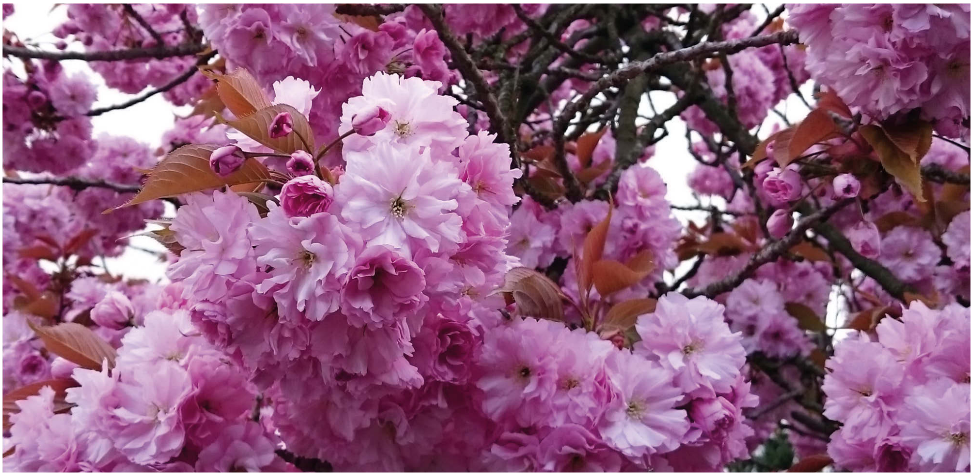
Foundations of computing Volume 8

Manuel Sorge

Be sparse! Be dense! Be robust!

Elements of parameterized algorithmics



Universitätsverlag der TU Berlin

Technische Universität Berlin

Manuel Sorge

Be sparse! Be dense! Be robust!

Elements of parameterized algorithmics

Die Schriftenreihe *Foundations of computing* der Technischen Universität Berlin wird herausgegeben von:
Prof. Dr. Stephan Kreutzer,
Prof. Dr. Uwe Nestmann,
Prof. Dr. Rolf Niedermeier.

Foundations of computing | 8

Manuel Sorge

Be sparse! Be dense! Be robust!
Elements of parameterized algorithmics

Universitätsverlag der TU Berlin

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

Universitätsverlag der TU Berlin, 2017
http://www.verlag.tu-berlin.de

Fasanenstr. 88, 10623 Berlin
Tel.: +49 (0)30 314 76131 / Fax: -76133
E-Mail: publikationen@ub.tu-berlin.de

Zugl.: Berlin, Techn. Univ., Diss., 2016
1. Gutachter: Prof. Dr. Rolf Niedermeier
2. Gutachter: Prof. Dr. Hans L. Bodlaender
3. Gutachter: Dr. Dan Hermelin
Die Arbeit wurde am 23. März 2016 an der Fakultät IV unter Vorsitz von Prof. Dr. Stefan Kreutzer erfolgreich verteidigt.

Druck: docupoint GmbH
Satz/Layout: Manuel Sorge

Umschlagfoto:
Manuel Sorge, 2016
CC BY 4.0 | http://creativecommons.org/licenses/by/4.0

ISBN 978-3-7983-2885-3 (print)
ISBN 978-3-7983-2886-0 (online)

ISSN 2199-5249 (print)
ISSN 2199-5257 (online)

Zugleich online veröffentlicht auf dem institutionellen Repositorium der Technischen Universität Berlin:
DOI 10.14279/depositonce-5635
http://dx.doi.org/10.14279/depositonce-5635

Zusammenfassung

In dieser Dissertation untersuchen wir die Berechnungskomplexität von fünf NP-vollständigen Graphproblemen. Es wird weithin angenommen, dass NP-vollständige Probleme im Allgemeinen nicht effizient gelöst werden können, das heißt, dass sie keine Polynomialzeitalgorithmen erlauben. Diese Annahme basiert auf vielen bisher nicht erfolgreichen Versuchen das Gegenteil zu beweisen. Aus diesem Grund versuchen wir Eigenschaften der Eingabe herauszuarbeiten, die das betrachtete Problem handhabbar oder unhandhabbar machen. Solche Eigenschaften messen wir mittels *Parametern*, das heißt, Abbildungen, die jeder möglichen Eingabe eine natürliche Zahl zuordnen. Für einen gegebenen Parameter κ versuchen wir dann *Fixed-Parameter Algorithmen* zu entwerfen, also Algorithmen, die auf Eingabe q eine obere Laufzeitschranke von $\phi(\kappa) \cdot |q|^c$ erlauben, wobei ϕ eine berechenbare Funktion ist, $|q|$ die Länge der Eingabe, und c eine Konstante. Natürlich ist c dabei eine möglichst kleine einstellige Zahl und ϕ eine möglichst schwach wachsende Funktion, die allerdings exponentiell sein muss.

In den Graphproblemen, die wir in dieser Dissertation studieren, repräsentiert unsere Eingabe eine Situation in der wir einen Lösungsgraph finden sollen. Zusätzlich sollen die Lösungsgraphen bestimmte problemspezifische Eigenschaften haben. Wir betrachten drei Varianten dieser Eigenschaften:

Zunächst suchen wir einen Graphen, der *sparse* sein soll. Das heißt, dass er wenige Kanten enthalten soll.

Dann suchen wir einen Graphen, der *dense* sein soll. Das heißt, dass er viele Kanten enthalten soll.

Zuletzt suchen wir einen Graphen, der *robust* sein soll. Das heißt, dass er eine gute Lösung bleiben soll, selbst wenn er einige kleine Modifikationen durchmacht.

Be sparse! In diesem Teil der Arbeit analysieren wir zwei ähnliche Probleme. In beiden ist die Eingabe ein Hypergraph $\mathcal{H}$, bestehend aus einer Knotenmenge V und einer Familie $\mathcal{E}$ von Teilmengen von V, genannt Hyperkanten. Die Aufgabe ist

einen *Support* für $\mathcal{H}$ zu finden, einen Graphen G, sodass für jede Hyperkante $W \in \mathcal{E}$ der induzierte Teilgraph $G[W]$ verbunden ist.

Motiviert durch Anwendungen im Netzwerkdesign betrachten wir SUBSET INTERCONNECTION DESIGN, worin wir eine natürliche Zahl f als zusätzliche Eingabe bekommen, und der Support höchstens $|V| + f - 1$ Kanten enthalten soll. Wir zeigen, dass SUBSET INTERCONNECTION DESIGN einen Fixed-Parameter Algorithmus in Hinsicht auf die Zahl der Hyperkanten im Eingabegraph erlaubt, und einen Fixed-Parameter Algorithmus in Hinsicht auf $f+d$, wobei d die Größe einer größten Hyperkante ist.

Motiviert durch eine Anwendung in der Hypergraphvisualisierung studieren wir r-OUTERPLANAR SUPPORT, worin der Support für $\mathcal{H}$ r-outerplanar sein soll, das bedeutet, er soll eine kantenkreuzungsfreie Einbettung in die Ebene erlauben mit höchstens r Schichten. Wir zeigen, dass r-OUTERPLANAR SUPPORT einen Fixed-Parameter Algorithmus in Hinsicht auf $m + r$ zulässt, wobei m die Anzahl der Hyperkanten im Eingabehypergraphen $\mathcal{H}$ ist.

Be dense! In diesem Teil der Arbeit studieren wir zwei Probleme, die durch Community Detection in sozialen Netzwerken motiviert sind. Dabei ist die Eingabe ein Graph G und eine natürliche Zahl k. Wir suchen einen Teilgraphen G' von G, der (genau) k Knoten enthält und dabei eine von zwei mathematisch präzisen Definitionen davon, *dense* zu sein, aufweist.

In μ-CLIQUE, $0 < \mu \leq 1$, soll der gesuchte Teilgraph G' mindestens $\mu\binom{k}{2}$ Kanten enthalten. Wir studieren die Berechnungskomplexität von μ-CLIQUE in Hinsicht auf drei Parameter des Eingabegraphen G: dem maximalen Knotengrad Δ, dem h-Index h, und der Degeneracy d. Es gilt $\Delta \geq h \geq d$ für jeden Graphen und sowohl h als auch d nehmen kleine Werte in Graphen an, die aus sozialen Netzwerken abgeleitet sind. Für Δ und h erhalten wir Fixed-Parameter Algorithmen für μ-CLIQUE und wir zeigen, dass für $d + k$ wahrscheinlich kein Fixed-Parameter Algorithmus existiert. Unsere positiven algorithmischen Resultate erhalten wir indem wir ein allgemeines Framework zum Optimieren von Zielfunktionen über Teilgraphen mit k Knoten entwickeln.

In HIGHLY CONNECTED SUBGRAPH soll in dem gesuchten Teilgraphen G' (mit k Knoten) jeder Knoten Knotengrad mindestens $\lfloor k/2 \rfloor + 1$ haben. Wir analysieren einen Teil der sogenannten Parameter Ecology für HIGHLY CONNECTED SUBGRAPH. Das heißt, wir navigieren im Raum der möglichen Parameter auf der Suche nach einem vernünftigen Trade-off zwischen kleinen Parameterwerten in der Praxis und effizienten oberen Laufzeitschranken. Die Highlights hier sind, dass es keine Algorithmen mit $2^{o(n)} \cdot \text{poly}(n)$-Laufzeit für HIGHLY CONNECTED SUBGRAPH gibt, es sei denn die Exponential Time Hypothesis stimmt nicht; ein Algorithmus mit $O(4^{\gamma} \cdot n^2)$-Laufzeit, wobei γ die Anzahl der Kanten ist, die aus dem Lösungsgraphen G' herausgehen; und ein Algorithmus mit $2^{O(\sqrt{\alpha}\log\alpha)} + \alpha^2 nm$-Laufzeit, wobei α die Anzahl der Kanten ist, die nicht in G' enthalten sind.

Be robust! In diesem Teil der Arbeit untersuchen wir das Problem VECTOR CONNECTIVITY, in dem wir einen Graphen G, ein Knotenlabeling $\lambda\colon V(G) \to \{1,\dots,d\}$, sowie eine natürliche Zahl k gegeben haben. Wir sollen eine Knotenteilmenge $S \subseteq V(G)$ der Größe höchstens k finden, sodass jeder Knoten $v \in V(G) \setminus S$ mindestens $\lambda(v)$ knotendisjunkte Pfade von v nach S in G hat. Solch eine Menge S ist beispielsweise nützlich, wenn wir Server in einem Netzwerk platzieren sollen, sodass Robustness-of-Service-Bedingungen erfüllt sind. Wir zeigen, dass VECTOR CONNECTIVITY einen randomisierten Fixed-Parameter Algorithmus in Hinsicht auf k zulässt, keine polynomielle Kernelisierung in Hinsicht auf $k + d$ zulässt, aber wenn dagegen d als eine Konstante betrachtet wird, eine Kernelisierung mit $O(k)$ übrigbleibenden Knoten erlaubt.

Abstract

In this thesis we study the computational complexity of five NP-complete graph problems. It is widely accepted that, in general, NP-complete problems cannot be solved efficiently, that is, in polynomial time, due to many unsuccessful attempts to prove the contrary. Hence, we aim to identify properties of the inputs other than their length, that make the problem tractable or intractable. We measure these properties via *parameters*, mappings that assign to each input a nonnegative integer. For a given parameter κ, we then attempt to design *fixed-parameter algorithms*, algorithms that on input q have running time bounded from above by $\phi(\kappa(q)) \cdot |q|^c$, where ϕ is a computable function, $|q|$ is the length of q, and c is a constant. Of course, it is prefereable that c is a small single digit number and that ϕ grows as slow as possible, although it has to be exponential.

In each of the graph problems treated in this thesis, our input represents the setting in which we shall find a solution graph. In addition, the solution graphs shall have a certain property specific to our five graph problems. This property comes in three flavors.

First, we look for a graph that shall *be sparse!* That is, it shall contain few edges.

Second, we look for a graph that shall *be dense!* That is, it shall contain many edges.

Third, we look for a graph that shall *be robust!* That is, it shall remain a good solution, even when it suffers several small modifications.

Be sparse! In this part of the thesis, we analyze two similar problems. The input for both of them is a hypergraph $\mathcal{H}$, which consists of a vertex set V and a family $\mathcal{E}$ of subsets of V, called hyperedges. The task is to find a *support* for $\mathcal{H}$, a graph G such that for each hyperedge $W \in \mathcal{E}$ we have that $G[W]$ is connected. Motivated by applications in network design, we study Subset Interconnection Design, where we additionally get an integer f, and the support shall contain at most $|V| + f - 1$ edges. We show that Subset Interconnection Design admits a fixed-parameter algorithm with respect to the number of hyperedges in the input hypergraph, and

a fixed-parameter algorithm with respect to $f + d$, where d is the size of a largest hyperedge.

Motivated by an application in hypergraph visualization, we study r-OUTERPLANAR SUPPORT where the support for $\mathscr{H}$ shall be r-outerplanar, that is, admit a edge-crossing free embedding in the plane with at most r layers. We show that r-OUTERPLANAR SUPPORT admits a fixed-parameter algorithm with respect to $m+r$, where m is the number of hyperedges in the input hypergraph $\mathscr{H}$.

Be dense! In this part of the thesis, we study two problems motivated by community detection in social networks. Herein, the input is a graph G and an integer k. We look for a subgraph G' of G containing (exactly) k vertices which adheres to one of two mathematically precise definitions of being dense.

In μ-CLIQUE, $0 < \mu \leq 1$, the sought k-vertex subgraph G' should contain at least $\mu\binom{k}{2}$ edges. We study the complexity of μ-CLIQUE with respect to three parameters of the input graph G: the maximum vertex degree Δ, h-index h, and degeneracy d. We have $\Delta \geq h \geq d$ in every graph and h as well as d assume small values in graphs derived from social networks. For Δ and for h, respectively, we obtain fixed-parameter algorithms for μ-CLIQUE and we show that for $d + k$ a fixed-parameter algorithm is unlikely to exist. We prove the positive algorithmic results via developing a general framework for optimizing objective functions over k-vertex subgraphs.

In HIGHLY CONNECTED SUBGRAPH we look for a k-vertex subgraph G' in which each vertex shall have degree at least $\lfloor k/2 \rfloor + 1$. We analyze a part of the so-called parameter ecology for HIGHLY CONNECTED SUBGRAPH, that is, we navigate the space of possible parameters in a quest to find a reasonable trade-off between small parameter values in practice and efficient running time guarantees. The highlights are that no $2^{o(n)} \cdot n^c$-time algorithms are possible for n-vertex input graphs unless the Exponential Time Hypothesis fails; that there is a $O(4^\gamma \cdot n^2)$-time algorithm for the number γ of edges outgoing from the solution G'; and we derive a $2^{O(\sqrt{\alpha}\log\alpha)} + \alpha^2 nm$-time algorithm for the number α of edges *not* in the solution.

Be robust! In this part of the thesis, we study the VECTOR CONNECTIVITY problem, where we are given a graph G, a vertex labeling $\lambda\colon V(G) \to \{1,\dots,d\}$, and an integer k. We are to find a vertex subset $S \subseteq V(G)$ of size at most k such that each vertex $v \in V(G) \setminus S$ has $\lambda(v)$ vertex-disjoint paths from v to S in G. Such a set S is useful when placing servers in a network to satisfy robustness-of-service demands. We prove that VECTOR CONNECTIVITY admits a randomized fixed-parameter algorithm with respect to k, that it does not allow a polynomial kernelization with respect to $k+d$ but that, if d is treated as a constant, then it allows a vertex-linear kernelization with respect to k.

Preface

In this thesis, I summarize some of my findings during my research from April 2011 to December 2015. During this time, I have been a member of the research group of Prof. Rolf Niedermeier at the Technische Universität Berlin. I gratefully acknowledge financial support by Deutsche Forschungsgemeinschaft, project DAPA (NI 369/12), from April 2011 up to the date of writing.

My research was almost exclusively in design and analysis of algorithms for NP-complete graph problems. Within this frame of reference, however, I have been interested in diverse topics and this is reflected in the present thesis. Most of my research has been in close collaboration with my coauthors; in the articles covered in this thesis, my coauthors were, in alphabetical order, René van Bevern, Jiehua Chen, Falk Hüffner, Iyad A. Kanj, Christian Komusiewicz, Stefan Kratsch, Rolf Niedermeier, Ondřej Suchý, and Mathias Weller.

The research on hypergraph supports (Chapters 3 and 4) was initiated during a retreat of the research group in Zinnowitz in March 2012. Falk Hüffner proposed to study minimum-edge hypergraph supports (Chapter 3). I immediately stepped into the trap of removing "superfluous" vertices contained in the same hyperedges (twins). We later discovered that the same happened to several researchers before. Back in Berlin, Jiehua Chen, and Christian Komusiewicz proved me wrong by a counterexample.

The kernelization with respect to the number of hyperedges was developed mainly by Christian Komusiewicz. The fixed-parameter tractability result with respect to the maximum hyperedge size and feedback edge number of the support were developed jointly by all coauthors and me. I was responsible for the detailed correctness proofs for the crucial Rules 3.7 and 3.8. Mathias Weller and I also tried to develop a more top-down approach, characterizing hypergraphs which require large feedback edge number in supports, but were not successful. This I would like to revisit in the future. The example of an instance with large feedback edge number is by Ondřej Suchý.

I presented our work on the 24th International Symposium on Algorithms and Computation (ISAAC '13) [Che+13] and prepared the journal version for *SIAM Journal on Discrete Mathematics* [Che+15].

The research on planar hypergraph supports (Chapter 4) was initiated together with René van Bevern, Christian Komusiewicz, and Rolf Niedermeier when Iyad A. Kanj came to our group for a stay from October 2014 to March 2015. Jointly, we developed a uniform fixed-parameter algorithm with respect to the number of hyperedges in the input graph by introducing the new concept of well-formed separator sequences, a sequence of separators which cut the graph by layers, each separator in the same way [Bev+15a]. I contributed several key ideas to the proof and wrote some of the main parts of the decomposition-based approach for finding long well-formed separator sequences. I later discovered that, for our purpose, so-called sphere-cut branch decompositions [Dor+10] can be used instead of the more constrained well-formed separator sequences. I wrote an alternative proof using these decompositions. A non-uniform tractability result (given in a generalized form in Section 2.1) was pointed out by a reviewer. I prepared the submission which appeared in *24th International Symposium on Graph Drawing and Network Visualization (GD '16)* [Bev+16].

The μ-CLIQUE problem (Chapter 6) was the first problem that I worked on for this thesis. I was nudged towards it by my colleague and then office mate Christian Komusiewicz. He asked me about the complexity of μ-CLIQUE on bounded-degree graphs. After toiling on this question for some time, I found a somewhat convoluted NP-completeness proof [Sor13], only to discover afterwards that there was a very elegant one in the literature [FS97]. I also observed the W[1]-hardness with respect to the degeneracy and solution size combined.

Together, we developed a fixed-parameter algorithm for μ-CLIQUE with respect to the maximum degree Δ. This algorithm has a subroutine which enumerates k-vertex connected subgraphs in bounded-degree graphs. We published our work in the *Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC '12)* [KS12], and I gave the presentation. After the symposium, I improved the enumeration routine so that the exponential part of the running time was asymptotically optimal. Afterwards, Mikko Koivisto (University of Helsinki) pointed out to us that the corresponding upper bound on the number of k-vertex connected subgraphs were already present in the literature [Bol06]. This meant that proving the running time came down to the simpler matter of showing that no subgraph is enumerated too often. Nevertheless, we think that the algorithmization of this bound is a useful black-box relevant to the community. Following

Christian's hints, I also worked out the details of balancing out the enumeration versus dynamic programming. Finally, I generalized the whole procedure to the subclass of FIXED-CARDINALITY OPTIMIZATION given in Chapter 6. A long version of our article appeared in *Discrete Applied Mathematics* [KS15a].

Christian and I also guided Kolja Stahl's bachelor thesis [Sta13] which consisted of an implementation of an algorithm based on the theoretical results in Chapter 6 and engineering the resulting implementation for efficiency. Subsequently, the three of us further developed the algorithm and I presented the results at the *14th International Symposium on Experimental Algorithms (SEA '15)* [KSS15].

Christian Komusiewicz also relayed the HIGHLY CONNECTED SUBGRAPH problem to me (Chapter 7) as a simplified version of a problem which arised in his work with his coauthors on a related clustering algorithm [Hüf+14]. I proved that HIGHLY CONNECTED SUBGRAPH is unlikely to admit subexponential-time algorithms with respect to the number of vertices in the input graph. As a way to tackle practical instances, I proposed to analyze the edge isolation parameter, the number of edges outgoing from the solution. From the diagram of parameter relations also naturally arose the edge deletion parameter. I developed the framework of reduction rules that the fixed-parameter tractability results for both edge isolation and deletion parameters are based on. I also proved fixed-parameter tractability for both parameters; the single exponential running time for the edge isolation parameter is due to Falk Hüffner. It seemed natural to try and get a subexponential-time algorithm with respect to the edge deletion parameter. After some calculations I could get it to work, indeed. We published our results in the *Proceedings of the 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '15)* [HKS15] and I gave the presentation. For this thesis, I wanted to see how far we can push the simple algorithmic approach that I had for the edge deletion parameter, resulting in the improved running time given in Chapter 7. I also added simple fixed-parameter tractability results for the maximum degree and h-index parameters to show the similarities between μ-clique and highly connected subgraph problems with regard to maximum degree, h-index, and degeneracy.

Regarding VECTOR CONNECTIVITY (Chapter 8), Stefan Kratsch approached me in July 2014. He had designed two reduction rules for VECTOR CONNECTIVITY (Rules 8.1 and 8.2) and asked whether we can use them to find a problem kernel if the maximum demand is upper bounded by a constant. After some discussion, we concluded that this should be the case. I worked out the details for the missing Rule 8.3. Two referees for the resulting paper pointed out the result by Fomin et al. [Fom+13], leading to an existence result for the problem kernel. Another referee pointed out a simplification in the design of Rule 8.3 which Stefan and I incorporated when Stefan visited Berlin in September 2014. I was not satisfied with the way that we treated three distinct cases in the proof and subsequently unified them, resulting in the current version of Lemma 8.13. We published our results in the *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC '15)* [KS15b], and I gave the presentation. We received the Excellent Student Paper Award, and were invited to a special issue of the *Algorithmica* journal [KS16].

During the time that I worked on this thesis, I was also involved in several projects which I do not cover here, including Golomb ruler construction [Sor+14], vehicle routing with tour-length constraints [Sor+11; Sor+12], with capacity constraints [Bev+14b; BKS15], and with bounded-overlap constraints [Flu+15], board game design [DKS14], balanced graph partitioning [Bev+14a], Google Scholar H-index manipulation [Bev+15b], feature selection [Fro+16], the Information System on Graph Classes and their Inclusions[1], and I contributed to a survey about the complexity of arc routing problems [Bev+14c] with a chapter on variants of the CHINESE POSTMAN problem.

[1] www.graphclasses.org

Contents

Chapter 1

Introduction

In this work, we design and analyze algorithms for computationally hard graph problems. In most of the problems we study, we are given a graph, consisting of a set of vertices and a set of pairs of vertices, called edges. We search for another graph that fulfills certain constraints.

This work consists of three parts, and in each of them, we treat one type of graph problem. The constraints on the output graph in the three types of problems are, respectively:

Be sparse! Be dense! Be robust!

What does it mean for a graph to be sparse, dense, or robust? Roughly speaking, a graph is *sparse* if it contains few edges in comparison to the maximum number of edges that it could have. In contrast, a graph is *dense* if it has a large number of the edges it could have. A graph is *robust* with respect to a certain property if it keeps this property after several small modifications. There are numerous mathematical definitions for sparsity [NM12], density [BP13; PYB13; SB13], and robustness [BEN09]. Of course, it depends on the particular context in which we use our graph problem which of these definitions are meaningful.

Defining sparsity, density, and robustness is not the point of this work. Rather, we take several such notions from the literature and applications, and analyze our graph problems with respect to these. The notions that we study are often not very complicated; however, the problems arising from them are computationally hard.

How are sparsity, density, and robustness useful? As mentioned, this depends on the particular application. Thus, let us give a brief overview over the types of problems that we study.

In Part I (*Be sparse!*) we study the design of interconnection graphs, a problem arising in communication systems, for example. Herein, we are given a set of servers, and a family of subsets of these servers, each of which represents servers that want to communicate with each other without involving the remaining servers. Our goal is to design an interconnection network, that is, a graph that contains all servers as vertices and that interconnects each of the given subsets of the servers. The overall efficiency of the resulting communication network corresponds to the sparsity of the graph, which we hence want to maximize. This and related problems have been studied in various different contexts by different communities [JP87; DM88; CDS04; Cho+07; AAR10; Buc+11].

In Part II (*Be dense!*) we study the problem of finding cohesive subgroups in a given network, for example, we want to find communities in social networks. If we model individuals as vertices and the friendship relations as edges, then this problem corresponds to finding a dense subgraph of a given graph. Including one of Karp's 21 original NP-hard problems [Kar72] as a special case, this is one of the most studied problems in algorithmics.

In Part III (*Be robust!*) we study the problem of selecting nodes in a network as servers, so that each of a given set of clients, also nodes in the network, enjoys interruption-free service, even if a specified number of nodes in the network fail. In other words, we aim to find a robust subnetwork containing clients and servers and in which no client can be separated from the servers by deleting the specified number of nodes. The study of this problem started only recently [Bor+14; CMR15].

Finally, we mention that sparsity can also be a restriction on the *input* that is relevant to applications of the graph problems studied here and that sparsity in the input graph is often algorithmically useful.

Elements of Parameterized Algorithmics

Most of the problems that we study are computationally hard, that is, NP-hard in general. However, it is often the case that the problem instances constructed in the NP-hardness proof have certain properties, which instances from real-world applications often do not have. For instance, in the canonical NP-hardness proof for finding communities in a given social network (the problem from Part II), the social network that we construct is dense. Social networks are sparse in practice, however [ELS13]. Hence, NP-hardness is merely a starting point for classifying the computational complexity. To obtain results of relevance to the actual application, we need refined analyses.

What is parameterized algorithmics? In this work, we study the *parameterized complexity* of graph problems. Herein, our aim is to determine the influence of certain properties of the input on the computational complexity or the influence of restrictions on the output on the computational complexity. These properties and restrictions are made tangible by *parameters*, mappings that assign to each input an integer, the *parameter value*. For example, we can measure the sparsity of an input graph by the maximum (vertex) degree, the maximum over all vertices v of the number of edges incident with v. Clearly, if this parameter has a small value, then the input is sparse. The question that we pursue is then: does our problem become tractable if we assume that the input has a small parameter value?

The maximum degree is an example of a parameterization of the input. In this work, we will equally frequently use parameters of the output. For example, in the interconnection graph design problem in Part I, we use the so-called feedback edge number of the output graph, another measure of sparsity.

What are the elements of parameterized algorithmics? Parameterized complexity and, with it, parameterized algorithmics have been pioneered by Downey and Fellows [DF99]. Since then, numerous case studies, treating the parameterized complexity of a given problem, were performed, leading to more and more refined techniques for deriving algorithms and running-time lower bounds. We mention here just two simple but popular algorithmic techniques:

Search tree algorithms encode (partial) solutions as vertices in a rooted tree whose depth and maximum degree is bounded from above by a small function of the parameter value. This implies that the overall tree has size bounded from above by a function of the parameter value and that, if the parameter has a small value, then also the tree has a reasonably small size. The algorithm then traverses the tree in order to find a solution.

Data reduction is to remove irrelevant parts of the input or to replace parts of the input by smaller ones which change the solution in some known way. The goal of data reduction is to shrink the input instance efficiently prior to applying an algorithm that finds a solution, for example, prior to applying a search tree algorithm. Parameters give a way to prove guarantees on the effectiveness of data reduction; they make it possible to bound from above the size of the instance resulting from data reduction.

In this work, we add to the body of case studies for parameterized complexity by studying the three graph problems mentioned before. The techniques which we explore can be summarized as follows.

In Part I we use data reduction as our main tool. Owing to the vague constraints on the solution that the interconnection graph problem imposes, an important technique is to take an (unknown) optimal solution, and to rewrite it until it has a certain structure while remaining optimal. Based on this structure, we can then design data reduction rules.

In Part II we provide both upper and lower running time bounds for data-driven parameterizations. More specifically, we use parameters which have been verified to assume small values in practice and also consider the trade-off between small parameter values and useful running-time upper bounds for the corresponding fixed-parameter algorithms. We emphasize simple search tree algorithms, which can be implemented easily and allow for many heuristic tweaks.

In Part III we use separator-based techniques to design data reduction rules. Separators in a graph are vertex sets whose removal disconnects the graph. One of the reduction rules replaces a part of the graph with a smaller one, changing the solution in some known way. One of the main challenges here is to define what it means to change the solution in a known way.

Before going into more details, we describe our notation and give basic definitions in Section 1.1. A list of the problems referenced in this thesis is given in Appendix A.

1.1. Preliminaries

In this section we describe our notation used throughout this work and we give some basic definitions that we need. We assume that the reader is familiar with some basics of set theory, discrete mathematics, calculus, computational complexity, and analysis of algorithms.

Some general notation that we use throughout is as follows.

Lower-case letters denote elements of fundamental sets, for example integers ℓ, n, m or vertices u, v, w of a graph.

Upper-case letters denote sets of elements, for example, a set V of vertices.

Calligraphic letters denote families of sets or entities related to families of sets, for example, a partition $\mathcal{P}$ of the vertices of a graph, a hypergraph $\mathcal{H}$, or a matroid $\mathcal{M}$.

Blackboard bold letters denote universal sets, for example, the set $\mathbb{G}$ of finite graphs or the set $\mathbb{N}$ of nonnegative integers.

Fraktur letters denote geometric objects, for example, a sphere $\mathfrak{S}$ or a plane $\mathfrak{R}^2$.

Lower-case greek letters denote functions and relations, for example, an integer labeling λ of the vertices of a graph.

By $A \uplus B$ we denote the union of two disjoint sets A and B. For a family of sets $\mathcal{F}$, we write $\bigcup\mathcal{F}$ in place of $\bigcup_{S\in\mathcal{F}} S$. For equivalence relations ϱ over some set S we use $[v]_\varrho$ to denote the equivalence class of $v \in S$ in ϱ. For a set S and a positive integer $i \in \mathbb{N}$, by $\binom{S}{i}$ we denote the family of i-element subsets of S.

1.1.1. Graphs and hypergraphs

We now describe our notation for graphs and hypergraphs. Our graph notation is mostly canon and leans on the one used in the book by Diestel [Die10]. For hypergraphs, it is important to notice the difference between induced subhypergraphs and shrunken subhypergraphs, both of which are natural generalizations of induced subgraphs.

Graphs. A (undirected) *graph* G is a tuple (V, E) consisting of a *vertex set* V, also denoted $V(G)$, and an *edge set* E, also denoted $E(G)$. The edge set $E(G)$ consists of two-element subsets of V. The vertices u, v are also called the *ends* of an edge $\{u, v\}$. Unless stated otherwise, all graphs are undirected, and do not contain any self-loops or parallel edges. The *order* of a graph is number of its vertices. Where it is not ambiguous, we denote $n := |V(G)|$ and $m := |E(G)|$.

We say that an edge and a vertex are *incident* if the edge contains the vertex. Two vertices are called *adjacent* or *neighbors* if they are incident with the same edge. The *open neighborhood* $N_G(v)$, or simply *neighborhood*, of a vertex v contains all its neighbors in G. We omit the subscript G from $N_G(v)$ if it is not ambiguous. The *degree* of a vertex v is the cardinality $|N_G(v)|$ of its neighborhood. We sometimes extend the notion of neighborhood to sets of vertices, that is, for a vertex subset $S \subseteq V$ we denote by $N(S) := \bigcup_{v\in S} N(v) \setminus S$ the *neighborhood of S*. The *closed neighborhood $N_G[v]$* of a vertex v is $N_G(v) \cup \{v\}$.

A *subgraph* of a graph G is a graph G' such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. We say that G' is a *proper subgraph* if $V(G') \subsetneq V(G)$ or $E(G') \subsetneq E(G)$. For $W \subseteq V(G)$, the *subgraph of G induced by W* is a graph

$$(W, \{e \mid (e \subseteq W) \ \wedge \ e \in E(G)\})$$

which we also denote by $G[W]$. *Removing a vertex subset $S \subseteq V(G)$ from G* means to take $G[V(G) \setminus S]$. We also write $G - S := G[V(G) \setminus S]$ as a shorthand. *Removing an edge subset $F \subseteq E(G)$ from G* means to take the graph $G - F := (V(G), E(G) \setminus F)$.

A *walk* between two vertices u, v in graph G is an alternating sequence of vertices and edges that starts with u and ends with v such that consecutive elements in the sequence are incident with one another. Two vertices are *connected* if they have a walk between them. The *distance* of two connected vertices u, v is the length of a shortest path between u and v. A set of vertices is *connected* if the vertices in the set are pairwise connected. A graph G is *connected* if $V(G)$ is connected. A *connected component* C in a graph G is a connected subgraph of G which is not a proper subgraph of any other connected subgraph of G. In other words, $C = G[D]$ for a vertex subset $D \subseteq V(G)$ and either $D = V(G)$ or no vertex v of G can be added to D such that the subgraph induced by $D \cup \{v\}$ is connected. For notational convenience, we sometimes identify connected components with their vertex sets. A *separator* in a graph G is a vertex subset S such that $G - S$ has more connected components than G. A *cut* in G is an edge subset $F \subseteq E(G)$ such that $G - F$ has more connected components than G. A *bridge* is an edge e such that $\{e\}$ is also a cut.

The *complement* $\overline{G}$ of a graph G is the graph $\left(V(G), \binom{V(G)}{2} \setminus E(G)\right)$.

Special vertex subsets. Important special types of vertex subsets are as follows.

- An *independent set* in a graph is a vertex subset of pairwise nonadjacent vertices.
- A *vertex cover* in a graph G is a vertex subset S such that $V(G) \setminus S$ is an independent set. Equivalently, each edge in G has at least one end in S.

- A *dominating set* in a graph G is a vertex subset S such that each vertex in G is contained in S or adjacent to some vertex in S.

Special graphs. Important special types of graphs are as follows.

- A *cycle* is a connected graph in which each vertex has degree exactly two. The *length* of a cycle is its number of edges.
- A *path* is a connected graph in which exactly two vertices have degree one and the remaining vertices have degree two. We also call the vertices with degree one the *ends*, *starting points* or *endpoints* of the path. The *length* of a path is the number of its edges. We also say that a path is *between* its ends or that it *runs* from one end to the other.
- A *forest* is a graph that does not contain a cycle as a subgraph.
- A *tree* is a connected forest. In a tree, the vertices with degree one are called *leaves* and the remaining vertices *inner vertices*.
- A *star* is a tree with precisely one inner vertex.
- A *clique* or a *complete graph* is a graph in which all vertices are pairwise adjacent. We usually use K_n to denote an n-vertex clique.
- A *bipartite graph* G admits a bipartition (U, W) of its vertex set such that U and W are independent sets. Vertex sets U and W are also called the *partite sets* of G.

A *Hamiltonian cycle* in a graph G is subgraph of G that is a cycle and contains all vertices of G. A *Hamiltonian path* is defined analogously.

Boundaried graphs and gluing. For a nonnegative integer $b \in \mathbb{N}$, a *b-boundaried graph* is a tuple (G, B, β) where G is a graph, where $B \subseteq V(G)$ such that $|B| = b$, and where β is a bijection $\beta\colon B \to \{0, \ldots, b\}$. Vertex subset B is also called the *boundary* and β the *boundary labeling*. For ease of notation we also refer to (G, B, β) as the b-boundaried graph G with boundary B and boundary labeling β. For brevity, we also denote by *β-boundaried graph G* that b-boundaried graph G whose boundary is the domain of β and whose boundary labeling is β.

For a nonnegative integer b, we define the *gluing* operation $\circ_b$ as a mapping that maps two b-boundaried graphs to an ordinary graph as follows: Given two b-boundaried graphs G_1, G_2 with corresponding boundaries B_1, B_2 and boundary labelings β_1, β_2, to obtain the graph $G_1 \circ_b G_2$ take the disjoint union of G_1 and G_2, and identify each $v \in B_1$ with $\beta_2^{-1}(\beta_1(v)) \in B_2$. We omit the index b in $\circ_b$ if it is clear from the context.

Directed graphs. A *directed graph* D is a tuple (V, A) consisting of a *vertex set* V, also denoted $V(D)$, and an *arc set* A, also denoted $A(D)$. The arc set consists of ordered pairs of vertices in V. Many definitions for undirected graphs have straightforward analogs in directed graphs; the following are special, however. The *indegree* of a vertex $v \in V$ is the cardinality of $\{(u, v) \mid (u, v) \in A\}$ and its *outdegree* is the cardinality of $\{(v, u) \mid (v, u) \in A\}$. A vertex in a directed graph is a *source* if its outdegree is at least 1 and its indegree is 0. Conversely, a vertex is a *sink* if its indegree is at least 1 and its outdegree is 0.

Hypergraphs. A *hypergraph* $\mathcal{H}$ is a tuple $(V,\mathcal{E})$ consisting of a *vertex set* V, also denoted $V(\mathcal{H})$ and a *hyperedge set* $\mathcal{E}$, also denoted $\mathcal{E}(\mathcal{H})$. The hyperedge set $\mathcal{E}$ is a family of subsets of V, that is, $F \subseteq V$ for every hyperedge $F \in \mathcal{E}$. Where it is not ambiguous, we denote $n := |V|$ and $m := |\mathcal{E}|$. When specifying running times, we use $|\mathcal{H}|$ to denote $|V(\mathcal{H})| + \sum_{F \in \mathcal{E}(\mathcal{H})} |F|$. The *size* $|F|$ of a hyperedge F is the number of vertices in it. Unless stated otherwise, we assume that hypergraphs do not contain hyperedges of size at most one or multiple copies of the same hyperedge. (These do not play any role for the problems under consideration, and removing them can be done easily and efficiently.)

A vertex $v \in V$ and a hyperedge $F \in \mathcal{E}$ are *incident* with one another if $v \in F$. For a vertex $v \in V(\mathcal{H})$, we denote $\mathcal{E}_{\mathcal{H}}(v) := \{F \in \mathcal{H} \mid v \in F\}$. If it is not ambiguous, we omit the subscript $\mathcal{H}$ from $\mathcal{E}_{\mathcal{H}}$. A vertex u *covers* a vertex v if $\mathcal{E}(v) \subseteq \mathcal{E}(u)$. Two vertices $u, v \in V$ are *twins* if $\mathcal{E}(v) = \mathcal{E}(u)$. Clearly, the relation τ on V defined by $\forall u, v \in V \colon (u, v) \in \tau \Leftrightarrow \mathcal{E}(u) = \mathcal{E}(v)$ is an equivalence relation. The equivalence classes $[u]_\tau$, $u \in V$, are called *twin classes*.

The *subhypergraph induced by* $V' \subseteq V$ is the hypergraph $\mathcal{H}[V'] := (V',\mathcal{E}')$ where $\mathcal{E}' = \{F \subseteq V' \mid F \in \mathcal{E}\}$. *Removing a vertex subset* $S \subseteq V(\mathcal{H})$ from a hypergraph $\mathcal{H} = (V,\mathcal{E})$ results in the hypergraph $\mathcal{H} - S := (V \setminus S,\mathcal{E}')$ where $\mathcal{E}'$ is obtained from $\{F \setminus S \mid F \in \mathcal{E}\}$ by removing empty, singleton, and duplicate sets. For brevity, we also write $\mathcal{H} - v$ instead of $\mathcal{H} - \{v\}$. The *subhypergraph shrunken to* $V' \subseteq V$ is the hypergraph $\mathcal{H}|_{V'} := \mathcal{H} - (V \setminus V')$.

A *hyperwalk* between two vertices u and v is an alternating sequence of vertices and hyperedges starting in u and ending in v such that consecutive elements are incident with one another. A hypergraph is *connected* if there is a hyperwalk between each pair of vertices.

The *incidence graph* of a hypergraph $\mathcal{H}$ is a bipartite graph G with $V(G) = V(\mathcal{H}) \uplus \mathcal{E}(\mathcal{H})$, and $E(G) = \{\{v, F\} \mid (v \in V(\mathcal{H})) \wedge (F \in \mathcal{E}(\mathcal{H})) \wedge (v \in F)\}$. The *dual hypergraph* of a hypergraph $\mathcal{H}$ is the hypergraph $(\mathcal{E}(\mathcal{H}), \{\mathcal{E}(v) \mid v \in V(\mathcal{H})\})$.

1.1.2. Classical complexity

We now recall some basic notions from classical complexity theory and algorithmics and introduce some related notation.

Decision problems. Most of the computational problems treated in this work are formulated as decision problems, that is, for a fixed language, we are given a string and we are to decide whether this string is contained in that language.

