37(Database-Issue):396–403, 2009. (3/19/2010). Cited on pp. 115, 128, and 152.

V. Batagelj and M. Zaversnik. An O(m) algorithm for cores decomposition of networks. Technical report, 2003. Cited on p. 104.

M. Bertolacci and A. Wirth. Are approximation algorithms for consensus clustering worthwhile? In *Proceedings of the 7th SIAM International Conference on Data Mining (SDM '07)*, pages 437–442. SIAM, 2007. Cited on p. 72.

N. Betzler, M. R. Fellows, C. Komusiewicz, and R. Niedermeier. Parameterized algorithms and hardness results for some graph motif problems. In *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM '08)*, volume 5029 of *Lecture Notes in Computer Science*, pages 31–43. Springer, 2008. Cited on p. iv.

N. Betzler, R. Bredereck, and R. Niedermeier. Partial kernelization for rank aggregation: Theory and experiments. In *Proceedings of the 5th International Symposium on Parameterized and Exact Computation (IPEC '10)*, volume 6478 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2010a. Cited on pp. 73, 76, 90, and 91.

N. Betzler, J. Guo, C. Komusiewicz, and R. Niedermeier. Average parameterization and partial kernelization for computing medians. In *Proceedings of the 9th Latin American Theoretical Informatics Symposium (LATIN '10)*, volume 6034 of *Lecture Notes in Computer Science*,

pages 60–71. Springer, 2010b. Cited on p. iv.

N. Betzler, R. van Bevern, C. Komusiewicz, M. R. Fellows, and R. Niedermeier. Parameterized algorithmics for finding connected motifs in biological networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1296–1308, 2011a. Cited on p. iv.

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, 2011b. Cited on pp. iv, 72, 73, and 91.

R. van Bevern, C. Komusiewicz, H. Moser, and R. Niedermeier. Measuring indifference: Unit interval vertex deletion. In *Proceedings of the 36th Workshop in Graph-Theoretic Concepts in Computer Science (WG '10)*, volume 6410 of *Lecture Notes in Computer Science*, pages 232–243, 2010. Cited on p. iii.

A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC '07)*, pages 67–74. ACM, 2007. Cited on pp. 118 and 123.

G. Blin, F. Sikora, and S. Vialette. GraMoFoNe: a Cytoscape plugin for querying motifs without topology in Protein-Protein Interactions networks. In *Proceedings of the 2nd International Conference on Bioinformatics and Computational Biology (BICoB'10)*, pages 38–43. International Society for Computers and their Applications (ISCA), 2010a. Cited on pp. 114, 128, and 135.

G. Blin, F. Sikora, and S. Vialette. Querying graphs in protein-protein interactions networks using feedback vertex set. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(4):628–635, 2010b. Cited on pp. 98, 135, and 137.

S. Böcker. A golden ratio parameterized algorithm for cluster editing. In *Proceedings of the International Workshop on Combinatorial Algorithms (IWOCA '11)*, Lecture Notes in Computer Science. Springer, 2011. To appear. Cited on p. 19.

S. Böcker and P. Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011. Cited on p. 23.

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. 17, 18, 19, 21, 33, and 43.

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. 17, 19, and 21.

H. L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14 (1):1–23, 1993. Cited on p. 75.

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 *Lecture Notes in Computer Science*, pages 17–37. Springer, 2009. Cited on pp. 10, 91, and 162.

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. 10, 91, and 162.

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 p. 19.

P. Bonizzoni, G. D. Vedova, R. Dondi, and T. Jiang. On the approximation of correlation clustering and consensus clustering. *Journal of Computer and System Sciences*, 74(5):671–

696, 2008. Cited on p. 72.

N. Bousquet, J. Daligault, and S. Thomassé. Multicut is FPT. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC '11)*, pages 459–468. ACM, 2011. Cited on p. 19.

E. I. Boyle, S. Weng, J. Gollub, H. Jin, D. Botstein, J. M. Cherry, and G. Sherlock. GO::TermFinder–open source software for accessing gene ontology information and finding significantly enriched gene ontology terms associated with a list of genes. *Bioinformatics*, 20(18):3710–3715, 2004. Cited on pp. 115, 129, and 152.

G. Brevier, R. Rizzi, and S. Vialette. Complexity issues in color-preserving graph embeddings. *Theoretical Computer Science*, 411(4–5):716–729, 2010. Cited on pp. 137 and 154.

S. Bruckner, F. Hüffner, R. M. Karp, R. Shamir, and R. Sharan. Torque: Topology-free querying of protein interaction networks. *Nucleic Acids Research*, 37(Web Server issue):W106-W108, 2009. Cited on pp. 114 and 128.

S. Bruckner, F. Hüffner, R. M. Karp, R. Shamir, and R. Sharan. Topology-free querying of protein interaction networks. *Journal of Computational Biology*, 17(3):237–252, 2010. Cited on pp. 98, 113, 114, 120, 124, 125, 128, 134, and 149.

D. Brügmann, C. Komusiewicz, and H. Moser. On generating triangle-free graphs. *Electronic Notes in Discrete Mathematics*, 32:51–58, 2009. Cited on p. iii.

L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. Cited on pp. 44, 46, 47, and 51.

L. Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3): 415–429, 2003. Cited on p. 162.

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 *Lecture Notes in Computer Science*, pages 60–71. Springer, 2010. Cited on p. 23.

Y. Cao, T. Jiang, and T. Girke. A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics*, 24(13):366–374, 2008. Cited on p. 154.

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. 19.

J. Chen and J. Meng. On parameterized intractability: Hardness and completeness. *The Computer Journal*, 51(1):39–59, 2008. Cited on p. 9.

J. Chen and J. Meng. A $2k$ kernel for the cluster editing problem. In *Proceedings of the 16th Annual International Computing and Combinatorics Conference (COCOON '10)*, volume 6196 of *Lecture Notes in Computer Science*, pages 459–468. Springer, 2010. Long version to appear in *Journal of Computer and System Sciences*, electronically available. Cited on pp. 17, 19, and 21.

Y. Chen, J. Flum, and M. Grohe. Machine-based methods in parameterized complexity theory. *Theoretical Computer Science*, 339(2-3):167–199, 2005. Cited on p. 132.

V. J. Cook, S. J. Sun, J. Tapia, S. Q. Muth, D. F. Argüello, B. L. Lewis, R. B. Rothenberg, P. D. McElroy, and the Network Analysis Project Team. Transmission network analysis in tuberculosis contact investigations. *Journal of Infectious Diseases*, 196:1517–1527, 2007. Cited on p. 44.

P. Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory of Computing Systems*, 46(2):261–283, 2010. Cited on p. 19.

T. J. P. van Dam and B. Snel. Protein complex evolution does not involve extensive network rewiring. *PLoS Computuational Biology*, 4(7):e1000132, 2008. Cited on pp. 95, 101, and 135.

F. K. H. A. 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 *Lecture Notes in Computer Science*, pages 13–24. Springer, 2006. Cited on pp. 17 and 19.

H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC '10)*, pages 251–260. ACM, 2010. Cited on p. 162.

M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and ids. volume 5555 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2009. Cited on pp. 10, 91, and 162.

R. Dondi, G. Fertin, and S. Vialette. Weak pattern matching in colored graphs: Minimizing the number of connected components. In *Proc. 10th ICTCS*, volume 4596 of *WSPC*, pages 27–38. World Scientific, 2007. Cited on p. 130.

R. Dondi, G. Fertin, and S. Vialette. Complexity issues in vertex-colored graph pattern matching. *Journal of Discrete Algorithms*, 9(1):82–99, 2011. Cited on pp. 113, 114, and 118.

F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010. Cited on pp. 74 and 91.

M. Dörnfelder, J. Guo, C. Komusiewicz, and M. Weller. On the parameterized complexity of consensus clustering. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC '11)*, 2011. To appear. Cited on p. 90.

B. Dost, T. Shlomi, N. Gupta, E. Ruppin, V. Bafna, and R. Sharan. QNet: A tool for querying protein interaction networks. *Journal of Computational Biology*, 15(7):913–925, 2008. Cited on pp. 97, 98, 135, and 137.

R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995a. Cited on p. 9.

R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1&2):109–131, 1995b. Cited on pp. 9 and 106.

R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. Cited on pp. 3, 8, 9, 10, 106, 120, and 131.

R. G. Downey and D. M. Thilikos. Confronting intractability via parameters. Technical report, 2011. Cited on p. 9.

D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC '10), Part 1*, volume 6506 of *Lecture Notes in Computer Science*, pages 403–414. Springer, 2010. Cited on p. 104.

P. Erdös and A. Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Academiae Scientiarum Hungaricae*, 17:61–99, 1966. Cited on p. 103.

I. Fagnot, G. Lelandais, and S. Vialette. Bounded list injective homomorphism for comparative analysis of protein-protein interaction graphs. *Journal of Discrete Algorithms*, 6(2):178–191, 2008. Cited on pp. 135, 136, 137, 138, 146, 148, and 149.