More formally, let Σ be an alphabet, that is, any finite set. By Σ^* we denote the set of all finite strings, sequences of elements of the alphabet Σ. For a string $p \in \Sigma^*$ we denote by $|p|$ its *length*, that is, the number of elements in the string p. A *language* is a subset of Σ^*. A *decision problem* for a language $P \subseteq \Sigma^*$ asks whether $p \in P$ for a given string $p \in \Sigma^*$. For convenience, we denote a decision problem also by the corresponding language.

Usually, for p to be in P, it must fulfill certain implicit, easily-checkable well-formedness conditions, for example, that p encodes a tuple of a graph and an integer. If p fulfills the well-formedness conditions implicit in the considered decision problem P, then we say that p is an *instance* of P. If additionally $p \in P$, then we say that p is a *yes-instance* or, if it is clear that p is an instance of P, then we simply say that p is *yes* for the problem P. Conversely, an instance p of P for which $p \notin P$ is a *no-instance* or simply *no*.

P, NP, and reductions. We assume that the reader is familiar with the concept of (deterministic) Turing machines. Definitions can be found in the literature, see Papadimitriou [Pap94] and Arora and Barak [AB09], for example. The *time* needed by a Turing machine for a computation is the number of elementary computation steps it performs. A Turing machine *decides* a decision problem $P \subseteq \Sigma^*$ if on every input $p \in \Sigma^*$ it halts after a finite number of steps and, furthermore, its memory tape is empty after halting if and only if $p \in P$. The class P contains all decision problems that can be decided by a deterministic Turing machine within time that is bounded from above by a polynomial in the input length. The class NP contains all decision problems P with the following property. There is a Turing machine M and for each $p \in P$ there is a polynomial ϕ and a string $q(p) \in \Sigma^*$, such that $|q(p)| \leq \phi(|p|)$ and M decides $\{(p, q(p)) \mid p \in P\}$ in polynomial time.

It is generally believed that not all problems in NP can be decided in polynomial time by a Turing machine, that is, $\mathsf{P} \neq \mathsf{NP}$. As a proxy for studying this conjecture, Cook [Coo71] and Karp [Kar72] introduced the concept of NP-hardness. To define it, we need a notion of reduction. A *polynomial-time many-one reduction* from a decision problem P to a decision problem Q is a mapping $\varrho\colon \Sigma^* \to \Sigma^*$ such that for each $p \in \Sigma^*$ we have that $p \in P$ if and only if $\varrho(p) \in Q$ and such that there is a Turing machine that computes ϱ in polynomial time. A decision problem P is NP-*hard* if for every problem Q in NP there is a a polynomial-time many-one reduction from Q to P. Most of the problems studied here are NP-hard.

Algorithm analysis. The computational model that we use to analyze algorithms is based on a Random Access Machine. We give an informal description, for details see Papadimitriou [Pap94], for example. A *Random Access Machine* is a computer which has access to an array with unbounded number of entries, each of which is an arbitrarily large integer initialized to 0, and two special registers, the *accumulator* and the *program counter*. Both these special registers contain arbitrarily large integers and are initialized with 0. The Random Access Machine carries out a *program*, a sequence of instructions. In each step, the instruction with the index stored in the program counter is executed, and the program counter is incremented (except for some special instructions). Herein, an *instruction* in the program is one of the following:

- reading a bit of the (binary) input string,
- writing a bit to the output string,
- loading a given integer or an integer stored in the array into the accumulator,
- writing the content of the accumulator into some specified cell of the array,
- checking whether the accumulator contains a given integer or is less than a given integer,
- adding or subtracting a given integer to or from the accumulator,
- given an integer n, removing the last n bits from the integer stored in the accumulator,
- setting the program counter to a given integer, and
- stopping the computation.

To analyze the running time of a Random Access Machine we assume that each of the above instructions takes constant time. Note the absence of a multiplication instruction. Allowing constant-time multiplication would mean that the resulting computer can solve NP-hard problems in polynomial time [Sch79]. In contrast, a Random Access Machine as described above (without constant-time multiplication) can be simulated by a Turing machine such that the time needed by the Turing machine is upper bounded by a polynomial of the time needed by the Random Access Machine [Sch79].

We assume that, on top of the above instructions, more elaborate control structures, like benign if/else and for- and while-loops, are implemented as well as convenience features like support for variables etc., all with a time step overhead of at most a constant factor.

Our running times are given in the "big Oh" or Landau notation, see Cormen et al. [Cor+09], for example. For an integer n, we also use the term poly(n) for quantity x, which means that there exists a constant c such that $x \in \mathrm{O}(n^c)$.

1.1.3. Parameterized complexity

One of the most central conecepts in this thesis is parameterized complexity which was pioneered by Downey and Fellows [DF99]. More recent textbooks include Flum and Grohe [FG06], Niedermeier [Nie06], Downey and Fellows [DF13], and Cygan et al. [Cyg+15].

Basics

Let Σ be an alphabet. A *parameter* is a mapping $\Sigma^* \to \mathbb{N}$. For a string q, $\kappa(q)$ is the *parameter value*. A *parameterized problem* is a tuple (Q, κ) of a language Q over some alphabet Σ and a parameter κ. We say that an algorithm is a *fixed-parameter algorithm* with respect to a parameter κ if the algorithm has running time $\phi(\kappa(q)) \cdot$ poly$(|q|)$ where q is the input and ϕ is some computable function. We also call a running-time function ϕ as above a *fixed-parameter running-time function* with respect to κ. We usually use the term fixed-parameter algorithm in the context of a decision problem P and then we assume that the algorithm decides P. However, sometimes we also use a fixed-parameter algorithm to compute a given function.

We note that sometimes a parameter κ is stated explicitly in the instances of a problem. In such cases, we omit the reference to κ and replace it by the value as stated in the corresponding problem definition. For example, if we have a problem of finding a solution of size k, then we write "k" for the solution size parameter, that is, the mapping that takes an instance and extracts the value of k. Moreover, when specifying running times with respect to a parameter κ, we often replace $\kappa(q)$ by the referenced value if the instance q is clear from the context.

For $k \in \mathbb{N}$, the kth *slice* of a parameterized problem (Q, κ) is $\{q \in Q \mid \kappa(q) = k\}$. A parameterized problem (Q, κ) is *non-uniformly fixed-parameter tractable* if for every $k \in \mathbb{N}$ there is a fixed-parameter algorithm with respect to κ that decides the kth slice of (Q, κ). Problem (Q, κ) is *uniformly fixed-parameter tractable* if there is a fixed-parameter algorithm that decides Q with an arbitrary, possibly not computable, fixed-parameter running time function. Problem (Q,κ) is *strongly uniformly fixed-parameter tractable* if there is a fixed-parameter algorithm with respect to κ that decides Q. Unless stated otherwise, when writing *fixed-parameter tractable* we mean strongly uniformly fixed-parameter tractable.

The class FPT contains all (strongly uniformly) fixed-parameter tractable parameterized problems. The class XP contains all parameterized problems which can be solved in polynomial time for constant parameter values. That is, each parameterized problem (Q,κ) in XP admits an algorithm that decides Q within running time $\phi(\kappa(q)) \cdot |q|^{\phi(\kappa(q))}$ for each input $q \in \Sigma^*$, where ϕ is a computable function. We also call such an algorithm an XP-*algorithm* and such a running time an XP-*running time*.

Reductions and hardness. To differentiate between FPT and XP, Downey and Fellows [DF99] describe the following hardness theory. A *parameterized reduction* from (Q,κ) to (P,λ) is a mapping $\varrho\colon Q \to P$ such that

- for every $q \in \Sigma^*$ we have $q \in Q$ if and only if $\varrho(q) \in P$,
- there is a function ϕ such that for every $q \in \Sigma^*$ we have $\lambda(\varrho(q)) \leq \phi(\kappa(q))$, and
- there is a fixed-parameter algorithm with respect to κ that computes ϱ.

We also say that (Q,κ) is *parameterized reducible* to (P,λ).

Downey and Fellows [DF99] introduced a hierarchy of classes $\mathsf{W[t]}$, $0 < t \in \mathbb{N}$, of parameterized problems as follows

$$\mathsf{FPT} \subseteq \mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \ldots \subseteq \mathsf{XP}.$$

For this work, the precise definitions of all these classes are not important. For our purposes, it suffices to consider only $\mathsf{W[1]}$ and $\mathsf{W[2]}$, we define them below. A parameterized problem (Q,κ) is called $\mathsf{W[t]}$-*hard* if every problem in $\mathsf{W[t]}$ is parameterized reducible to (Q,κ). Each of the inclusions in the hierarchy above is thought to be proper. In particular, it is widely believed that no $\mathsf{W[1]}$-hard problem admits a fixed-parameter algorithm.

The classes $\mathsf{W[1]}$ and $\mathsf{W[2]}$ in the above hierarchy can be defined as follows. The class $\mathsf{W[1]}$ contains all parameterized problems which are parameterized reducible to the following problem parameterized by k [Fel+09].

Multicolored Clique
Input: A nonnegative integer k and an undirected graph G along with a partition of its vertex set into k independent sets.
Question: Does G have a k-vertex clique as a subgraph?

The class W[2] contains all parameterized problems which are parameterized reducible to the following problem parameterized by k.

Dominating Set
Input: An undirected graph G and an integer k.
Question: Is there a dominating set of size k in G?

Clearly, Multicolored Clique is W[1]-hard and Dominating Set is W[2]-hard. Moreover, W[1]- or W[2]-hardness for a given parameterized problem can be shown by a parameterized reduction from Multicolored Clique or Dominating Set, respectively.

Data reduction

A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction* [GN07; Bod09; Kra14]. The goal is to remove needless information from the input so to reduce its size or to obtain some desirable properties of the input. Such small or well-formed instances can then be exploited by algorithms that produce a solution: small size of the input implies a small search space of the solution algorithm, and similarly, well-formed instances may be easier to solve.

Data reduction is usually presented as a series of *reduction rules*. These are polynomial-time algorithms that take as input an instance of some decision problem and produce another instance of the same problem as output. A reduction rule is *correct* if for each input instance I, the corresponding output instance of the rule is a yes-instance if and only if I is a yes-instance. We call an instance I of a parameterized problem *reduced* with respect to a reduction rule if the reduction rule does not apply to I. That is, carrying out that reduction rule yields an unchanged instance.

The notion of problem kernels captures the idea of reduction rules with effectiveness guarantee. A *kernelization* or *problem kernel* for a parameterized problem (Q,κ) is a parameterized reduction ϱ from (Q,κ) to itself such that ϱ is computable in polynomial time and there is a function ϕ such that for every $q \in \Sigma^*$ we have $|\varrho(q)| \leq \phi(\kappa(q))$. We also call ϕ the *size* of ϱ. If ϕ is polynomial, then we also call ϱ a *polynomial kernelization* or *polynomial problem kernel*.

Lower bounds on problem kernel sizes. Small, that is, polynomial problem kernels are particularly desirable. However, some parameterized problems are unlikely to admit problem kernels of polynomial size. To show such results, Bodlaender et al. [Bod+09] introduced the composition technique which was later refined by Bodlaender, Jansen, and Kratsch [BJK14] in so-called cross-compositions. Both are generalizations of the notion of reductions that takes as input *multiple* instances. To define cross-compositions, we need the following notion. A *polynomial equivalence relation* is an equivalence relation ϱ that has the following two properties.

- There is an algorithm that, given two strings $p, q \in \Sigma^*$, decides whether p and q belong to the same equivalence class of ϱ in time polynomial in $|p| + |q|$.
- For every finite set $S \subseteq \Sigma^*$, the equivalence relation ϱ partitions the elements of S into a number of equivalence classes that is bounded from above by a polynomial in the length of the longest string in S.

Let Q be a language and ϱ a polynomial equivalence relation on Q. An *or-cross-composition* from Q into a parameterized problem (P, κ) is a polynomial-time algorithm that, given t instances $q_1, \ldots, q_t \in \Sigma^*$ of Q belonging to the same equivalence class of ϱ, computes an instance $p \in \Sigma^*$ such that

$$\kappa(p) \le \operatorname{poly}\left(\log t + \max_{i=1}^{t} |q_i|\right)$$

and $p \in P$ if and only if for some $i \in \{1, \ldots, t\}$ we have $q_i \in Q$. If there is such an or-cross-composition, we also say that Q *or-cross-composes* into (P, κ). The equivalence relation ϱ herein can be chosen at will and is used to simplify the task of developing cross-compositions. Bodlaender, Jansen, and Kratsch [BJK14] proved the following, using a result of Fortnow and Santhanam [FS11].

Theorem 1.1. Let $Q \subseteq \Sigma^*$ be an NP-hard language and (P, κ) be a parameterized problem. If Q or-cross-composes into (P, κ) and (P, κ) admits a polynomial problem kernel, then $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Yap [Yap83] showed that $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ implies that the polynomial hierarchy collapses to the third level. As this is widely assumed not to be the case, Theorem 1.1 and a cross-composition give good evidence that the problem at hand does not admit a polynomial problem kernel. The consequence $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ also follows when replacing or-cross-compositions by so-called and-compositions [Dru15; Del14]. However, we only need or-cross-compositions in this work.

We mention that the hardness result implied by Theorem 1.1 transfers within NP-complete problems via so-called polynomial-parameter transformations [BTY11], reductions that preserve the parameter polynomially. More formally, a *polynomial-parameter transformation* from a parameterized problem (Q,κ) to another parameterized problem (P,λ), both over an alphabet Σ, is a polynomial-time many-one reduction ϱ from Q to P such that there is a polynomial ϕ such that for every $q \in \Sigma^*$ we have $\lambda(\varrho(q)) \le \phi(\kappa(q))$. Using this definition, if Q is NP-hard and P is in NP and if (P,λ) admits a polynomial problem kernel, then also (Q,κ) does: Simply pipeline the polynomial-parameter transformation from (Q,κ) to (P,λ), the polynomial kernel for (P,λ), and the polynomial-time many-one reduction from P to Q.

A more general notion of data reduction. Turing kernelization is another way to formalize effective data reduction [Lok09; Sch+12; Bin+12]. For a data reduction-based algorithm to be efficient, it is arguably enough to produce a small number of small instances rather than just one instance as in the definition of polynomial problem kernels.

The small instances that are produced by the data reduction procedure are formalized as oracle questions, which the oracle answers in O(1) time for the sake of analyzing the data reduction algorithm. In practice, the oracle is replaced by a decision algorithm. Formally, let Q be a language and $s \in \mathbb{N}$ be a nonnegative integer. An *s-bounded oracle* for Q is an operation which determines in O(1) time whether $q \in Q$ for any string $q \in \Sigma^*$ such that $|q| \le s$. Let $\phi\colon \mathbb{N} \to \mathbb{N}$ be a function. A *Turing kernelization* or a *Turing kernel* for a parameterized problem (Q,κ) is a polynomial-time algorithm that on every input $q \in \Sigma^*$ has access to a $\phi(\kappa(q))$-bounded oracle for Q and decides whether $q \in Q$. The function ϕ is also called the *size* of the Turing kernelization.

Weller [Wel13] observed that certain special cases of Turing kernelizations can be excluded using cross-compositions under the assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. For the general case, however, we are not aware of a technique that would imply a similar collapse in classical complexity. Relatedly, in pursuit of a general hardness theory for (Turing) kernelization Hermelin et al. [Her+15] introduced a hierarchy of classes of parameterized problems which presumably do not admit polynomial-size (Turing) kernelizations.

Refined running time lower bounds

In this section we describe how we can exploit the Exponential Time Hypothesis for giving lower bounds on running times. The lower bounds which we achieve are refinements of the gaps polynomial time versus exponential time and fixed-parameter running time versus XP running time.

Much research was dedicated to improving the running time of algorithms for NP-hard problems [Woe03; FK10]. Incremental improvements of exponential running times have been achieved for many problems, though, for several of them indefinitely continuing improvements in the base of the exponential running time would be a major breakthrough. An example of such a problem is the following.

> k-CNF-SATISFIABILITY
> *Input:* A boolean formula ϕ in conjunctive normal form with at most k literals in each clause.
> *Question:* Is there a truth assignment of the variables that makes ϕ true?

Impagliazzo and Paturi [IP01] introduced the following hypothesis that captures the difficulty when trying to improve the trivial $2^n \cdot \text{poly}(n)$-time algorithm for n-variable formulas.

Hypothesis 1.1 (Exponential Time Hypothesis)**.** There is a real d, $0 < d < 1$, such that every Turing machine that decides 3-CNF-SATISFIABILITY on n-variable formulas requires at least 2^{dn} time.

Note that, if the Exponential Time Hypothesis holds, then $\mathsf{P} \neq \mathsf{NP}$. Furthermore, if the Exponential Time Hypothesis holds, then n-variable 3-CNF-SATISFIABILITY does not admit a $2^{o(n)}$-time algorithm. This has been strengthened by Impagliazzo, Paturi, and Zane [IPZ01] to the fact that, unless the Exponential Time Hypothesis fails, there is no $2^{o(m)}$-time algorithm for 3-CNF-SATISFIABILITY with m clauses via the so-called Sparsification Lemma.

Subsequently, researchers discovered many implications of the Exponential Time Hypothesis for other problems, where progress on improving the running time upper bounds seemed difficult, too. See Lokshtanov, Marx, and Saurabh [LMS11] and Cygan et al. [Cyg+15, Chapter 14] for expositions.

A simple way to exclude $2^{o(\kappa(q))} \cdot \text{poly}(|q|)$-time algorithms for a parameterized problem (Q, κ) on input q is via a polynomial-time many-one reduction from n-variable 3-CNF-SATISFIABILITY to Q such that the parameter value $\kappa(q)$ is upper bounded by a linear function in n (or, analogously, by m) for each instance q created by the reduction. For example, using the canonical reduction from 3-CNF-SATISFIABILITY we can infer that the well-known CLIQUE problem does not admit a $2^{o(n)} \cdot \text{poly}(n)$-time algorithm unless the Exponential Time Hypothesis fails, where n is the number of vertices in the input graph.

CLIQUE
Input: An undirected graph G and a nonnegative integer k.
Question: Does G have a k-vertex complete subgraph (a clique) as a subgraph?

Chen et al. [Che+05] showed that also an algorithm with running time $n^{o(k)}$ for CLIQUE would contradict the Exponential Time Hypothesis. Finally, we mention that $\mathsf{FPT} \neq \mathsf{W}[1]$ implies that the Exponential Time Hypothesis holds [ADF95].

1.1.4. Calculus

We recall Stirling's approximation of the factorial $t! = t\cdot(t-1)\cdot\ldots\cdot 1$ of a non-negative integer $t \in \mathbb{N}$: The upper and lower bounds

$$t! \leq \sqrt{2\pi}\, t^{t+\frac{1}{2}} e^{-t+1} \text{ and} \tag{1.1}$$

$$t! \geq \sqrt{2\pi}\, t^{t+\frac{1}{2}} e^{-t} \tag{1.2}$$

can be derived from an exposition by Robbins [Rob55].

The below result on a special binomial coefficient follows from Stirling's approximation. We use it in Sections 3.5, 6.2.2, and 6.2.3.

Proposition 1.1. For every two nonnegative integers $\ell, k \in \mathbb{N}$ we have

$$\binom{\ell k}{k} \geq \frac{1}{e^2}\sqrt{\frac{\ell}{2\pi(\ell-1)k}}\left(\frac{\ell}{\ell-1}\right)^{\ell k}(\ell-1)^k.$$

Proof. By applying the definition of binomial coefficients to $\binom{\ell k}{k}$ and then Inequalities (1.1) and (1.2), we obtain the following

$$\begin{aligned}\binom{\ell k}{k} &= \frac{(\ell k)!}{((\ell-1)k)!k!} \\ &\geq \frac{\sqrt{2\pi}(\ell k)^{\ell k+\frac{1}{2}} e^{-\ell k}}{((\ell-1)k)!k!} \\ &\geq \frac{(\ell k)^{\ell k+\frac{1}{2}} e^{-\ell k+1}}{\sqrt{2\pi}((\ell-1)k)^{(\ell-1)k+\frac{1}{2}} e^{-(\ell-1)k+1} k^{k+\frac{1}{2}} e^{-k+1}} \\ &= \frac{1}{e^2}\sqrt{\frac{\ell k}{2\pi(\ell-1)k^2}}\cdot\frac{(\ell k)^{\ell k}}{((\ell-1)k)^{(\ell-1)k}k^k}. \end{aligned} \tag{1.3}$$

We rearrange the last fraction in the right-hand side of Equation (1.3) and obtain the following

$$\frac{(\ell k)^{\ell k}}{((\ell-1)k)^{(\ell-1)k}k^k} = \left(\frac{\ell k}{(\ell-1)k}\right)^{\ell k} \cdot \left(\frac{(\ell-1)k}{k}\right)^{k}. \tag{1.4}$$

Thus, by Equation (1.3) and Equation (1.4), we have

$$\binom{\ell k}{k} \geq e\sqrt{\frac{\ell}{2\pi(\ell-1)k}}\left(\frac{\ell}{\ell-1}\right)^{\ell k}(\ell-1)^k. \qquad \square$$

Part I
Be sparse!

Chapter 2

Introduction to hypergraph supports

In this part we investigate the computational complexity of two problems pertaining to finding hypergraph supports. A *support* for a hypergraph $\mathscr{H}$ is a graph G on the same vertex set such that each hyperedge in $\mathscr{H}$ induces a connected subgraph in G. Figure 2.1 shows two examples; the right hypergraph highlights that the hyperedges may overlap in such a way, that they enforce cycles in any support. Finding supports has surprisingly varied applications. We highlight four of them below, showcasing the ways in which supports are useful. It is striking that in all these applications, we aim to find a sparse support in one way or another, that is, a support with a small number of edges. Generally, the corresponding computational problems are NP-complete; finding a dense support, on the other hand, is trivial because a clique is always a support.

Network design. In publish-subscribe communication networks, agents can subscribe to topics and inform each other about news items within their subscribed topics. Suppose that we are to design a peer-to-peer communication network for the agents under the premise that each topic should be self-sufficient, that is, the agents of one topic should be able to inform each other without depending on agents not subscribed to that topic. Then we search for a support for the hypergraph on the subscribers, with a hyperedge containing the set of subscribers for each topic. This support should additionally have properties that are beneficial for *efficient* communication networks. In efficient communication networks, the overall load of communication should be minimized. This quantity can be captured by the number of edges in the support [Cho+07; Hos+12; OR11].

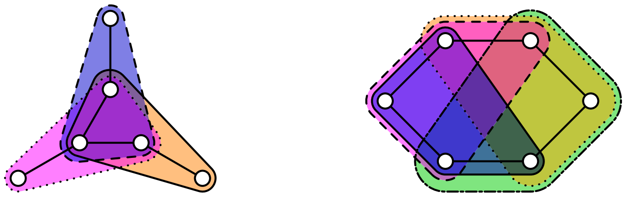


Figure 2.1.: Two hypergraphs and supports for them. Both supports have the fewest possible number of edges. The hypergraphs are drawn in the so-called *subset standard* [Mäk90]: Herein, we draw vertices as white circles and hyperedges by grouping their incident vertices together inside a closed curve which we fill semi-transparently. We draw edges of the supports as direct lines between their end points.

Hypergraph visualization. When drawing a hypergraph, we would like to have a support that can be embedded in the plane without crossings. We can use such a *planar support* to derive a special plane embedding of the hypergraph, called *subdivision drawing*. In such a drawing, each vertex corresponds to a subregion of the plane and for each hyperedge F, the union of the regions that correspond to the vertices in F is a connected region [JP87]. Figure 2.2 shows a subdivision drawing derived from the support of the hypergraph on the right in Figure 2.1. Since computing a planar support is NP-complete [JP87], research focused particularly on several simplifying restrictions of planar supports, aiming for nicer drawings and for tractability of computing supports [KKS08; Buc+11; Bra+12; Bra+11; KMN14]. In this application, it is not the primary objective to find a sparse support. However, every planar graph is also a sparse graph.

Combinatorial auctions. In a combinatorial auction, bidders can bid on bundles of indivisible items. We then want to distribute the items to the bidders (the *winners*) in such a way that the revenue is maximized. The bundles define a hypergraph that has the items as vertices and hyperedges corresponding to the bundles. For determining the winners, it is useful to have a support of bounded treewidth for this hypergraph. In that case, the winners can be determined efficiently, which is NP-complete in general [CDS04]. Similarly to above, it is not the main goal to obtain a sparse support, but graphs of bounded treewidth are also sparse.

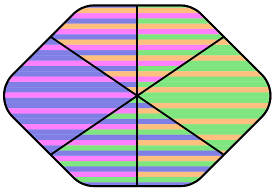


Figure 2.2.: A *subdivision drawing* of the hypergraph on the right in Figure 2.1: Each vertex is represented by a minimal region that is enclosed by black lines. Each hyperedge is represented by a color, that is, if a vertex is contained in a hyperedge, then the region of the vertex contains a color corresponding to the hyperedge. Note that, for each hyperedge, the vertex regions with the color of that hyperedge form a connected region. The placement of the vertex regions directly corresponds to the placement of the vertices in the (planar) support shown in Figure 2.1.

Network inference. A hypergraph on a set of individuals is defined by memes: A hyperedge corresponds to a subset of individuals who propagate a meme. Sometimes we know this hypergraph from observations about posts in social media, but cannot effectively determine the underlying structure of the social network. In this case, a support for the meme-hypergraph could be helpful as a first approximation of structure of the social network [AAR10]. A support with the minimum number of edges represents a maximum-parsimony explanation for the spreading of memes.

Apart from the above, supports find applications in vacuum system design [DM88; DK95], processor design [Fan+08], database systems [Bee+83; TY84; Gol88], and supports also appear as a tool in the analysis of colorability of hypergraphs where they are called *spanning subgraphs* [KKV04] or *host graphs* [BTV11]. Similarly to the application in combinatorial auctions, supports of special structure can also serve to identify tractable special cases of hypergraph problems. For example, Guo and Niedermeier [GN06] found that the SET COVER problem is solvable in polynomial time if the dual of the input hypergraph has a tree support and Jansen [Jan17] found that HITTING SET is fixed-parameter tractable with respect to a parameter that measures the tree-likeness of a special support of the input hypergraph.

Indicative of the importance of hypergraph supports is that several communities discovered and rediscovered results on them independently from one another. For example, there are three NP-hardness proofs for finding minimum-edge supports

[Cho+07; DM88; Fan+08] and two polynomial-time algorithms for finding tree supports (one by combining results of Beeri et al. [Bee+83] and Tarjan and Yannakakis [TY84], and one by Conitzer, Derryberry, and Sandholm [CDS04]).

Appetizer. In this work we study two problems arising from the first and the second one of the above applications, respectively. Regarding the first one, network design, we investigate the following problem in Chapter 3.

> SUBSET INTERCONNECTION DESIGN
> *Input:* A connected hypergraph $\mathcal{H}$ and a nonnegative integer f.
> *Question:* Does $\mathcal{H}$ admit a support with at most $|V(\mathcal{H})| - 1 + f$ edges?

The formulation of SUBSET INTERCONNECTION DESIGN is motivated by the fact that sparsity is one of the properties desired for good resulting communication networks [Cho+07]. Furthermore, the parameter f, also called *feedback edge number*, measures the distance from the polynomial-time special case of constructing a tree support [CDS04; Bee+83; TY84]. We prove that SUBSET INTERCONNECTION DESIGN is fixed-parameter tractable with respect to the number of hyperedges and also with respect to the largest hyperedge size and the feedback edge number f combined. The results are obtained via several data reduction rules.

In Chapter 4 we analyze the following problem which is motivated by applications in hypergraph visualization.

> r-OUTERPLANAR SUPPORT
> *Input:* A connected hypergraph $\mathcal{H}$ and a nonnegative integer r.
> *Question:* Does $\mathcal{H}$ admit an r-outerplanar support?

Similarly to previous research for planar hypergraph supports [Buc+11; Bra+11], we restrict our attention to r-outerplanar supports for tractability and aesthetic reasons. A graph is *r-outerplanar* if it admits an embedding in the plane without edge crossings such that the graph becomes empty after at most r steps of deleting all vertices incident with the outer face. See Section 4.2 for details. We prove that r-OUTERPLANAR SUPPORT is fixed-parameter tractable with respect to r and the number of hyperedges combined. The result is obtained via a single data reduction rule.

In both r-OUTERPLANAR SUPPORT and SUBSET INTERCONNECTION DESIGN we use the parameterization by the number m of hyperedges as a most natural candidate. A series of papers proved that SUBSET INTERCONNECTION DESIGN is solvable in polynomial time for $m = 2, 3, 4$ [Du86; Tan89; XF95] and a paper studied drawings of hypergraphs with $m \leq 8$ [VV04], warranting further investigation. Indeed, there is a simple argument showing that both r-OUTERPLANAR SUPPORT and SUBSET INTERCONNECTION DESIGN are non-uniformly fixed-parameter tractable with respect to m (see Section 2.1). However, we argue in Section 2.1 below that (strongly) uniform results are desirable. We deliver them in Chapters 3 and 4, respectively, where we also provide more details regarding related work for r-OUTERPLANAR SUPPORT and SUBSET INTERCONNECTION DESIGN.

Related problems. We now mention several problems related to finding supports of special structure that were studied in the literature. They impose different requirements on the type of support or tighten or relax the constraint that each hyperedge should induce a connected graph.

As mentioned above, supports of small treewidth are useful when determining winners in combinatorial auctions. More precisely, the winner determination problem can be formulated as WEIGHTED SET PACKING and this problem is in XP with respect to the treewidth of a given support [CDS04]. However, deciding whether there is a support of treewidth three is NP-complete [GG13]. As far as we know, the complexity in the case of treewidth two is currently unknown. Gottlob and Greco [GG13] also improved Conitzer, Derryberry, and Sandholm's [CDS04] XP-result, by showing that WEIGHTED SET PACKING is in XP with respect to the so-called hypertree width of the dual hypergraph and that this parameter is upper bounded by the treewidth of a support.

In network design, further desirable properties that supports may possess apart from sparsity have been considered. These properties are related to designing efficient and fault-tolerant communication networks. To keep the communication load on each node in the network low, it is desirable that the corresponding supports have small maximum vertex degree [Cho+07]. For small latencies, also the diameter of the subgraphs induced by the hyperedges should be small [Cho+07]. Fault-tolerance in a communication network translates to k-connectedness of the subgraphs induced by hyperedges for some prespecified parameter k [CVJ13]. Finding a k-connected support possessing the minimum number of edges is NP-complete [CVJ13]. Onus and Richa [OR11] addressed the problem of minimizing the maximum degree of the support, provided a polynomial-time logarithmic-factor approximation algorithm, and proved that an asymptotic improvement in the approximation factor is not possible unless P = NP. Obviously, minimizing the diameters of the hyperedge-induced subgraphs only makes sense when we simultaneously aim for other properties in the support like sparsity or small maximum degree, for example. To the best of our knowledge, such combined optimization criteria have not been studied from a computational complexity point of view yet.

Dinitz and Wilfong [DW12] studied the so-called CONSTRAINED CONNECTIVITY problem in which we are given a set of vertices V and a *safe set* $S_{u,v} \subseteq V$ for each pair of vertices $u, v \in V$. The goal is to find a graph G with vertex set V such that for each pair $u, v \in V$ the subgraph $G[S_{u,v}]$ induced by the safe set contains a path between u and v. That is, in contrast to the support problem, we only require one pair of vertices to be connected in each "hyperedge" $S_{u,v}$. As optimization criteria Dinitz and Wilfong considered minimizing the number of edges and minimizing the maximum degree, respectively. The corresponding optimization problems have applications in network routing. Dinitz and Wilfong [DW12] provided NP-hardness and approximability results.

Bixby and Wagner [BW88] considered the problem of searching for a tree support such that each hyperedge induces a path. They showed that this is equivalent to deciding whether a given binary matroid is *graphic*, that is, whether its independent sets correspond to forest subgraphs of a graph. Bixby and Wagner proposed an algorithm for this problem with linear running time multiplied by an inverse Ackermann-function factor.

Finally, finding a path support is equivalent to checking whether a given binary matrix has the consecutive ones property, that is, whether there is a reordering of the columns such that the ones appear consecutive in each row. This problem can be solved in linear time in the size of the matrix via a famous algorithm by Booth and Lueker [BL76]. See also Dom [Dom09] for a survey on algorithms for the consecutive ones property.

In summary, hypergraph supports are surprisingly ubiquitous with varied fields of application, making for a worthwhile field of study.

2.1. Parameterization by the number of hyperedges

For both r-OUTERPLANAR SUPPORT and SUBSET INTERCONNECTION DESIGN we examine the parameterization by m, the number of hyperedges, as a very natural restriction of the input structure. We now briefly reflect on a counter-intuitive property of the parameterization m and we prove that a class of problems encompassing both r-OUTERPLANAR SUPPORT and SUBSET INTERCONNECTION DESIGN is *nonuniformly* fixed-parameter tractable with respect to m. This motivates the search for *uniform* fixed-parameter tractability results.

Twins. At first glance, it seems trivial to prove that both problems are fixed-parameter tractable with respect to m: Let us call two vertices $u, v \in V$ of a hypergraph $\mathcal{H} = (V, \mathcal{E})$ *twins*, if they are in the same hyperedges. That is, $\mathcal{E}(u) = \mathcal{E}(v)$. Twins do not seem to add any new information, and if there are no twins then clearly the number $|V(\mathcal{H})|$ of vertices is upper bounded by a function of m. This would imply that we can find a support in fixed-parameter time by removing all twins and trying all possible graphs. However, we prove in Section 4.3 for r-OUTERPLANAR SUPPORT and in Section 3.3 for SUBSET INTERCONNECTION DESIGN that twins can actually make or break the instance, that is, removing just one twin from a yes-instance can result in a no-instance.

Adding twins in both problems is not detrimental, however. For example, if we have an r-outerplanar support for $\mathcal{H}$, then we can make a new twin adjacent to one of its already present twins with a bit of care, so that the resulting graph remains r-outerplanar[2]: We just have to make sure that the new twin does not end-up in a new $(r+1)$th layer by placing it in a face as close to the outer face as possible. Similarly, adding a twin as a degree-one neighbor to its present twin in a support with feedback edge number f results again in a support with feedback edge number f.

Nonuniform fixed-parameter tractability. Reversing the idea above, from each hypergraph with an r-outerplanar support by deleting twins we can obtain a *minimal* hypergraph $\mathcal{H}'$ which also has an r-outerplanar support but from which no further twins can be deleted while maintaining the property of having an r-outerplanar support. Using Dickson's lemma (see below for details) it is not hard to show that there is a function ϕ such that, for each fixed number m of hyperedges, there are only $\phi(m)$ such minimal hypergraphs. (A priori, we do not know of a way to compute ϕ, however. A way to compute ϕ arises from the uniform fixed-parameter tractability result that we give in Chapter 4.) By hard-wiring the minimal hypergraphs, we obtain for each value of m an algorithm that decides whether a given hypergraph has an r-outerplanar support. This idea works in the same way for supports of fixed feedback edge number f and, more generally, for supports satisfying an arbitrary graph property $\mathbb{P}$ that is *closed under adding degree-one vertices*, that is, to an arbitrary vertex in a graph in $\mathbb{P}$, we may add a degree-one neighbor and again obtain a graph in $\mathbb{P}$.[3]

[2] Recall that a graph is *r-outerplanar* if it admits an embedding in the plane without edge crossings such that the graph becomes empty after at most r steps of deleting all vertices incident with the outer face. See Section 4.2 for details.

[3] Even more generally, we can allow graph properties $\mathbb{P}$ such that, for every graph $G \in \mathbb{P}$ and every vertex $u \in V(G)$, there is a graph $G' \in \mathbb{P}$ with $V(G') = V(G) \cup \{v\}$ and $E(G') \supseteq E(G) \cup \{\{u, v\}\}$. This observa-

We now formalize the above approach. For a graph property $\mathbb{P}$, say that a hypergraph $\mathcal{H}$ is *$\mathbb{P}$-supportable* if $\mathcal{H}$ admits a support contained in $\mathbb{P}$. We prove the following.

Theorem 2.1. Let $\mathbb{P}$ be a graph property that is closed under adding degree-one vertices. It is nonuniformly fixed-parameter tractable with respect to the number of hyperedges to decide whether a given hypergraph is $\mathbb{P}$-supportable. That is, for each $m \in \mathbb{N}$, there is a fixed-parameter algorithm with respect to m that decides whether a given hypergraph with m hyperedges is $\mathbb{P}$-supportable.

The proof is given via two lemmas. First, we observe that being $\mathbb{P}$-supportable is a hypergraph property that is closed under adding twins. A hypergraph property $\mathbb{S}$ is *closed under adding twins* if taking any hypergraph $\mathcal{H} \in \mathbb{S}$ and adding a twin to it yields another hypergraph in $\mathbb{S}$. Adding a twin means to take a vertex u in $\mathcal{H}$ and to add a new vertex v that is contained in precisely the same hyperedges as u.

Denote by $\mathbb{S}_\mathbb{P}$ the hypergraph property of being $\mathbb{P}$-supportable, that is, $\mathbb{S}_\mathbb{P}$ is the set containing all $\mathbb{P}$-supportable hypergraphs. If $\mathbb{P}$ is closed under adding degree-one vertices, then we can simply add a new twin as a degree-one neighbor to one of its siblings in a support. Hence, we have following.

Lemma 2.1. If the graph property $\mathbb{P}$ is closed under adding degree-one vertices, then the hypergraph property $\mathbb{S}_\mathbb{P}$ is closed under adding twins.

Next we show that every hypergraph property that is closed under adding twins is non-uniformly fixed-parameter decidable with respect to m.

Lemma 2.2. Let $\mathbb{S}$ be a hypergraph property that is closed under adding twins. For every $m \in \mathbb{N}$ there is a fixed-parameter algorithm with respect to m that decides $\mathbb{S}$ on hypergraphs with m hyperedges.

Proof. We first define a quasi-order $\preceq$ on the family of hypergraphs with m hyperedges. (A *quasi-order* is reflexive and transitive.) To define $\preceq$, we say that $\mathcal{H} \preceq \mathcal{G}$ if $\mathcal{H}$ can be obtained from $\mathcal{G}$ by iteratively removing a vertex that has a twin. If we allow zero removals so that $\preceq$ is reflexive, it is clear that $\preceq$ is a quasi-order. Moreover, if $\mathcal{H} \in \mathbb{S}$ and $\mathcal{H} \preceq \mathcal{G}$, then $\mathcal{G} \in \mathbb{S}$ since $\mathbb{S}$ is closed under adding twins.

Next we show that, for every $m \in \mathbb{N}$, the family $\mathbb{F}_m$ of $\mathbb{P}$-supportable hypergraphs that are minimal under $\preceq$ is finite. Consider the representation of an m-hyperedge hypergraph $\mathcal{H}$ as a 2^m-tuple $t_\mathcal{H} \in \mathbb{N}^{2^m}$, each entry of which represents the size of

tion may be useful when treating graph properties $\mathbb{P}$ which are not closed under adding degree-one vertices like cographs, for example.

a distinct twin class. The set of such tuples is quasi-ordered by a natural extension of $\leq$, namely $(a_1,\dots,a_\ell) \leq (b_1,\dots,b_\ell)$ if $a_i \leq b_i$ for each $i \in \{1,\dots,\ell\}$. We now lead the assumption that $\mathbb{F}_m$ is infinite to a contradiction by using Dickson's Lemma [Dic13, Lemma A].

If $\mathbb{F}_m$ is infinite, then there is an infinite subset $\mathbb{F}'_m$ of hypergraphs which have the same (nonempty) twin classes. For hypergraphs $\mathcal{H},\mathcal{G}$ with the same twin classes, $t_{\mathcal{H}} \leq t_{\mathcal{G}}$ implies $\mathcal{H} \preceq \mathcal{G}$. Thus, $\mathbb{F}'_m$ gives an infinite set T of tuples that are pairwise incomparable under $\leq$. Dickson's Lemma states that for every set $S \subseteq \mathbb{N}^\ell$ there exists a finite subset $S' \subseteq S$ such that for each $s \in S$ there is an $s' \in S'$ with $s' \leq s$. This is a contradiction to T containing infinitely many incomparable tuples. Hence, $\mathbb{F}_m$ is finite.

Finally, to obtain an algorithm for every fixed m as in the lemma, we embed the family $\mathbb{F}_m$ of hypergraphs in $\mathbb{S}$ that are minimal with respect to $\preceq$ as a constant into the algorithm. The algorithm simply checks whether its input hypergraph $\mathcal{H}$ fulfills $\mathcal{F} \preceq \mathcal{H}$ for some $\mathcal{F} \in \mathbb{F}_m$, which clearly can be done in polynomial time for each $\mathcal{F} \in \mathbb{F}_m$. □

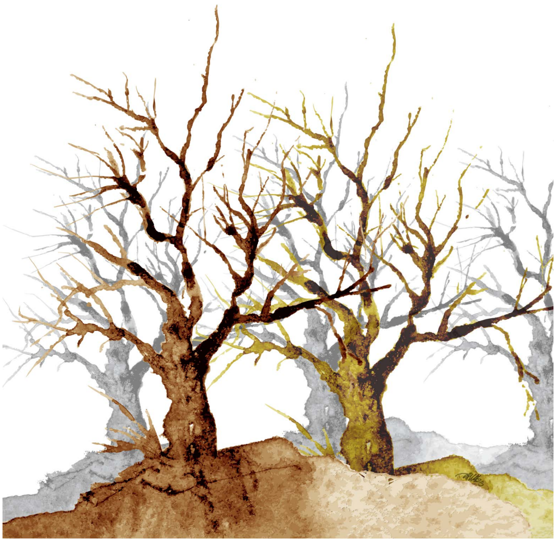
Theorem 2.1 directly follows from Lemmas 2.1 and 2.2. We observed before that, for $\mathbb{P} \in \{r\text{-outerplanar, feedback edge number} = f\}$, to every $\mathbb{P}$-supportable hypergraph we can add any new twin and again obtain a $\mathbb{P}$-supportable hypergraph. Hence, we have by Theorem 2.1 a fixed-parameter algorithm for every fixed value of parameter m, the number of hyperedges, for both r-OUTERPLANAR SUPPORT and f-SUBSET INTERCONNECTION DESIGN, where we look for a support with feedback edge number f, .

Corollary 2.1. For each $m \in \mathbb{N}$ there are fixed-parameter algorithms with respect to m that decide r-OUTERPLANAR SUPPORT and f-SUBSET INTERCONNECTION DESIGN, respectively, on hypergraphs with m hyperedges.

The need for uniform results. We emphasize that the proof of Theorem 2.1 does not provide insight into why it is that a non-$\mathbb{P}$-supportable hypergraph suddenly becomes $\mathbb{P}$-supportable when adding a twin. That is, we currently do not know how to compute the set $\mathbb{F}_m$ of minimal yes-instances, making the nonuniform fixed-parameter tractability result inapplicable at the moment. Moreover, it is in general impossible to replace any nonuniform fixed-parameter tractability result with a strongly uniform one [DF99, Theorem 19.21]. Hence, to eventually obtain implementable algorithms that are able to deal with any input, it is important to find a uniform result instead. To this end, we prove uniform fixed-parameter tractability with respect to m for both r-OUTERPLANAR SUPPORT and SUBSET INTERCONNECTION DESIGN in Chapter 4 and Chapter 3 respectively. We leave it as a question for future research whether these results can be unified to yield uniform fixed-parameter tractability with respect to m for a general class of hypergraph properties that encompass having r-outerplanar supports and supports with small feedback edge sets.

Chapter 3

Minimum-edge hypergraph supports



This chapter is based on “Polynomial-Time Data Reduction for the Subset Interconnection Design Problem” by Jiehua Chen, Christian Komusiewicz, Rolf Niedermeier, Manuel Sorge, Ondřej Suchý, and Mathias Weller (SIAM Journal on Discrete Mathematics [Che+15]).

3.1. Introduction

In several applications, practitioners construct an *overlay graph* that connects the vertices of each hyperedge of a given hypergraph. In this chapter we study an NP-complete problem where we want to construct an overlay graph containing few edges. The formal definition of the corresponding decision problem is as follows.

> SUBSET INTERCONNECTION DESIGN
> *Input:* A connected hypergraph $\mathscr{H}$ and a nonnegative integer f.
> *Question:* Does $\mathscr{H}$ admit a support with at most $|V(\mathscr{H})| - 1 + f$ edges?

Recall that a *support* for hypergraph $\mathscr{H}$ is a graph G on the same vertex set such that each hyperedge in $\mathscr{H}$ induces a connected subgraph of G. In essence, SUBSET INTERCONNECTION DESIGN is the decision problem for finding a support of minimum number of edges. We formulated it with the parameter f, that is, an upper bound on the number of edges in the desired support diminished by $|V(\mathscr{H})| - 1$, because each support for a connected hypergraph contains at least $|V(\mathscr{H})| - 1$ edges. Hence, to minimize the number of edges, we indeed minimize f. We call this quantity the *feedback edge number* of the support; in the literature, it is also known as *cyclomatic number*, *circuit rank*, and *nullity*. Throughout this chapter, we say that a support is *optimal* if it contains the minimum number of edges or, equivalently, if it has the minimum feedback edge number. Figure 3.1 shows two examples of hypergraphs and optimal supports. Although we present our results SUBSET INTERCONNECTION DESIGN as a decision problem it is easy to adapt the positive algorithmic results to its optimization version.

SUBSET INTERCONNECTION DESIGN is a fundamental problem concerning hypergraph and graph structures that has many applications. As mentioned in Chapter 2, SUBSET INTERCONNECTION DESIGN was studied in the context of designing vacuum systems [DM88; DK95], publish-subscribe information systems [Cho+07; Hos+12; OR11], reconfigurable interconnection networks [Fan+08], and, in a weighted variant, in the context of inferring a most-likely social network [AAR10]. To the best of our knowledge, SUBSET INTERCONNECTION DESIGN was first defined by Du [Du86] and the first NP-completeness proof was presented by Du and Miller [DM88].

Applications. We briefly describe applications that motivate the study of SUBSET INTERCONNECTION DESIGN. In vacuum systems, we are to connect a set of working places (vertices of a hypergraph) using valves (edges of the support for the hypergraph), so that a low-pressure environment can be made available at each working

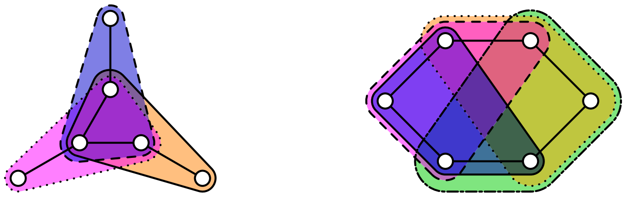


Figure 3.1.: Left: A hypergraph with six vertices (white circles) and three hyperedges (lines enclosing vertices). Its support (edges indicated by direct lines between their endpoints) has feedback edge number 0. Right: A hypergraph with six vertices and four hyperedges. Its optimal support has feedback edge number 1.

place. Hyperedges correspond to different requirements on the pressure level, provided by different vacuum pumps. An optimal support corresponds to a minimum-size set of valves that have to be provided. Apart from minimizing the costs of the construction, an optimal support also maximizes the efficiency of the resulting vacuum system because of pressure leaks [DK95].

In publish-subscribe information systems, we are given a set of agents, each subscribed to a certain set of topics. The task is to design an *overlay network* through which the agents can inform each other about news in their topic. Furthermore, the agents of each topic should be able to inform each other without having to rely on agents not subscribed to that topic. Thus, we are looking for a support for the hypergraph that has the agents as vertices and a hyperedge for the set of subscribers of each topic. Chockler et al. [Cho+07] note that it is desirable that the corresponding overlay network should be efficient, which means that the support should have low average and maximum (vertex) degree. Clearly, the optimal support has the smallest-possible average degree.

SUBSET INTERCONNECTION DESIGN is a generalization of TREE SUPPORT where we are given a hypergraph $\mathcal{H}$ and we are asked whether $\mathcal{H}$ admits a support that is a tree. We also call a hypergraph *hypertree* if it has a tree support [Bra+98]. Hypertrees have received much attention for two reasons. First, they are related to chordal graphs, a well studied graph class [BLS99]: A graph is *chordal* if and only if the dual hypergraph of its clique hypergraph is a hypertree. (The *clique hypergraph* of a graph G has the same vertex set as G and a hyperedge for the vertex set of each maximal clique in G.) Hypertrees were studied extensively because of this connection to chordal graphs as well as to other related graph classes, see Brandstädt, Le, and Spinrad [BLS99] for a survey.

The second motivation for studying hypertrees stems from database systems. Here, hypergraphs occur naturally by taking the attributes managed in the database tables as vertices and defining a hyperedge for each relation scheme, a set of attributes whose relation is managed in a table. Beeri et al. [Bee+83] argued that it is desirable for the hypergraph $\mathcal{H}$ as above to be a *dual hypertree*, that is, the dual hypergraph of a hypertree (see also Fagin [Fag83] for a gentle introduction).[4] For example, if $\mathcal{H}$ is a dual hypertree, then the underlying database has a rather simple *consistency* check, that is, whether no subset of tables in the database contain contradictory values.

Finally, supports of small feedback edge number may serve as a way to identify tractable special cases of generally hard problems. For example, Guo and Niedermeier [GN06] showed that the NP-complete SET COVER problem is polynomial-time solvable, if the dual of the input hypergraph has a tree support. Moreover, Jansen [Jan17] showed that HITTING SET is fixed-parameter tractable with respect to the feedback edge number of a given support G that fulfills the additional condition that each hyperedge in the input hypergraph induces a path in G. (If more general induced subgraphs are allowed, HITTING SET remains W[1]-hard, however.)

In this chapter, we study the influence of three parameters on the computational complexity of SUBSET INTERCONNECTION DESIGN. The parameters are the size $d := \max_{F \in \mathcal{E}(\mathcal{H})} |F|$ of the largest hyperedge, the number $m := |\mathcal{E}(\mathcal{H})|$ of hyperedges in the input hypergraph $\mathcal{H}$, and an upper bound f on the feedback edge number $|E(G)| - |V(G)| + 1$ of the support G. Parameters m and d measure sparseness of the input, whereas f measures sparseness of the output.

The feedback edge number f of the support is clearly motivated by the applications above. It also makes sense to assume that the number m of hyperedges is small, for example, in the vacuum systems and database systems applications. Moreover, a series of papers showed that SUBSET INTERCONNECTION DESIGN can be solved in polynomial time if $2 \le m \le 4$ [Du86; Tan89; XF95]. This begs the question whether there is an XP-time or FPT-time algorithm for SUBSET INTERCONNECTION DESIGN parameterized by m. Finally, the parameter d, the maximum hyperedge

[4] Beeri et al. [Bee+83] called dual hypertrees *acyclic*. We refrain from using this notation because dual hypertrees can contain cycles. The notion of acyclicity of Beeri et al. [Bee+83] is equivalent to being a dual hypertree due to a characterization of Slater [Sla78]. See also Brandstädt, Le, and Spinrad [BLS99, Theorem 1.3.1].

size, is a technical restriction which we need to obtain tractability for small f and which we could not circumvent so far.

Our contribution. We perform a parameterized complexity analysis with respect to the parameters m, d, and f, and we show that small parameters yield efficient algorithms. Our core working machinery is to develop numerous polynomial-time data reduction rules.

We start with the parameter m, the number of hyperedges. As mentioned in Section 2.1, at first glance, SUBSET INTERCONNECTION DESIGN should be fixed-parameter tractable with respect to m because it seems superfluous to have two vertices which are in the same hyperedges; we call such vertices *twins*. We should be able to remove one of the two twin vertices, say v. Then we should be able to find an optimal support of the resulting hypergraph and attach v as a degree-one neighbor of its twin to obtain an optimal support of the original hypergraph. However, as we show in Section 3.3, removing just one twin in a hypergraph can increase the feedback edge number f of the optimal support by at least one. In fact, removing twins (or removing vertices in a more general relation) has been proposed as a reduction rule for SUBSET INTERCONNECTION DESIGN [Fan+08; Hos+12]; we provide a counterexample to the correctness of this rule.[5] Based on this, we provide both refined and completely new data reduction rules, and prove their correctness and effectiveness.

We go on to show that SUBSET INTERCONNECTION DESIGN can be solved in linear time if the input hypergraph contains only a constant number m of hyperedges (Section 3.4). This implies that SUBSET INTERCONNECTION DESIGN is fixed-parameter tractable with respect to m. To achieve this, we prove that every hypergraph with m edges admits a support with at most $\phi(m)$ vertices in each twin class which do not have degree one for some function ϕ. (A *twin class* is a maximal set of pairwise twins.) In each of the at most 2^m twin classes, we can safely remove the remaining vertices, yielding a problem kernel.

For hyperedges of size $d \leq 4$ we show that SUBSET INTERCONNECTION DESIGN has a problem kernel with a linear number of vertices with respect to f, the feedback edge number of the support (Section 3.6.1). The proof is based on several data reduction rules, which allow us to make successively more assumptions about an optimal support. Ultimately, we can assume that no vertex in this support is in a *dangling tree* of size at least two, a tree which can be separated from the rest of the graph via removing a single vertex. Furthermore, apart from at most $O(fd)$ vertices,

[5]Hosoda et al. [Hos+15] subsequently corrected their result.

each vertex has degree at least three. This then implies that there are few vertices overall, since the number of vertices of degree at least three that are not in dangling trees is bounded from above linearly in f.

For arbitrary d we make use of our data reduction rules and show in Section 3.6.2 that SUBSET INTERCONNECTION DESIGN can be solved in $d^{18df} \cdot \text{poly}(|\mathcal{H}|)$ time. (Recall that $|\mathcal{H}|$ denotes $|V(\mathcal{H})| + \sum_{F \in \mathcal{E}(\mathcal{H})} |F|$.)

Known results and related work. As mentioned before, SUBSET INTERCONNECTION DESIGN has been studied independently in different communities under different names. Several NP-completeness proofs have appeared [Cho+07; DM88; Fan+08]. Fan et al. [Fan+08] and Hosoda et al. [Hos+12] showed that NP-completeness holds even when each hyperedge in the input hypergraph has size $d = 3$, while for $d \leq 2$ the input hypergraph is itself an optimal support. If the hypergraph is closed under intersections, that is, every intersection of two hyperedges is also a hyperedge in the hypergraph, then SUBSET INTERCONNECTION DESIGN becomes polynomial-time solvable; this follows from techniques used by Buchin et al. [Buc+11, Lemma 1]. Polynomial-time approximability was also studied: Angluin, Aspnes, and Reyzin [AAR10], Chockler et al. [Cho+07], and Hosoda et al. [Hos+12] provided various logarithmic-factor approximation algorithms. Angluin, Aspnes, and Reyzin [AAR10] and Hosoda et al. [Hos+12] gave inapproximability results, implying that logarithmic-factor approximation algorithms are likely to be optimal. The currently fastest exact algorithm for SUBSET INTERCONNECTION DESIGN has a running time of $O(n^{2k}/4^k + n^2)$ where $k = |V(\mathcal{H})| - 1 + f$ is the upper bound on the number of edges the support [Hos+12].

From Theorem 2.1 it follows that SUBSET INTERCONNECTION DESIGN is nonuniformly fixed-parameter tractable with an unknown running time function, of which we in particular do not know whether it is computable. In contrast, we show in this chapter that this result can be made uniform, that is, there is *one* fixed-parameter algorithm for every m and we give a concrete running time.

TREE SUPPORT, the problem whether a given hypergraph has a tree support, can be decided in linear time. This follows from a characterization of Slater [Sla78] which is equivalent to the dual hypergraph being *acyclic* (independently discovered by two other authors, see Brandstädt, Le, and Spinrad [BLS99, Theorem 1.3.1 and Theorem 8.1.1]). Tarjan and Yannakakis [TY84] showed that acyclicity can be decided in linear time. This result was seemingly unknown to Conitzer, Derryberry, and Sandholm [CDS04], who later gave a polynomial time algorithm for constructing a tree support.

A variant of TREE SUPPORT where the edges incur costs was studied in the context of communication network design [KS03] and hypergraph drawing [KMN14]; here, we are to find a tree support of minimum cost. This variant can be solved in $O(n^2(m + \log(n)))$ time [KMN14] (recall that $n = |V(\mathcal{H})|$ and $m = |\mathcal{E}(\mathcal{H})|$).

3.2. Specific preliminaries

A *feedback edge set* of a graph G is a set of edges whose removal makes G a forest. The *feedback edge number* f of G is the size of any minimum feedback edge set. If G is connected, then the feedback edge number is $|E(G)| - |V(G)| + 1$.

Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph. Recall that $\mathcal{E}(v)$ denotes the set of hyperedges that contain v and that a vertex $u \in V$ *covers* another vertex $v \in V$ if $\mathcal{E}(v) \subseteq \mathcal{E}(u)$. The *covering graph* of hypergraph $\mathcal{H} = (V, \mathcal{E})$ is the directed graph $G_C = (V, \{(u, v) \mid \mathcal{E}(v) \subseteq \mathcal{E}(u)\})$. In other words, G_C contains an arc (u, v) if and only if u covers v. Note that G_C is transitive. Some of our reduction rules construct the covering graph of $\mathcal{H}$ as a subroutine. The following lemma bounds the running time for this step.

Lemma 3.1. Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$ we can construct the covering graph G_C in $O(n \cdot |\mathcal{H}|)$ time.

Proof. Initialize G_C as the directed graph $(V, V \times V)$. Then, for each $F \in \mathcal{E}$, remove the arcs in $(V \setminus F) \times F$ from G_C in $O(n \cdot |F|)$ time. Clearly, if (u, v) is an arc of the resulting directed graph G_C, then there is no hyperedge containing v but not u or, equivalently, $\mathcal{E}(u) \supseteq \mathcal{E}(v)$. If G_C does not contain the arc (u, v), then there is a hyperedge $F \in \mathcal{E}$ such that $v \in F$ but $u \notin F$, by the construction of G_C. Thus, G_C contains exactly the arcs (u, v) such that u covers v. $\square$

3.3. Beware of removing twins

In this section, we show that a previously proposed data reduction rule for SUBSET INTERCONNECTION DESIGN is incorrect. We also provide properties of SUBSET INTERCONNECTION DESIGN's optimal supports and simple data reduction rules that we use later.

A very natural approach to identifying edges of an optimal support is to look for vertices u and v such that u covers v, that is, $\mathcal{E}(v) \subseteq \mathcal{E}(u)$. The following shows that, in every support, degree-one vertices are adjacent to vertices that cover them.

Observation 3.1. If the hypergraph $\mathcal{H} = (V, \mathcal{E})$ has a support G such that some $v \in V$ has only one neighbor u in G, then u covers v.

Proof. Let $F \in \mathcal{E}$ be a hyperedge with $v \in F$. Since $G[F]$ is connected, v has degree at least one in $G[F]$. Since u is the only neighbor of v in G it follows that $u \in F$. □

It is thus tempting to design a data reduction rule that adds an edge between such vertices: creating a degree-one vertex should be optimal since every vertex needs at least one incident edge. Indeed, such a reduction rule was proposed for vertex pairs u, v that are *twins*, that is, they are in the same hyperedges [Fan+08], or where one covers the other [Hos+12]. The variant of these reduction rules that applies less often reads as follows.

Rule 3.1. If the hypergraph $\mathcal{H} = (V, \mathcal{E})$ contains twins u and v, that is, $\mathcal{E}(u) = \mathcal{E}(v)$, then remove u from $\mathcal{H}$.

Unfortunately, Rule 3.1 is not correct, as the following counterexample shows.

Lemma 3.2. Let $f \geq 3$. There is a hypergraph that admits a support with feedback edge number f and that contains precisely two twins u and v such that removing either u or v yields a hypergraph that does not admit a support with feedback edge number f.

Proof. Consider the hypergraph $\mathcal{H} = (V, \mathcal{E})$, with vertex set

$$V = \{u, v, a_1, \ldots, a_f, b_1, \ldots, b_f\}$$

and hyperedge set $\mathcal{E}$ which is the union of the following sets of hyperedges:

$$\begin{aligned}
\mathcal{E}_1 &= \{\{a_i, b_i\} \mid i \in \{1, \ldots, f\}\},\\
\mathcal{E}_2 &= \{\{u, v, a_i, b_i\} \mid i \in \{1, \ldots, f\}\},\\
\mathcal{E}_3 &= \{\{u, v, a_i, b_i, a_j\} \mid i, j \in \{1, \ldots, f\}, i \neq j\}, \text{ and}\\
\mathcal{E}_4 &= \{\{u, v, a_i, b_i, b_j\} \mid i, j \in \{1, \ldots, f\}, i \neq j\}.
\end{aligned}$$

Note that the graph $G = (V, E)$ with $E := \mathcal{E}_1 \cup \{\{a_i, u\}, \{b_i, v\} \mid i \in \{1, \ldots, f\}\}$ is a support for $\mathcal{H}$ containing $3f$ edges, that is, G has feedback edge number

$$3f - (2f + 2) + 1 = f - 1.$$

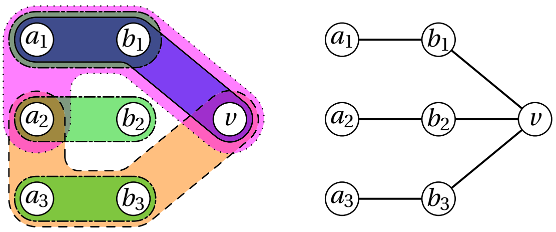


Figure 3.2.: An example that shows parts of the hypergraph $\mathcal{H}'$ in the proof of Lemma 3.2; herein the upper bound on the feedback edge number $f = 3$. Left: some hyperedges from $\mathcal{H}'$. Right: the $2f$ edges that can be assumed to be in an optimal support for $\mathcal{H}'$.

Let $\mathcal{H}' = (V', \mathcal{E}')$ be the hypergraph that results from $\mathcal{H}$ by applying Rule 3.1 to u and v, that is, removing u from $\mathcal{H}$. Figure 3.2 shows parts of the hypergraph $\mathcal{H}'$ for $f = 3$. Then, $V' = V \setminus \{u\}$ and $\mathcal{E}'$ consists of the following hyperedges:

$$\begin{aligned}
\mathcal{E}_1 &= \{\{a_i, b_i\} \mid i \in \{1, \ldots, f\}\},\\
\mathcal{E}'_2 &= \{\{v, a_i, b_i\} \mid i \in \{1, \ldots, f\}\},\\
\mathcal{E}'_3 &= \{\{v, a_i, b_i, a_j\} \mid i, j \in \{1, \ldots, f\}, i \neq j\}, \text{ and}\\
\mathcal{E}'_4 &= \{\{v, a_i, b_i, b_j\} \mid i, j \in \{1, \ldots, f\}, i \neq j\}.
\end{aligned}$$

We show that every support for $\mathcal{H}'$ has at least $3f$ edges and, thus, it has feedback edge number at least f.

First, every support for $\mathcal{H}'$ contains the f edges corresponding to the size-two hyperedges of $\mathcal{E}_1$. Furthermore, due to the hyperedges in $\mathcal{E}'_2$, for each $i \in \{1, \ldots, f\}$, every support contains $\{v, a_i\}$ or $\{v, b_i\}$. By the symmetry between a_i and b_i in the hypergraph $\mathcal{H}'$, assume without loss of generality that an optimal support contains the edge $\{v, b_i\}$ for all $i \in \{1, \ldots, f\}$ (the set of these edges plus $\mathcal{E}_1$ is shown in Figure 3.2). Now, let $G' = (V', E')$ be such a support for $\mathcal{H}'$. Let $A_1 = \{a_i \mid \{v, a_i\} \notin E'\}$ be the set of a_is that are *not* adjacent to v in G' and let A_2 denote the set of the remaining a_is. If $A_1 = \emptyset$, then G' contains at least $3f$ edges. We show that, if $A_1 \neq \emptyset$, then the graph G' also has at least $3f$ edges. Assume that G' is optimal and that every optimal support has at least $k > 0$ vertices in A_1. For every hyperedge $F = \{v, a_i, b_i, a_j\}$ with $a_j \in A_1$ and $i \in \{1, \ldots, f\} \setminus \{j\}$, G' has an edge between a_j

and $\{v, a_i, b_i\}$, since $G'[F]$ is connected. Note that if G' contains the edge $\{b_i, a_j\}$, then we can replace this edge by $\{v, a_j\}$: Since $j \neq i$, there is only one hyperedge, namely F, that contains both b_i and a_j. Clearly, $G'[F]$ can also be made connected by adding $\{v, a_j\}$ instead. This implies an optimal support with $k-1$ vertices in A_1, contradicting our choice of k. Hence, G' contains no edges $\{b_i, a_j\}$ with $i \neq j$. Consequently, in order to make each hyperedge $\{v, a_i, b_i, a_j\} \in \mathscr{E}_3'$ with $a_j \in A_1$ connected, there is an edge between a_i and a_j.

Hence, G' has $k\cdot(f-k)$ edges between A_1 and A_2, $\binom{k}{2}$ edges between vertices in A_1 and $f-k$ further edges between v and A_2. Altogether the total number of edges in G' is thus at least

$$\begin{aligned} & 2f + k\cdot(f-k) + \binom{k}{2} + f - k \\ = \; & 3f + \frac{k\cdot(2f-k-3)}{2} \;\geq\; 3f + \frac{k\cdot(f-3)}{2} \;\geq\; 3f. \end{aligned}$$

This implies that $\mathscr{H}'$ does not have a support with feedback edge number f whereas $\mathscr{H}$ does. ☐

As one can see from the proof, the main reason for the incorrectness of Rule 3.1 is the incorrect assumption that there is an optimal support which adds an edge between twins. However, with some additional conditions, rules similar to Rule 3.1 are correct (Rules 3.2 to 3.4, 3.6, and 3.8 below). First, if a vertex u is adjacent to some vertex v covering u in an optimal support, then there is an optimal support that shifts all other edges incident with u to v.

Lemma 3.3. Let u, v be two vertices in a hypergraph $\mathscr{H}$ with v covering u. If $\mathscr{H}$ has an optimal support G containing the edge $\{u, v\}$, then $\mathscr{H}$ also has an optimal support with u being adjacent only to v.

Proof. Assume that u has degree at least two in G, that is, u has some neighbor $w \neq v$. Then, obtain a modified graph G' by replacing the edge $\{u, w\}$ by the edge $\{v, w\}$. The modified graph G' has the same number of edges as G. Furthermore, the two endpoints of the removed edge $\{u, w\}$ are still connected in each hyperedge that contains them: Since v covers u, such a hyperedge also contains v, which is a common neighbor of u and w in G'. Hence, G' is also an optimal support. ☐

The above lemma immediately leads to the following data reduction rule.

Rule 3.2. If hypergraph $\mathscr{H} = (V, \mathscr{E})$ contains vertices u, v such that v covers u and if there is an optimal support G containing the edge $\{u, v\}$, then remove u from $\mathscr{H}$.

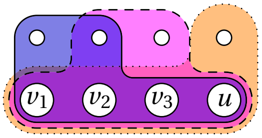


Figure 3.3.: A hypergraph to which Rule 3.3 applies with $q = 3$: Each vertex v_i covers u, and each hyperedge incident with u has size at most $q + 3 = 6$.

Of course, it is not clear how to efficiently determine whether such an optimal support exists. We later exhibit hypergraph structures that enforce the precondition of Rule 3.2 and then use it as a subroutine.

Lemma 3.4. Rule 3.2 is correct.

Proof. If there is a support of feedback edge number at most f for $\mathcal{H}$, then the precondition of the Rule 3.2 and Lemma 3.3 imply that there is a support G of feedback edge number at most f in which u has degree one and is adjacent to v. Then, $G - \{u\}$ is a support of feedback edge number at most $f - 1$ for $\mathcal{H}$ with u removed. Conversely, if there is a support G' of feedback edge number at most $f - 1$ for $\mathcal{H}$ with u removed, then adding the edge $\{u, v\}$ gives a support of feedback edge number at most f for $\mathcal{H}$ since all hyperedges F containing u also contain v and the corresponding subhyperedges $F \setminus \{u\}$ induce connected subgraphs of G'. □

Note that the correctness of Rule 3.2 together with Lemma 3.2 implies that there are instances in which a twin class is an independent set in every optimal support.

In the counterexample to Rule 3.1, there are only two twins and they are contained in hyperedges of size five, that is, the size-five hyperedges containing these two vertices have three other "unrelated" vertices. In the following, we show that the counterexample is tight in the sense that, if each hyperedge containing u also contains at most *two* vertices that do not cover u, then Rule 3.1 is correct.

Rule 3.3. Let $q \in \mathbb{N}$. If there are $q + 1$ vertices u and $v_1, \ldots, v_q$ in the hypergraph $\mathcal{H} = (V, \mathcal{E})$ such that for each $i \in \{1, \ldots, q\}$, vertex v_i covers u and, if for each hyperedge $F \in \mathcal{E}(u)$ it holds that $|F| \leq q + 3$, then remove u from $\mathcal{H}$.

See Figure 3.3 for a hypergraph to which Rule 3.3 applies with $q = 3$. Another example is the hypergraph $\mathcal{H}$ defined in Lemma 3.2 without the hyperedges in $\mathcal{E}_3 \cup \mathcal{E}_4$: Then, $q = 1$ and all hyperedges are of size at most four.

Lemma 3.5. Rule 3.3 is correct and can be applied exhaustively in $O(n \cdot |\mathcal{H}|)$ time.

Proof. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph, let Q be the vertex set $\{v_1, \ldots, v_q\} \subseteq V$, and let u be a vertex in V such that Rule 3.3 applies. Further, let N denote the set of neighbors of u in an optimal support G. If $N \cap Q \neq \emptyset$, then the correctness of Rule 3.3 follows from the correctness of Rule 3.2. Hence, we assume that $N \cap Q = \emptyset$. We distinguish two cases.

Case 1: N contains a neighbor w of some $v \in Q$ in G. Let G' be the result of removing the edge $\{u, w\}$ from G and adding $\{u, v\}$. If there is some hyperedge F such that $G'[F]$ is disconnected, then $u, w \in F$. Since v covers u, also $v \in F$. Then, there is a path between u and w via v in $G'[F]$, implying that $G'[F]$ is connected.

Case 2: N contains no neighbor of any $v \in Q$. Then, for each $F \in \mathcal{E}(u)$, $|F \cap N| \leq 1$ since, otherwise, $|F| \leq q + 3$ implies that $F = \{u\} \cup Q \cup (F \cap N)$ and then $G[F]$ does not contain a path between u and any vertex in Q. Thus, $G' := G - \{u\}$ is a support for the hypergraph $\mathcal{H}'$ that results from $\mathcal{H}$ by removing u. Note that G' has at least one edge less than G. Since all hyperedges of $\mathcal{H}$ that contain u are supersets of Q, adding u with the edge $\{v_1, u\}$ to G' results in a support for $\mathcal{H}$, which is optimal since $|E(G')| + 1 \leq |E(G)|$.

Hence, Rule 3.3 is correct. It remains to prove the running time bound. For a vertex u, let Q be the set of vertices covering u. It is not hard to see that if Rule 3.3 applies to any subset of Q, then it also applies to Q. Reversing all arcs in the covering graph G_C of $\mathcal{H}$ allows us to compute $|Q|$ in constant time per vertex. Thus, assuming that the size of a hyperedge can be computed in constant time, Rule 3.3 can be applied exhaustively to $\mathcal{H} = (V, \mathcal{E})$ in $O(n \cdot |\mathcal{H}| + n \cdot \sum_{u \in V} |\mathcal{E}(u)|) = O(n \cdot |\mathcal{H}|)$ time. To compute the size of a hyperedge in constant time, we initialize once an array mapping hyperedges to their sizes in $O(|\mathcal{H}|)$ time. Whenever we remove a vertex, we update this array. The running time for all update steps amounts to $O(|\mathcal{H}|)$. $\square$

As a corollary of Lemma 3.5, we also obtain correctness of the following rule since it is a special case of Rule 3.3. This rule will be useful in Section 3.6.1 where we prove a linear-vertex kernel with respect to the feedback edge number of the support for SUBSET INTERCONNECTION DESIGN with hyperedges of size at most 4.

Rule 3.4. If there are two vertices u and v such that v covers u and $|F| \leq 4$ for each hyperedge $F \in \mathcal{E}(u)$, then remove u from $\mathcal{H}$.

Note that the condition $|F| \leq 4$ in Rule 3.4 is also tight in the sense that, if u is incident with hyperedges of size at least five, then Rule 3.4 is not correct as shown by the hypergraph constructed in Lemma 3.2.

3.4. Data reduction rules for instances with few hyperedges

In this section, we show that SUBSET INTERCONNECTION DESIGN is fixed-parameter tractable with respect to the number m of hyperedges. A previous fixed-parameter tractability result for this parameter [Hos+12, Theorem 8] relied on Rule 3.1 and is therefore incorrect.[6] The intuition behind Rule 3.1 is that it is optimal to connect a twin class by a spanning tree and to subsequently represent this twin class by a single vertex. This approach results in an instance with at most 2^m vertices which would imply fixed-parameter tractability. As the counterexample in Lemma 3.2 shows, a twin class can be disconnected in the subgraph induced by every optimal support. Thus, in order to restore the fixed-parameter tractability result, we need a slightly more involved rule whose correctness proof makes use of the following upper bound on the number of edges needed in the support.

Lemma 3.6. Every instance of SUBSET INTERCONNECTION DESIGN with $f \geq \binom{2^m}{2}$ is a yes-instance.

Proof. We show how to construct a support G with less than $\binom{2^m}{2} + n$ edges. Note that such a support G has feedback edge number at most $\binom{2^m}{2}$. The main idea is to first partition the vertex set into two subsets V' and $V \setminus V'$. We then construct a support G' for the hypergraph $\mathcal{H}'$ resulting from $\mathcal{H}$ by removing the vertices in $V \setminus V'$. Finally, we obtain a support for $\mathcal{H}$ by adding each time one edge connecting each vertex in $V \setminus V'$ to the graph G'.

Recall that a twin class is a maximal set of vertices that mutually cover each other. Let V' consist of exactly one vertex from each twin class of $\mathcal{H}$. Clearly, $|V'| \leq 2^m$ where m is the number of hyperedges. Let $\mathcal{H}'$ be the hypergraph resulting from removing all vertices that are not in V' from $\mathcal{H}$. Obviously, a complete graph G' for V' is a support for hypergraph $\mathcal{H}'$. This support has $\binom{2^m}{2}$ edges.

We now extend G' to a support G for $\mathcal{H}$ as follows. Add each vertex $v \in V \setminus V'$ to G'. Furthermore, add an edge incident to v and its twin in V' (recall that at least one vertex of each twin class is in V'). Let G be the resulting graph. Since there is at least one twin class, $|V \setminus V'| \leq n-1$ and hence G contains at most $\binom{2^m}{2} + n - 1$ edges.

[6] The theorem of Hosoda et al. [Hos+12] states that an optimal support can be computed in polynomial time if $m \leq \phi(n)$ where ϕ is some specific function. This is equivalent to fixed-parameter tractability: if $m \leq \phi(n)$, then one can apply the polynomial-time algorithm, otherwise $m > \phi(n)$ implies that the instance size depends only on m. Using our modified twin reduction rule (Rule 3.6 below), the result of Hosoda et al. [Hos+12] has since been restored [Hos+15].

Graph G is a support for $\mathcal{H}$ since for each hyperedge $F \in \mathcal{E}$, the subgraph $G[F]$ is connected: $G[V' \cap F]$ is a complete graph and each vertex in $F \setminus V'$ is adjacent to its twin in $F \cap V'$. Thus, $G[F]$ is connected. □

The upper bound provided by Lemma 3.6 grows exponentially in the number of hyperedges. It would be interesting to replace this exponential dependence by a polynomial function. However, this is not possible without further reduction rules; there are instances that require a support with feedback edge number at least $2^{\Omega(m)}$, see Section 3.5.

Lemma 3.6 directly yields the correctness of the following data reduction rule.

Rule 3.5. If $f \geq \binom{2^m}{2}$, then answer yes.

Our aim is now to shrink the size each of the 2^m twin classes so that it is bounded from above by a function of m. For this, the following rule removes vertices from large twin classes.

Rule 3.6. Let $(\mathcal{H}, f)$ be an instance that is reduced with respect to Rule 3.5. If there is a twin class T in $\mathcal{H}$ with $|T| > 4^m + 7 \cdot 2^m + 1$, then remove an arbitrary vertex $v \in T$ from $\mathcal{H}$.

In the following, by *degree-ℓ vertices* we refer to vertices of degree exactly ℓ. To prove the correctness of Rule 3.6, we show that there is a support G that has the following property concerning its low-degree vertices.

Lemma 3.7. Let $\mathcal{H} = (V, \mathcal{E})$ be a connected hypergraph and $|V| \geq 3$. Then, there is a support $G = (V, E)$ such that for each twin class T of $\mathcal{H}$, graph G has
(i) at most one vertex $t \in T$ that has degree-one neighbors in G, and
(ii) at most one degree-two vertex $t' \in T$ in G.

Proof. Let G be a support for $\mathcal{H}$. We show how to transform G into a support G^* which fulfills both properties of the lemma. First, suppose that there is a nonempty set $\mathcal{T} = \{T_1, \dots, T_q\}$ of twin classes of $\mathcal{H}$ such that at least two vertices of each T_i have degree-one neighbors in G. Now, modify G as follows. Pick an arbitrary vertex t_i from each T_i such that t_i has a degree-one neighbor and label t_i as the *one* vertex in twin class T_i that will have degree-one neighbors in the modified support G^*.

Then, as long as $\mathcal{T}$ is nonempty, pick an arbitrary T_i and an arbitrary vertex $t' \in T_i \setminus \{t_i\}$. Let X be the set of degree-one neighbors of t' in G. Note that $t_i \notin X$ as, otherwise, t_i and its neighbor would form a connected component of size two in G. This contradicts the fact that G is a support because by definition $\mathcal{H}$ is connected and contains at least three vertices. We remove all edges between X and t' and make all vertices in X adjacent to t_i. Let G' be the resulting graph. Observe that G' has the same number of edges as G. Moreover, G' is also a support: The vertices of X were not part of any shortest path between two vertices in $V \setminus X$. Hence, $G[V \setminus X]$ is a support for the hypergraph obtained from $\mathcal{H}$ by removing each vertex in X. In particular, for each $F \in \mathcal{E}$ containing t_i, vertex t_i is connected to all other vertices in $G[F \setminus X]$. Therefore, adding each vertex in X as a degree-one neighbor to t_i (which results in the graph G') produces a support for $\mathcal{H}$.

The above shows the correctness of the first modification step. After this step we update the set $\mathcal{T}$ by performing the modification step, again for some arbitrary twin class of $\mathcal{H}$ that contains at least two vertices with degree-one neighbors in G'. We repeat this process until $\mathcal{T}$ is empty. In order to show that we can obtain a support G^* in which $\mathcal{T}$ is empty, we need to show that we only have to perform a finite number of modification steps. To this end, note that in each step the degree of the labeled vertex $t_i \in T_i$ increases and only the degree of t' decreases. Since t' is a twin of t_i, the degree of all other labeled vertices stays the same. Hence the number of modification steps is finite. Summarizing, there is a support that fulfills Property (i) of the lemma.

We now show that such a support can be further modified such that it also fulfills Property (ii). Let G be a support that fulfills Property (i) and let $\mathcal{T} = \{T_1, \dots, T_q\}$ denote the twin classes for which there are at least two degree-two vertices in G. For each T_i do the following. Since the first property is satisfied, T_i contains at least one degree-two vertex u such that its two neighbors a and b have degree at least two, respectively. Then, remove the two edges incident with u, make a and b adjacent, and make u adjacent to either the one vertex in T_i that has degree-one neighbors, or, if such a vertex does not exist, another degree-two vertex in T_i. See Figure 3.4 for an illustration of this modification step. Note that the neighbors of t_i and u do not have to be contained in T_i. Nevertheless, for each hyperedge F the connected component of a and b is connected to the one of t_i (if it exists) in $G[F]$, which we now exploit to show the correctness of the modification step.

Let G' be the modified graph and let v denote the new neighbor of u in G'. Clearly, G' has the same number of edges as G. To show that G' is also a support, we consider each $F \in \mathcal{E}$ that contains u and at least one of a and b and show that $G'[F]$ is connected. We distinguish two cases.

Case 1: Exactly one of a and b is in F. Then, u has degree one in $G[F]$, and thus $G[F] - \{u\}$ is connected. In particular, the twin v of u has a path to all vertices in $G[F] - \{u\}$. Hence, “reinserting” u and making it adjacent to v results in a connected graph isomorphic to $G'[F]$.

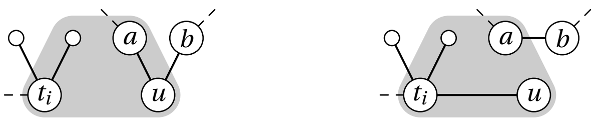


Figure 3.4.: A possible situation before (left) and after the second modification step (right) in the proof of Lemma 3.7. The gray region represents a twin class.

Case 2: Both a and b are in F. Since $G[F]$ is connected, the graph G'' that is obtained from $G[F]$ by removing u and making a and b adjacent is also connected. Again, this means that the twin v of u (in T_i) has in G'' a path to all other vertices. As above, "reinserting" u and making it adjacent to v yields a connected graph isomorphic to $G'[F]$.