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 *Lecture Notes in Computer Science*, pages 276–277. Springer, 2006. Cited on p. 10.

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 *Lecture Notes in Computer Science*, pages 2–10. Springer, 2009. Cited on p. 10.

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 *Lecture Notes in Computer Science*, pages 312–321. Springer, 2007. Cited on pp. 17 and 18.

M. R. Fellows, D. Hermelin, F. A. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009. Cited on p. 60.

M. R. Fellows, G. Fertin, D. Hermelin, and S. Vialette. Upper and lower bounds for finding connected motifs in vertex-colored graphs. *Journal of Computer and System Sciences*, 77(4): 799–811, 2011a. Cited on pp. 111, 113, and 116.

M. R. Fellows, J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011b. Cited on pp. iii and 19.

X. Z. Fern and C. E. Brodley. Solving cluster ensemble problems by bipartite graph partitioning. In *Proceedings of the 21st International Conference on Machine Learning (ICML '04)*, volume 69 of *ACM International Conference Proceeding Series*, pages 36–. ACM, 2004. Cited on p. 72.

H. Fernau, F. V. Fomin, D. Lokshtanov, D. Raible, S. Saurabh, and Y. Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS '09)*, volume 3 of *LIPIcs*, pages 421–432. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2009. Cited on pp. 75, 91, and 158.

G. Fertin, R. Rizzi, and S. Vialette. Finding occurrences of protein complexes in protein-protein interaction graphs. *Journal of Discrete Algorithms*, 7(1):90–101, 2009. Cited on p. 137.

V. Filkov and S. Skiena. Integrating microarray data by consensus clustering. *International Journal on Artificial Intelligence Tools*, 13(4):863–880, 2004. Cited on pp. 71 and 72.

J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. Cited on pp. 3 and 8.

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. 10, 91, and 162.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL http://archive.ics.uci.edu/ml. Cited on p. 91.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. Cited on pp. 3, 8, and 95.

A. Goder and V. Filkov. Consensus clustering algorithms: Comparison and refinement. In *Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX '08)*, pages 109–117. SIAM, 2008. Cited on pp. 69, 71, and 72.

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. 17, 18, 21, and 45.

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, 1973. Cited on p. 12.

M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45:59–96, 1989. Cited on p. 19.

S. Guillemot and F. Sikora. Finding and counting vertex-colored subtrees. In *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS '10)*, volume 6281 of *Lecture Notes in Computer Science*, pages 405–416. Springer, 2010. Cited on pp. 113 and 114.

J. Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8–10):718–726, 2009. Cited on pp. 17, 18, 21, and 23.

J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007. Cited on p. 10.

J. Guo, F. Hüffner, and R. Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2004. Cited on p. 162.

J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of vertex cover variants. *Theory of Computing Systems*, 41(3):501–520, 2007. Cited on pp. 132 and 133.

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 *Lecture Notes in Computer Science*. Springer, 2008. To appear. Cited on p. iii.

J. Guo, I. A. Kanj, C. Komusiewicz, and J. Uhlmann. Editing graphs into disjoint unions of dense clusters. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC '09)*, volume 5878 of *Lecture Notes in Computer Science*, pages 583–593. Springer, 2009a. Cited on pp. iv, 45, 55, and 68.

J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. A more relaxed model for graph-based data clustering: s-plex editing. In *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management (AAIM '09)*, volume 5564 of *Lecture Notes in Computer Science*, pages 226–239. Springer, 2009b. Cited on p. iv.

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)*. AAAI Press, 2010a. Cited on pp. iii and 19.

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, 2010b. Cited on pp. iv, 47, and 55.

J. Guo, I. A. Kanj, C. Komusiewicz, and J. Uhlmann. Editing graphs into disjoint unions of dense clusters. *Algorithmica*, 2011. To appear, electronically available. Cited on p. iv.

F. Harary. The maximum connectivity of a graph. *Proceedings of the National Academy of Science of the United States of America*, 48(7):1142–1146, 1962. Cited on p. 61.

E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4–6):175–181, 2000. Cited on pp. 17 and 19.

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), Revised Papers*, volume 6410 of *Lecture Notes in Computer Science*, pages 171–183, 2010. Cited on p. 20.

J. E. Hopcroft and R. M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. In *Conference Record of the Twelfth Annual Symposium on Switching and Automata Theory (FOCS '71)*, pages 122–125. IEEE, 1971. Cited on p. 102.