Note that since neither a nor b are degree-one vertices and by the choice of v, the above modification does not result in a support in which a twin class has more than one vertex that is a neighbor of degree-one vertices. Consequently, the modification can be applied to all T_i's without losing the first property. Hence, there is a support fulfilling both properties. □

With the above lemma at hand, we can show the correctness of Rule 3.6. The outline of the proof is as follows. Using Lemma 3.7, we show that only $O(2^m)$ vertices of every twin class T have degree at least three but at least one low-degree neighbor. Consequently, for sufficiently large $|T|$ we can assume that one vertex of T has degree one in G: Otherwise, there are many degree-(≥ 3) vertices in G that have only degree-(≥ 3) neighbors in G. This, however, pushes the feedback edge number of G above the guarantee given by Rule 3.5.

Lemma 3.8. Rule 3.6 is correct.

Proof. Let $(\mathcal{H}, f)$ denote the original instance and $(\mathcal{H}', f)$ an instance resulting from one application of Rule 3.6. We show that both instances are equivalent. It is easy to see that if $(\mathcal{H}', f)$ is a yes-instance, then $(\mathcal{H}, f)$ is also a yes-instance: Let G' be the support for $\mathcal{H}'$. Pick one vertex $u \in T \setminus \{v\}$, make v adjacent to u, and call the resulting graph G. Note that G has feedback edge number f. Then, G has one more edge than G'. Moreover, G is also a support since for each $F \in \mathcal{E}$ containing v, the subgraph $G'[F \setminus \{v\}]$ is connected. This implies that $G[F]$ is also connected (since $F \setminus \{v\}$ contains u).

For the other direction of the equivalence, suppose that G is a support for $\mathcal{H}$. Since $\mathcal{H}$ is reduced with respect to Rule 3.5, graph G has feedback edge number $f < \binom{2^m}{2}$. Let twin class T and vertex v be as described in Rule 3.6. To show that $(\mathcal{H}', f)$ is also a yes-instance, we show, using $|T| > 4^m + 7 \cdot 2^m + 1$, that hypergraph $\mathcal{H}$ has also a support with feedback edge number f in which vertex v has degree one.

First, we prove that there are at most $4^m + 7 \cdot 2^m$ vertices whose degree in G is at least three. Obviously, this upper-bounds the number of vertices with degree at least three in the twin class T as well. We consider two subsets of this vertex set: by X we denote the vertices of degree at least three that have only neighbors of degree at least three in G, and by Y we denote the other vertices of degree at least three. Note that Lemma 3.7 implies that Y has at most $(1+2) \cdot 2^m = 3 \cdot 2^m$ vertices: First, the number of twin classes is at most 2^m. Second, in G each twin class has at most one vertex that has degree-one neighbors. Finally, in G each twin class has at most one degree-two vertex. Hence, there are at most $2 \cdot 2^m$ neighbors of degree-two vertices.

It remains to upper-bound the size of X. We do this by deriving a lower bound on the number of edges in G and then show that, for large X, this lower bound exceeds $|V| - 1 + f$, contradicting the fact that $(\mathcal{H}, f)$ is a yes-instance. To this end, let $Z = V \setminus (X \cup Y)$ denote the set of vertices that have degree one or two in G. We partition the edges of G into two subsets: the set $E_{X \cup Y}$ which contains edges with both endpoints on vertices of degree at least three, and the set $E_{Y \cup Z}$ which contains all remaining edges. Since we assume that G is connected, $|E_{Y \cup Z}| \geq |Z| - 1$. The number of edges in $E_{X \cup Y}$ is at least $3|X|/2$ since all vertices in X have degree at least three and only neighbors in $X \cup Y$. If $|X| \geq 2 \cdot \left(\binom{2^m}{2} + 3 \cdot 2^m\right)$, then

$$|E_{X \cup Y}| \;\geq\; \frac{3|X|}{2} = |X| + \frac{|X|}{2} \geq |X| + |Y| + \binom{2^m}{2},$$

where the last inequality holds because $|Y| \leq 3 \cdot 2^m$. Hence, the number of edges in G is

$$|E_{X \cup Y}| + |E_{Y \cup Z}| \;\geq\; |X| + |Y| + \binom{2^m}{2} + |Z| - 1 \;=\; n + \binom{2^m}{2} - 1.$$

This contradicts the assumption that $(\mathcal{H}, f)$ is a yes-instance, since $f < \binom{2^m}{2}$ after application of Rule 3.5. Hence, we have $|X| < 2 \cdot (\binom{2^m}{2} + 3 \cdot 2^m) = 4^m + 5 \cdot 2^m$.

Now we can upper-bound the number of vertices in the twin class T that have degree at least two. In addition to the vertices of X, class T can contain at most one vertex that is a neighbor of degree-one vertices, at most $2 \cdot 2^m$ vertices that are neighbors of degree-two vertices, and at most one degree-two vertex. Therefore, if T contains more than $4^m + 7 \cdot 2^m + 1$ vertices, then at least one of them is a degree-one vertex in G. Without loss of generality, we can assume that this is v. □

It now only remains to combine the above results to obtain a problem kernel for Subset Interconnection Design parameterized by the number m of hyperedges. Moreover, this kernel can be computed in linear time, thus yielding linear-time fixed-parameter tractability.

Theorem 3.1. An instance of Subset Interconnection Design be reduced to an equivalent one of size at most $O(8^m \cdot m)$ in $O(|\mathcal{H}|)$ time.

Proof. The kernelization algorithm first applies Rule 3.5 and then exhaustively applies Rule 3.6. After this, the size of each twin class in $\mathcal{H}$ is at most $O(4^m)$. Hence, the instance has size $O(8^m \cdot m)$: The input hypergraph $\mathcal{H}$ has at most 2^m twin classes, each containing $O(4^m)$ vertices. Therefore, the total number n of vertices is $O(8^m)$. The overall instance size follows.

The running time of the kernelization algorithm can be upper bounded as follows. Clearly, Rule 3.5 runs in $O(|\mathcal{H}|)$ time. In order to apply Rule 3.6, we first compute a partition of V into the twin classes. This can be done as follows. We start with one set containing V and then consider an arbitrary hyperedge $F \in \mathcal{E}$. The vertices that are in F are in a different twin class than the vertices that are not in F. Hence, using F we partition V into two subsets. We repeat the partitioning for all hyperedges, each time using the current hyperedge to update the partition. The partitioning can be done in $O(|F|)$ time [HPV99, Lemma 1]. This process thus takes $O(|\mathcal{H}|)$ time. Its output is a partition of V into all different twin classes. For each twin class, we check whether Rule 3.6 can be applied. Instead of applying the rule right away, we label the vertices of the twin classes that will be removed and decrease k by the overall number of labeled vertices. After all twin classes have been processed, we remove all labeled vertices from each $F \in \mathcal{E}$, again in linear time. Thus, the overall running time is $O(|\mathcal{H}|)$. □

By applying a brute-force search on the problem kernel, we obtain the following.

Corollary 3.1. Subset Interconnection Design be solved in $2^{O(m8^m)} + O(|\mathcal{H}|)$ time.

Proof. Applying Theorem 3.1 we get from our input instance obtain in $O(|\mathcal{H}|)$ time an equivalent instance with at most $c8^m$ vertices for some constant c. Hence, by Lemma 3.6, we can assume that the number k of edges in any support is less than

$$\binom{2^m}{2} + c8^m \leq 4^m + c8^m \leq (c+1)8^m.$$

Now we try all subsets of k edges out of the at most $\binom{c8^m}{2} \leq c^2 8^{2m}$ possible edges on at most $c8^m$ vertices. For each of them we test whether it is a support in $O(mk) = O(m8^m)$ time. Since there are at most

$$\binom{c^2 8^{2m}}{k} \leq \binom{c^2 8^{2m}}{(c+1)8^m} \leq (c^2 8^{2m})^{(c+1)8^m} = 2^{O(m8^m)}$$

such sets, we get the $2^{O(m8^m)}$ bound for the running time of this part of the algorithm. □

3.5. A family of hypergraphs requiring supports with large feedback edge number

In this section we show that the bound given by Lemma 3.6 and used in Section 3.4 is optimal by giving an almost matching lower bound with respect to m on the size of a feedback vertex set. To this end, we construct a hypergraph $\mathcal{H}$ with $m = 2q+1$ hyperedges $F_1, \dots, F_m$, where $q \geq 2$, such that each vertex is in exactly q hyperedges and each intersection of any q hyperedges consists of exactly one vertex: For each vector in $\{0,1\}^m$ with exactly q one-entries we introduce one vertex. Hence, there are overall $n = \binom{m}{q}$ vertices $v_1, \dots, v_n$. Each entry in the vector of a vertex corresponds to one hyperedge, that is, a hyperedge F_i, $i \in \{1, \dots, m\}$, consists of those vertices whose vector has a one-entry in the ith position and $q-1$ one-entries in the other $m-1 = 2q$ positions. For each i, there are $\binom{2q}{q-1}$ such vectors. Hence, each hyperedge contains exactly $\binom{2q}{q-1}$ vertices. This completes the construction. Note that in each support for the resulting hypergraph $\mathcal{H}$ the subgraph induced by any hyperedge must contain at least $\binom{2q}{q-1} - 1$ edges.

Suppose that graph G is a support for $\mathcal{H}$. If each edge in G is contained in exactly one induced subgraph $G[F_i]$ for some hyperedge F_i, then G would need

$$(2q+1) \cdot \left(\binom{2q}{q-1} - 1 \right)$$

edges. Each edge $\{u, v\}$ in G can be contained in at most $q-1$ subgraphs $G[F_i]$, $1 \leq i \leq m$ (corresponding to the positions where both u and v have one-entries in their vector representations). Therefore, each support contains at least

$$k_{\text{opt}} \geq \frac{2q+1}{q-1} \cdot \left(\binom{2q}{q-1} - 1 \right)$$

edges. Now we compare this to the number of vertices. We have

$$\begin{aligned}
k_{\text{opt}} - n &\geq \frac{2q+1}{q-1} \cdot \left(\binom{2q}{q-1} - 1 \right) - \binom{2q+1}{q} \\
&= \frac{1}{q-1} \left(\frac{q \cdot (2q+1)!}{q! \cdot (q+1)!} - \frac{(q-1) \cdot (2q+1)!}{q! \cdot (q+1)!} \right) - \frac{2q+1}{q-1} \\
&= \frac{1}{q-1} \left(\frac{(2q+1)!}{q! \cdot (q+1)!} \right) - \frac{2q+1}{q-1} \\
&= \frac{1}{q-1} \binom{2q+1}{q} - \frac{2q+1}{q-1} \\
&= \frac{2q+1}{(q-1)(q+1)} \binom{2q}{q} - \frac{2q+1}{q-1}.
\end{aligned}$$

Using Proposition 1.1 and $q \geq 2$, we obtain

$$\begin{aligned}
k_{\text{opt}} - n &\geq \frac{2q+1}{(q-1)(q+1)} \cdot \frac{2^{2q-1}}{\sqrt{q}} - 5 \\
&> \frac{2^{2q}}{(q+1)\sqrt{q}} - 5.
\end{aligned}$$

Thus, $k_{\text{opt}} \geq 2^{\Omega(q)} + n = 2^{\Omega(m)} + n$. Succinctly, we have shown the following.

Proposition 3.1. For each odd integer $m \geq 5$ there is a hypergraph $\mathcal{H}$ with m hyperedges such that each support for $\mathcal{H}$ contains at least $2^{\Omega(m)} + n$ edges, where n is the number of vertices in $\mathcal{H}$.

3.6. Data reduction rules for sparse supports

In this section, we present a set of reduction rules that identify and remove parts of the instance which either produce tree-like induced subgraphs or long induced paths in the support. We analyze the power of these data reduction rules by showing that, for maximum hyperedge size $d \le 4$, they yield a problem kernel for the parameter feedback edge number f and that, for general d, they can be used to obtain fixed-parameter tractability for the combined parameter (d, f). The reason for obtaining a weaker result for $d > 4$ is that Rule 3.1 is indeed correct if $d \le 4$ (see Rule 3.4) but not in general. Applying this rule removes a certain class of vertices from the input graph that seem to be hard to identify for $d > 4$.

Although f appears in our data reduction rules, they are applicable regardless of the value of f. Hence, the data reduction rules can be applied also to the optimization version of SUBSET INTERCONNECTION DESIGN.

3.6.1. A problem kernel for the parameter feedback edge number f for $d \le 4$

We now describe how we can remove all but $O(f)$ vertices from a SUBSET INTERCONNECTION DESIGN instance with $d \le 4$ in $O(n \cdot m^3)$ time by using Rule 3.4 (Section 3.3) and an additional reduction rule (Rule 3.7 below). Informally, the parameter f upper-bounds the number of vertices that are in cycles and have degree at least three. Using Rule 3.4 we can remove vertices that have degree one in supports. Hence, to reduce the overall number of vertices to $O(f)$, we also have to remove vertices that are in long induced paths in the support. This is the purpose of Rule 3.7. This rule is also needed in Section 3.6.2 which deals with the case $d \ge 5$. Therefore, we formulate the rule in a more general way than needed for the special case $d \le 4$.

Rule 3.7. Let $(\mathcal{H} = (V, \mathcal{E}), f)$ be an instance of SUBSET INTERCONNECTION DESIGN. If $\mathcal{H}$ contains a vertex set $P := \{p_0, \dots, p_{2d}\}$ with the set $\mathcal{E}_P := \bigcup_{p \in P} \mathcal{E}(p)$ of incident hyperedges such that

(i) no $p_i \in P$ covers any $p_j \in P$ with $j \neq i$,
(ii) for each $F \in \mathcal{E}_P$ we have $F \cap P = \{p_i, \dots, p_j\}$ for some $0 \le i \le j \le 2d$,
(iii) for each $F \in \mathcal{E}_P$ with $F \cap \{p_0, p_{2d}\} = \emptyset$, and for every vertex $v \in F \setminus P$, there is a vertex $p \in P$ that covers v, and
(iv) there is no hyperedge $F \in \mathcal{E}_P$ such that $F \cap P = \{p_i\}$ for any $0 < i < 2d$,

then do the following.

For every $F \in \mathcal{E}_P$ with $F \cap \{p_0, p_{2d}\} = \emptyset$, remove all vertices in $F \setminus P$ from $\mathcal{H}$. Furthermore, remove the vertices $p_2, \dots, p_{2d-2}$ from $\mathcal{H}$ and decrease f by $d - 1$.

Intuitively, Conditions (i) and (ii) plus the fact that the hyperedges have size at most d enforce that there is a support G that makes $G[P]$ a long induced path with

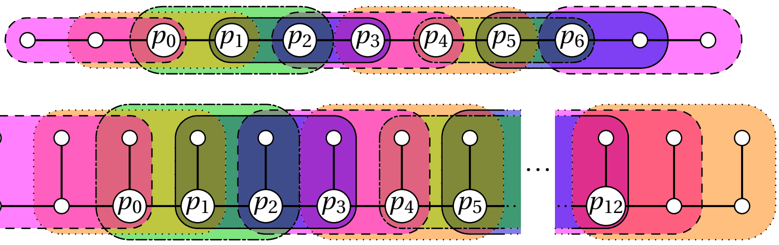


Figure 3.5.: Two hypergraphs to which Rule 3.7 applies along with optimal supports.

endpoints p_0 and p_{2d}. Below, we use $P_I := \{p_1, \ldots, p_{2d-1}\}$ to denote the set of inner vertices of this path structure. Condition (iii) ensures that in the support in which $G[P]$ is a path all vertices that are in hyperedges with only vertices of P_I can be made degree-one neighbors of some vertex of P_I.

Figure 3.5 shows two examples of hypergraphs with maximum hyperedge size $d = 3$ and $d = 6$, respectively, to which Rule 3.7 applies.

In order to prove the correctness and running time of the rule, we first make some observations on the structure of the subhypergraph that consists of the hyperedge set $\mathcal{E}_P$. The first observation is about the structure of the hyperedges along the presumed path containing P.

Observation 3.2. Let $\mathcal{H}$ be a hypergraph and $P \subseteq V$ satisfying the conditions of Rule 3.7. For every $p_i \in P_I$ there is a hyperedge F_i^+ such that $p_{i-1} \notin F_i^+$ and $\{p_i, p_{i+1}\} \subseteq F_i^+$ and also a hyperedge F_i^- such that $\{p_{i-1}, p_i\} \subseteq F_i^-$ and $p_{i+1} \notin F_i^-$. Moreover, there is a hyperedge F_0^- such that $F_0^- \cap P = \{p_0\}$ and a hyperedge F_{2d}^+ such that $F_{2d}^+ \cap P = \{p_{2d}\}$.

Proof. Consider some $i \in \{1, \ldots, 2d-1\}$. By Condition (i) of Rule 3.7, there is some hyperedge containing p_i but not p_{i-1}. Now, by Condition (iv), this hyperedge contains at least one further vertex from P, and by Condition (ii) it thus contains p_{i+1}. Hence, there is a hyperedge F_i^+ containing p_i and p_{i+1} that does not contain p_{i-1}. The existence of F_i^- follows from a symmetric argument. Finally, Condition (i) implies the existence of F_0^- and F_{2d}^+. □

For the second observation we consider the hyperedges that only intersect with the set of inner vertices P_I. To this end, let $\mathcal{E}_I = \{F \in \mathcal{E}_P \mid F \cap P \subseteq P_I\}$ be the hyperedges of $\mathcal{E}_P$ that contain neither p_0 nor p_{2d} and let $W := \bigcup_{F \in \mathcal{E}_I} F$ be the union of all vertices that are incident with some hyperedge in $\mathcal{E}_I$. Due to Observation 3.2,

$\mathcal{E}_I$ is not empty and $W \cap P = P \setminus \{p_0, p_{2d}\}$. Each vertex v in $W \setminus P$ is covered by some vertex of $P \setminus \{p_0, p_{2d}\}$: By the definition of W, there is a hyperedge $F \in \mathcal{E}_I$ with $F \cap \{p_0, p_{2d}\} = \emptyset$ that is incident with v. By Condition (iii), v is covered by some $p_i \in P$. Since $F \cap \{p_0, p_{2d}\} = \emptyset$, we have $p_i \neq p_0$ and $p_i \neq p_{2d}$. Altogether, W thus has the following property.

Observation 3.3. For each $v \in W$ either $v \in P_I$ or there is a $p_i \in P_I$ such that p_i covers v.

Informally, the above two observations imply that the subhypergraph $\mathcal{H}[W]$ has a support in which P_I is an induced path and all other vertices of W have degree one. The final observation is used to show that this support for $\mathcal{H}[W]$ is also optimal. It is based on the fact that any support for a connected input hypergraph is a connected graph.

Observation 3.4. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph and let G be a support for $\mathcal{H}$. If the subhypergraph $\mathcal{H}[V']$ induced by a vertex subset $V' \subseteq V$ is connected, then $|E(G[V'])| \geq |V'| - 1$.

Using these observations, we can now show the correctness of the rule.

Lemma 3.9. Rule 3.7 is correct.

Proof. Let $\mathcal{H}$, P and $\mathcal{E}_P$ be as described in Rule 3.7 and, as above, denote $P_I := \{p_1, \ldots, p_{2d-1}\}$, $\mathcal{E}_I := \{F \in \mathcal{E}_P \mid F \cap P \subseteq P_I\}$, and $W := \bigcup_{F \in \mathcal{E}_I} F$. Consider an arbitrary optimal support G' and let G be the graph obtained from G' by removing all edges in $G'[W]$ and then adding the edges $\{p_i, p_{i+1}\}$ for $i \in \{1, \ldots, 2d-2\}$, and, for every $v \in W \setminus P$ adding one edge $\{v, p_i\}$ where p_i covers v. We prove that G is an optimal support.

First, we show that G is a support, that is, $G[F]$ is connected for all $F \in \mathcal{E}$. For this, let $F \in \mathcal{E}$ and $F_W = F \cap W$. Observe that if $|F_W| < 2$, then $G[F]$ is connected. Hence, assume $|F_W| \geq 2$ and let $u, v \in F_W$ such that $\{u, v\} \in E(G')$. We show that u and v are connected in $G[F_W]$. This directly implies that $G[F]$ is connected, since any path in $G'[F]$ using $\{u, v\}$ can then use the path between u and v in $G[F_W]$ instead. By Observation 3.3, we have $F_W \cap P \supseteq \{p_i, p_j\}$ for some $1 \leq i \leq j \leq 2d-1$, and Condition (ii) yields $F_W \cap P \supseteq \{p_i, p_{i+1}, \ldots, p_j\}$. Without loss of generality, we may assume that either $u = p_i$ or $\{u, p_i\} \in E(G)$. Similarly, we may assume that either $v = p_j$ or $\{v, p_j\} \in E(G)$. Hence, it suffices to show that $G[\{p_i, \ldots, p_j\}]$ is connected. This is directly implied by the construction of G. Thus, there is a path between u and v and G is a support.

We now prove the optimality of G. For this, we first show that the induced subhypergraph $\mathcal{H}[W] = (W, \mathcal{E}_I)$ is connected, which implies a lower bound on the number of edges in $G'[W]$. Let $u, v \in W$. We construct a hyperwalk between u and v using only hyperedges of $\mathcal{E}_I$. By Observation 3.3, it suffices to show that there is a hyperwalk between any p_i, p_{i+1}, $1 \le i \le 2d-2$. Assume without loss of generality that $i \le d$. By Observation 3.2 there is some hyperedge $F \in \mathcal{E}$ with $p_i, p_{i+1} \in F$ and $p_{i-1} \notin F$ for each $i \in \{1, \ldots, 2d-1\}$. Condition (ii) states that $F \cap P = \{p_i, \ldots, p_j\}$ and, since $|F| \le d$, we have $j \le i + d - 1 < 2d$. Hence, $F \in \mathcal{E}_I$ and $\mathcal{H}[W]$ is connected.

The facts that hypergraph $\mathcal{H}[W]$ is connected and Observation 3.4 imply that $G'[W]$ contains at least $|W| - 1$ edges. The graph $G[W]$ also contains $|W| - 1$ edges. Since W contains all vertices p_i for $1 \le i \le 2d-1$, by the construction of graph G', graphs G' and G share all edges not contained in $G[W]$. Hence, G is optimal.

We have thus shown that there is an optimal support G where each vertex $v \in W \setminus P$ is adjacent to a vertex in P that covers v. Combining this with Rule 3.2 we conclude that removing any vertex in $W \setminus P$ from the hypergraph results in an equivalent instance. Hence, in the following we assume that $W \subseteq P$, and hence $W = \{p_1, \ldots, p_{2d-1}\}$. It remains to show that the remaining modifications for $p_2, \ldots, p_{2d-2}$ are correct.

Let $(\mathcal{H}, f)$ be the original instance and let $(\mathcal{H}', f')$ be the instance resulting from an application of Rule 3.7 to $(\mathcal{H}, f)$. Note that $f' = f - d + 1$ since we assume that $W = \{p_1, \ldots, p_{2d-1}\}$ and that, after the application of Rule 3.7, only the vertices p_i with $2 \le i \le 2d-2$ are removed. Let P^* denote the set of the removed vertices. We prove that the instances are equivalent.

First, let G be a support with feedback edge number f for $\mathcal{H}$, as constructed in the first part of the proof. We show that we can obtain a support for $\mathcal{H}'$ with feedback edge number at most f from G. Consider a hyperedge $F \in \mathcal{E}$ such that $G[F \setminus P^*]$ is disconnected. Clearly, $F \cap P^* \neq \emptyset$ and $F \setminus P^* \neq \emptyset$. Because of Condition (ii) and $|F| \le d$, the set $F \setminus P^*$ then contains exactly one of p_1 and p_{2d-1}. Without loss of generality, let $p_1 \in F$. Remove each edge $\{v, p_i\}$ in $G[F]$ with $i \in \{2, \ldots, d\}$ and add $\{v, p_1\}$. Let G^* be the graph obtained by iterating the replacements as long as possible. We obtain a support for $\mathcal{H}'$ by taking $G^*[V \setminus P^*]$: We have that $G^*[F \setminus P^*]$ is connected because any path in $G[F]$ between two vertices in $F \setminus P^*$ that used $\{v, p_i\}$ can now use $\{v, p_1\}$ instead. Finally, since $2d-2$ edges are incident with P^* in G^*, and the same number of vertices are removed from G^*, it follows that $G^*[V \setminus P^*]$ has feedback edge number at most f' edges. Hence, if $(\mathcal{H}, f)$ is a yes-instance, then also $(\mathcal{H}', f')$ is a yes-instance.

For the converse, let G' be an optimal support for $\mathcal{H}'$ with at most k' edges. We claim that adding the edges $\{p_i, p_{i+1}\}$ for $i \in \{1, \ldots, 2d-2\}$ yields a support G for $\mathcal{H}$.

Clearly, it suffices to consider hyperedges F that intersect P^*. Condition (ii) implies that $G[F \cap P^*]$ is connected. Hence, if $F \subseteq P^*$, then $G[F]$ is connected. Otherwise, $G'[F \setminus P^*]$ is connected since G' is a support. By Condition (ii), either $p_1 \in F$ or $p_{2d-1} \in F$. Since $F \cap P^*$ is connected to p_1 or to p_{2d-1} in G, this implies that $G[F]$ is connected. Note that G contains at most k edges since it contains exactly $2d-2$ edges more than G'. Hence, if $(\mathcal{H}', k')$ is a yes-instance, then also $(\mathcal{H}, k)$ is a yes-instance and the rule is correct. □

We now show the running time of Rule 3.7.

Lemma 3.10. It is possible to apply Rule 3.7 or to decide that it does not apply to the hypergraph in $O(m^3 d^3)$ time.

Proof. First, recall Observation 3.2. In order to find an application of Rule 3.7 we will consecutively find hyperedges that correspond to the definitions of F_i^- and F_i^+. The algorithm is as follows.

We start by building the covering graph of $\mathcal{H}$. Then, we construct an auxiliary hypergraph $\mathcal{H}^* = (V^*, \mathcal{E}^*)$ as follows: Start with the hypergraph $\mathcal{H}$ and while there is a vertex in the covering graph that has an incoming arc, remove this vertex from $\mathcal{H}$. Note that V^* does not contain any pair of vertices u and v such that u covers v. The running time for constructing $\mathcal{H}^*$ is dominated by the $O(n \cdot |\mathcal{H}|) = O(m^2 \cdot d^2)$ time needed for constructing the covering graph.

Intuitively, we successively discover the vertices in $p_1, \ldots, p_{2d-1}$ from Rule 3.7 that are contained in the F_i^- and F_i^+, giving rise to successively new hyperedges F_i^-, F_i^+. By considering only V^*, the F_i^- and F_i^+ form a very regular structure, and when p_i is fixed, then there is little choice for p_{i+1}. Hence, we can basically guess p_1 and greedily determine the successive p_i, $i > 1$, until either sufficiently many are found, or there are no choices left. Subsequently, we make this approach more formal.

We guess, trying all possibilities, which of the sets F in $\mathcal{E}^*$ is the set $F_1^+ \cap V^*$. Let us assume that our guess was correct. We then check for vertices v having the following property:

Property (a): there is a hyperedge $F' \in \mathcal{E}^*$ such that $F \cap F' = \{v\}$.

Discard the guess of F if there are more than two or less than two vertices with Property (a). Otherwise, for both choices to denote one of these vertices by p_1, proceed as follows.

Set $i := 1$ and repeat the following as long as $F \setminus \{p_1, \ldots, p_i\} \neq \emptyset$. Check whether there are vertices v in $F \setminus \{p_1, \ldots, p_i\}$ with

> Property (b): there is a hyperedge $F' \in \mathcal{E}^*$ such that F' contains v and p_i, and $F' \cap F \subseteq \{p_1, \ldots, p_i, v\}$.

If there is exactly one vertex v in $F \setminus \{p_1, \ldots, p_i\}$ with Property (b), then denote this vertex p_{i+1} and set $i := i+1$. Otherwise discard the guess of p_1.

Now, set $i = |F|$ and repeat the following as long as $i < 2d-1$. Check for each vertex v in $V^* \setminus \{p_1, \ldots, p_i\}$ whether it has

> Property (c): there is a hyperedge $F' \in \mathcal{E}^*$ such that F' contains v, $F' \cap \{p_1, \ldots, p_i\} = \{p_i\}$ and for every $u \in F' \setminus \{v, p_i\}$, and every hyperedge $F'' \in \mathcal{E}^*$ containing u and p_i we have that F'' contains v.

If there is exactly one vertex with Property (c), then denote this vertex v as p_{i+1} and set $i := i+1$. Otherwise discard the guess of p_1.

Finally, let $L = \bigcap\{F' \in \mathcal{E}^* \mid F' \not\subseteq \{p_1, \ldots, p_{2d-1}\} \wedge (p_1 \in F')\} \setminus \{p_1, \ldots, p_{2d-1}\}$ and similarly $R = \bigcap\{F' \in \mathcal{E}^* \mid F' \not\subseteq \{p_1, \ldots, p_{2d-1}\} \wedge (p_{2d-1} \in F')\} \setminus \{p_1, \ldots, p_{2d-1}\}$. Let p_0 be an arbitrary vertex in L and p_{2d} an arbitrary vertex in R. We now check whether $p_0, \ldots, p_{2d}$ satisfies the conditions of Rule 3.7 in the original input hypergraph $\mathcal{H}$. We claim that if there is an application of Rule 3.7, then the above algorithm finds it.

Denote the vertices in P in the application of the rule by $p'_0, \ldots, p'_{2d}$. Let F_i^+ and F_i^- be as in Observation 3.2 and denote $F'^+_i = F_i^+ \cap V^*$ and $F'^-_i = F_i^- \cap V^*$, respectively. If $p'_0, \ldots, p'_{2d}$ satisfies the assumptions of Rule 3.7 and $\mathcal{E}(u) = \mathcal{E}(p'_i)$, then also $p'_0, \ldots, p'_{i-1}, u, p'_{i+1}, \ldots, p'_{2d}$ satisfies the assumptions of Rule 3.7 and, hence, we can assume $p'_0, \ldots, p'_{2d} \in V^*$.

Consider the branch in which $F = F'^+_1$. The sets F'^-_1 and $F'^+_{|F|}$ witness the Property (a) for p'_1 and $p'_{|F|}$, respectively, and no other vertex can satisfy Property (a) by Conditions (ii) and (iv). Hence, we may assume $p_1 = p'_1$. For each $i \in \{1, \ldots, |F|-1\}$, the set F'^-_{i+1} witnesses the Property (b) for p'_{i+1} and no other vertex in F has Property (b).

Next, for $i \in \{|F|, \ldots, 2d-2\}$, p'_{i+1} has Property (c), and for any other vertex taking $u = p'_{i+1}$ (which must be contained in any F') and $F'' = F'^-_{i+1}$ shows that it does not have Property (c). Finally, it remains to note that every hyperedge which contains $p'_1 = p_1$ and p'_0 also contains the whole set L and, in particular, p_0 (similarly, R contains p_{2d}). Hence, also $p_0, p'_1, \ldots, p'_{2d-1}, p_{2d}$ satisfy the assumptions of Rule 3.7, and the algorithm finds an application.

As to the running time, there are $|\mathcal{E}^*| \le m$ possible choices of F, any of the Properties (a)–(c) can be tested for all vertices in $O(m^2 \cdot d^2)$ time by first selecting F' and F'' and then v and u inside them. As successfully testing a condition implies finding one more vertex of $p'_0, \ldots, p'_{2d}$, we test the conditions at most $2d+1$ times in each branch. The sets L and R can be found in $O(m \cdot d)$ time. Finally, the assumptions of Rule 3.7 for a given sequence $p_0, \ldots, p_{2d}$ can be tested in $O(m \cdot d^2)$ time. Hence the total running time is $O(m \cdot ((2d+1) \cdot m^2 \cdot d^2 + m \cdot d^2)) = O(m^3 d^3)$. □

We now derive an upper bound on the number of vertices in instances that are reduced with respect to Rule 3.4 and Rule 3.7. As Rule 3.4 reduces all vertices that have degree one in any support, the upper bound on the overall number of vertices will be obtained by upper-bounding the number of vertices with degree at least two in the support. As mentioned above, we will use Rule 3.7 in Section 3.6.2, where we also need to upper-bound the number of vertices with degree at least two. Hence we use the concept of 2-cores, which was introduced in the context of social network analysis.

Definition 3.1 (Seidman [Sei83])**.** The *2-core* of a graph G is the induced subgraph of G that has the maximum number of vertices such that each vertex has degree at least two.

The 2-core of a graph is unique [Sei83]. The application of Rule 3.7 alone does not necessarily yield a bounded number of vertices in the 2-core in a support. Instead, the application of Rule 3.4 for $d \leq 4$ (and of a different rule for $d \geq 5$) "prepares the hypergraph" such that application of Rule 3.7 yields a size upper bound on the 2-core of an optimal support G. We define the notion of being prepared as follows.

Definition 3.2. We say that a hypergraph $\mathcal{H} = (V, \mathcal{E})$ is *cleared* if there is an optimal support G for $\mathcal{H}$ such that each vertex of degree at least two is in the 2-core of G and, furthermore, for each $P := \{p_0, \ldots, p_{2d}\}$ with $P \subseteq V$ and $\mathcal{E}' := \bigcup_{p \in P} \mathcal{E}(p)$ that satisfy Conditions (i) to (iii) of Rule 3.7, it holds that $\mathcal{H}$ and P also satisfy Condition (iv).

The intuition behind the definition is as follows. The first part of the definition guarantees that the support for a cleared hypergraph consists of the 2-core plus possibly some degree-one vertices attached to this 2-core. The second part of the definition guarantees that any hypergraph with long paths in the 2-core of its support can be reduced further by applying Rule 3.7.

For $d \leq 4$ it is sufficient to apply Rule 3.4 in order to clear a hypergraph.

Lemma 3.11. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph with $d \leq 4$ that is reduced with respect to Rule 3.4. Then, $\mathcal{H}$ is cleared.

Proof. The reducedness of $\mathcal{H}$ directly implies that there is a support without any degree-one vertices, hence, the first property of being cleared is satisfied. For the second property, consider P and $\mathcal{E}$ as in Rule 3.7 and assume that there is a hyperedge $F \in \mathcal{E}$ such that $F \cap P = \{p_i\}$ for some $1 < i < 2d$. Then, by Condition (iii) we have $\mathcal{E}(p_i) \supseteq \mathcal{E}(u)$ for each $u \in F \setminus P$ and, hence, Rule 3.4 applies, a contradiction. □

We now upper bound the size of reduced instances. We also use this bound in Section 3.6.2 and, hence, prove it in a slightly more general form than needed for $d \le 4$.

Lemma 3.12. Let $(\mathcal{H}, f)$ be a yes-instance of SUBSET INTERCONNECTION DESIGN such that $\mathcal{H}$ is connected, cleared, and reduced with respect to Rule 3.7. Then, there is a support $G = (V, E)$ for $(\mathcal{H}, f)$ such that the 2-core of G has at most

$$\max\{(9d-1)(f-1), (3d-1)f\}$$

vertices and, hence, at most $9d \cdot f$ edges.

Proof. Among all optimal supports G for $(\mathcal{H}, f)$ such that each vertex of degree at least two is in the 2-core of G, choose G such that it contains the maximum number of degree-one vertices. Now consider the 2-core G' of G. Note that G and G' have the same feedback edge number f. We show an upper bound on the number of vertices in G' which then gives a bound on the number of edges in G'.

Let $V_{\ge 3}$ denote the vertices with degree at least three in G' and V_2 the degree-two vertices in G'. We first bound the number of components in $G'[V_2]$. If $V_{\ge 3} = \emptyset$, then, clearly, we have at most one component. Otherwise, consider the graph G^* with loops and parallel edges obtained from G' by replacing each maximal path with inner vertices in V_2 by a single edge. The number of edges in G^* is an upper bound on the number of components in $G'[V_2]$. The number of edges in G^* is

$$|V_{\ge 3}| - 1 + f \;=\; \sum_{v \in V_{\ge 3}} \deg_G(v)/2 \;\ge\; 3|V_{\ge 3}|/2.$$

The above relation implies that $|V_{\ge 3}| \le 2f - 2$. Thus, the number of edges in G^* and the number of components in $G'[V_2]$ is at most $\max\{1, 3f - 3\}$.

We now show that each connected component of $G'[V_2]$ contains fewer than $3d$ vertices. Consider a connected component C of $G'[V_2]$ with $c + 1 \ge 3d$ vertices. Since C is a path, its vertices admit an ordering $p_0, p_1, \ldots, p_c$ with $\{p_i, p_j\} \in E \Leftrightarrow j = i + 1$ for all $0 \le i \le j \le c$. Let $P := \{p_0, p_1, \ldots, p_{2d}\}$ and let $\mathcal{E}' := \bigcup_{p \in P} \mathcal{E}(p)$. We show that Rule 3.7 applies to $\mathcal{H}$, contradicting its reducedness.

First, assume that Condition (ii) of Rule 3.7 does not hold for some $F \in \mathcal{E}'$. That is, there are $0 \le i < j < \ell \le 2d$ such that $p_i, p_\ell \in F$ and $p_j \notin F$. Since $c + 1 \ge |P| + d$, a shortest p_i-p_ℓ-path in $G - \{p_j\}$ contains at least $d+1 > |F|$ vertices. Thus, $G[F]$ is not connected, contradicting G being a support for $(\mathcal{H}, k)$. Hence, Condition (ii) of Rule 3.7 is satisfied.

Now, Condition (i) of Rule 3.7 can be seen as follows. Assume that there is a pair p_i, p_j, $i < j$, of vertices in C such that p_j covers p_i. By the above, all hyperedges that contain p_i and p_j also contain $\{p_{i+1}, \dots, p_{j-1}\}$. Hence, p_{i+1} also covers p_i. Since p_i and p_{i+1} are adjacent in G, this implies that a new support can be obtained by making all neighbors (except p_{i+1}) of p_i adjacent to p_{i+1} instead. The new graph is also a support, and has one additional degree-one vertex. This contradicts our choice of G. Hence, Condition (i) of Rule 3.7 holds.

Let $F \in \mathcal{E}$ with $F \cap \{p_0, p_{2d}\} = \emptyset$ and note that $G[F]$ is connected. Since $\mathcal{H}$ is cleared, and since there is some $p_j \in F \cap P$, we know that $G[F]$ consists entirely of vertices in P plus some degree-one vertices. Each degree-one vertex v is covered by its neighbor $p \in P$ in G. Thus Condition (iii) of Rule 3.7 holds.

Finally, Condition (iv) of Rule 3.7 follows since $\mathcal{H}$ is cleared. Thus, Rule 3.7 applies, a contradiction.

By the above, each connected component in $G' - V_{\geq 3}$ has less than $3d$ vertices. Hence, $|V_2| \leq (3d-1)\max\{3f-3, 1\}$. Altogether, this implies

$$|V_2| + |V_{\geq 3}| \;\leq\; (3d-1)\max\{3f-3,1\} + 2f - 2 \;=\; \max\{(9d-1)(f-1), (3d-1)f\}.$$

By definition of f this means that the 2-core of G has at most $9d \cdot f$ edges. ☐

For $d \leq 4$, after applying Rule 3.4 the size bound on the 2-core immediately gives a bound on the overall instance size.

Theorem 3.2. An instance of SUBSET INTERCONNECTION DESIGN with maximum hyperedge size $d \leq 4$ can be reduced to an equivalent one with at most $\max\{35(f-1), 11f\}$ vertices in $O(n \cdot m^3)$ time.

Proof. The kernelization algorithm exhaustively applies Rule 3.4 and Rule 3.7. Reducedness with respect to Rule 3.4 ensures that there are no degree-one vertices in any optimal support and, by Lemma 3.11, that the hypergraph is cleared. Consequently, being reduced with respect to Rule 3.7 implies that any support contains at most $\max\{(9d-1)(f-1), (3d-1)f\}$ vertices of degree at least two. Altogether this implies the bound on the number of vertices.

For the running time, we apply Rule 3.4 exhaustively, then apply Rule 3.7 exhaustively, and repeat until neither applies anymore. Since each application of one of the rules removes at least one vertex, we iterate at most n times. Applying the running time bounds given by Lemmas 3.5 and 3.10 we obtain the overall running time bound. $\square$

3.6.2. A fixed-parameter algorithm for f and d

Our polynomial-time data reduction in Section 3.6.1 does not generalize to arbitrary d. The reason is that the condition $|F| \leq 4$ is necessary for the correctness of Rule 3.4 or, equivalently, that Rule 3.1 is incorrect if $d > 5$. Using an additional reduction rule (Rule 3.8 below), we can obtain the same bound on the number of vertices in the 2-core of a support. However, many degree-one vertices may still remain and it seems unclear how to remove them for $d \geq 5$. Nevertheless, using the fact that the 2-core of a support has bounded size we obtain a branching algorithm with running time $O(d^{18df} \cdot d \cdot n \cdot m + n \cdot m^3 \cdot d^2)$. The algorithm first applies Rule 3.7 and Rule 3.8 to simplify the structure of the support that we are looking for. Then, we apply a branching rule that branches into $\mathrm{O}(d^2)$ cases and finds at least one of the edges in the 2-core of a support. If the branching rule does not apply, then an optimal support can be found in polynomial time.

First, to obtain the bound on the 2-core, we replace Rule 3.4 with Rule 3.8 to clear the input hypergraph and to make Lemma 3.12 applicable.

Rule 3.8. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph, $F \in \mathcal{E}$, and $F = \{u, u_1, \ldots, u_\ell\}$ such that u covers each u_i. Then, remove the vertices $u_1, \ldots, u_\ell$ from $\mathcal{H}$.

Lemma 3.13. Rule 3.8 is correct and one application takes $\mathrm{O}(n \cdot m \cdot d)$ time.

Proof. We first show the correctness of the rule. We show that there is an optimal support G such that $G[F]$ is a star with center vertex u. Let G' be any optimal support for $\mathcal{H}$ and assume that there are two adjacent vertices in F none of which is u. Note that we may choose two such adjacent vertices u_i, u_j such that $\{u, u_j\} \subseteq N_{G'}(u_i)$; this is possible because otherwise $G'[F]$ is not connected. Remove the edge $\{u_i, u_j\}$ from G' and add $\{u, u_j\}$ to obtain G. We prove that the graph G is a support. Consider any hyperedge $F' \in \mathcal{E}$ and any path P' between two vertices in $G'[F']$. If P' contains the edge $\{u_i, u_j\}$, then we may replace it by $\{u_i, u\}, \{u, u_j\}$ to obtain the walk P in G. Since u covers both u_i and u_j the path P is contained in $G[F']$. Thus, $G[F']$ is connected for each $F' \in \mathcal{E}$ meaning that G is an optimal support. By repeating the replacement of edges described above, we may arrange that $G[F]$ is a star.

Note that the action of Rule 3.8 can be expressed as a series of applications of Rule 3.2. Clearly, Rule 3.2 applies to u_1, so one can safely remove u_1. Afterwards, Rule 3.2 still applies to u_2, so one can remove u_2. This can be repeated until all u_i's are removed from $\mathcal{H}$.

The running time of Rule 3.8 can be seen as follows. First, we construct the covering graph G_C for $\mathcal{H}$ in $O(n \cdot m \cdot d)$ time using Lemma 3.1. Then, for each hyperedge $F \in \mathcal{E}$ we check whether there is a vertex u such that there are arcs (u, v) in G_C for every $v \in F \setminus \{u\}$. If so, then we remove each vertex in $F \setminus \{u\}$. This costs $O(m \cdot d^2)$ time. It is easy to check that this procedure finds an application of Rule 3.8 if there is one. $\square$

For $d \geq 5$, we can now use Rule 3.8 to clear hypergraphs.

Lemma 3.14. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph that is reduced with respect to Rule 3.8. Then, $\mathcal{H}$ is cleared.

Proof. We first prove the first statement of being cleared, namely, that there is an optimal support G for $\mathcal{H}$ such that each vertex not in the 2-core of G has degree one. We use the notion of "pending trees". Let G be a graph. If $G[V']$ is the 2-core of G, then a *pending tree* of G is a connected component of $G[V \setminus V']$ plus its unique neighbor in V'.

Pick an optimal support G and consider its pending trees and their vertex sets $C_1, \ldots, C_c$. For each pending tree C_i there is a unique vertex x both in C_i and in the 2-core of G. Denote $x =: \mathrm{root}(C_i)$. If the 2-core of G is empty, then there is only one pending tree C_1 and we choose $\mathrm{root}(C_1)$ to be an arbitrary vertex of degree one instead. Next, consider optimal supports that contain the maximum number of degree-one vertices. Among these optimal supports, choose G such that

$$\mathrm{val}(G) := \sum_{i=1}^{c} \sum_{\substack{v \in C_i \\ \deg_G(v)=1}} \mathrm{dist}(\mathrm{root}(C_i), v)$$

is minimized, where $\mathrm{dist}(u, v)$ is the length of a shortest path between u and v in G.

Assume now that there is a pending tree with vertex set $C \subseteq V$ such that $G[C]$ contains at least one vertex with degree at least two in $G[C]$. Choose $u \in C$ with degree at least two such that a shortest path between u and $\mathrm{root}(C)$ has maximum length. Consider the neighbors of u in G. Let v be the neighbor of u on the shortest path from u to $\mathrm{root}(C)$. All neighbors of u different from v must be of degree one due to the choice of u according to the maximum length path to $\mathrm{root}(C)$. Let us call them $u_1,\dots,u_\ell$. By Observation 3.1, $\mathcal{E}(u) \supseteq \mathcal{E}(u_i)$ for all $i \in \{1,\dots,\ell\}$. Consider some u_i. Since $\mathcal{H}$ is reduced with respect to Rule 3.8, there is no subset U of the degree-one neighbors of u and no hyperedge $F \in \mathcal{E}$ such that $F = U \cup \{u, u_i\}$. Hence, each hyperedge incident with u_i also contains some vertex other than u and its degree-one neighbors. We conclude that each hyperedge $F \in \mathcal{E}$ that contains u_i also contains v since, otherwise, $G[F]$ is not connected. Thus, the graph G' obtained from G by removing the edge $\{u_i, u\}$ and adding the edge $\{u_i, v\}$ is an optimal support. However, the distance of u_i to $\operatorname{root}(C)$ is smaller in G' than in G. Hence, either u_i is the only degree-one neighbor of u in G, contradicting the choice of G according to the maximum number of degree-one vertices, or G' exhibits a smaller $\operatorname{val}(G')$ contradicting the choice of G according to the minimum $\operatorname{val}(G)$. Hence, there are no pending trees C that contain degree-two vertices and the first statement of the cleared-definition now follows.

It remains to prove the implications of the conditions of Rule 3.7. Let $\mathcal{H}$, $P := \{p_0,\dots,p_{2d}\}$ with $P \subseteq V$, and $\mathcal{E}' := \bigcup_{p\in P} \mathcal{E}(p)$ satisfy the Conditions (i) to (iii) of Rule 3.7. If there is a hyperedge $F \in \mathcal{E}$ such that $F \cap P = \{p_i\}$ for some $0 < i < 2d$, then we have $\mathcal{E}(u) \subseteq \mathcal{E}(p_i)$ for every $u \in F \setminus \{p_i\}$ by Condition (iii), and Rule 3.8 applies to $p_i, u_1, \dots, u_\ell$, where $\{p_i, u_1, \dots, u_\ell\} = F$. Hence, if $\mathcal{H}$ is reduced with respect to Rule 3.8, then Condition (iv) is satisfied. □

Now, Lemma 3.12 is applicable to hypergraphs that are reduced with respect to Rule 3.8, that is, we can reduce any input instance in polynomial time to one such that there is a support with at most $9d \cdot f$ edges in the 2-core. Based on this fact, we devise a search tree algorithm for the parameter (d, f). In the algorithm we maintain a partial support for the input hypergraph and in each search tree node, we try to add one new edge to the partial support. Eventually, either the partial support cannot be extended to a full support or it can be done in a greedy fashion.

In order to obtain a search tree whose size depends only on d and f, we ensure that the search tree has depth at most $9d \cdot f$ and that the algorithm branches into at most $\binom{d}{2}$ cases in each step. As mentioned, the basic idea is to determine the 2-core of the support by adding one edge to it in each branching step. Thus, applying the upper bound $9d \cdot f$ on the number of edges in the 2-core from Lemma 3.12, we perform at most $9d \cdot f$ branches. After we have determined a candidate for the 2-core, we can check whether it is correct because, if it is, then the remaining vertices can be attached to the 2-core as degree-one vertices in a greedy fashion. We now make this approach formal.

In the following, G denotes the partial support that we have built by branching so far. Initially it is an edgeless graph. Furthermore, we assume that the input hypergraph $\mathcal{H}$ is reduced with respect to Rule 3.8.

Branching rule 3.1. Let F be a hyperedge of $\mathcal{H}$ such that, with $F_0 \subseteq F$ denoting the vertices of F whose degree in G is zero, $G[F]$ is disconnected and cannot be made connected by, for each $u \in F_0$, adding an edge between u and some vertex $v \in F \setminus F_0$ that covers u. Then, branch into all possibilities to add an edge to $G[F]$.

We now show that Branching rule 3.1 is correct. For this, say that a graph G is a *partial 2-core support* for the hypergraph $\mathcal{H}$ if all the edges of G are contained in the 2-core of some optimal support of $\mathcal{H}$. Note that the edgeless graph is a partial 2-core support for every hypergraph and, hence, the edgeless graph is a suitable start for the search tree algorithm.

Lemma 3.15. Let $G = (V, E)$ be a graph and $\mathcal{H}$ a hypergraph such that Branching rule 3.1 is applicable. Then, G is a partial 2-core support for $\mathcal{H}$ if and only if some graph obtained by the application of Branching rule 3.1 is a partial 2-core support for $\mathcal{H}$.

Proof. ($\Leftarrow$): Clearly, removing an edge out of a partial 2-core support yields again a partial 2-core support.

($\Rightarrow$): Let $\tilde{G}$ be an optimal support such that its 2-core $\hat{G}$ contains all edges of G. We show that $\hat{G}[F]$ contains an edge which is not in $G[F]$ but in some graph obtained by applying Branching rule 3.1.

Recall that we assume $\mathcal{H}$ to be reduced with respect to Rule 3.8. Hence, $\mathcal{H}$ is cleared and we can assume that all edges of $\tilde{G}[F]$ that are not in the 2-core $\hat{G}$ are incident with degree-one vertices in $\tilde{G}$. Assume towards a contradiction that all edges in $\tilde{G}[F]$ are not in the 2-core $\hat{G}$. Thus, they are incident with degree-one vertices in $\tilde{G}$. Hence, all vertices in F_0 have degree one in $\tilde{G}$. Since $G'[F]$ is connected, each vertex $u \in F_0$ is connected to some $v \in F \setminus F_0$. By Observation 3.1, v covers u and, thus, the condition of Branching rule 3.1 is not satisfied, a contradiction. Thus, there is one edge in $G[F]$which is not in G but in the 2-core $\hat{G}$. Clearly, in one of the branches, an edge of $\hat{G}$ is selected and added to G. □

Next, we show that, if Branching rule 3.1 does not apply, meaning that its preconditions are not fulfilled for any choice of hyperedge F, then we can solve the instance by greedily assigning the remaining vertices.

Lemma 3.16. Let $\mathcal{H}$ be a hypergraph and let G be a graph such that there is a support for $\mathcal{H}$ that is a supergraph of G and Branching rule 3.1 does not apply to $\mathcal{H}$ and G. Then, an optimal support for $\mathcal{H}$ can be computed in $\mathrm{O}(n \cdot d \cdot m)$ time.

Proof. Let $\mathcal{H} = (V, \mathcal{E})$, let V_0 be the set of degree-zero vertices in G, and let G' be obtained from G by adding exactly one edge $\{u, v\}$ to G for each $u \in V_0$ where $v \in V \setminus V_0$ is an arbitrary vertex covering u. Note that G' can be computed in $\mathrm{O}(n \cdot d \cdot m)$ time, by first computing the covering graph of $\mathcal{H}$ using Lemma 3.1 and then iterating over each vertex and adding the appropriate edges. We now show that G' is an optimal support.

We first show that for each hyperedge $F \in \mathcal{E}$, the graph $G'[F]$ is connected. Let $F_0 = F \cap V_0$ for a hyperedge $F \in \mathcal{E}$. Since Branching rule 3.1 does not apply to the instance, $G[F]$ can be made connected by adding an edge for each $u \in F_0$ between u and some vertex $w \in F \setminus F_0$ that covers u. Thus, because each $u \in F_0$ gets at most one incident edge in this way, $G[F \setminus F_0]$ is connected. Since, for each $u \in F_0$, its neighbor v in G' covers u and we know that $v \in F \setminus F_0$, also $G'[F]$ is connected.

It remains to show that G' is optimal. Note that G' has exactly $|V_0|$ more edges than G. Let G^* be any support that is a supergraph of G. Now merge in G^* all vertices in $V \setminus V_0$ into one vertex. The resulting graph contains at least $|V_0|$ edges, since G^* is connected. Hence, G^* has $|V_0|$ edges which are incident with vertices from V_0. Thus, at least $|V_0|$ edges have to be added to G' to obtain a support. □

Combining all of the above, we arrive at the main result of this section.

Theorem 3.3. Subset Interconnection Design can be solved in $O(d^{18df} \cdot d \cdot n \cdot m + n \cdot m^3 \cdot d^2)$ time.

Proof. Let $\mathcal{H} = (V, \mathcal{E})$ be the input hypergraph. The branching algorithm works as follows. It starts by exhaustively applying Rule 3.8, then exhaustively applying Rule 3.7 and repeating until neither applies anymore. Since each application removes at least one vertex the applications of Rule 3.8 take $\mathrm{O}(n \cdot (d \cdot n \cdot m))$ time by Lemma 3.13. The applications of Rule 3.7 take $\mathrm{O}(n/d \cdot (m^3 \cdot d^3)) = \mathrm{O}(n \cdot m^3 \cdot d^2)$ time by Lemma 3.10 and the fact that each application removes at least $2d - 2 > d$ vertices. Note that the running time contribution of Rule 3.7 dominates the one of Rule 3.8. Lemma 3.14 and Lemma 3.12 imply that there is a support whose 2-core has at most $9d \cdot f$ edges if the input is a yes-instance. Thus, we apply Branching rule 3.1 as often as possible or until the number of edges in G exceeds $9d \cdot f$. Each application creates at most $\binom{d}{2}$ branches, and since each branch adds one edge to G, there are at most $9d \cdot f$ branches. The correctness of this branching follows by Lemma 3.15. Finally, if Branching rule 3.1 does not apply, then we can compute an optimal support satisfying the constraints of the search tree node in $O(n \cdot d \cdot m)$ time (by Lemma 3.16).

To check whether Branching rule 3.1 applies, we maintain the covering graph throughout the search tree. At each node, we iterate over each edge F, add the edges between vertices in F and arbitrary covering vertices to G, and check whether the result is connected. This needs $O(m \cdot (d + d^2))$ time at each node. Initializing the covering graph can be done in $O(d \cdot n \cdot m)$ time (Lemma 3.1). The size of the search tree is at most $\binom{d}{2}^{9df} \leq d^{18df}$ and hence the overall running time is

$$\begin{aligned} &O(n \cdot m^3 \cdot d^2 + d \cdot n \cdot m + d^{18df} \cdot (m \cdot d^2 + n \cdot m)) \\ &= O(d^{18df} \cdot d \cdot n \cdot m + n \cdot m^3 \cdot d^2). \end{aligned} \qquad \square$$

3.7. Concluding remarks

The main contributions of this chapter are as follows. First, we show that twins can be crucial to obtain a support with a fixed feedback edge number, but the number of crucial twins in each twin class is upper bounded by some function of m, the number of hyperedges. Subsequently this leads to a linear-time algorithm for constant values of m. Second, we provide data reduction rules and a branching rule, showing that SUBSET INTERCONNECTION DESIGN is fixed-parameter tractable with respect to the maximum hyperedge size d and the feedback edge number of the support.

The linear-time algorithm for a constant number m of hyperedges that follows from our data reduction rules seems foremost of theoretical interest. In practice, it seems unlikely that the corresponding data reduction rules apply often because they require a twin class of size exponential in m. An interesting future research direction is thus to improve the size bounds on the twin classes. However, we conjecture that an upper bound polynomial in m on the size of the whole remaining instance, that is, a polynomial-size problem kernel for parameter m, cannot be achieved.

Of more practical value seem to be the remaining Rules 3.2 to 3.4, 3.7, and 3.8 which repair the incorrect Rule 3.1, the "twin reduction rule" from the literature [Fan+08; Hos+12]. Indeed, studying the cases when Rule 3.1 applies, but none of the repaired rules do, would be useful and could lead to further relevant data reduction rules.

Above we showed that finding tree-like supports in hypergraphs with small hyperedges is fixed-parameter tractable. Here, algorithm engineering is needed to improve the running times and to merge our reduction rules with existing solution strategies [Fan+08]. On the theoretical side, we did not resolve yet whether SUBSET INTERCONNECTION DESIGN is fixed-parameter tractable with respect to the feedback edge number f of the support *alone*. A possible line of attack would be to first find out whether, in our results, we can replace the parameter d (maximum hyperedge size) by the smaller parameter "size of the largest intersection between two input hyperedges". Obtaining fixed-parameter tractability far beyond the feedback edge number seems unlikely, however, because preliminary considerations indicate that SUBSET INTERCONNECTION DESIGN is W[1]-hard with respect to the feedback vertex number of the support, that is, with respect to the smallest number of vertices that can be removed so that the support is cycle-free.

Regarding data reduction rules for small feedback edge number of the support, we note that, while there clearly is structural insight behind the reduction rules, they were developed in a rather ad-hoc fashion. There is indeed a characterization via forbidden "chordless hypercycles" in the dual hypergraph for the special case of *tree* supports. This characterization was given by Goodman and Shmueli [GS83] (see also Brandstädt, Le, and Spinrad [BLS99, Theorem 8.1.1]). It would be interesting to try and combine this characterization with the insights from Section 3.6 to develop obstructions for supports of small feedback edge number.

In terms of solution algorithms it would be interesting to significantly improve on the straightforward exponential upper bound $2^{O(n^2)}$ when solving SUBSET INTERCONNECTION DESIGN parameterized by the number n of vertices.

Regarding the application in communication networks, it is interesting to consider data reduction for variants of SUBSET INTERCONNECTION DESIGN that ask to minimize the maximum vertex degree additionally to the average degree (see Onus and Richa [OR11]). We note that our reduction rules may create pathologically large maximum degrees, for example, because of Rule 3.8. We therefore believe that a different style of data reduction is needed for obtaining supports of small maximum vertex degree. Finally, it is also of practical interest to deal with edge weights for the constructed network [KS03; KMN14]; our methods only cover the unweighted case.

Chapter 4

Planar hypergraph supports

This chapter is based on “Twins in Subdivision Drawings of Hypergraphs” by René van Bevern, Iyad A. Kanj, Christian Komusiewicz, Rolf Niedermeier, and Manuel Sorge (24th International Symposium on Graph Drawing and Network Visualization (GD ’16) [Bev+16]).

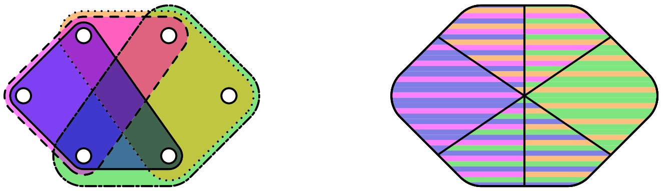


Figure 4.1.: Two drawings of the same hypergraph. On the left, we see a drawing in the *subset standard* in which the vertices (white circles) are enclosed by curves that correspond to hyperedges. On the right, we see a *subdivision drawing* in which we assign vertices to regions (enclosed by black lines) and we color these regions with colors that one-to-one correspond to the hyperedges; for each hyperedge, the regions of the vertices in that hyperedge are connected. Note that the drawing on the left is not a subdivision drawing, since it contains regions which are not assigned to any vertex.

4.1. Introduction

In this chapter we study r-OUTERPLANAR SUPPORT, a problem with applications in hypergraph drawing. Hypergraph drawings are useful as visual aid in designing electronic circuits or in examining relational database schemes. In electronic circuit design, hypergraphs are formed by taking components of circuits as vertices and grouping components in a hyperedge if they should be electrically in common [EGB06]. In relational databases, a hypergraph on the set of attributes managed by the database naturally arises from sets of attributes whose relation is managed in each table [Bee+83; Mäk90].

There are several methods for embedding hypergraphs in the plane. The combinatorial problem that we study stems from obtaining *subdivision drawings* [JP87; KKS08]. Herein, given a hypergraph $\mathcal{H}$, we divide the plane into closed regions that one-to-one correspond to the vertices of $\mathcal{H}$ in such a way that for each hyperedge e the union of the regions corresponding to the vertices in e forms a connected region. Subdivision drawings have also been called *vertex-based Venn diagrams* [JP87]. If a hypergraph admits a subdivision drawing, then we call it *vertex-planar*. Figure 4.1 shows an example for such a drawing.

Vertex planarity is a natural extension of planarity for ordinary graphs. A graph is planar if and only if it is vertex-planar when viewed as a hypergraph. For hypergraphs, vertex planarity is a rather general concept of planar embeddings, as, for example, each vertex planar hypergraph is also Zykov planar (meaning that the incidence graph is planar) and has a well-formed Euler diagram (see Flower, Fish, and Howse [FFH08]). For specifics on the relation of different kinds of planar embeddings for hypergraphs, see Kaufmann, Kreveld, and Speckmann [KKS08] and Brandes et al. [Bra+11].

Vertex planarity was introduced by Johnson and Pollak [JP87]. They also noted that it is equivalent for a hypergraph to be vertex-planar and to have a support that is planar. (Recall that a *support* for a hypergraph $\mathcal{H}$ is a graph G on the same vertex set as $\mathcal{H}$ such that each hyperedge induces a connected subgraph of G.) We refer to Kaufmann, Kreveld, and Speckmann [KKS08] for a method to obtain a subdivision drawing from a planar support. Unfortunately, it is NP-complete to decide whether a given hypergraph has a planar support; Johnson and Pollak [JP87] showed this using a reduction from the NP-complete HAMILTON PATH problem [GJ79] on cubic planar graphs.

Given Johnson and Pollak's NP-completeness result, it is natural to ask whether it is algorithmically easier to answer whether a given hypergraph has a more restricted planar embedding by restricting the type of support we are looking for. Unfortunately, it is still NP-complete to decide whether a given hypergraph has a 3-outerplanar and even NP-complete to decide whether a given hypergraph has a 2-outerplanar support. These results were obtained by Buchin et al. [Buc+11] using a reduction from the NP-complete 3-CNF-SATISFIABILITY problem [GJ79]. To the best of our knowledge, determining the classical complexity of finding outerplanar supports is currently an open problem.

Going further, it is also natural to restrict the structure of the hypergraph. To this end we parameterize by the number m of hyperedges as an obvious candidate. We study the following problem.

r-OUTERPLANAR SUPPORT
Input: A connected hypergraph $\mathcal{H}$ and a nonnegative integer r.
Question: Does $\mathcal{H}$ admit an r-outerplanar support?

Contribution. We prove the following theorem.

Theorem 4.1. r-OUTERPLANAR SUPPORT has a linear-time computable problem kernel with at most $2^{2^{O((m+\log(r))r^2)}}$ vertices, where m is the number of hyperedges and r is the outerplanarity number of the support.

Hence, r-OUTERPLANAR SUPPORT is fixed-parameter tractable with respect to $m+r$. Apart from being natural restrictions on the input and output, it is also conceivable that the parameters m and r are small in practical instances: For large number m of hyperedges it is plausible that we obtain only hardly legible drawings unless the hyperedges adhere to some special structure. Thus, it makes sense to deal with the case of small m separately. Small outerplanarity number r leads to few layers in the drawing which may lead to aesthetically pleasing drawings.

Two vertices in a hypergraph are called *twins*, if they are contained precisely in the same hyperedges. Previous works on r-OUTERPLANAR SUPPORT assumed that the input hypergraph is *twinless*, that is, for every subset of hyperedges, there is at most one vertex contained precisely in these hyperedges. This assumption was used by Mäkinen [Mäk90, p. 179], Buchin et al. [Buc+11, p. 535], and Kaufmann, Kreveld, and Speckmann [KKS08, p. 399]. Clearly, in the case of twinless hypergraphs, Theorem 4.1 is trivial, because then the input hypergraph contains at most 2^m vertices. The intuition behind assuming twinlessness is that twins do not seem useful at first glance; whatever role one vertex can play to obtain a support, its twin can also fulfill. In Section 4.3, however, we show that this intuition is wrong. More specifically, we give a hypergraph with two twins that has a (2-outer-)planar support but, removing one twin, it ceases to have a planar support. Thus, twins may indeed be helpful to find a solution. If the input hypergraph is not twinless, however, it seems much more demanding to prove a result like Theorem 4.1.

The proof of Theorem 4.1 is divided into two parts, the outline of the proof is as follows. Our aim is to prove that in each twin class larger than some function $\psi(m+r)$, we can safely forget at least one vertex. To do this we show first that, if there is an r-outerplanar support, then it has at most $\psi(m+r)$ *crucial* vertices in each twin class, and the remaining vertices can be attached as degree-one neighbors to their twins. Being crucial will receive a mathematically precise meaning in Section 4.5. Hence, our goal is to show for a tentative support that, if it has a large number of vertices, then we can modify it such that it remains a support and such that it has at least one vertex that is not crucial. To show this, we give a long sequence of nested separators in the support. Here, *nested* means that each separator separates the graph into a *left* side and a *right* side, and the left sides form a sequence of vertex sets that is increasing with respect to the subset ordering. Furthermore, the sequence of separators has the additional property that, for any pair of separators S_1, S_2, we can glue the left side of S_1 and the right side of S_2, obtaining another r-outerplanar graph.

In the second part of the proof, we show that, in a long sequence of nested separators as above, there are two separators with the following property. We can glue their left and right sides and reattach the vertices which we discarded in the process of gluing as degree-one vertices in such a way that the resulting graph is an r-outerplanar support. The reattached degree-one vertices then are not crucial, and by obtaining them we have reached our goal. Namely, there is at least one vertex which can be removed from the hypergraph.

Known results and related work. As mentioned before, Johnson and Pollak [JP87] showed that finding a *planar* support is NP-complete. Buchin et al. [Buc+11] even proved that r-OUTERPLANAR SUPPORT is NP-complete for $r = 2, 3$. By adapting the NP-hardness proof for finding a 3-outerplanar support [Buc+11] we can derive that r-OUTERPLANAR SUPPORT is also NP-complete for every $r > 3$. This is possible due to a property of the reduction that Buchin et al. use. Namely, given a formula ϕ in 3CNF, they construct a hypergraph $\mathcal{H}$ that has a *planar* support if and only if ϕ is satisfiable. Due to the way in which $\mathcal{H}$ is constructed, if there is any planar support, then it is 3-outerplanar. Thus, to obtain NP-hardness for outerplanarity number $r > 3$, we simply add to the construction of the hypergraph a disjoint copy arbitrary graph (viewed as a hypergraph) that is r-outerplanar but not $r-1$-outerplanar.

Towards determining the complexity of finding an *outerplanar* hypergraph support, Brandes et al. [Bra+11] gave a polynomial-time algorithm for cactus supports (graphs in which each edge is contained in at most one cycle). They also showed that finding an outerplanar support (or planar support) can be done in polynomial time, if in the input hypergraph each intersection or difference of two hyperedges is either a singleton or again a hyperedge in the hypergraph.

A tree support can be found in linear time [Bee+83; TY84]. Buchin et al. [Buc+11] gave a polynomial-time algorithm that can deal with the additional constraint that each vertex in the tree support has to have degree at most a given number. Klemz, Mchedlidze, and Nöllenburg [KMN14] studied so-called area-proportional Euler diagrams, for which the corresponding computational problem reduces to finding a minimum-weight tree support. Such supports can also be found in polynomial time [KS03; KMN14].

In a wider scope, motivated by drawing metro maps and metro map-like diagrams, Brandes et al. [Bra+12] studied the problem of finding *path-based* planar hypergraph supports, that is, planar supports that fulfill the additional constraint that each hyperedge contains a Hamiltonian path. They observed that Johnson and Pollak's [JP87] NP-completeness result holds also for finding path-based planar supports and, among other results, showed that path-based tree supports can be found in polynomial time. Finding path-based tree supports is also known as the GRAPH REALIZATION problem, for which several polynomial-time algorithms were already known. See Bixby and Wagner [BW88] and their references.

Since Zykov planarity (planarity of the incidence graph, see Section 1.1.1) is testable in linear time [HT74], it is of interest to minimize crossings in the corresponding embeddings of hypergraphs that are not Zykov planar. Chimani and Gutwenger [CG07] gave a definition for what it means for two hyperedges to cross, and developed an ILP-formulation and heuristics for the (NP-complete) problem of minimizing the number of crossings.

Outline of this chapter. Section 4.2 contains notation, definitions, and a result on sphere-cut branch decompositions from the literature that we need. In Section 4.3 we give an example that shows that twins can be crucial for a hypergraph to have a planar support. In Section 4.4 we give a long sequence of separators in an r-outerplanar graph, pairwise gluable, such that we again get an r-outerplanar graph. In Section 4.5 we apply this sequence of separators to r-OUTERPLANAR SUPPORT and prove that r-OUTERPLANAR SUPPORT is (strongly uniformly) fixed-parameter tractable with respect to m (Theorem 4.1).

4.2. Specific preliminaries

We deal with embeddings of graphs into the plane and sphere. For this we need some notions from topology. Branch-decompositions and sphere-cut branch decompositions are central tools that we also recapitulate.

Topology. A *topological space* is a tuple $\mathfrak{X} = (X, \mathscr{F})$ of a set X, called *universe*, and a collection $\mathscr{F}$ of subsets of X, called *topology*, that satisfy the following properties:

- The empty set $\emptyset$ and X are in $\mathscr{F}$.
- The union of the elements of any subcollection of $\mathscr{F}$ is in $\mathscr{F}$.
- The intersection of the elements of any finite subcollection of $\mathscr{F}$ is in $\mathscr{F}$.

Each set in $\mathscr{F}$ is called *open*. A *closed set* is the complement of an open set. (The empty set and X are both open and closed.)

We consider here the topological space $\mathfrak{R}^\ell = (\mathbb{R}^\ell, \mathscr{F})$ where $\mathscr{F}$ is the standard topology of $\mathbb{R}^\ell$, that is, $\mathscr{F}$ is the closure under union and finite intersection of the open balls $\{\vec{x} \in \mathbb{R}^\ell \mid \|\vec{x} - \vec{y}\| < d\}$ for $d \in \mathbb{R}$, $\vec{y} \in \mathbb{R}^\ell$, where $\|\cdot\|$ is the Euclidean norm.

A *topological subspace* $\mathfrak{T} \subseteq \mathfrak{S}$ of a topological space $\mathfrak{S}$ is a topological space whose universe is a subset of the universe of $\mathfrak{S}$. We always assume topological subspaces to carry the *subspace topology*, that is, the open sets of $\mathfrak{T}$ are the intersections of the open sets of $\mathfrak{S}$ with the universe of $\mathfrak{T}$. We also say that $\mathfrak{T}$ is the topological subspace *induced* by the universe of $\mathfrak{T}$.

Important topological subspaces of $\mathfrak{R}^\ell$ are, with a slight abuse of notation,

- the *plane* $\mathfrak{R}^2$,
- the *sphere*, whose universe is $\{(x,y,z) \in \mathbb{R}^3 \mid x^2+y^2+z^2 = 1\}$,
- the *closed disk*, whose universe is $\{(x,y) \in \mathbb{R}^2 \mid x^2+y^2 \leq 1\}$,
- the *open disk*, whose universe is $\{(x,y) \in \mathbb{R}^2 \mid x^2+y^2 < 1\}$, and
- the *circle*, whose universe is $\{(x,y) \in \mathbb{R}^2 \mid x^2+y^2 = 1\}$.

A *homeomorphism* ϕ between two topological spaces is a bijection ϕ between the two corresponding universes such that both ϕ and ϕ^{-1} are continuous. We often refer to a *subspace* $\mathfrak{X}$ *in a topological space* $\mathfrak{Y}$ (for example, a circle in a plane), by which we mean a topological subspace of $\mathfrak{Y}$ which is homeomorphic to $\mathfrak{X}$.

An *arc* is a topological space that is homeomorphic to the closed interval $[0,1] \subseteq \mathfrak{R}^1$. The images of 0 and 1 under a corresponding homeomorphism are the *endpoints* of the arc, which *links* them and runs *between* them. Being linked by an arc forms an equivalence relation on the universe of a topological space. The topological subspaces induced by the equivalence classes of this relation are called *regions*. We say that a closed set C in a topological space $\mathfrak{S}$ *separates* $\mathfrak{S}$ into the regions of the subspace of $\mathfrak{S}$ induced by $S \setminus C$ where S is the universe of $\mathfrak{S}$.

For more on topology, see Munkres [Mun00], for example.

Embeddings of graphs into the plane and sphere. An *embedding* of a graph $G = (V,E)$ into the plane $\mathfrak{R}^2$ (into the sphere $\mathfrak{S}$) is a tuple $(\mathfrak{V},\mathscr{E})$ and a bijection $\phi\colon V \to \mathfrak{V}$ such that

- $\mathfrak{V} \subseteq \mathfrak{R}^2$ ($\mathfrak{V} \subseteq \mathfrak{S}$),
- $\mathscr{E}$ is a set of arcs in $\mathfrak{R}^2$ (in $\mathfrak{S}$) with endpoints in $\mathfrak{V}$,
- the interior of any arc in $\mathscr{E}$ (that is, the arc without its endpoints) contains no point in $\mathfrak{V}$ and no point of any other arc in $\mathscr{E}$, and
- $u, v \in V$ are adjacent in G if and only if $\phi(u)$ is linked to $\phi(v)$ by an arc in $\mathscr{E}$.

The regions in $\mathfrak{R}^2 \setminus (\bigcup\mathscr{E})$ (in $\mathfrak{S} \setminus (\bigcup\mathscr{E})$) are called *faces*.

A *planar graph* is a graph which has an embedding in the plane or, equivalently, in the sphere. A *plane graph* $G = (V,E)$ is a planar graph given with a fixed embedding in the plane. An $\mathfrak{S}$*-plane graph* G is a planar graph given with a fixed embedding in the sphere. For notational convenience, we refer to the sets V and $\mathfrak{V}$ as well as E and $\mathscr{E}$ interchangeably. Moreover, we sometimes identify G with the set of points $\mathfrak{V} \cup \bigcup\mathscr{E}$.

A *noose in an $\mathfrak{S}$-plane graph G* is a circle in $\mathfrak{S}$ whose intersection with G is contained in $V(G)$. Note that every noose separates $\mathfrak{S}$ into two open disks.

Layer decompositions, outerplanar graphs. The face of unbounded size in the embedding of a plane graph G is called *outer face*. The *layer decomposition* of G with respect to the embedding is a partition of V into layers $L_1 \uplus \cdots \uplus L_r$ and is defined inductively as follows. Layer L_1 is the set of vertices that lie on the outer face of G, and, for each $i \in \{2,\dots,r\}$, layer L_i is the set of vertices that lie on the outer face of $G - (\bigcup_{j=1}^{i-1} L_j)$. The graph G is called *r-outerplanar* if it has an embedding with a layer decomposition consisting of at most r layers. We denote by the *outerplanarity number* of G the minimum r such that G is r-outerplanar. If $r = 1$, then G is simply said to be *outerplanar*. A *face path* is an alternating sequence of faces and vertices such that two consecutive elements are incident with one another. The first and last element of a face path are called its *ends*. Note that the ends of a face path may be two vertices, two faces, or a face and a vertex. The *length* of a face path is the number of faces in the sequence. Note that a vertex v in layer L_i has a face path of length i from v to the outer face. Moreover, a graph is r-outerplanar if and only if each vertex has a face path of length at most r to the outer face.

Branch decompositions. A *branch decomposition of a graph G* is a tuple (T, λ) where T is a ternary tree, that is, each internal vertex has degree three, and λ is a bijection between the leaves of T and $E(G)$. Every edge $e \in E(T)$ defines a bipartition of $E(G)$ into A_e, B_e corresponding to the leaves in the connected components of $T - e$. Define the *middle set* $M(e)$ of an edge $e \in E(T)$ to be the set of vertices in G which are incident with both an edge in A_e and B_e. That is,

$$M(e) := \{v \in V(G) \mid \exists a \in A_e \exists b \in B_e \colon v \in a \cap b\}.$$

The *width of an edge $e \in E(T)$* is $|M(e)|$ and the *width of a branch decomposition* (T, λ) is the largest width of an edge in T. The *branchwidth of a graph G* is the smallest width of a branch decomposition of G.

A *sphere-cut branch decomposition of an $\mathfrak{S}$-plane graph G* is a branch decomposition (T, λ) of G fulfilling the following additional condition. For every edge $e \in E(T)$, there is a noose $\mathfrak{N}_e$ whose intersection with G is precisely $M(e)$ and, furthermore, the open disks $\mathfrak{D}_1, \mathfrak{D}_2$ into which the noose $\mathfrak{N}_e$ separates $\mathfrak{S}$, can be indexed in such a way that $\mathfrak{D}_1 \cap G = A_e \setminus M(e)$ and $\mathfrak{D}_2 \cap G = B_e \setminus M(e)$. We use the following theorem.

Theorem 4.2 ([ST94; Dor+10; MP15])**.** Let G be a connected, bridgeless, $\mathfrak{S}$-plane graph of branchwidth at most b. There exists a sphere-cut branch decomposition for G of width at most b.

Dorn et al. [Dor+10] first noted that Seymour and Thomas [ST94] implicitly proved a variant of Theorem 4.2 in which G is required to have no degree-one vertices rather than no bridges. Marx and Pilipczuk [MP15] observed a flaw in Dorn et al.'s derivation, showing that bridgelessness is required (and sufficient). The sphere-cut branch decomposition in Theorem 4.2 can be computed in $O(|V(G)|^3)$ time (see Gu and Tamaki [GT08]), but we do not need to explicitly construct it below.

4.3. Beware of removing twins

As mentioned in Section 4.1, previous works about r-OUTERPLANAR SUPPORT assumed that the input hypergraph is twinless. More precisely, this assumption was used by Mäkinen [Mäk90, p. 179], Buchin et al. [Buc+11, p. 346], and Kaufmann, Kreveld, and Speckmann [KKS08, p. 399]. In Figure 4.2, however, we provide a concrete example that shows that twins can be necessary to obtain a (2-outer-)planar support. We now describe the construction of the corresponding hypergraph $\mathcal{H} = (V, \mathcal{E})$.

The vertex-set of the hypergraph $\mathcal{H}$ shown in Figure 4.2 is

$$V := \{a, b, c, d, v_a, v_b, v_d, u_b, u_c, u_d, t, t'\}.$$

We choose the hyperedges in such a way that t and t' are twins and $\mathcal{H}$ has a planar support but $\mathcal{H} - t$ does not. First, we add to the set of hyperedges $\mathcal{E}$ of $\mathcal{H}$ the size-two hyperedges represented by solid lines between the corresponding vertices in Figure 4.2. The corresponding "solid" hyperedges incident with (and only with) a, b, c, d form a K_4 and have the purpose of essentially fixing the embedding of each support G: Since K_4 is a 3-connected graph, it has only one planar embedding up to the choice of the outer face [Tut63, p. 747]. The remaining solid hyperedges (incident with v_a, v_b, v_d and u_a, u_c, u_d) have the purpose of anchoring the u- and v-vertices within two different faces of the embedding of the K_4: These hyperedges form two connected components that are adjacent to a, b, d and b, c, d, respectively. Hence, these connected components reside in those (unique) faces of the K_4 that are incident with a, b, d and b, c, d, respectively.

With the following additional hyperedges, our goal is to enforce that t and t' are used as conduits to connect the v-vertices to c via both and a and b, and to connect

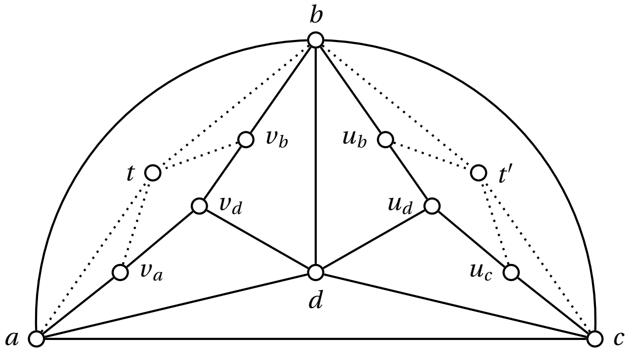


Figure 4.2.: The hypergraph $\mathcal{H}$ and its support, showing that twins can be essential for obtaining a (2-outer)planar support. The set of hyperedges consists of size-two hyperedges that are drawn as solid lines between the corresponding vertices and, additionally, $\{a, v_a, t, t', c\}$, $\{a, v_b, t, t', c\}$, $\{b, v_a, t, t', c\}$, $\{b, v_b, t, t', c\}$, $\{b, u_b, t, t', a\}$, $\{b, u_c, t, t', a\}$, $\{c, u_b, t, t', a\}$, and $\{c, u_c, t, t', a\}$. Note that the vertices t and t' are twins. Hypergraph $\mathcal{H}$ has a (2-outer)planar support whose edges are indicated by the solid and dotted lines. However, $\mathcal{H} - t$ does not have a planar support.

the u-vertices to a via both b and c. As we explain below, this is achieved by the following hyperedges:

$$\begin{array}{llll} \{a, v_a, t, t', c\}, & \{a, v_b, t, t', c\}, & \{b, v_a, t, t', c\}, & \{b, v_b, t, t', c\}, \\ \{b, u_b, t, t', a\}, & \{b, u_c, t, t', a\}, & \{c, u_b, t, t', a\}, & \{c, u_c, t, t', a\}. \end{array}$$

Clearly, t and t' are twins.