F. Hüffner, S. Wernicke, and T. Zichner. FASPAD: fast signaling pathway detection.

Bioinformatics, 23(13):1708–1709, 2007. Cited on pp. 4, 98, and 135.

F. Hüffner, S. Wernicke, and T. Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 52(2):114–132, 2008. Cited on pp. 98 and 137.

F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Isolation concepts for clique enumeration: Comparison and computational experiments. *Theoretical Computer Science*, 410(52):5384–5397, 2009. Cited on p. iii.

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 p. iii.

R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. Cited on p. 23.

K. Inoue, W. Li, and H. Kurata. Diffusion model based spectral clustering for protein-protein interaction networks. *PloS one*, 5(9), 2010. Cited on pp. 17 and 42.

H. Jiang and B. Zhu. Weak kernels. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:5, 2010. Cited on pp. 74 and 158.

M. Karpinski and W. Schudy. Faster algorithms for feedback arc set tournament, Kemeny rank aggregation and betweenness tournament. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC '10), Part 1*, volume 6506 of *Lecture Notes in Computer Science*, pages 3–14, 2010. Cited on p. 39.

B. P. Kelley, R. Sharan, , R. M. Karp, T. Sittler, D. E. Root, B. R. Stockwell, and T. Ideker. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proceedings of the National Academy of Sciences of the United States of America*, 100(20): 11394–11399, 2003. Cited on p. 98.

C. Komusiewicz and J. Uhlmann. A cubic-vertex kernel for flip-consensus tree. In *Proceedings of the 28th Foundations of Software Technology and Theoretical Computer Science Conference (FSTTCS '08)*, volume 08004 of *Dagstuhl Seminar Proceedings*, pages 280–291. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2008. Cited on p. iii.

C. Komusiewicz and J. Uhlmann. Alternative parameterizations for cluster editing. In *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*, volume 6543 of *Lecture Notes in Computer Science*, pages 344–355. Springer, 2011. Cited on pp. iv and 22.

C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38-40):3640–3654, 2009. Cited on pp. iii and 104.

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. iii and 162.

S. Kosub. Local density. In *Network Analysis*, volume 3418 of *Lecture Notes in Computer Science*, pages 112–142. Springer, 2004. Cited on p. 44.

N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, S. Pu, N. Datta, A. P. Tikuisis, T. Punna, J. M. Peregrín-Alvarez, M. Shales, X. Zhang, M. Davey, M. D. Robinson, A. Paccanaro, J. E. Bray, A. Sheung, B. Beattie, D. P. Richards, V. Canadien, A. Lalev, F. Mena, P. Wong, A. Starostine, M. M. Canete, J. Vlasblom, S. Wu, C. Orsi, S. R. Collins, S. Chandran, R. Haw, J. J. Rilstone, K. Gandi, N. J. Thompson, G. Musso, P. S. Onge, S. Ghanny, M. H. Lam, G. Butland, A. M. Altaf-Ul, S. Kanaya, A. Shilatifard, E. O'Shea, J. S. Weissman, C. J. Ingles, T. R. Hughes, J. Parkinson, M. Gerstein, S. J. Wodak, A. Emili, and J. F. Greenblatt. Global

landscape of protein complexes in the yeast saccharomyces cerevisiae. *Nature*, 440(7084): 637–643, 2006. Cited on p. 17.

M. Křivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986. Cited on pp. 18, 19, 23, 24, and 72.

V. Lacroix, C. G. Fernandes, and M.-F. Sagot. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):360–368, 2006. Cited on pp. 98, 111, 112, 113, 130, and 134.

J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. Cited on p. 13.

D. Lokshtanov. *New Methods in Parameterized Algorithms and Complexity*. PhD thesis, Universitetet i Bergen, Bergen, Norway, 2009. Cited on pp. 75, 91, and 158.

D. Lokshtanov and D. Marx. Clustering with local restrictions. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP '11), Part 1*, volume 6755 of *Lecture Notes in Computer Science*, pages 785–797. Springer, 2011. Cited on p. 20.

S. Lu, F. Zhang, J. Chen, and S.-H. Sze. Finding pathway structures in protein interaction networks. *Algorithmica*, 48(8):363–374, 2007. Cited on p. 134.

D. Marx and I. Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC '11)*, pages 469–478. ACM, 2011. Cited on p. 19.

M. Meilă. Comparing clusterings: an axiomatic view. In *Proceedings of the 22nd International Conference on Machine Learning (ICML '05)*, volume 119 of *ACM International Conference Proceeding Series*, pages 577–584. ACM, 2005. Cited on p. 69.