As can easily be verified, adding t and t' and the dotted edges in Figure 4.2 to the graph induced by the solid edges gives a planar support for $\mathcal{H}$.

We now show that t and t' have to reside in different faces for each planar support G for $\mathcal{H}$. First, observe that, in G, either v_a is not adjacent to b or v_b is not adjacent to a. Moreover, neither of v_a and v_b is adjacent to c. Thus, to connect the subgraphs induced by the hyperedges that contain v_a or v_b, either vertex t or its twin t' must be adjacent to one of the two vertices in G. For the same reason, t or t' must be adjacent to u_b or u_c. Since there is no face in that is simultaneously incident with one of v_a or v_b and one of u_b or u_c, vertices t and t' thus have to be in different faces. This implies that it is impossible to obtain a planar support if one of t and t' is missing. Therefore, removing one vertex of a twin class can transform a yes-instance of r-OUTERPLANAR SUPPORT into a no-instance.

Generalization to arbitrarily large twin classes. To show that the above example is not a pathology of having only one pair of twins, we extend it so that an arbitrarily large set of twins is required for the existence of a planar support. By introducing several copies of the hypergraph constructed above, we enforce that each vertex of a twin class is contained in a distinct face of any planar embedding of the support.

Fix an integer $\ell \in \mathbb{N}$. To construct the hypergraph $\mathcal{H}$, copy the vertex set V from above ℓ times, and let

$$V_i := \{a_i, b_i, c_i, d_i, v_{i,a}, v_{i,b}, v_{i,d}, u_{i,b}, u_{i,c}, u_{i,d}, t_i, t_i'\}$$

denote the vertex set of the ith copy. Within each copy, add the size-two solid hyperedges as above. Then, add a distinguished vertex v^*, and add the size-two hyperedges $\{a_i, v^*\}$, $\{b_i, v^*\}$, and $\{c_i, v^*\}$ to $\mathcal{H}$ for each $i \in \{1, \ldots, \ell\}$. Vertex v^* serves as a conduit to connect subgraphs induced by hyperedges that contain vertices from all the copies, which we are about to introduce.

Let $X = \{x \mid i \in \{1, \ldots, \ell\}\}$ for each

$$(X, x) \in \{(A, a_i), (B, b_i), (C, c_i), (V_a, v_{i,a}), (V_b, v_{i,b}), (U_b, u_{i,b}), (U_c, u_{i,c})\},$$

and let $T := \{t_i, t_i' \mid i \in \{1, \ldots, \ell\}\}$. The final hyperedges in $\mathcal{H}$ are

$$\begin{aligned}
&A \cup C \cup V_a \cup T \cup \{v^*\}, && A \cup C \cup V_b \cup T \cup \{v^*\},\\
&B \cup C \cup V_a \cup T \cup \{v^*\}, && B \cup C \cup V_b \cup T \cup \{v^*\},\\
&B \cup A \cup U_b \cup T \cup \{v^*\}, && B \cup A \cup U_c \cup T \cup \{v^*\},\\
&C \cup A \cup U_b \cup T \cup \{v^*\}, && \text{and } C \cup A \cup U_c \cup T \cup \{v^*\}.
\end{aligned}$$

Note that T forms a twin class in the resulting hypergraph $\mathcal{H}$. Hypergraph $\mathcal{H}$ has a planar support because v^* can be used to connect for each $\mathcal{H}[V_i]$ the partial supports that are obtained by copying the support for the simple example above.

We claim that, for each $t \in T$, hypergraph $\mathcal{H} - t$ does not have a planar support. To see this, assume that there is a support G and consider the planar embedding of $G - T$. Note that, in this embedding, no vertex $v_{a,i}, v_{b,i} \in V_a \cup V_b$ is in the same face as any vertex $u_{b,i}, u_{c,i} \in U_b \cup U_c$, because these vertices are in two connected components incident with d_i, a_i, and b_i, or d_i, b_i, and c_i, respectively, as in the simple example before. In addition, no vertex in $V_a \cup V_b \cup U_b \cup U_c$ from the ith copy is incident with the same face as a vertex in $V_a \cup V_b \cup U_b \cup U_c$ from the jth copy, where $i \neq j$. This is because all the vertices in $A \cup B \cup C$ have to be incident with the same face, due to the connections of v^* to each copy of a, b, and c. Thus, each pair of copies of $\{a, b, v_a, v_b, v_d\}$ or $\{a, b, u_b, u_c, u_d\}$ defines one of 2ℓ distinct faces. As before, each of these faces has to contain a vertex from T, thus, if $|T| < 2\ell$, then there is no planar support.

4.4. A sequence of gluable edge bipartitions

In this section, given an r-outerplanar graph, we provide a sequence of separators, each of size at most $2r$, that has the following properties.

- Each of the separators separates the graph into a well-defined *left* and *right* side (we say below that this is the left and right side of the separator).
- The separators are *nested*, meaning that each left side of a separator contains all left sides of separators with smaller index in the sequence.
- For every two separators S_i, S_j with $j > i$, gluing the left side of S_i with the right side of S_j yields an r-outerplanar graph. Gluing means to pairwise identify the vertices of S_i and S_j, in particular, $|S_i| = |S_j|$.

We lower-bound the length of the sequence in terms of the number n of vertices in G and the outerplanarity number r.

In Section 4.5 we then use such a sequence to show that, if a hypergraph is large, then it contains vertices which are not crucial for having an r-outerplanar support. Essentially, we take an r-outerplanar support for the hypergraph, construct the sequence of separators, and use two separators S_1, S_2 in the sequence to show that we can glue the left side of S_1 and the right side of S_2, that is, remove all vertices not contained in the left and right sides and reattach them as (non-crucial) degree-one vertices. Furthermore, we can do this in such a way that we again get an r-outerplanar support.

To formally define the sequence of separators, we use the following notation. Although the intuition about separators is instructive, it is more convenient to define our sequence of separators in terms of edge bipartitions.

For an edge bipartition $A, B \subseteq E(G)$ of a graph G, let $M(A, B)$ be the set of vertices in G which are adjacent with both an edge in A and in B, that is,

$$M(A, B) := \{v \in V(G) \mid \exists a \in A \exists b \in B \colon v \in a \cap b\}.$$

We call $M(A, B)$ the *middle set* of A, B, similarly to the middle sets in branch decompositions. For an edge set $A \subseteq E(G)$, denote by $G\langle A\rangle := (\bigcup_{e \in A} e, A)$ the subgraph induced by A. Recall also from Section 1.1 the definitions of graph gluing, boundary, and boundary labeling.

We prove the following theorem.

Theorem 4.3. For every connected, bridgeless, r-outerplanar graph G with n vertices there is a sequence $((A_i, B_i, \beta_i))_{i=1}^{s}$ where each pair $A_i, B_i \subseteq E(G)$ is an edge bipartition of G and $\beta_i \colon M(A_i, B_i) \to \{1, \dots, |M(A_i, B_i)|\}$ such that $s \geq \log(n)/(r+1)^{32r^2+8r}$, and, for every i, j, $1 \leq i < j \leq s$,

(i) $|M(A_i, B_i)| = |M(A_j, B_j)| \leq 2r$,

(ii) $A_i \subsetneq A_j$, $B_i \supsetneq B_j$, and

(iii) $G\langle A_i \rangle \circ G\langle B_j \rangle$ is r-outerplanar, where $G\langle A_i \rangle$ is understood to be β_i-boundaried and $G\langle B_j \rangle$ is understood to be β_j-boundaried.

The proof relies crucially on sphere-cut branch decompositions [Dor+10; MP15]. A *sphere-cut branch decomposition* is a ternary tree T whose leaves one-to-one correspond to the edges of the graph G embedded in the sphere (without edge crossings) that fulfills the following property. For each edge e in T, there is a circle in the sphere that meets G in precisely the middle set of the edge bipartition (A, B) of G induced by the connected components of $T - e$, and moreover, that circle cuts the sphere into two disks such that one of the disks contains only edges from A and the other only from B. Such a circle is also called *noose*. For the precise definitions, see Section 4.2.

Outline of the proof of Theorem 4.3. We first transform the planar embedding of G into an embedding in the sphere and apply Theorem 4.2 from which we obtain a sphere-cut branch decomposition for G of width at most $2r$. The edge bipartitions in Theorem 4.3 are defined based on the edges in a longest path in the decomposition tree corresponding to the sphere-cut branch decomposition. The longest path in the decomposition tree has length at least $2\log(n)$, and the edges on this path will define a sequence of edge bipartitions, a supersequence of the one in Theorem 4.3. We define a signature for each bipartition, a string containing $(32r^2+8r)\cdot\log(r+1)+1$ bits, which determines the pairs of edge bipartitions that can be glued so that we obtain an r-outerplanar graph. The sequence in Theorem 4.3 is then obtained from those bipartitions which have the same signature. The sphere-cut property of the branch decomposition gives one noose in the sphere for each edge bipartition in the sequence, such that it separates the parts in the edge bipartition from one another. The nooses of the sphere-cut branch decomposition will be crucial in the proof of Statement (iii) in Theorem 4.3, that is, the r-outerplanarity of the glued graphs.

We now give some more details concerning the r-outerplanarity of the glued graphs. After sanitizing the nooses, we can assume that they separate the sphere into *left* disks and *right* disks in such a way that each left disk contains all left disks with smaller index. Hence, for each pair of nooses, we can cut out a left disk and a right disk, and glue them along their corresponding nooses such that we again get a sphere. Alongside the sphere, we get a graph embedded in it that corresponds to the left and right sides of the separators induced by the nooses. It then remains to make the gluing so that the graph remains r-outerplanar, that is, it results in a graph embedded without edge crossings such that each vertex has a face path of length at most r to the outer face. For this we define a signature for each edge bipartition and we keep only the largest subsequence of edge bipartitions that have the same signature.

Expanding on the definition of signatures, we use it to ensure that the layer of each vertex in $G\langle A_i\rangle \circ G\langle B_j\rangle$ only decreases in comparison to G. For this, we note in the signature for each face touched by the noose that corresponds to (A_i, B_j) how far it is away from the outer face (or, rather, the face in the sphere corresponding to the outer face in the plane), and we note for each pair of faces touched by the noose how far they are away from each other. Then, if two edge bipartitions have the same signature, each vertex in the glued graph will be at most as far away from the faces touched by the noose and, hence, at most as far away from the outer face.

As we will see below, each edge bipartition signature can be encoded in $(32r^2 + 8r)\cdot\log(r+1)+1$ bits. Thus, out of the $2\log(n)$ edge bipartitions that we obtain from the longest path in the decomposition tree, there are at least $\log(n)/(r+1)^{32r^2+8r}$ edge bipartitions with the same signature.

The remainder of this section is dedicated to the formal proof of Theorem 4.3.

Proof of Theorem 4.3. In the following, fix an arbitrary r-outerplanar embedding of G.

An initial sequence $\mathcal{T}$ of edge bipartitions. Consider the canonical embedding of G into a sphere $\mathfrak{S}$ that we obtain by taking a circle that encloses but does not intersect G and identifying all points in the unbounded region of the plane which is separated off by this circle. Since G is r-outerplanar it follows that it has branchwidth at most $2r$ [Bie15]. By Theorem 4.2, there is a sphere-cut branch decomposition (T, λ) for G of width at most $2r$. We define the sequence in Theorem 4.3 based on (T, λ).

Consider a longest path P in T. Denote by e_1 the edge of G which is the preimage of the first vertex of P under the mapping λ. Since each edge in T induces a bipartition of the edges in G, so does each edge on P. Define the sequence $\mathcal{T} := ((C_i, D_i))_{i=1}^{t}$, where (C_i, D_i) is the bipartition of $E(G)$ induced by the ith edge on P

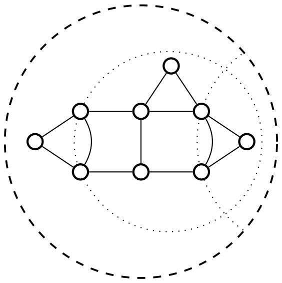
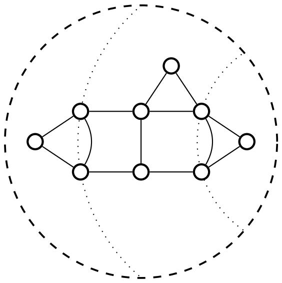



Figure 4.3.: A graph embedded in the sphere and two crossing nooses (dotted, left) and two noncrossing nooses (dotted, right). We projected the sphere into the plane by replacing a point in the sphere with a circle (dashed) and drawing all remaining points inside this circle. Both pairs of nooses represent the same edge bipartitions. Note that the two nooses on the right share exactly one point on the sphere.

such that $e_1 \in C_i$. We have $C_i \subsetneq C_{i+1}$ and $D_i \supsetneq D_{i+1}$ because T is a ternary tree and λ is a bijection. We later need a lower bound on the length of $\mathcal{T}$. For this, observe that P contains at least $2\log(n)$ edges, because G contains at least n edges (there are no vertices of degree one) and T is a ternary tree. Hence, sequence $\mathcal{T}$ also has at least $2\log(n)$ entries. The sequence in Theorem 4.3 is defined based on a subsequence of $\mathcal{T}$.

Obtaining a sequence of noncrossing nooses. To define the desired subsequence of $\mathcal{T}$, we choose one noose $\mathfrak{N}_i$ for each $(C_i, D_i) \in \mathcal{T}$ such that the resulting sequence of nooses has the following property. Denote by $\mathfrak{C}_i, \mathfrak{D}_i$ the open disks in which $\mathfrak{N}_i$ separates $\mathfrak{S}$ such that $C_i \subseteq \mathfrak{C}_i$ and $D_i \subseteq \mathfrak{D}_i$. Then, it shall hold that for any two i, j, $i < j$, we have $\mathfrak{C}_i \subsetneq \mathfrak{C}_j$ and $\mathfrak{D}_i \supsetneq \mathfrak{D}_j$. We say that the nooses $\mathfrak{N}_i$ and $\mathfrak{N}_j$ are *noncrossing* and *crossing* otherwise. See Figure 4.3 for examples.

To see that we can choose the nooses in this way, first choose them arbitrarily and then consider two crossing nooses $\mathfrak{N}_i, \mathfrak{N}_j$, $i < j$, that is, $\mathfrak{C}_i \cap \mathfrak{D}_j \neq \emptyset$. We define a noose $\tilde{\mathfrak{N}}_i$ which we obtain from $\mathfrak{N}_i$ by replacing each maximal subsegment contained in $\mathfrak{D}_j$ by the corresponding subsegment of $\mathfrak{N}_j$ which is contained in $\mathfrak{C}_i$. There is no edge of G contained in $\mathfrak{C}_i \cap \mathfrak{D}_j$ because such an edge then would also be in $C_i \cap D_j \subseteq C_i \cap D_i$, a contradiction to the fact that C_i, D_i is a bipartition of $E(G)$. Hence, noose $\tilde{\mathfrak{N}}_i$ separates $\mathfrak{S}$ into two open disks $\tilde{\mathfrak{C}}_i, \tilde{\mathfrak{D}}_i$ such that $C_i = \tilde{\mathfrak{C}}_i \cap E(G)$

and $D_i = \tilde{\mathfrak{C}}_i \cap E(G)$. Thus, $\tilde{\mathfrak{N}}_i$ fulfills the conditions for the nooses in sphere-cut branch decompositions and we may choose $\tilde{\mathfrak{N}}_i$ for (C_i, D_i) instead of $\mathfrak{N}_i$.

Clearly, $\tilde{\mathfrak{N}}_i$ and $\mathfrak{N}_j$ are noncrossing. Moreover, any noose $\mathfrak{N}_k$, $k > i$, that crosses $\tilde{\mathfrak{N}}_i$ also crosses $\mathfrak{N}_i$ because $\tilde{\mathfrak{C}}_i \subseteq \mathfrak{C}_i$. Thus, by replacing $\mathfrak{N}_i$ with $\tilde{\mathfrak{N}}_i$, the number of pairs of crossing nooses with indices at least i is strictly decreased. This means that after a finite number of such replacements we reach a sequence of pairwise noncrossing nooses.

Signatures that allow gluing. Based on the sequence $\mathcal{T}$ of edge bipartitions of G and the nooses we have fixed above for each edge bipartition, we now define a tuple, the signature, for each edge bipartition that can be encoded using $(32r^2 + 8r) \cdot \log(r+1) + 1$ bits and that has the property that, if two edge bipartitions have the same signature, then the corresponding graphs can be glued in a way that results in an r-outerplanar graph, as stated in Theorem 4.3.

We need some notation and definitions. Denote by $\mathfrak{F}$ the face in the sphere embedding of G that corresponds to the outer face of G in the planar embedding. Pick a point $y \in \mathfrak{F}$ in such a way that y is not equal to any vertex and not contained in any edge or noose $\mathfrak{N}_i$. For every noose $\mathfrak{N}_i$ we define a bijection $\beta_i \colon M(C_i, D_i) \to \{1, \ldots, |M(C_i, D_i)|\}$ corresponding to the order in which the vertices in $M(C_i, D_i)$ appear in a traversal of $\mathfrak{N}_i$ that starts in an arbitrary point. We furthermore define a map γ_i from each face touched by $\mathfrak{N}_i$ to its occurrences in the traversal of $\mathfrak{N}_i$ above. More precisely, if face $\mathfrak{G}$ occurs in the traversal of $\mathfrak{N}_i$ between vertex $\beta_i^{-1}(j)$ and $\beta_i^{-1}(j+1)$ (wherein we set $|M(C_i, D_i)| + 1$ equal to 1), then $j \in \gamma_i(\mathfrak{G})$. Finally, say that a face path P is *contained* in a closed disk $\mathfrak{E}$ if each vertex in P is contained in $\mathfrak{E}$.

The *signature* of (C_i, D_i) is a tuple (b, L_1, L_2) defined as follows.

- $b = 1$ if $y \in \mathfrak{C}_i$ and $b = 0$ otherwise.
- L_1 is the set containing the tuple $(k, \xi, \mathfrak{X}, \ell)$, for each $k \in \{1, \ldots, |M(C_i, D_i)|\}$, for each $\xi \in \{\beta, \gamma\}$, and for each $\mathfrak{X} \in \{\mathfrak{C}, \mathfrak{D}\}$, where ℓ is the length of a shortest face path from $\xi_i^{-1}(k)$ to $\mathfrak{F}$ that is contained in $\mathfrak{X}_i \cup \mathfrak{N}_i$.
- L_2 is the set containing $(k_1, k_2, \xi, \psi, \mathfrak{X}, \ell)$, for each $k_1, k_2 \in \{1, \ldots, |M(C_i, D_i)|\}$, for each pair $\xi, \psi \in \{\beta, \gamma\}$, and for each $\mathfrak{X} \in \{\mathfrak{C}, \mathfrak{D}\}$, where ℓ is the length of a shortest face path from $\xi_i^{-1}(k_1)$ to $\psi_i^{-1}(k_2)$ that is contained in $\mathfrak{X}_i \cup \mathfrak{N}_i$.

If the paths above do not exist, or the lengths are larger than r, then put ∞ instead of the length ℓ.

Definition of the desired edge bipartition sequence. Take

$$\mathcal{S} := ((C_i, D_i, \beta_i))_{i=1}^{s}$$

where, in a slight abuse of notation, $((C_i, D_i))_{i=1}^{s}$ is the longest subsequence of $\mathcal{T}$ in which all edge bipartitions (C_i, D_i) have the same signature. Two edge bipartitions (defined via nooses) which have the same signature are shown in Figure 4.3 and in Figure 4.4. We claim that $\mathcal{S}$ fulfills the conditions of Theorem 4.3.

Length of the sequence. To see that the length s of $\mathcal{S}$ is large enough, recall that sequence $\mathcal{T}$ contains at least $2\log(n)$ entries. The longest subsequence of $\mathcal{T}$ with pairwise equal signatures has length at least $2\log(n)$ divided by the number of different signatures (b, L_1, L_2). It is not hard to see that there are at most two possibilities for b, at most $(r+1)^{2r\cdot 2\cdot 2} = (r+1)^{8r}$ possibilities for L_1, and at most $(r+1)^{2r\cdot 2r\cdot 2\cdot 2\cdot 2} = (r+1)^{32r^2}$ possibilities for L_2, giving an overall upper bound on the number of different signatures of

$$2\cdot(r+1)^{8r}\cdot(r+1)^{32r^2} \;=\; 2\cdot(r+1)^{32r^2+8r}.$$

Thus $\mathcal{S}$ has length at least $\log(n)/(r+1)^{32r^2+8r}$.

Outerplanarity number of the glued graphs. For each (C_i, D_i), $(C_j, D_j) \in \mathcal{S}$, $i < j$, we have $C_i \subsetneq C_j$ and $D_i \supsetneq D_j$. Thus to prove Theorem 4.3 it remains to show that $G_{ij} := G\langle C_i\rangle \circ G\langle D_j\rangle$ is r-outerplanar. To see this, we first describe how to obtain an r-outerplanar embedding for a supergraph G' of G_{ij} from G's embedding in the sphere. Graph G' is defined below and is isomorphic to G_{ij} except that it may contain multiple copies of an edge in G_{ij}.

Recall that the nooses $\mathfrak{N}_i$ and $\mathfrak{N}_j$ are noncrossing. Hence the closed disks $\mathfrak{C}_i \cup \mathfrak{N}_i$ and $\mathfrak{D}_j \cup \mathfrak{N}_j$ can intersect only in their nooses $\mathfrak{N}_i$ and $\mathfrak{N}_j$. We now consider dislocating these disks from the sphere, and identifying their boundaries $\mathfrak{N}_i$ and $\mathfrak{N}_j$, creating another sphere. Figure 4.4 shows an example.

Recall that the vertices in $M(C_i, D_i)$ and $M(C_j, D_j)$ are enumerated by β_i and β_j, respectively, according to traversals of the corresponding nooses. Hence, there is an open disk $\tilde{\mathfrak{C}}_i$ with $\tilde{\mathfrak{C}}_i \cap \mathfrak{C}_i = \emptyset$ and a homeomorphism $\phi\colon \mathfrak{C}_i \cup \mathfrak{N}_i \to \tilde{\mathfrak{C}}_i \cup \mathfrak{N}_j$ that has the following properties.

(i) For the two traversals of the nooses that define β_i and β_j, respectively, we have that the initial points of the traversals are mapped onto each other by ϕ and, if z comes after x in the traversal of $\mathfrak{N}_i$, then $\phi(z)$ comes after $\phi(x)$ in the traversal of $\mathfrak{N}_j$.

(ii) For each $k \in \{1, \ldots, |M(C_i, D_i)|\}$ we have $\phi(\beta_i^{-1}(k)) = \beta_j^{-1}(k)$.

Denote by G' the $\mathfrak{S}$-plane graph induced by the point set $\phi(G \cap \mathfrak{C}_i) \cup (G \cap \mathfrak{D}_i)$. We claim that from G' we can derive an r-outerplanar embedding of G_{ij}.

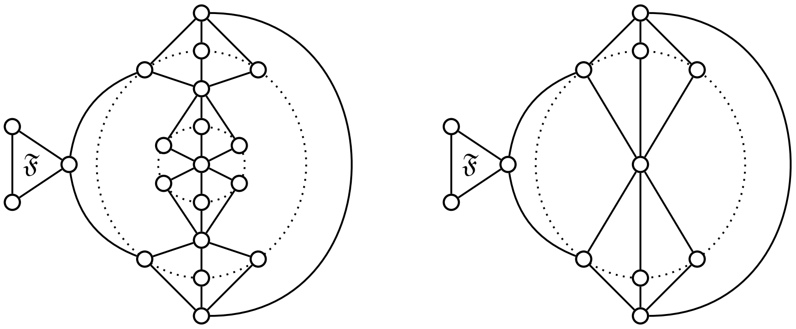


Figure 4.4.: Left: A graph embedded in a subdisk of the sphere which has been projected onto the plane. We show two nooses (dotted) that induce edge bipartitions. The signatures of the two edge bipartitions are the same if we assume that both left sides (the C_i) of the bipartitions contain the outermost edges in the drawing and if we furthermore assume that the corresponding mappings β_i are the clockwise orderings of the vertices on the noose starting with the topmost vertex.
Right: The graph resulting from gluing along the two nooses.

We first prove that G_{ij} is an edge-induced subgraph of G' without loss of generality: We may assume that G and G_{ij} have the same vertex set without loss of generality by Property (ii) of homeomorphism ϕ. Since each edge $e \in C_i$ is contained in $\mathfrak{C}_i$, it is also present in $\phi(\mathfrak{C}_i)$ and thus in G'. Moreover, each edge in $e \in D_j$ is trivially contained in $\mathfrak{D}_j$, hence, also in G'. Thus, we may assume that G_{ij} is an edge-induced subgraph of G' whence from any r-outerplanar embedding of G' we obtain an r-outerplanar embedding of G_{ij}.

Graph G' has a sphere embedding due to the way it was constructed. We now prove that from this embedding we can obtain an r-outerplanar one. This then finishes the proof of Theorem 4.3. Note that there is a face in the sphere embedding of G' that contains y or $\phi(y)$ due to the flag b in the signatures. In a slight abuse of notation, we denote this face by $\mathfrak{F}$. By removing a point contained in the face $\mathfrak{F}$ from the sphere, we obtain a topological space homeomorphic to the plane. Fix a corresponding homeomorphism δ and note that, applying δ to G', we obtain a planar embedding of G' with the outer face $\delta(\mathfrak{F})$. In the following we assume that G' is embedded in this way and, for the sake of simplicity, denote $\delta(\mathfrak{F})$ by $\mathfrak{F}$.

To conclude the proof it remains to show that the embedding of G' is an r-outer-planar one. Recall that a graph is r-outerplanar if and only it has an embedding in the plane such that each vertex v has an incident face with a face path of length at most r to the outer face $\mathfrak{F}$. Call such a path *good* with respect to v.

It remains to show that each vertex in G' has a good face path. It suffices to prove this for vertices in $\mathfrak{C}_i$ whose good paths in G are not contained in $\mathfrak{C}_i$ and vertices in $\mathfrak{D}_j$ whose good paths in G are not contained in $\mathfrak{D}_j$ as the remaining ones are also present in G'. Consider a vertex in $\mathfrak{C}_i$ whose good face path P is not contained in $\mathfrak{C}_i$. We claim that we can replace every maximal face subpath of P which is contained in $\mathfrak{D}_i \cup \mathfrak{N}_i$ by a face path contained in $\mathfrak{D}_j \cup \mathfrak{N}_j$ in such a way that the resulting sequence P' is a face path in G'. Moreover, P' is at most as long as P.

Consider a maximal face subpath S of P which is contained in $\mathfrak{D}_i \cup \mathfrak{N}_i$. Each end of S is either a vertex in $M(C_i, D_i)$, or a face. If an end of S is a face, then it can either be the outer face $\mathfrak{F}$ or a face $\mathfrak{G} \neq \mathfrak{F}$ which is intersected by $\mathfrak{N}_i$. (Note that not both ends of S can be $\mathfrak{F}$ as P is a shortest path to $\mathfrak{F}$.)

If one end of S is $\mathfrak{F}$, then associate with S a tuple $(k, \xi, \mathfrak{D}, \ell)$ where $\xi = \beta$ if the other end of S is a vertex and $\xi = \gamma$ otherwise, and where ℓ is the length of S. The first entry, k, is an integer equal to $\xi_i^{-1}(v)$ if the end of S is a vertex v, and otherwise, if the end is a face $\mathfrak{G} \neq \mathfrak{F}$, then k is defined as follows. Draw an arc $\mathfrak{A}$ contained in $\mathfrak{G}$ between the two vertices that P visits before and after $\mathfrak{G}$ such that $\mathfrak{A}$ and $\mathfrak{N}_i$ have the smallest-possible intersection. Note that $\mathfrak{A}$ and $\mathfrak{N}_i$ intersect in precisely one point y since S is maximal. Define $k \in \mathbb{N}$ such that in the traversal of $\mathfrak{N}_i$ that defines β_i vertex $\beta_i^{-1}(k)$ comes before y and $\beta_i^{-1}(k+1)$ comes after y (where we set $k+1 = 1$ if $k = |M(C_i, D_i)|$).

There is a tuple $(k, \xi, \mathfrak{D}, \ell')$ with $\ell' \leq \ell$ saved in L_1 of the signature of (C_i, D_i), since S has length at most r. Thus, $(k, \xi, \mathfrak{D}, \ell')$ is also saved in L_1 of the signature of (C_j, D_j) since the signatures of (C_i, D_i) and (C_j, D_j) are the same. Hence, there is a face path S' in $\mathfrak{D}_j$ with the ends $\mathfrak{F}$ and $\xi_j^{-1}(k)$.

We claim that $\xi_j^{-1}(k)$ and $\xi_i^{-1}(k)$ describe the same entities in G'. Indeed, if $\xi = \beta$, that is, the end of S is a vertex, then $\xi_i^{-1}(k) = \beta_i^{-1}(k)$ which is equal to $\beta_j^{-1}(k) = \xi_j^{-1}(k)$ by Property (ii) of homeomorphism ϕ.

If $\xi = \gamma$, then consider the face $\mathfrak{G} = \xi_i^{-1}(k)$ and the face $\mathfrak{H} = \xi_j^{-1}(k)$, both in G. By definition, $\mathfrak{G}$ intersects $\mathfrak{N}_i$ in the segment $\mathfrak{S}_i$ of the traversal defining β between $\beta_i^{-1}(k)$ and $\beta_i^{-1}(k+1)$. Similarly, $\mathfrak{H}$ intersects $\mathfrak{N}_i$ in the segment $\mathfrak{S}_j$ between $\beta_j^{-1}(k)$ and $\beta_j^{-1}(k+1)$. In G', face $\mathfrak{G}$ is represented by $\phi(\mathfrak{G} \cap (\mathfrak{C}_i \cup \mathfrak{N}_i))$ and face $\mathfrak{H}$ is represented by $\mathfrak{H} \cap (\mathfrak{D}_j \cup \mathfrak{N}_j) = \mathfrak{H} \cap (\mathfrak{D}_j \cup \mathfrak{N}_i)$. Moreover, segments $\mathfrak{S}_i$ and $\mathfrak{S}_j$ are identified by homeomorphism ϕ because of its Property (i). Hence, $\phi(\mathfrak{G} \cap (\mathfrak{C}_i \cup \mathfrak{N}_i))$ and

$\mathfrak{H} \cap (\mathfrak{D}_j \cup \mathfrak{N}_i)$ are merged into one face in G'. Thus, indeed $\xi_j^{-1}(k)$ and $\xi_i^{-1}(k)$ describe the same entities in G'. This implies that we can replace S by S' in P and the predecessors and successors of the ends of S' in P are incident with one another.

The proof that we can replace S by a corresponding path S' in P in the case that S does not have $\mathfrak{F}$ as an end is analogous to the above and omitted. Hence, replacing all maximal face subpaths of P that are not contained in $\mathfrak{C}_i$, we obtain a good path in G'. Finally, the case that the good path of a vertex in $\mathfrak{D}_j$ is not contained in $\mathfrak{C}_i$ is symmetric to the above and also omitted.

Summarizing, since each vertex in G has a good path, so has each vertex in G', meaning that G' is r-outerplanar. Since G_{ij} is an edge-induced subgraph of G', also G_{ij} is r-outerplanar. This concludes the proof of Theorem 4.3. □

4.5. A problem kernel for r-Outerplanar Support

Assume that the hypergraph has an r-outerplanar support. Clearly, we have the desired problem kernel if the number n of vertices is upper bounded in terms of the number m of hyperedges and the outerplanarity number r. Otherwise, if $m, r \ll n$, then, by Theorem 4.3, there exists a sequence of edge bipartitions that is long in comparison with m. In this case, intuitively speaking, for at least two edge bipartitions, their "status" must be the same with respect to their induced separators and the hyperedges of $\mathcal{H}$ crossing them. These two edge bipartitions can be glued resulting in a new graph. This new graph is not a support for $\mathcal{H}$ since it has less vertices. The missing vertices, however, can be reattached to this graph, obtaining an r-outerplanar support for $\mathcal{H}$. We formalize this approach next.

Definition 4.1 (Representative support)**.** Let $\mathcal{H}$ be a hypergraph. A graph G is a *representative support* for $\mathcal{H}$ if $V(G) \subseteq V(\mathcal{H})$, graph G is a support for subhypergraph $\mathcal{H}|_{V(G)}$ shrunken to $V(G)$, and every vertex in $V(\mathcal{H}) \setminus V(G)$ is covered in $\mathcal{H}$ by some vertex in $V(G)$.

Using Theorem 4.3, we show that the size of a smallest representative r-outerplanar support is upper bounded by a function of the number m of hyperedges of $\mathcal{H}$ plus the outerplanarity number r of a support. To this end, we first formally define the notion of two separators having the same status with respect to the hyperedges that cross the separators.

Definition 4.2 (Edge-bipartition signature)**.** Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph and let G be a representative planar support for $\mathcal{H}$. Let (A, B, β) be a tuple where (A, B) is an edge bipartition of G, and $\beta\colon M(A,B) \to \{1,\dots,|M(A,B)|\}$. Denote $\ell := |M(A,B)|$. The *signature* of (A,B,β) is a triple $(\mathcal{T},\phi,K)$, where

- $\mathcal{T} := \{[u]_\tau \mid u \in \bigcup A\}$ is the set of twin classes in $\bigcup A$,
- $\phi : \{1,\dots,\ell\} \to \{[u]_\tau \mid u \in V\}\colon j \mapsto [\beta^{-1}(j)]_\tau$ maps each index of a vertex in $M(A,B)$ to the twin class of that vertex, and
- $K := \{\gamma_F \mid F \in \mathcal{E}\}$, where γ_F is the relation on $\{1,\dots,\ell\}$ defined by $(i,j) \in \gamma_F$ whenever $\beta^{-1}(i), \beta^{-1}(j) \in F$ and $\beta^{-1}(i)$ is connected to $\beta^{-1}(j)$ in $G\langle B\rangle[F \cap \bigcup B]$, that is, in the subgraph of $G\langle B\rangle$ induced by $F \cap \bigcup B$.

We have the following upper bound.

Lemma 4.1. In a sequence $((A_i,B_i,\beta_i))_{i=1}^{s}$ as in Theorem 4.3 the number of distinct edge-bipartition signatures is upper bounded by $2^{m\cdot(2r^2+r+1)}$.

Proof. Denote the signature of (A_i,B_i,β_i) by $(\mathcal{T}_i,\phi_i,K_i)$. There are at most $2^m - 1$ twin classes in $\mathcal{T}_i$. Furthermore, for every i,j, $i<j$, we have $A_i \subsetneq A_j$, which implies $\mathcal{T}_i \subseteq \mathcal{T}_j$. Thus, either $\mathcal{T}_i = \mathcal{T}_{i+1}$ or $\mathcal{T}_{i+1}$ comprises at least one additional twin class. Since the number of twin classes can increase at most $2^m - 2$ times, the number of different $\mathcal{T}_i$ is less than 2^m. Next, there are at most 2^m choices for a twin class for each $\beta^{-1}(i) \in M(A_i,B_i)$, leading to at most $2^{m\ell}$ different possibilities where $\ell = |M(A_i,B_i)|$. For the last part of the signature, K_i, for each γ_e there are $2^{(\ell^2-\ell)/2}$ possibilities, leading to $2^{m(\ell^2-\ell)/2}$ possibilities for K_i. Since the size ℓ of the middle sets in Theorem 4.3 is at most $2r$ we have the following upper bound on the number of possible signatures:

$$2^m \cdot 2^{2mr} \cdot 2^{m\cdot(2r^2-r)} = 2^{m\cdot(2r^2+r+1)}. \qquad \square$$

Denote $\psi(m,r) := 2^{6r\cdot 2^{m\cdot(2r^2+r+1)}\cdot(r+1)^{32r^2+8r}}$.

Lemma 4.2. If a hypergraph $\mathcal{H} = (V,\mathcal{E})$ has an r-outerplanar support, then it has a representative r-outerplanar support with at most $\psi(m,r)$ vertices.

Proof. Let $G = (W,E)$ be a representative r-outerplanar support for $\mathcal{H}$ with the minimum number of vertices and fix a corresponding planar embedding. Assume towards a contradiction that $|W| > \psi(m,r)$. We show that there is a representative support for $\mathcal{H}$ with less than $\psi(m,r)$ vertices.

We aim to apply Theorem 4.3 to G. For this we need that G is connected and does not contain any bridges. Indeed, if G is not connected, then add edges between its connected components in a tree-like fashion. This does not affect the outerplanarity number of G (although it adds bridges). If G has a bridge $\{u,v\}$, then at least one of its ends, say v, has degree at least two because $|W| > \psi(m,r)$. One neighbor $w \neq u$ of v is incident with the same face as u, because $\{u,v\}$ is a bridge. After adding the edge $\{u,w\}$, edge $\{u,v\}$ ceases to be a bridge. We can embed $\{u,w\}$ in such a way that the face $\mathfrak{F}$ incident with u, v, and w is split into one face incident with only $\{u,v,w\}$ and one face $\mathfrak{F}'$ incident with all the vertices that are incident with $\mathfrak{F}$. Thus, each face path that used $\mathfrak{F}$ can now use $\mathfrak{F}'$ instead. This implies that each vertex retains a face path of length at most r to the outer face, meaning that G remains r-outerplanar. Thus, we may assume that G is connected, bridgeless, and r-outerplanar.

Graph G contains more than $\psi(m,r)$ vertices. Thus, there exists a sequence $\mathcal{S} = ((A_i,B_i,\beta_i))_{i=1}^{s}$ that satisfies the conditions of Theorem 4.3 and is of length at least

$$s \geq \frac{\log(\psi(m,r))}{(r+1)^{32r^2+8r}} = \frac{6r \cdot 2^{m\cdot(2r^2+r+1)} \cdot (r+1)^{32r^2+8r}}{(r+1)^{32r^2+8r}} = 6r \cdot 2^{m\cdot(2r^2+r+1)}.$$

Since there are less than $2^{m\cdot(2r^2+r+1)}$ different signatures in $\mathcal{S}$ (Lemma 4.1), there are $6r$ elements of $\mathcal{S}$ that have the same signature. Note that each middle set $M(A_i,B_i)$ induces a plane graph in G and, since $|M(A_i,B_i)| \leq 2r$, thus induces at most

$$\max\{1, 3|M(A_i,B_i)| - 6\} \leq \max\{1, 6r-6\}$$

edges. Thus, there are two edge bipartitions (A_i,B_i,β_i) and (A_j,B_j,β_j), $i < j$, in $\mathcal{S}$ with the same signature such that the middle sets $M(A_i,B_i)$, $M(A_j,B_j)$ differ in at least one vertex.

Let $G_{ij} := G\langle A_i\rangle \circ G\langle B_j\rangle$, wherein $G\langle A_i\rangle$ is β_i-boundaried and $G\langle B_j\rangle$ is β_j-boundaried. Let $W' := V(G_{ij})$, where we assume that $W' \cap M(A_j,B_j) \subseteq M(A_i,B_i)$ for the sake of a simpler notation. Note that $W \setminus W' \neq \emptyset$ since the middle sets of the two edge bipartitions differ in at least one vertex and since $A_i \subsetneq A_j$.

We prove that G_{ij} is a representative support for $\mathcal{H}$. That is, we show that each vertex $V \setminus W'$ is covered by some vertex in W' in $\mathcal{H}$ and that G_{ij} is a support for $\mathcal{H}|_{W'}$. Since G_{ij} is r-outerplanar by Theorem 4.3 Statement (iii), this contradicts the choice of G according to the minimum number of vertices, thus proving the lemma.

To prove that each vertex $V \setminus W'$ is covered by some vertex in W', we show that $\{[u]_\tau \mid u \in V\} = \{[u]_\tau \mid u \in W'\}$. Since $G = (W,E)$ is a representative support, $\{[u]_\tau \mid u \in V\} = \{[u]_\tau \mid u \in W\}$. Furthermore, by the definition of signature, we have $\{[u]_\tau \mid u \in \bigcup A_i\} = \{[u]_\tau \mid u \in \bigcup A_j\}$. Thus, for each vertex $u \in W \setminus W'$, there is a vertex $v \in W'$ with $[u]_\tau = [v]_\tau$, meaning that, indeed, $\{[u]_\tau \mid u \in V\} = \{[u]_\tau \mid u \in W'\}$.

To show that G_{ij} is a representative support it remains to show that it is a support for $\mathcal{H}|_{W'}$, that is, each hyperedge F' of $\mathcal{H}|_{W'}$ induces a connected graph $G_{ij}[F']$.

Let F be a hyperedge of $\mathcal{H}$ such that $F \cap W' = F'$. Observe that such a hyperedge F exists and that $G[F \cap W]$ is connected since G is a representative support of $\mathcal{H}$. Denote by S_k the middle set $M(A_k, B_k)$ of (A_k, B_k) in G for $k \in \{i, j\}$ and by S the middle set $M(A_i, B_j) = S_i = S_j$ of (A_i, B_j) in G_{ij}.

To show that $G_{ij}[F']$ is connected, consider first the case that $F \cap (S_i \cup S_j) = \emptyset$. Since each vertex in $V \setminus W'$ is covered by a vertex in W' we have that each vertex in F is contained in either $G\langle A_i \rangle$ or $G\langle B_j \rangle$ along with all edges of $G[F]$. All these edges are also present in G_{ij} whence $G_{ij}[F']$ is connected.

Now consider the case that $F \cap (S_i \cup S_j) \neq \emptyset$. Since S_i and S_j are separators in G, each vertex in $F \setminus (S_i \cup S_j)$ is connected in $G[F]$ to some vertex in S_i or S_j via a path with internal vertices in $F \setminus (S_i \cup S_j)$. We consider the connectivity relation of their corresponding vertices in S. To this end, for a graph H and $T \subseteq V(H)$ use $\gamma(T, H)$ for the equivalence relation on T of connectivity in H. That is, for $u, v \in T$ we have $(u, v) \in \gamma(T, H)$ if u and v are connected in H. Using this terminology, since both S_i and S_j equal S in G_{ij}, to show that $G_{ij}[F']$ is connected, it is enough to prove that the transitive closure δ of $\gamma(F' \cap S, G_{ij}\langle A_i \rangle) \cup \gamma(F' \cap S, G_{ij}\langle B_j \rangle)$ contains only one equivalence class.

Denote by $\hat{G}$ the graph obtained from G by identifying each $v \in S_i$ with $\beta_j^{-1}(\beta_i(v)) \in S_j$, hence, identifying S_i and S_j, resulting in the set S. Relation $\alpha := \gamma(F \cap S, \hat{G})$ has only one equivalence class and, moreover, it is the transitive closure of

$$\gamma(F \cap S_i, G\langle A_i \rangle) \,\cup\, \gamma(F \cap S, \hat{G}\langle B_i \setminus B_j \rangle) \,\cup\, \gamma(F \cap S_j, G\langle B_j \rangle),$$

wherein we identify each $v \in S_i$ with $\beta_j^{-1}(\beta_i(v)) \in S_j$ as above and, thus, $S_i = S_j = S$. We have

$$\gamma(F' \cap S, G_{ij}\langle A_i \rangle) \;=\; \gamma(F \cap S_i, G\langle A_i \rangle)$$

and

$$\gamma(F' \cap S, G_{ij}\langle B_j \rangle) \;=\; \gamma(F \cap S_j, G\langle B_j \rangle).$$

Thus for $\alpha \;=\; \delta$ it suffices to prove that

$$\gamma(F \cap S, \hat{G}\langle B_i \setminus B_j \rangle) \;\subseteq\; \gamma(F' \cap S_j, G_{ij}\langle B_j \rangle).$$

Indeed, the left-hand side $\gamma(F \cap S, \hat{G}\langle B_i \setminus B_j \rangle)$ is contained in $\gamma(F \cap S_i, G\langle B_i \rangle)$. Let $(\mathcal{T}, \phi, \mathcal{C})$ be the signature of (A_i, B_i, β_i) and (A_j, B_j, β_j) and $(F, \gamma_F) \in \mathcal{C}$. Note that

$$\gamma(F \cap S_i, G\langle B_i \rangle) \;=\; \gamma_F \;=\; \gamma(F \cap S_j, G\langle B_j \rangle)$$

where we abuse notation and set $u = \beta_i(u)$ for $u \in S_i$ and $v = \beta_j(v)$ for $v \in S_j$. Hence,

$$\gamma(F \cap S, \hat{G}\langle B_i \setminus B_j\rangle) \subseteq \gamma(F \cap S_j, G\langle B_j\rangle) = \gamma(F' \cap S_j, G\langle B_j\rangle) = \gamma(F' \cap S_j, G_{ij}\langle B_i\rangle).$$

Thus, indeed, $\delta = \alpha$, that is, F' is connected. □ □

We now use the upper bound on the number of vertices in representative supports to obtain a problem kernel for r-OUTERPLANAR SUPPORT. First, we show that representative supports can be extended to obtain a support.

Lemma 4.3. Let $G = (W, E)$ be a representative r-outerplanar support for a hypergraph $\mathcal{H} = (V, \mathcal{E})$. Then $\mathcal{H}$ has an r-outerplanar support in which all vertices of $V \setminus W$ have degree one.

Proof. Let G' be the graph obtained from G by making each vertex v of $V \setminus W$ a degree-one neighbor of a vertex in W that covers v (such a vertex exists by the definition of representative support). Clearly, the resulting graph is plane. It is also r-outerplanar, which can be seen by adapting an r-outerplanar embedding of G for G': If the neighbor v of a new degree-one vertex u is in L_1, then place u in the outer face. If $v \in L_i$, $i > 1$, then place u in a face which is incident with v and a vertex in L_{i-1} (such a face exists since otherwise v is not in layer L_i).

It remains to show that G' is a support for $\mathcal{H}$. Consider a hyperedge $F \in \mathcal{E}$. Since G is a representative support for $\mathcal{H}$, we have that $F \cap W$ is nonempty and that $G[F \cap W]$ is connected. In G', each vertex $u \in F \setminus W$ is adjacent to some vertex $v \in W$ that covers u. This implies that $v \in F$. Thus, $G'[F]$ is connected as $G'[F \cap W]$ is connected and all vertices in $F \setminus W$ are neighbors of a vertex in $F \cap W$. □

We now use Lemma 4.3 to show that, if there is a twin class that contains more vertices than a small representative support, then we can safely remove one vertex from this twin class.

Lemma 4.4. Let $\ell \in \mathbb{N}$, let $\mathcal{H}$ be a hypergraph, and let $v \in V(\mathcal{H})$ be a vertex such that $|[v]_\tau| \geq \ell$. If $\mathcal{H}$ has a representative r-outerplanar support with less than ℓ vertices, then $\mathcal{H} - v$ has an r-outerplanar support.

Proof. Let $G = (W, E)$ be a representative r-outerplanar support for $\mathcal{H}$ such that $|W| < \ell$. Then at least one vertex of $[v]_\tau$ is not in W and we can assume that this vertex is v without loss of generality. Thus, $\mathcal{H}$ has an r-outerplanar support G' in which v has degree one by Lemma 4.3. The graph $G' - v$ is a support for $\mathcal{H} - v$: For each hyperedge e in $\mathcal{H} - v$, we have that $G'[F \setminus \{v\}]$ is connected because v is not a cut-vertex in $G'[F]$ (since it has degree one). □

Now we combine the observations above with the fact that there are small r-outerplanar supports to prove that the following reduction rule is correct for r-OUTERPLANAR SUPPORT. As before, let $\psi(m,r) = 2^{6r\cdot 2^{m\cdot(2r^2+r+1)}\cdot(r+1)^{32r^2+8r}}$.

Rule 4.1. Let $\mathcal{H}$ be a hypergraph with m edges. If there is a twin class with more than $\psi(m,r)$ vertices, then remove one vertex out of this class.

By proving the correctness, we arrive at our desired result.

Theorem 4.4. r-OUTERPLANAR SUPPORT has a linear-time computable problem kernel with at most $2^m\cdot\psi(m,r)$ vertices. Hence, r-OUTERPLANAR SUPPORT is fixed-parameter tractable with respect to $m+r$.

Proof. It suffices to prove that Rule 4.1 is correct and that it can be applied exhaustively in linear time.

For the correctness, consider an instance $\mathcal{H}=(V,\mathcal{E})$ of r-OUTERPLANAR SUPPORT to which Rule 4.1 is applicable and let $v\in V$ be a vertex to be removed, that is, v is contained in a twin class of size more than $\psi(m,r)$. By Lemma 4.2, if $\mathcal{H}$ has an r-outerplanar support, then it has a representative r-outerplanar support with at most $\psi(m,r)$ vertices. By Lemma 4.4, this implies that $\mathcal{H}-v$ has an r-outerplanar support. Moreover, if $\mathcal{H}-v$ has an r-outerplanar support, then this r-outerplanar support is a representative r-outerplanar support for $\mathcal{H}$. By Lemma 4.3, this implies that $\mathcal{H}$ has an r-outerplanar support. Therefore, $\mathcal{H}$ and $\mathcal{H}-v$ are equivalent instances, and v can be safely removed from $\mathcal{H}$.

Rule 4.1 can be applied exhaustively in linear time because the twin classes can be computed in linear time [HPV99]. After this, each twin class contains at most $\psi(m,r)$ vertices; the claimed overall size bound follows since the number of twin classes is at most 2^m. □

Note that Theorem 4.1 directly follows from Theorem 4.4.

4.6. Concluding remarks

The main contribution of this chapter is to show that twins may be crucial for instances of r-OUTERPLANAR SUPPORT but the number of crucial twins is bounded in terms of the number m of hyperedges and the outerplanarity number r of a support. As a result, we can safely remove non-crucial twins, leading to a problem kernel for r-OUTERPLANAR SUPPORT and, in turn, to (strongly uniform) fixed-parameter tractability with respect to $m+r$. It is fair to say, however, that this result is still not instructive for obtaining r-outerplanar supports efficiently. In this regard, there are two directions for future research.

First, above we only showed how to reduce the size of the input instance. We also need an efficient algorithm to construct an r-outerplanar support for such an instance. As a first step, it would be interesting to improve on the $n^{O(n)}$-time brute-force algorithm that simply enumerates all n-vertex planar graphs and tests whether one of them is an r-outerplanar support.[7]

Second, it is interesting to gear the parameters under consideration more towards practice. In Section 4.5 above we attached signatures to each edge bipartition in a sequence of edge bipartitions of a support and we could reduce our input only if there were sufficiently many edge bipartitions with the same signature. This signature contained among other information the twin class of each vertex of the separator induced by the edge bipartition. Clearly, if all of these at least 2^{mr} different types of signatures are present, this will lead to an illegible drawing of the hypergraph (and still, in absence of better upper bounds, we cannot reduce our input). It seems thus worthwhile to contemplate parameters that capture legibility of the hypergraph drawing by restricting further the number of possible signatures.

A blatant open question is whether finding a *planar* support is (strongly uniformly) fixed-parameter tractable with respect to the number of hyperedges only. In this case we do not directly have the sequence of small separators which we constructed in Section 4.4 for r-outerplanar graphs. Our belief is, however, that there is a planar support whose treewidth is upper bounded by a function of the number of hyperedges, but we did not find a way to prove this so far. From bounded treewidth we could infer fixed-parameter tractability with respect to m by slightly adapting the proof in this chapter. To obtain an upper bound on the treewidth, in the spirit of irrelevant vertices [RS12], a direction to try is to take a subgraph of the support with large treewidth and to prove that an edge in this subgraph is not necessary to connect all hyperedges, and to iterate this argument until no subgraph of large treewidth exists anymore. This would imply that the whole graph has bounded treewidth (see Chuzhoy [Chu15], for example).

Finally, unrelated to the above, an interesting research direction is to try and generalize the polynomial-time algorithm of Brandes et al. [Bra+11] for constructing planar (or outerplanar) supports for hypergraphs whose family of hyperedges is basically closed under taking intersections and differences. Fixed-parameter algo-

[7]Recall that each planar graph has an ordering of the vertices such that each vertex has at most five neighbors later in the ordering. To achieve $n^{O(n)}$ enumeration time we simply guess such an ordering and then for each vertex its at most five later neighbors.

rithms for parameters that measure the distance to this polynomial-time case could be of practical interest [GHN04].

Part II
Be dense!

Chapter 5

Introduction to dense-subgraph problems

Dense graphs are an important model for tightly interacting groups in social, biological, or financial domains. Dense graphs may represent communities in social networks [WF94], segments of the stock market in financial networks [BBP03], or functional units of a cell in biological networks [BH03], for instance.

An illustrative example is the *market graph*, a graph that has capital stocks as vertices and an edge between two stocks if the correlation coefficient of the daily price changes, a measure of the similarity of the price behavior, is above a certain threshold. This graph was first analyzed by Boginski, Butenko, and Pardalos [BBP03] and later also by Komusiewicz et al. [Kom+09], Hüffner et al. [Hüf+09], and Zeng et al. [Zen+07]. Anecdotal evidence suggests that a large dense subgraph in the market graph forms a *market segment*, that is, the stocks in the subgraph correspond to companies that occupy similar economic niches. For example, Komusiewicz et al. [Kom+09] found a large clique, a graph with pairwise adjacent vertices, in the market graph consisting of energy stocks and Zeng et al. [Zen+07, Fig. 1] found a large clique of municipal finance stocks. Intuitively, while any two stock prices can correlate just by chance or due to general market movements, it is unlikely that several prices of different stocks correlate pairwise unless there is a systemic reason. Clearly, the partition of the market into segments is a very important information in stock portfolio management. For example, in order to limit risk, we would like to avoid extreme exposure of our capital to any particular segment.

Below, to abstract from communities, market segments, and biological functional units, we subsume these concepts as *clusters*. There are numerous ways in which we can define mathematically what a cluster is [Kos05; BP13; PYB13]. Naturally, different models are suited more or less to any particular application. The most simple model is of course the clique as in the example above. However, demanding that all vertices are pairwise connected is too strict in most applications due to both conceptual and practical reasons. Regarding concepts, for example, pairwise connectedness does not model communities in social networks for obvious reasons. In practice, furthermore, edges can be missing due to errors in the data. This is in particular the case in biological applications [Yu+06]. A good model of clusters should also be robust to such complications. Hence, it is common to study *clique relaxations* as cluster or dense-subgraph models [Kos05; BP13; PYB13].

We investigate here the computational complexity of finding a largest-possible cluster in a given network with respect to two clique relaxations which we define below. Such clusters give a point of reference for further analysis of the clusters in the network. Furthermore, algorithms for determining large clusters can be used as a subroutine for *graph clustering* (see Xu and Wunsch [XW05], for example), where, as in the market graph example above, we seek to partition a given graph into dense subgraphs such that between each pair of these subgraphs there are only few connections. For example, we can recursively take a maximum-size cluster into a clustering, or find clusters which improve the edge coverage of a given clustering [Hüf+14]. Furthermore, in a preprocessing step for enumerating all maximum-size clusters, we can first compute a maximal set of maximum-size clusters and then define data-reduction rules based on this set [Ebl+12]. Finally, finding dense subgraphs serves as a proxy to study the complexity of graph clustering because we can think of finding one cluster as a subproblem of determining a clustering.

The two clique relaxations that we study have the segragating property that the constraints that they impose take into account the size of the clusters that we are looking for. In a sense, the constraints scale with the size, leading to similar restrictions on large clusters as on small clusters. Concretely, we study μ-cliques, where we relax the density constraint of cliques, and highly connected graphs, where we relax the edge-connectivity constraint of cliques. Formally, they are defined as follows.

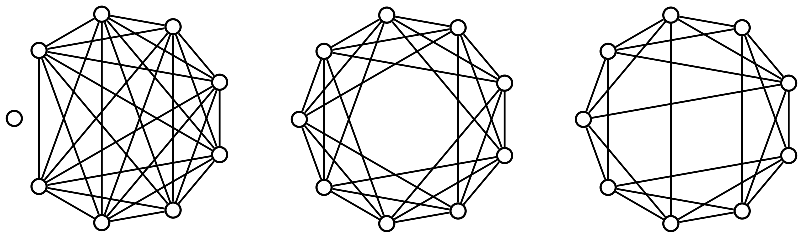


Figure 5.1.: From left to right: A 3/4-clique that is not highly connected (it is not even connected), a highly connected 3/4-clique, and a highly connected graph which is not a 3/4-clique. The leftmost graph contains the fewest number of vertices among disconnected 3/4-cliques.

Definition 5.1.

- The *density* of an n-vertex graph with m edges is $m/\binom{n}{2}$. A graph is a *μ-clique* if its density is at least μ.
- The *edge connectivity* of a graph is the size of a minimum cut. An n-vertex graph is *highly connected* if its edge connectivity is larger than $n/2$. Equivalently, each vertex has degree at least $\lfloor n/2 \rfloor + 1$.[8]

Figure 5.1 shows examples of μ-cliques and highly connected graphs. Clearly, the two conditions are quite different, because the lower bound on the density is a global constraint, whereas the size-dependent lower bound on the vertex degrees places a tangible constraint on each vertex.

The μ-clique concept was introduced by Abello, Resende, and Sudarsky [ARS02] under the name *γ-quasi-cliques*.[9] It is clearly fundamental and this in itself makes studying the computational complexity of finding μ-cliques worthwhile. From a practical point of view, a criticism is that μ-cliques are not necessarily connected (see Figure 5.1). This is clearly not a property that we would expect from a cluster, for example, from a community in a social network. However, the rather relaxed global constraint allows to capture a broader range of cluster structures. Intuitively, μ-cliques are useful if we perform an exploratory analysis of our network and we do not want to limit ourselves to more strict models of clusters which could prevent us from finding the true nature of the clusters that we are looking for. In this case, we

[8]The equivalence follows through a proof by Chartrand [Cha66, Theorem 1].

[9]Abello, Resende, and Sudarsky [ARS02] additionally required μ-cliques to be connected but this constraint was dropped in later works on μ-cliques [Pat+13; PMB14; PYB13].

can use μ-cliques as a “catch-all” model of clusters and, after a preliminary analysis of the clusters that we find in this way, subsequently limit our search to more precise restrictions of the desirable clusters that we observed.

Hartuv and Shamir [HS00] introduced the concept of highly connected graphs as an improvement over defining a cluster as a graph that has edge connectivity of at least some fixed constant. The idea is that clusters may show edge connectivity proportional to their size in some scenarios. Hence, requiring some fixed edge connectivity might be too strict for small clusters and too lax for large clusters. Highly connected graphs have been applied in diverse biological contexts [PWJ04; KSV05; Par+11; Hüf+14; HSP13]. For instance, Hartuv et al. [Har+00] used highly connected graphs successfully in an application where a partition of a DNA similarity graph into clusters is desired. This suggests that highly connected graphs may be good models for clusters in similarity graphs. This is also concordant with intuition, because similarity relations should be approximately transitive and, in general, we do not expect to find clusters of homogeneous size.

Comparing both concepts, every highly connected graph is a $1/2$-clique [HS00, Theorem 4 a)] but, clearly, not every $1/2$-clique is highly connected. For larger μ neither implies the other. This is demonstrated in Figure 5.1 for $\mu = 3/4$.

In comparison to other commonly studied clique relaxations [Kos05; BP13; PYB13], μ-cliques and highly connected graphs have two segragating features. First, they both incorporate constraints whose effects are sensitive to the size of the cluster. In other words, both small clusters and large clusters adhere to the same intuitive meaning of density. This stands in contrast to s-plexes, for example, where we require that each vertex is not connected to at most $s-1$ other vertices in the cluster for some constant s [SF78]. Clearly, this constraint is much stronger for larger clusters and hence, many larger clusters may be discounted when using s-plexes whereas they are included when using μ-cliques or highly connected graphs.

Second, the property of being a μ-clique and the property of being highly connected are not *hereditary*. That is, removing a vertex from a μ-clique does not necessarily yield another μ-clique and analogously for highly connected graphs. Heredity is usually a useful property for the corresponding maximization problems where we want to maximize the size of the cluster, that is, the number k of vertices in the cluster. This is because, then, we can successively increase k and know that, once we cannot find a solution for k, there is also no solution of size greater than k. For both of our cluster concepts, however, we cannot rely on heredity. On the positive side, being a μ-clique is *quasi-hereditary*, meaning that there is a vertex in every μ-clique that we can remove in order to obtain another μ-clique. In contrast, being highly connected is not even quasi-hereditary: from a triangle, a highly connected graph, no vertex can be deleted in order to obtain another highly connected graph. This example generalizes to arbitrary size graphs via a complete bipartite graph, whose partite sets are of equal size, with a universal vertex attached to the graph, that is, a vertex incident with all remaining vertices.

In this part we are interested in efficient *exact* algorithms for finding μ-cliques and highly connected graphs. Inspired by an argument by Aloise et al. [Alo+10], we want to emphasize that efficient exact algorithms for problems related to cluster detection are desirable for the following reasons. Although they are usually only suitable for small to medium-size instances, they serve as a benchmark for heuristics. Furthermore, exact algorithms can be used to evaluate the cluster model that they are built upon. This is not possible with heuristics or approximation algorithms, because peculiarities in their results can be due to the algorithms themselves rather than the cluster model. Finally, exact algorithms often inform the design of heuristics via the structural insights that are usually required to get small running-time upper bounds.

Appetizer. We now give the precise problem formulations and outline our results. Both problems are formalized as decision problems. We mention, however, that the corresponding algorithms can be adapted for the maximization problems by trying all possible values of k.

In Chapter 6 we study the following problem.

> μ-Clique
> *Input:* An undirected 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?

We analyze μ-Clique with respect to several distinct parameters, obtaining fixed-parameter tractability and hardness results. Our focus is mainly on the three parameters *maximum (vertex) degree* Δ, *h-index* h, and *degeneracy* d. Informally, bounded maximum degree means that *all* vertices have few neighbors, bounded h-index means that *most* vertices have few neighbors, and bounded degeneracy means that in every subgraph *there is always* a vertex with few neighbors. For every graph we have $\Delta \geq h \geq d$, and in many applications $h \leq 100$ and $d \leq 20$ [ELS13; ES12]. For $\phi \in \{\Delta, h\}$ we obtain fixed-parameter algorithms with a running time of the order $\phi^{O(\phi)} n$ where n is the number of vertices in the input graph. With respect to d, however, μ-Clique is W[1]-hard. We obtain the fixed-parameter algorithms for Δ and h, respectively, via developing a general framework for fixed-cardinality optimization, where we want to optimize a given objective function over all vertex subsets of fixed-cardinality k of a graph.

In Chapter 7 we study the following problem.

> HIGHLY CONNECTED SUBGRAPH
> *Input:* An undirected graph $G = (V, E)$ and a nonnegative integer k.
> *Question:* Is there a vertex set $S \subseteq V$ of size exactly k such that $G[S]$ is highly connected?

Similarly to μ-CLIQUE we also analyze HIGHLY CONNECTED SUBGRAPH with respect to several different parameters. The focus is a little different, however. Apart from maximum degree, h-index, and degeneracy, we consider the parameter *edge deletion*, the number of edges not in the solution, and the parameter *edge isolation*, the number of edges emanating from the solution. For edge deletion we obtain a subexponential fixed-parameter algorithm and for edge isolation a single-exponential fixed-parameter algorithm. The algorithm for the edge deletion parameter is foremost of theoretical interest, whereas the algorithm for the edge isolation parameter seems to offer a good trade-off between parameter size in practice and a good running time guarantee.

Both the above studies are related to the *parameter ecology* for the corresponding problems [FJR13; Nie10; KN12; Har14]. Herein, we navigate the space of possible parameters of the input (mostly parameters of input graph). By proving upper and lower running-time bounds we close in on parameters which simultaneously lend themselves to tractability results and are small in practice. From an algorithmic perspective, finding highly connected subgraphs is more accessible than finding μ-cliques, because the strong and local connectivity and degree constraints make it easier to develop data-reduction rules.

Related cluster concepts and related work. We now briefly point out relations of highly connected graphs and μ-cliques to some other concepts of clusters and some related work. Pattillo, Youssef, and Butenko [PYB13] described several further cluster definitions and their relations.

The first concept is that of an s-club; a graph has *diameter s* if every shortest path between two vertices contains at most s edges. Graphs with diameter s are called *s-club*. Each highly connected graph is also a 2-*club* [HS00, Theorem 1]. In contrast, μ-cliques do not have constant diameter. The s-club concept has been widely studied as another natural relaxation of cliques, see Pattillo, Youssef, and Butenko [PYB13], Shahinpour and Butenko [SB13], and Komusiewicz [Kom16] for surveys. Of special relevance is the work by Hartung et al. [Har+15] wo analyzed the problem of finding a 2-club of size k in a given graph with respect to various parameters of the input graph. Notably, in contrast to μ-cliques and highly connected graphs, finding a 2-club of size k is W[1]-hard with respect to the h-index of the input graph. A criticism of the s-club concept is that a star is an s-club for every $s \geq 2$, but arguably does not represent a community in the common sense.

A graph G is an *s-plex* if each vertex has degree at least $|V(G)| - 1 - s$. A simple calculation shows that every s-plex G with $|V(G)| \geq 2s + 3$ is highly connected and that, vice versa, every highly connected graph G is an s-plex for every s satisfying $s \leq (|V(G)| - 3)/2$. Clearly, an s-plex G is also a μ-clique for some μ (uninformatively) lower bounded by $|V(G)|$ and s. No converse holds, though. Similarly to the above, s-plexes with a constant s may be too lax for small clusters and too strict for large ones and thus are better suited for applications where the expected clusters have similar size. For a survey on results for s-plexes, see [PYB13].

Finally, Brunato, Hoos, and Battiti [BHB08] combined the global density constraint of μ-cliques and the local degree lower bound of s-plexes into a hybrid cluster concept which has the advantages of both and performed some preliminary empirical analysis.

Chapter 6

Subgraphs of fixed minimum density

This chapter is based on “An Algorithmic Framework for Fixed-Cardinality Optimization in Sparse Graphs Applied to Dense Subgraph Problems” by Christian Komusiewicz and Manuel Sorge (Discrete Applied Mathematics [KS15a]).

6.1. Introduction

The *density* of an n-vertex graph G with m edges is $m/\binom{n}{2}$. A graph is a *μ-clique* if its density is at least μ. A μ-clique is a general notion of a cluster with applications in computational biology [JP09], social network analysis, and VLSI layout design [Hol+06], for example. It is one example of a *clique relaxation*, a notion of cluster which is less restrictive than a clique [Kos05; BP13; PYB13]. In this chapter, we study exact algorithms for the following NP-complete problem.

μ-CLIQUE
Input: An undirected 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?

Clearly, μ-CLIQUE is NP-complete and W[1]-hard with respect to k because contains as a special case of the CLIQUE problem. μ-CLIQUE can be seen as a decision variant of the DENSEST k-SUBGRAPH problem [FPK01; KS09; Bha+10] in which we want to find a k-vertex set S which maximizes the density μ in $G[S]$. Some of our positive algorithmic results apply also to DENSEST k-SUBGRAPH and, more generally, to a subclass of graph optimization problems with cardinality constraints. A generic problem formulation can be given as follows.

FIXED-CARDINALITY OPTIMIZATION
Input: An undirected graph $G = (V, E)$, an objective function $\phi\colon 2^V \to \mathbb{Q}^+$, and a nonnegative integer k.
Task: Find the maximum of $\phi(S)$ over all sets $S \subseteq V$ such that $|S| = k$.

Herein, we assume that ϕ is given as an algorithm that receives G and k and $S \subseteq V$ as input and computes $\phi(S)$ in $T(k, G)$ time. Note that, in general, the cardinality constraint could also apply to the number of edges of the solution and that fixed-cardinality optimization is not restricted to graph inputs.

In this work, we determine how two types of parameters influence the complexity of μ-CLIQUE, DENSEST k-SUBGRAPH, and FIXED-CARDINALITY OPTIMIZATION. The first type comprises the classic parameter solution size k and its *dual* parameter $|V| - k$. Parameters of the second type measure the sparseness of the input graph G: maximum (vertex) degree Δ, h-index h, and degeneracy d. Informally, small maximum degree means that *all* vertices have few neighbors, small h-index means that *most* vertices have few neighbors, and small degeneracy means that in every subgraph *there is always* a vertex with few neighbors. Formally, the h-index and degeneracy are defined as follows.

Definition 6.1.

- The *degeneracy* of a graph G is the smallest integer d such that each subgraph of G contains at least one vertex of degree at most d.
- The *h-index* of a graph G is the largest integer h such that G contains at least h vertices of degree at least h.

By definition, $\Delta \geq h$-index $\geq d$. The study of these three parameters is motivated by two facts: First, many real-world networks such as biological and social networks are relatively sparse since they contain many vertices of low degree and only few vertices of high degree (the network *hubs*). Second, the otherwise notoriously hard CLIQUE problem is much easier on sparse graphs. For example, all maximal cliques can be enumerated in $O(3^{d/3} \cdot d \cdot |V|)$ time on graphs with degeneracy d [ELS13].

The degeneracy is a well-studied graph parameter, also known under the names *k-core number* [Sei83], *width* [Fre82], and *linkage* [KT96], and it is one less than the *coloring number* [EH66]. The degeneracy is small in many applications; for example, and very relevant to this work, it is below 20 for many graphs arising in social network analysis [ELS13].

The h-index parameter was introduced by Eppstein and Spiro [ES12] in the context of triangle counting in dynamic graphs. Eppstein and Spiro computed the h-index of 136 real-world graphs commonly used as benchmarks for cluster detection. Barring one exception, the h-index was below 100 with a median of 12.

Hence, efficient algorithms for small degeneracy and h-index are desirable. The maximum degree parameter we mainly use for its simplicity and as an intermediate step to obtain efficient algorithms if the h-index is small.

Our contribution. We first provide in Section 6.2.1 a fixed-parameter algorithm for the restriction of FIXED-CARDINALITY OPTIMIZATION to sets S that induce *connected* graphs $G[S]$, parameterized by the solution size k and the maximum degree Δ combined. It is based on an uncomplicated enumeration of all the possible connected subgraphs. The enumeration algorithm runs in $O((e(\Delta-1))^{k-1} \cdot (\Delta + k) \cdot n)$ time and the exponential term herein is asymptotically optimal. The proof is based on an upper bound on the number of k-vertex connected subgraphs in graphs of bounded degree by Bollobás [Bol06]. Even though the number of fixed-order connected subgraphs of a graph is a fundamental issue in graph theory, at the time of developing the result we were not aware of an algorithmic treatment as above. Parallel to our work, a similar result was provided by Elbassioni [Elb15] (see the related work below). Connected subgraph enumeration was used by Katrenic and Schiermeyer [KS11], Bonnet et al. [Bon+15] and Hermelin et al. [Her+13, Lemma 5], for example. They used the worse, easy-to-prove upper bound of $\Delta^{2k} \cdot \operatorname{poly}(n)$, however, it was not their goal to optimize the running time.

In Section 6.2.3 we then go on to treat also possibly disconnected graphs $G[S]$ in FIXED-CARDINALITY OPTIMIZATION. Motivated by the fact that the objective function in DENSEST k-SUBGRAPH, viewed as a special case of FIXED-CARDINALITY OPTIMIZATION, allows to treat connected components individually, we introduce the following property of objective functions that captures this special case.

Definition 6.2. Let $G = (V, E)$ be a graph, let $\phi \colon 2^V \to \mathbb{Q}$ be an objective function, let $S, T \subseteq V$ be arbitrary disjoint vertex sets, and let $\chi \colon \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$ such that χ is non-decreasing in both arguments individually.

- We call ϕ *component sub-*χ if the functions ϕ and χ have the property that $\phi(S) \le \chi(\phi(W), \phi(S \setminus W))$ where W is an arbitrary connected component of $G[S]$.
- We call ϕ *super-*χ if $\phi(S \cup T) \ge \chi(\phi(S), \phi(T))$.
- If both the above properties hold for ϕ and χ, then we call ϕ *component* χ*-linear.*

We give examples and non-examples of component-linear objective functions in Section 6.2.4. We obtain a randomized $\mathrm{O}(\gamma^{k-1} \cdot (\Delta + k) \cdot n \cdot T(k, G))$-time algorithm for optimizing component χ-linear objective functions ϕ where $\gamma \le 4.2 \cdot (\Delta - 1)$ and ϕ, χ are computable in $T(k, G)$ time. The algorithm works by color-coding the solution [AYZ95], then enumerating colorful connected solutions, and then using dynamic programming to find the optimal colorful, possibly disconnected, solution. The enumeration and dynamic programming steps have disproportionate running times, but by increasing the number of colors used in the coloring step [DBK07; HWZ08], we can balance the running times.

In Section 6.3 we apply the above to DENSEST k-SUBGRAPH and μ-CLIQUE. For an overview of our results, refer to Table 6.1. We obtain algorithms with running times $\Delta^{\mathrm{O}(\Delta)} n$ and $h^{\mathrm{O}(h)} n$ for μ-CLIQUE and an algorithm with running time $(\ell + d)^{\mathrm{O}(\ell)} n + \mathrm{O}(m)$ where $\ell = n - k$ is the dual parameter of the solution size k.

Finally, in Section 6.4 we show that we cannot extend our tractability results to the degeneracy parameter of the input graph. Specifically, we prove that μ-CLIQUE is W[1]-hard with respect to the solution size k and degeneracy d combined. Being a special case of DENSEST k-SUBGRAPH and FIXED-CARDINALITY OPTIMIZATION, this also applies to the more general problem. From the reduction used in this result, it also follows that there are no fixed-parameter algorithms with respect to Δ or h with subexponential running time for μ-CLIQUE, for any $0 < \mu \le 1$, unless the Exponential Time Hypothesis fails. A simple adaption of the reduction also shows that μ-CLIQUE does not admit polynomial-size problem kernels with respect to Δ or h.

Table 6.1.: Summary of our results and previous results for μ-CLIQUE and DENSEST k-SUBGRAPH. Note that hardness transfers from μ-CLIQUE to DENSEST k-SUBGRAPH and tractability transfers in the reverse direction. For fixed-parameter tractability (FPT) results, we write a rough estimate of the exponential running time factor. Herein, k denotes the solution size, $\ell := n - k$, Δ is the maximum degree, h is the h-index, and d is the degeneracy of the input graph.

Param.	μ-CLIQUE	DENSEST k-SUBGRAPH
Δ	FPT: $\Delta^{O(\Delta)}$ (Theorem 6.5), no poly. kernel (Theorem 6.10)	NP-complete for $\Delta = 3$ [FS97]
h	FPT: $h^{O(h)}$ (Theorem 6.6), no poly. kernel (Theorem 6.10)	NP-complete for $h = 3$ [FS97]
d	in XP (Lemma 6.5 Statement (iii))	NP-complete for $d = 2$ [FS97]
k, d	W[1]-hard (Theorem 6.8) in XP (trivial)	W[1]-hard (Theorem 6.8) in XP (trivial)
ℓ	W[1]-hard [KS15a] in XP (trivial)	W[1]-hard [Cai08] in XP (trivial)
ℓ, d	FPT: $(\ell + d)^{O(\ell)}$ (Theorem 6.7)	FPT: $(\ell + d)^{O(\ell)}$ (Theorem 6.7)

Known results and related work. For an overview of FIXED-CARDINALITY OPTIMIZATION problems, refer to [Bru+06]; the parameterized complexity of some special cases is studied by Cai [Cai08]. The Random Separation method [CCC06] yields fixed-parameter algorithms for a wide range of special cases of FIXED-CARDINALITY OPTIMIZATION and the combined parameter (Δ, k) where Δ is the maximum degree of G. For the special case of DENSEST k-SUBGRAPH, the randomized algorithm takes $O(2^{(\Delta+1)\cdot k} \cdot (\Delta + k) \cdot n)$ time to achieve a constant error probability. Derandomization of the algorithm adds a factor of $(\Delta k + k)^{O(\log(\Delta k))} \log n$ to the running time.

Regarding connected subgraphs in graphs of bounded maximum degree, Kangas et al. [Kan+14] provided upper bounds on the number of such subgraphs in terms of the number of vertices. Bollobás [Bol06] proved an upper bound on the number of connected k-vertex subgraphs containing a specific vertex in terms of k and the maximum degree Δ (see Theorem 6.1 below). This theorem is an integral part in the analysis of our subgraph enumeration algorithm. Combining Bollobás' theorem with a recent polynomial-delay algorithm for enumerating k-vertex connected subgraphs by Elbassioni [Elb15], we achieve a running time of $O((e(\Delta-1))^{k-1}k^3\Delta^2 \cdot n)$, which is worse than the running we obtain here by a factor of $k^2\Delta$.[10] Elbassioni's algorithm was developed in parallel to our result.

Specific results concerning μ-CLIQUE and DENSEST k-SUBGRAPH are as follows. For $\mu = 1$ μ-CLIQUE is equivalent to CLIQUE, the problem of finding a complete subgraph of order k, which is W[1]-hard with respect to the parameter k and fixed-parameter tractable with respect to the dual parameter $n-k$ [DF95; DF13]. The μ-CLIQUE problem remains NP-hard for every rational number μ, $0 < \mu < 1$ [Pat+13]. However, contrary to the CLIQUE problem, μ-CLIQUE is W[1]-hard with respect to the dual parameter $n-k$ [KS15a].

DENSEST k-SUBGRAPH is NP-complete and W[1]-hard with respect to k, as it is a generalization of CLIQUE. Moreover, DENSEST k-SUBGRAPH is W[1]-hard with respect to the parameter $n-k$ [Cai08]. It is, however, fixed-parameter tractable with respect to the maximum degree Δ and solution size k combined [CCC06]. Holzapfel et al. [Hol+06] showed that DENSEST k-SUBGRAPH remains NP-hard, even when looking only for subgraphs with average degree at least $2+\Omega(1/k^{1-\epsilon})$ for $0<\epsilon<2$. Finding k-vertex subgraphs of average degree at least $2+O(1/k)$, however, can be done in polynomial time [Hol+06]. DENSEST k-SUBGRAPH is NP-complete even in graphs with maximum degree three and degeneracy two [FS97]. The "densest subgraph" in the corresponding reduction, however, has very low, non-constant density.

Motivated by the above algorithmic hardness results, approximation algorithms for DENSEST k-SUBGRAPH have also received a lot of attention, see Khuller and Saha [KS09], Feige, Peleg, and Kortsarz [FPK01], and Khot [Kho04], for example.

A trivial exponential-time algorithm solves DENSEST k-SUBGRAPH in $2^n \cdot \text{poly}(n)$ time by checking all vertex-subsets. Chang et al. [Cha+14] showed that the running time can be improved to $1.7315^n \cdot \text{poly}(n)$ and Bourgeois et al. [Bou+13] presented improved exponential-time algorithms for some special cases of DENSEST k-SUBGRAPH.

A related problem is DENSEST SUBGRAPH, which is to find a subgraph that has maximum average degree (without constraint on the order). DENSEST SUBGRAPH is polynomial-time solvable using network flow techniques [GGT89].

[10]Elbassioni [Elb15] claims a much better running time of $O((e\Delta)^k k^2/(\Delta-1)+m+n)$ which is based on a claimed upper bound on the number of k-vertex connected subgraphs of $(e\Delta)^k/((\Delta-1)k)$ by Uehara [Ueh99]. This bound is erroneous as it does not increase when taking the disjoint union of two graphs.

6.2. A fixed-parameter algorithm for cardinality-constrained optimization

In this section we present and analyze our algorithm for FIXED-CARDINALITY OPTIMIZATION with respect to the parameters maximum degree Δ in the input graph and size k of the sought solution. As outlined in the introduction (Section 6.1), it is divided into two steps: enumeration of connected subgraphs (Section 6.2.1) and a dynamic programming procedure (Section 6.2.3). Applied to DENSEST k-SUBGRAPH, we achieve an asymptotic running-time improvement in the exponential part of the running time in comparison to the Random Separation method [CCC06]. In Section 6.2.2 we show the tightness of the running time in the enumeration procedure. This example and further ones are given Section 6.2.4. In Section 6.3 we apply our algorithm to μ-CLIQUE.

6.2.1. Enumerating connected graphs

First, we focus on the case that we only have to consider solutions S such that $G[S]$ is connected. We can find these by a simple but efficient enumeration of all possible solutions. Bounding the running time of the enumeration procedure builds on a result of Bollobás [Bol06, Equation (7)].

Theorem 6.1 (Bollobás [Bol06])**.** Let G be a graph with maximum degree Δ and let v be a vertex in G. There are at most $(e(\Delta-1))^{k-1}$ subtrees[11] of order k in G that contain v.

As Bollobás [Bol06] notes, since each connected graph has a spanning tree, it follows that also the number of connected induced subgraphs of order k in G that contain v is at most $(e(\Delta-1))^{k-1}$. Building on Theorem 6.1, we obtain an enumeration algorithm as follows.

Theorem 6.2. Let $G=(V,E)$ be a graph with maximum degree Δ that is represented as an adjacency list data structure and let v be a vertex in G. There are $O((e(\Delta-1))^{k-1})$ connected k-vertex subgraphs of G that contain v and their vertex sets can be enumerated in $O((e(\Delta-1))^{k-1}\cdot(\Delta+k))$ time.