N. Memon, K. C. Kristoffersen, D. L. Hicks, and H. L. Larsen. Detecting critical regions in covert networks: A case study of 9/11 terrorists network. In *Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES '07)*, pages 861–870. IEEE Computer Society, 2007. Cited on p. 44.

S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (FOCS '80)*. IEEE, 1980. Cited on p. 33.

Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2004. Cited on p. 3.

S. Monti, P. Tamayo, J. P. Mesirov, and T. R. Golub. Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. *Machine Learning*, 52(1–2):91–118, 2003. Cited on p. 69.

R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. Cited on pp. 3, 8, and 162.

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 *LIPIcs*, pages 17–32. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010. Cited on pp. 11 and 162.

R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73(3–4):125–129, 2000. Cited on p. 55.

M. Nikolski and D. J. Sherman. Family relationships: should consensus reign? - consensus clustering for protein families. *Bioinformatics*, 23(2):71–76, 2007. Cited on p. 69.

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 p. 4.

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 p. 18.

N. Przulj, D. A. Wigle, and I. Jurisica. Functional topology in a network of protein interactions. *Bioinformatics*, 20(3):340–348, 2004. Cited on pp. 4 and 17.

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 International Conference on Computational Systems Bioinformatics (CSB '07)*, volume 6 of *Computational Systems Bioinformatics*, pages 391–401. Imperial College Press, 2007. Cited on pp. 18 and 42.

A. Schäfer, C. Komusiewicz, H. Moser, and R. Niedermeier. Parameterized computational complexity of finding small-diameter subgraphs. *Optimization Letters*, 2011. To appear, electronically available. Cited on pp. iii, 75, and 91.

J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *Journal of Computational Biology*, 13(2):133–144, 2006. Cited on pp. 4, 135, and 137.

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. 44.

P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. Cited on p. 74.

SGD project. Saccharomyces genome database. http://downloads.yeastgenome.org/ (3/20/2010). Cited on pp. 115, 128, and 152.

R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1–2):173–182, 2004. Cited on pp. 18, 23, 24, and 45.

R. Sharan and T. Ideker. Modeling cellular machinery through biological network comparison. *Nature Biotechnology*, 24:427–433, April 2006. Cited on pp. 95, 97, and 135.

R. Sharan, T. Ideker, B. P. Kelley, R. Shamir, and R. M. Karp. Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. *Journal of Computational Biology*, 12(6):835–846, 2005. Cited on pp. 98, 101, and 135.

R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Molecular Systems Biology*, 3:88, 2007. Cited on pp. 4 and 17.

T. Shlomi, D. Segal, E. Ruppin, and R. Sharan. QPath: a method for querying pathways in a protein–protein interaction network. *BMC Bioinformatics*, 7:199, 2006. Cited on pp. 4, 95, 97, and 98.

A. Strehl and J. Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002. Cited on p. 72.

R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2): 146–160, 1972. Cited on p. 51.

E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006. Cited on p. 103.

S. Tweedie, M. Ashburner, K. Falls, P. Leyland, P. McQuilton, S. Marygold, G. H. Millburn, D. Osumi-Sutherland, A. Schroeder, R. Seal, and H. Zhang. Flybase: enhancing *Drosophila* Gene Ontology annotations. *Nucleic Acids Research*, 37(Database-Issue):555–559, 2009. (3/22/2010). Cited on pp. 115, 128, and 152.

J. Uhlmann. *Multivariate Algorithmics in Biological Data Analysis*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 2011. Cited on p. 23.

J. M. M. van Rooij, M. E. van Kooten Niekerk, and H. L. Bodlaender. Partition into triangles on bounded degree graphs. In *Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*, volume 6543 of *Lecture Notes in Computer Science*, pages 558–569. Springer, 2011. Cited on pp. 29 and 30.

V. V. Vazirani. *Approximation Algorithms*. Springer, 2001. Cited on pp. 3 and 8.

Y. Wakabayashi. The complexity of computing medians of relations. *Resenhas*, 3(3):323–350, 1998. Cited on p. 72.

M. Weller, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. On making directed graphs transitive. *Journal of Computer and System Sciences*, 2011. To appear, electronically available. Cited on pp. iii and 25.

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 p. 42.

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. 18.

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. 18.

G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–208. Springer, 2003. Cited on p. 23.

H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006. Cited on pp. 43, 105, and 137.

D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007. Cited on p. 102.

A. van Zuylen and D. P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research*, 34(3):594–620, 2009. Cited on pp. 19, 72, and 91.