We call a vertex subset $V'\subseteq V$ *admissible* if it induces a connected subgraph of G that contains v and is of order at most k. We describe a tree Γ, called *search*

[11] By a subtree we refer to a subgraph which is a tree.

tree below, where each of its nodes represents an admissible set and each admissible set is represented by some node in Γ. We then show that the order of Γ is at most $O((e(\Delta-1))^{k-1})$ and that computing Γ in a depth-first fashion can be implemented to run in $O((e(\Delta-1))^{k-1}\cdot(\Delta+k))$ time. Throughout this section vertex v is fixed. We also fix an arbitrary ordering of the vertices in G which assigns a unique *index* to every vertex.

Definition of search tree Γ. Let us describe the search tree $\Gamma=\Gamma(G,v,k)$. Each of its nodes $\mathcal{N}$ is associated with a tuple $\tau(\mathcal{N})=(P,W)$ where $W\subseteq P\subseteq V$. We show below that P is admissible. Intuitively, the subtree of Γ rooted at a node $\mathcal{N}$ associated with (P,W) represents those admissible supersets of P that do not contain a vertex of $N(W)\setminus P$. For this, we will choose a vertex $u\in P\setminus W$ and generate a child of $\mathcal{N}$ for each subset of the neighbors of u that extend P to another suitable admissible set. To avoid adding further neighbors of u to P deeper in the search tree Γ—which would correspond to traversing some parts of the search space multiple times—we use the set W. This set contains all "already processed" vertices that should not contribute any further neighbors to the admissible set P. Hence, when adding neighbors of u we have to avoid any neighbor of a vertex in W.

Continuing the definition of Γ, the root of Γ is associated with $(\{v\},\emptyset)$ and the remaining nodes are defined inductively as follows. Let $\mathcal{N}$ be any node in Γ with its associated tuple (P,W) such that $|W|<|P|<k$ and $N(P)\setminus N(W)\neq\emptyset$, and let u be the vertex with lowest index in $P\setminus W$. For every subset $M\subseteq N(u)\setminus(N(W)\cup P)$ that fulfills $|M|\leq k-|P|$, we add a new child to $\mathcal{N}$ associated with $(P\cup M, W\cup\{u\})$. (Note that $M=\emptyset$ is one of the choices for M.) There are no further nodes in Γ and this concludes its definition. Below we say that the *order* of Γ is its number of nodes.

Representation of admissible sets in search tree Γ. We now prove that each admissible set is represented in Γ. Then we show a technical lemma that proves that admissible sets of size precisely k are contained in the leaf nodes of Γ and that gives us a way to account in a tight way for the work done for inner nodes when constructing the search tree.

Lemma 6.1. (i) In each tuple (P,W) associated with some node of Γ the set P is admissible. Furthermore, (ii) for each admissible set P, there is a set W such that (P,W) is associated with some node of Γ.

Proof. To prove (i) it suffices to observe that the set P associated with the root is admissible and that, if P is admissible for some node $\mathcal{N}$, then the set P for each of its children is. The first part is obvious. For the second part, let $\mathcal{N}$ be associated with (P, W) and let one of its children be associated with (P', W'). Since $G[P]$ is connected and since $P' \setminus P \subseteq N(u)$, where u is the vertex with lowest index in P, also $G[P']$ is connected. Moreover, $|P' \setminus P| \leq k - |P|$ by definition of Γ and, hence, P' contains at most k vertices. Thus, we have proved (i).

It remains to prove (ii). Assume for the sake of contradiction that for some admissible set P there is no node in Γ associated with the set (P, W) for any set W. Consider a node $\mathcal{N}'$ with associated tuple (P', W') in Γ with the longest path to the root such that $P' \subseteq P$ and $N(W') \cap (P \setminus P') = \emptyset$; clearly, such a node exists. Consider the vertex $u' \in P' \setminus W'$ with lowest index and $C = N(u') \cap (P \setminus P')$. Note that $C \cap N(W') = \emptyset$ as $N(W') \cap (P \setminus P') = \emptyset$. Then, the child of $\mathcal{N}'$ with associated tuple $(P' \cup C, W' \cup \{u'\})$ also fulfills our condition on its tuple but has a longer path to the root. This contradicts our choice of $\mathcal{N}'$. □

In order to upper bound the running time of constructing Γ, we divide the nodes of Γ into different types. The first type are *interesting leaves* which are leaves $\mathcal{N}$ of Γ with $\tau(N) = (P, W)$ such that $|P| = k$. The second type are *boring leaves* which are the remaining leaves. Note that for boring leaves we have $|P| < k$ and $P = W$. The third type are the parents of the interesting leaves and the fourth type are all remaining inner nodes which we call *deep inner nodes*.

Lemma 6.2. The search tree Γ is of order at most $(3 + 2/(2^{\Delta-1} - 2))(e(\Delta - 1))^{k-1}$. Moreover, the number of interesting leaves and number of parents of interesting leaves is at most $(e(\Delta - 1))^{k-1}$ each, there are at most $\frac{1}{2^{\Delta-1}-2}(e(\Delta - 1))^{k-1}$ deep inner nodes, and there are at most $(1 + 1/(2^{\Delta-1} - 2))(e(\Delta - 1))^{k-1}$ boring leaves.

Proof. By definition, each node $\mathcal{N}$ with associated tuple $\tau(\mathcal{N}) = (P, W)$ and vertex u with lowest index in $P \setminus W$ has one child for every subset of $R(\mathcal{N}) := N(u) \setminus (P \cup N(W))$ of size at most $k - |P|$. Hence, node $\mathcal{N}$ has at most $\sum_{i=0}^{\min\{|R(\mathcal{N})|, |P|-k\}} \binom{|R(\mathcal{N})|}{i}$ children. In particular, the root of Γ has at most 2^{Δ} children and each deep inner node has at most $2^{\Delta-1}$ children (as some neighbor of u has to be in P already). Assume for the moment that a search tree $\Gamma^* = \Gamma^*(G^*, v^*, k)$ exists that achieves the maximum possible number of children in each node. Clearly, search tree Γ^* contains at least as many nodes as Γ because search tree Γ can be embedded into Γ^* in the natural way. Search tree Γ^* also contains at least as many deep inner nodes. Furthermore, every boring leaf $\mathcal{N}$ of search tree Γ can be mapped to a unique boring leaf of Γ^*: take the natural embedding of Γ into Γ^*, the node $\mathcal{N}^*$ to which $\mathcal{N}$ corresponds in Γ^* and then recursively follow the (unique) child of $\mathcal{N}^*$, $\tau(\mathcal{N}^*) = (P^*, W^*)$ such that no vertex is added to P^*. Hence, search tree Γ^* contains at least as many boring leaves as Γ. Similarly, search tree Γ^* also contains at least as many interesting leaves.

It thus suffices to upper bound the numbers of nodes of Γ^* instead of the corresponding numbers of nodes of Γ. Let us fix a concrete G^* and v^* in $\Gamma^* = \Gamma^*(G^*, v^*, k)$ (in particular, let us show that Γ^* as defined above exists). Take as G^* any tree that is rooted at some vertex v^*, such that each vertex in G^* has degree exactly Δ and that has depth at least $k-1$. As there are no two distinct paths from v^* to any other vertex, in each node $\mathcal{N}$ of Γ^* with associated tuple $\tau(\mathcal{N}) = (P, W)$ and vertex u with lowest index in $P \setminus W$ we indeed have $|N(u) \setminus (P \cup N(W))| = |N(u) \setminus P| = |N(u)| - 1 = \Delta - 1$, except for the root, where we have $|N(u) \setminus (P \cup N(W))| = |N(u)| = \Delta$. Hence, Γ^* achieves the maximum possible number of children in each node.

We first upper bound the number of interesting leaves in Γ^*. Since $|P| = k$ in an interesting leaf $\mathcal{N}$ with associated tuple (P, W), graph $G^*[P]$ induces an order-k subtree of G^* that contains v^*. This mapping from the leaves to the set of order-k subtrees of G^* that contain v is injective: Assume the contrary, that is, there are two interesting leaves $\mathcal{N}_1, \mathcal{N}_2$ of Γ^* that are mapped to the same tree. Consider the node $\mathcal{N}$ in Γ^* with the longest path to the root such that $\mathcal{N}$ lies on both paths from the root to $\mathcal{N}_1$ and $\mathcal{N}_2$. Consider the tuple $\tau(\mathcal{N}) = (P, W)$ associated with $\mathcal{N}$, the successors $\mathcal{N}_1', \mathcal{N}_2'$ of $\mathcal{N}$ on the corresponding paths, and their associated tuples $\tau(\mathcal{N}_1') = (P_1, W_1)$, $\tau(\mathcal{N}_2') = (P_2, W_2)$. There is a vertex w in exactly one of P_1, P_2 that is not in P, say $w \in P_1 \setminus P_2$. Thus, in the tree induced by leaf $\mathcal{N}_1$, the vertex w is present whereas in the tree induced by $\mathcal{N}_2$ the vertex w is missing. This is a contradiction to our assumption. Hence, there is an injective mapping from the interesting leaves of Γ^* to the set of order-k subtrees of G^* that contain v^* and, invoking Theorem 6.1, their number is at most $(e(\Delta-1))^{k-1}$. Clearly, the set of parents of interesting leaves also has at most this cardinality.

We next aim to upper bound the number of deep inner nodes of Γ^*. First note that any inner node of Γ^* has at most one boring leaf as child. Hence, if we remove the leaves from Γ^*, we obtain a tree Γ' in which each inner node has at least $2^{\Delta-1}-1$ children. Furthermore, as in each non-root node $\mathcal{N}$ of Γ' with associated tuple (P, W) we have $|N(u) \setminus (P \cup N(W))| = |N(u)| - 1$, each node $\mathcal{N}$ is the ancestor of a parent of a interesting leaf in Γ^*. Thus the leaves in Γ' are a subset of the parents of interesting leaves in Γ^*, that is, there are at most $(e(\Delta-1))^{k-1}$ leaves in Γ'. Using that each inner node in Γ' has at least $2^{\Delta-1}-1$ children, the number of inner nodes in Γ' is at most $1/(2^{\Delta-1}-2)(e(\Delta-1))^{k-1}$ which is also the number of deep inner nodes in Γ^*.

Finally, as noted above, each inner node of Γ^* has at most one boring leaf as child. Hence, the number of boring leaves is at most $(1 + 1/(2^{\Delta-1}-2))(e(\Delta-1))^{k-1}$. □

A corollary of Lemma 6.2 is that the number of subgraphs of G that contain v and are of order *at most* k is at most $3(e(\Delta-1))^{k-1}$.

Enumeration algorithm and proof of Theorem 6.2. Using Lemma 6.2 in a straightforward algorithm to construct the search tree Γ now yields a proof for Theorem 6.2.

Proof of Theorem 6.2. We construct Γ in a depth-first fashion, computing the associated tuples of each node as follows. Clearly, the associated tuple of the root is given. For every node, we will compute the associated tuples of all children. Hence, when processing a node of Γ, we may assume that its tuple is given. We represent P and W in a tuple (P, W) as well as all neighbors $N(W)$ of vertices in W as a set data structure that allows for addition and look-up of single elements in $O(1)$ time [BT93].

Let us describe the procedure for a certain node $\mathcal{N}$ associated with tuple $\tau(\mathcal{N}) = (P, W)$. We first report P as admissible set, which is correct by Lemma 6.1(i) and can be done in $O(k)$ time. Then we create a list containing the elements of $N(u) \setminus (P \cup N(W))$ in arbitrary order, where u is the vertex with lowest index in $P \setminus W$; vertex u is retrievable in $O(k)$ time. It is possible to create the list in $O(\Delta)$ time because of the set data structures used for P and $N(W)$. Once we created the list of $N(u) \setminus (P \cup N(W))$, we iterate over all its sublists L corresponding to some set M with $|M| \le k - |P|$ and, for each of them, make a recursive call for the child with associated tuple $(P \cup M, W \cup \{u\})$. The recursive call includes computing the tuple, which is possible in $O(k)$ time, and updating the set data structure for the vertices in $N(W)$, possible in $O(\Delta)$ time. As creating the sublists is possible in time linear in the number of sublists, processing each node $\mathcal{N}$ thus takes $O((\Delta + k)) \cdot (C(\mathcal{N}) + 1)$ time where $C(\mathcal{N})$ is the number of children of $\mathcal{N}$ in Γ.

We now derive the overall running time for constructing Γ. Clearly, each leaf of Γ contributes $O(\Delta + k)$ processing time, amounting to $O((e(\Delta-1))^{k-1}(\Delta + k))$ overall by Lemma 6.2. The overall processing time contributed by the parents of the leaves is $O(\Delta + k)$ times the number of leaves, hence yielding the same asymptotic upper bound. Next, each remaining node in Γ has at most 2^{Δ} children and hence contributes $O(2^{\Delta}(\Delta + k))$ processing time. Using the upper bound on their number from Lemma 6.2, their overall contribution is $O(2^{\Delta}(\Delta + k)) \cdot (1/(2^{\Delta-1} - 2) \cdot (e(\Delta-1))^{k-1} = O((e(\Delta-1))^{k-1} \cdot (\Delta + k))$. □

While the above enumeration scheme seems quite simple, its running time is asymptotically almost optimal. We show in Section 6.2.2 that, if G is a $(\Delta - 1)$-ary tree, then there are many admissible sets, closely matching the upper bound following from Lemma 6.2.

Consequence for FIXED-CARDINALITY OPTIMIZATION. Using Theorem 6.2 we obtain the following by simply starting the enumeration process from each vertex in G.

Theorem 6.3. Let (G, ϕ, k) be an instance of FIXED-CARDINALITY OPTIMIZATION such that

- G has maximum degree Δ,
- $\phi(S) = 0$ if $G[S]$ is not connected, and
- $\phi(S)$ can be evaluated in $T(k, G)$ time.

Then, (G, ϕ, k) can be solved in $O((e(\Delta-1))^{k-1} \cdot (\Delta + k) \cdot n \cdot T(k, G))$ time.

There are many natural FIXED-CARDINALITY OPTIMIZATION problems that fulfill the conditions of Theorem 6.3. For example, if we take $\phi_G(S)$ to be the diameter of $G[S]$, we arrive at an optimization version of the s-CLUB problem [BBT05; BP13; Sch+12] which asks to find a k-vertex subgraph that has diameter at most s. Note that this problem is nontrivial only if $k > \Delta$. As a consequence, the above theorem provides a refined running time in comparison to the previously reported overall running time of $O((k-2)^k \cdot k! \cdot kn + nm)$ for s-CLUB parameterized by k [Sch+12].

Using a straightforward application of the Random Separation method [CCC06], we obtain $O(2^{(\Delta+1)\cdot k} \cdot T(k, G) \cdot (n + m))$ running time for the special case of FIXED-CARDINALITY OPTIMIZATION above. Hence, in comparison to Random Separation, Theorem 6.3 gives an improved exponential part of the running time and we do not need to rely on randomization. Moreover, often $T(k, G) \leq f(k, \Delta)$ for some function f. In this case our algorithm runs in linear time for fixed values of Δ and k whereas the derandomization of the Random Separation approach adds a factor of $\log n$[CCC06].

6.2.2. A lower bound on the number of connected subgraphs

We now show that the exponential term $(e(\Delta-1))^{k-1}$ in Theorem 6.2 is asymptotically optimal. For the corresponding lower bound, we use the fact that the number of ℓ-ary ordered trees with k inner vertices is exactly $\binom{\ell k}{k}/((\ell-1)k+1)$ (see Hilton and Pedersen [HP91], for example). In ordered trees the order of the children of a vertex matters. For example, adding two children to the “left” leaf of a vertex with two leaf children yields a different binary ordered tree than adding two children to the “right” leaf. More formally, an *ℓ-ary ordered tree* is uniquely described by a vector in $\mathbb{N}^\ell$ for each vertex u, such that the number of vertices induced by the subtree of the ith child of u is exactly the ith entry in u's vector and each vertex has either ℓ or zero children.

Lemma 6.3. For every pair of integers k and Δ there is a graph G with maximum degree Δ and a vertex v such that there are at least $(1/((\Delta-2)k+1))\cdot\binom{(\Delta-1)k}{k}$ connected subgraphs of order at most k that contain v.

Proof. Take G to be a tree with root v where every vertex has either zero or $\Delta-1$ children and the path of each leaf to v is of length at least k. Give the children of every vertex in G an arbitrary order and consider an arbitrary *ordered* rooted tree T with k vertices such that every vertex has at most $\Delta-1$ children. Observe that T induces a subtree of G by identifying the roots and then embedding the remaining vertices in the natural way. In fact, in this way we obtain a bijection between the k-vertex subtrees of G and the corresponding ordered k-vertex trees. Furthermore, by taking an arbitrary ordered k-vertex tree such that each vertex has at most $\Delta-1$ children and adding leaves to each vertex which has less than $\Delta-1$ children we obtain a $(\Delta-1)$-ary ordered tree with k inner vertices. Vice versa, removing all leaves from a $(\Delta-1)$-ary ordered tree with k inner vertices we obtain a corresponding k-vertex tree. Hence, the number of k-vertex subtrees of G is lower-bounded by the number $\binom{(\Delta-1)k}{k}/((\Delta-2)k+1)$ of $(\Delta-1)$-ary ordered trees with k inner vertices. □

Consider the number of admissible sets in the search tree Γ from Section 6.2.1. Applying Proposition 1.1 to the binomial coefficient in Lemma 6.3, we obtain that in the worst case the number of admissible sets can be at least

$$c\left(\frac{\Delta-1}{\Delta-2}\right)^{(\Delta-1)k}\cdot\frac{(\Delta-2)^k}{\Delta k^{3/2}}$$

for some constant $c>0$ independent of k and Δ. Let us consider the ratio ϱ "upper bound divided by lower bound" for fixed k. The kth root of the ratio ϱ is

$$\Theta\left(\frac{e(\Delta-1)}{\left(\frac{\Delta-1}{\Delta-2}\right)^{\Delta-1}(\Delta-2)}\right)=\Theta\left(\frac{e}{\left(\frac{\Delta-1}{\Delta-2}\right)^{\Delta-2}}\right)=\Theta\left(\frac{e}{\left(1+\frac{1}{\Delta-2}\right)^{\Delta-2}}\right).$$

Because $\lim_{\Delta\to\infty}(1+1/(\Delta-2))^{\Delta-2}=e$, the ratio approaches some constant, meaning that the upper bound is asymptotically tight.

6.2.3. Combining solutions from different connected graphs

In the case of general objective functions ϕ, the optimal solution for ϕ could be a vertex set S such that $G[S]$ is not connected. This is in particular the case for

DENSEST k-SUBGRAPH: If the input graph consists for example of two cliques on $k/2$ vertices that are connected by a long path of degree-two vertices, then, for sufficiently large k, the optimal solution consists exactly of the two cliques and it is thus disconnected. However, the objective function of DENSEST k-SUBGRAPH has, as many objective functions for FIXED-CARDINALITY OPTIMIZATION, the useful property that connected components can be considered independently when evaluating the function. This can be formalized using component linearity; for convenience, we repeat below the definition from the introductory section.

Definition 6.3. Let $G = (V, E)$ be a graph, let $\phi\colon 2^V \to \mathbb{Q}$ be an objective function, let $S, T \subseteq V$ be arbitrary disjoint vertex sets, and let $\chi\colon \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$ such that χ is non-decreasing in both arguments individually.

- We call ϕ *component sub-χ* if the functions ϕ and χ have the property that $\phi(S) \le \chi(\phi(W), \phi(S \setminus W))$ where W is an arbitrary connected component of $G[S]$.
- We call ϕ *super-χ* if $\phi(S \cup T) \ge \chi(\phi(S), \phi(T))$.
- If both the above properties hold for ϕ and χ, then we call ϕ *component χ-linear*.

The idea behind component χ-linearity is as follows. The value of $\phi(S)$ can be evaluated by first evaluating subsets $S_1, S_2 \subseteq S$ of S. If ϕ is component χ-linear, then the only possibility for the value of $\phi(S)$ to be much larger than a "combination" of $\phi(S_1)$ and $\phi(S_2)$ is if there are edges between S_1 and S_2. Hence, connected components of $G[S]$ can be first evaluated separately and then be combined.

It is easy to check that, taking $\phi(S)$ as the number of edges in $G[S]$ and $\chi(a, b) = a + b$ we obtain an objective function corresponding to DENSEST k-SUBGRAPH which is component $+$-linear. In the following, we extend our enumeration algorithm to FIXED-CARDINALITY OPTIMIZATION with component χ-linear objective functions. In Section 6.2.4 we then demonstrate the usefulness of component χ-linearity via several examples. Our algorithm for FIXED-CARDINALITY OPTIMIZATION, based on the *color coding* technique [AYZ95], is randomized with false negatives. It can be derandomized with an additional running time factor of $2^{O(k)} \cdot \log n$ [AYZ95].

Algorithm description. Let S be a vertex set of size k such that $\phi(S)$ is maximum. The basic idea of color coding is to color the vertices of the input graph uniformly at random with a set C of k colors and to hope that S is *colorful*, that is, for each color in C there is exactly one vertex in S that has received this color. With some nonzero probability, the coloring will satisfy this property. Assuming the graph is colored this way, first use the enumeration algorithm to find all connected components of $G[S]$.

Then combine these connected graphs by applying dynamic programming. The fact that the solution is colorful enables us to define which connected components can be combined and which cannot. The color-coding/enumeration/dynamic programming routine is repeated sufficiently often to achieve constant error probability. The details are as follows.

Apply the coloring to the vertex set. After the coloring, first compute for every subset C' of C the connected subgraph $G[S']$ that maximizes $\phi(S')$ among all connected subgraphs whose vertices have color set C'. This can be easily achieved by adapting the enumeration algorithm of Theorem 6.2 to only report colorful connected graphs and then evaluating ϕ for each enumerated colorful graph $G[S']$. We fill a table $\mathcal{D}$ storing the currently best value for each color subset as follows. We initialize $\mathcal{D}$ by setting $\mathcal{D}(C') := -\infty$ for all $C' \subseteq C$. If during the enumeration we find some S' with color set C' and $\mathcal{D}(C') \leq \phi(S')$, then we set $\mathcal{D}(C') \leftarrow \phi(S')$. After the enumeration of all connected subgraphs the entry $\mathcal{D}(C')$ contains exactly the maximum objective value among all connected subgraphs with color set C'. Afterwards, we find the maximum objective value of any graph with color set C using another table $\mathcal{T}$. Here, the entry in $\mathcal{T}(C')$ for some color set $C' \subseteq C$ contains the maximum of $\phi(S)$ for all vertex sets S with color set C'. We can fill $\mathcal{T}(C')$ by the following recurrence:

Lemma 6.4. The tables $\mathcal{T}$ and $\mathcal{D}$ have the property that

$$\mathcal{T}(C') = \max\{\mathcal{D}(C'), \max_{C'' \subset C'} \chi(\mathcal{T}(C''), \mathcal{T}(C' \setminus C''))\}.$$

Proof. We first prove that the left-hand side is at most as large as the right-hand side and then the other direction.

"$\leq$": Let $S \subseteq V$ be a "witness" for $\mathcal{T}(C')$, that is, $G[S]$ is colorful with color set C' and $\phi(S) = \mathcal{T}(C')$. If $G[S]$ is connected, then $\mathcal{T}(C') = \mathcal{D}(C')$, as required. If $G[S]$ is disconnected, then for an arbitrary connected component W of S we have $\mathcal{T}(C') \leq \chi(\phi(W), \phi(S \setminus W))$ because ϕ is component sub-χ. Since χ is non-decreasing we have

$$\chi(\phi(W), \phi(S \setminus W)) \leq \chi(\mathcal{D}(C''), \mathcal{T}(C' \setminus C'')) \leq \chi(\mathcal{T}(C''), \mathcal{T}(C' \setminus C''))$$

where C'' is the set of colors in $G[W]$. Thus $\mathcal{T}(C') \leq \chi(\mathcal{T}(C''), \mathcal{T}(C' \setminus C''))$.

"$\geq$": We have $\mathcal{T}(C') \geq \mathcal{D}(C')$ by definition. Let S and T, $S \cap T = \emptyset$, be witnesses for $\mathcal{T}(C'')$ and $\mathcal{T}(C' \setminus C'')$, respectively, for a set $C'' \subseteq C'$. Since ϕ is super-χ, we have $\phi(S \cup T) \geq \chi(\phi(S), \phi(T))$ and hence also $\mathcal{T}(C') \geq \chi(\mathcal{T}(C''), \mathcal{T}(C' \setminus C''))$. □

Thus, after filling $\mathcal{T}$ according to the above recurrence, the maximum objective value of any colorful solution with k vertices is stored in $\mathcal{T}(C)$.

By repeating the above algorithm $O(e^k)$ times one can achieve a constant error probability that in one of the repetitions, called *trials*, the solution has indeed obtained a colorful coloring. If $T(k,G)$ is an upper bound on the time needed to evaluate ϕ and χ, then this leads to an overall running time of $O((e^2(\Delta-1))^{k-1} \cdot (\Delta+k)\cdot n)\cdot T(k,G)$ for the algorithm: For each trial the table $\mathcal{T}$ can be evaluated in $O(3^k)\cdot T(k,G)$ time,[12] and thus the dominating part of the running time is the subgraph enumeration procedure.

Speed-up using more colors. In the following, we describe how this running time can be further improved by employing a known speed-up trick for color coding (see Deshpande, Barzilay, and Karger [DBK07] and Hüffner, Wernicke, and Zichner [HWZ08]). The idea is to increase the number of colors, that is, to use ck colors when coloring the vertices, for an integer $c > 1$. This modification has two effects. On the one hand, it increases the probability that the solution is colorful which reduces the number of necessary trials. On the other hand, it increases the running time needed for the dynamic programming, since the table now has $\Theta(2^{ck})$ entries. Hence, there is a running time trade-off between the two parts of the algorithm. In our application, we can observe that the dominating part of the running time for each trial is the subgraph enumeration which does not depend on the number of colors but only on k. Hence, it makes sense to increase the number of colors as long as the dynamic programming part is still faster than the subgraph enumeration part. As a result, we can achieve a sizable speed-up, the concrete analysis is as follows.

First, the probability P_c, $c > 1$, that an optimal solution $S \subseteq V$ is colorful when coloring V uniformly at random with ck colors is

$$\begin{aligned} P_c &= \frac{\binom{ck}{k}k!}{(ck)^k} \geq \frac{1}{e^2}\sqrt{\frac{c}{2\pi(c-1)k}}\left(\frac{c}{c-1}\right)^{ck}(c-1)^k\sqrt{2\pi}k^{k+\frac{1}{2}}\frac{1}{(eck)^k} \\ &= \frac{1}{e^2}\sqrt{\frac{c}{c-1}}\left(\left(\frac{c}{c-1}\right)^c\frac{c-1}{ec}\right)^k = \frac{1}{e^2}\sqrt{\frac{c}{c-1}}\left(\left(\frac{c}{c-1}\right)^{c-1}\frac{1}{e}\right)^k \end{aligned}$$

where the inequality holds due to Stirling's approximation and Proposition 1.1. We make $\lceil 1/P_c\rceil$ coloring trials. The probability to have at least one trial in ℓ such that

[12]We include the function T in this time bound since the values produced by ϕ and χ could be very large in the general case.

the optimal solution S is colorful is $1-(1-P_c)^\ell$. The probability of success is thus at least $1-1/e$ because $1-(1-P_c)^{1/P_c} \geq 1-1/e$. Then the enumeration of the connected subgraphs takes $O(1/P_c \cdot (e(\Delta-1))^{k-1} \cdot (\Delta+k) \cdot n \cdot T(k,G))$ time overall, whereas the dynamic programming part contributes $O(1/P_c \cdot 3^{ck}) \cdot T(k,G)$ time. Hence, if we choose c in such a way that $3^{ck} \in O((e(\Delta-1))^{k-1})$, then the overall running time is dominated by the enumeration procedures. We claim that this is the case if we let c be smallest possible such that $c \geq (1-1/k)\cdot\log_3(e(\Delta-1))$ and such that ck is an integer. First we note that it is possible to choose c in this way, that is, $(1-1/k)\cdot \log_3(e(\Delta-1)) \geq 1$. Indeed, this is true if and only if $\log_3 e + \log_3(\Delta-1) \geq k/(k-1)$. Note that $\Delta, k \geq 3$ without loss of generality,[13] and thus $\log_3 e + \log_3(\Delta-1) \geq 3/2$ and $3/2 \geq k/(k-1)$. Hence we may choose c in this way. We see $3^{ck} \in O((e(\Delta-1))^{k-1})$ as follows. Because increasing c by $1/k$ increases the integer part of ck, we have $c \leq (1-1/k)\log_3(e(\Delta-1)) + 1/k$ and thus $3^{ck} \leq 3^{(k-1)\log_3(e(\Delta-1))+1} = 3(e(\Delta-1))^{k-1}$. Since the probability P_c is monotonously ascending for increasing c, the overall running time is in the order of

$$\frac{1}{P_{\log_3(e(\Delta-1))}} \cdot (e(\Delta-1))^{k-1} \cdot (\Delta+k)\cdot n \cdot T(k,G)$$

$$\sim \left(\left(\frac{\log_3(e(\Delta-1))-1}{\log_3(e(\Delta-1))}\right)^{\log_3(e(\Delta-1))-1} e^2(\Delta-1)\right)^{k-1} \cdot (\Delta+k)\cdot n\cdot T(k,G)$$

$$= \left(\left(\log_{e(\Delta-1)} \frac{e(\Delta-1)}{3}\right)^{\log_3 \frac{e(\Delta-1)}{3}} e^2(\Delta-1)\right)^{k-1} \cdot (\Delta+k)\cdot n\cdot T(k,G).$$

It seems complicated to bring the base in the exponential term, let us call the base γ, into a more readable form. We give some numerical evaluations in Table 6.2.

Note that, choosing c as above, the base in the exponential factor of P_c tends to one for increasing Δ. Thus the overall running time of the algorithm tends to $O((e(\Delta-1))^k \cdot (\Delta+k)\cdot n)\cdot T(k,G)$ as Δ increases. For example, for all $\Delta \geq 5$ we obtain the upper bound of $O(((e+0.9)(\Delta-1))^k \cdot (\Delta+k)\cdot n)\cdot T(k,G)$. Concrete values for γ are given in Table 6.2. Concluding, we obtain the following theorem.

[13] If $\Delta \leq 2$, then the number of k-vertex subgraphs is linear in the number of vertices and if $k \leq 2$, then the number of k-vertex subgraphs is $O(\Delta n)$. In both cases enumeration is possible in linear time in the number of subgraphs, and, together with color coding and dynamic programming, it is not hard to obtain a running time of $O(3^k \cdot \Delta \cdot n)\cdot T(k,G)$ for optimizing the objective function.

Δ	3	4	5	6	10
$\gamma/(\Delta-1)$	4.2	3.8	3.6	3.5	3.4
γ	8.4	11.3	14.4	17.5	29.8

Table 6.2.: Approximate values of the base γ in the exponential running time factor in the improved color coding/dynamic programming routine.

Theorem 6.4. Let (G,ϕ,k) be an instance of FIXED-CARDINALITY OPTIMIZATION such that

- G has maximum degree $\Delta > 2$,
- ϕ is component χ-linear, and
- ϕ as well as χ can be evaluated in $T(k,G)$ time.

Then, (G,ϕ,k) can be solved in $O(\gamma^{k-1}\cdot(\Delta+k)\cdot n)\cdot T(k,G)$ time, reporting a yes-instance as a no-instance with probability at most $1/e$. Herein,

$$\gamma = (\log_\beta(\beta/3))^{\log_3(\beta/3)} e\beta$$

and $\beta = e(\Delta-1)$. In particular, $\gamma \le 4.2\cdot(\Delta-1)$.

6.2.4. Concrete examples for component linear functions

We conclude this section with concrete examples for component-linearity (Definition 6.3) and the resulting algorithm for FIXED-CARDINALITY OPTIMIZATION (Theorem 6.4).

Random Separation. The algorithm underlying Theorem 6.4 applied to DENSEST k-SUBGRAPH has a worst-case running time of $O((4.2\cdot(\Delta-1))^{k-1}\cdot(\Delta+k)\cdot k^2\cdot n)$. Using Random Separation, DENSEST k-SUBGRAPH can be solved in $2^{O(\Delta k)}\cdot(\Delta+k)\cdot n$ time with one-sided error and constant error probability [CCC06]; our algorithm thus improves on this running time.

A non-example: Independent sets. An objective function that is *not* component χ-linear for any function χ as in Definition 6.3 is $\phi(S)$ defined as "maximum size of an independent set in $G[S]$". To see this, take any function χ that fulfills the two properties demanded by Definition 6.3. Then, $\chi(\phi(S),\phi(T)) \le \max\{\phi(S),\phi(T)\}$: For any i and j, the complete bipartite graph with partite sets A and B of size i and j, respectively, fulfills $\phi(A \cup B) = \max\{|A|, |B|\} = \max\{\phi(A), \phi(B)\}$. Since ϕ is super-χ we thus have $\chi(i, j) \leq \max\{i, j\}$. However, for vertex sets S and a connected component W in $G[S]$ we have

$$\phi(S) \;=\; \phi(W) + \phi(S \setminus W) \leq \chi(\phi(W), \phi(S \setminus W)) \leq \max\{\phi(W), \phi(S \setminus W)\}$$

because ϕ is component sub-χ. This is absurd because for nonempty W and $S \setminus W$ we clearly have $\phi(W) + \phi(S \setminus W) > \max\{\phi(W), \phi(S \setminus W)\}$.

Multiple objectives. We now demonstrate the merit of stating Definition 6.3 in the present general way. Suppose we are given a graph $G = (V, E)$ and are asked to decide whether among the densest k-vertex subgraphs of G, there is one with a connected component of size at least ℓ. We may define a corresponding objective function

$$\phi(S) \;:=\; |E(G[S])| \cdot 2^{\lceil \log k \rceil} + \max\{|W| \mid W \text{ is a connected component of } G[S]\},$$

that is, the last $\lceil \log k \rceil$ bits of $\phi(S)$ are reserved for the size of a largest connected component of $G[S]$. Then we can define a function $\chi(a, b)$ that sums the first bits and takes the maximum of the last $\lceil \log k \rceil$ bits. This function is monotonously ascending in both arguments and it is easy to check that ϕ is component sub-χ. To see that it is super-χ it suffices to observe that the number of edges in $G[S \cup T]$ is at least as large as the sum of the edges in $G[S]$ and $G[T]$ for disjoint S, T and similarly for the size of the largest component. Hence ϕ is component χ-linear and we may apply Theorem 6.4 to decide the above property of G.

Fixed-cardinality graph partitioning. Our framework is applicable to a subclass of Fixed-Cardinality Optimization, the so-called fixed-cardinality graph partitioning problems [SZ14; Bon+15]. In these problems a graph $G = (V, E)$ and an integer p is given and the task is to decide whether there is a k-vertex subset $S \subseteq V$ such that $\phi(S) := a|E(S)| + b|\delta(S)|$ is at least p. Here, a, b, and k are fixed constants, $E(S)$ is the set of edges in $G[S]$, and $\delta(S)$ is the set of edges with exactly one endpoint in S. It is not hard to show that, whenever $a \geq 2b$, then ϕ is component $+$-linear:

We extend the definition of δ to $\delta(S,T)$, which denotes the set of edges with one endpoint in S and one in T for disjoint vertex sets S,T. Clearly, for any two disjoint vertex sets S,T, we have

$$\begin{aligned}\phi(S\cup T) &= a|E(S\cup T)|+b|\delta(S\cup T)|\\ &= a(|E(S)|+|E(T)|+|\delta(S,T)|)+b(|\delta(S)|+|\delta(T)|-2|\delta(S,T)|)\\ &= \phi(S)+\phi(T)+(a-2b)\cdot|\delta(S,T)|.\end{aligned}$$

Hence, if T is a connected component of $S\cup T$, then $\phi(S\cup T)=\phi(S)+\phi(T)$ and thus ϕ is component sub-+. Further, if $a-2b\geq 0$, that is, $a\geq 2b$, then ϕ is super-+, showing that ϕ is component +-linear.

Thus, Theorem 6.4 is also applicable to any fixed-cardinality graph partitioning problem with $a\geq 2b$. Combining Theorem 6.4 with an $\Delta^k\cdot\text{poly}(n)$-time algorithm of Bonnet et al. [Bon+15] for the case $a\leq 2b$ yields an algorithm for general fixed-cardinality graph partitioning problems with running time $\gamma^{k-1}\cdot\text{poly}(n)$, where γ is as defined in Theorem 6.4. For all $\Delta\geq 4$ the resulting algorithm improves on the running time upper bound $4^{k+o(k)}\cdot\Delta^k\cdot\text{poly}(n)$ given by Shachnai and Zehavi [SZ14].

6.3. Application to finding μ-cliques

We now describe how to use the algorithms presented for FIXED-CARDINALITY OPTIMIZATION in order to obtain fixed-parameter algorithms for μ-CLIQUE. More precisely, this will lead to fixed-parameter algorithms for the parameters maximum degree Δ of G and the h-index of G, and for the combined parameter that comprises $n-k$ and degeneracy d of G. Before presenting these algorithms, we observe relationships between the vertices in μ-cliques and the sparsity parameters under consideration.

6.3.1. Upper-bounding the solution size

The relation between the order of a μ-clique and its maximum degree, h-index, and degeneracy is as follows.

Lemma 6.5. A μ-clique with

(i) maximum degree Δ contains at most $\Delta/\mu+1$ vertices.
(ii) h-index h contains at most $(h\cdot(h-1)+2\cdot(n-h)\cdot h)/(\mu\cdot(n-1))<2\cdot h/\mu$ vertices.
(iii) degeneracy d contains less than $(4\cdot d+\mu)/2\cdot\mu$ vertices.

Proof. Let G be a n-vertex μ-clique.

Statement (i): If G has maximum degree Δ it has at most $n \cdot \Delta/2$ edges. Since its density is at least μ it has at least $\mu \cdot \binom{n}{2}$ edges. Combining the two statements we get

$$\mu \cdot \frac{n \cdot (n-1)}{2} \leq \frac{n \cdot \Delta}{2}$$

and thus $n \leq \Delta/\mu + 1$.

Statement (ii): If G has h-index h it has at most h vertices of degree more than h. Hence, at least $n-h$ vertices have degree at most h. Let H denote the set of at most h vertices that have degree more than h, and let E_H denote the set of edges with both endpoints in H. Clearly, $|E_H| \leq \binom{h}{2}$. Since all vertices in $V \setminus H$ have degree at most h, there can be at most $(n-h) \cdot h$ edges incident with these vertices. Hence, the total number of edges in G is at most $\binom{h}{2} + (n-h) \cdot h$. Combining this with the lower bound $\mu \cdot \binom{n}{2}$ for the edge number of G we get

$$\mu \cdot \frac{n \cdot (n-1)}{2} \leq \frac{h \cdot (h-1)}{2} + (n-h) \cdot h$$

and thus

$$n \leq \frac{h \cdot (h-1) + 2 \cdot (n-h) \cdot h}{\mu \cdot (n-1)} \leq \frac{2 \cdot h}{\mu}.$$

Statement (iii): If G has degeneracy d, it has at most $d \cdot (n - \frac{d+1}{2})$ edges. Thus,

$$\mu \cdot \frac{n \cdot (n-1)}{2} \leq d \cdot \left(n - \frac{d+1}{2}\right),$$

which implies

$$n \leq \frac{2 \cdot d + \mu + \sqrt{4d^2 - 4d^2\mu + \mu^2}}{2 \cdot \mu} < \frac{4 \cdot d + \mu}{2 \cdot \mu}.$$ □

The upper bound $(h \cdot (h-1) + 2 \cdot (n-h) \cdot h)/(\mu \cdot (n-1))$ on the number of vertices in μ-cliques is tight as a graph consisting of a h-vertex clique and of $n-h$ further vertices that are an independent set but adjacent to all vertices of the clique has density exactly μ if n is equal to the upper bound. It is not hard to see that also the upper bound for the number of vertices with respect to the maximum degree is tight. The bound with respect to the degeneracy can be improved slightly.

6.3.2. Parameter maximum degree

The fixed-parameter algorithms for the maximum-degree parameter can be obtained by a straightforward application of the generic algorithms described in Section 6.2 with the size bounds given by Lemma 6.5.

In most application settings for μ-CLIQUE, one would add the further constraint that the solution has to induce a connected subgraph. By Lemma 6.5, we have $k \leq \Delta/\mu + 1$. Furthermore, for each vertex set $S \subseteq V$ of size at most k, we can compute the number of edges in $G[S]$ in $O(k \cdot \Delta)$ time: for each vertex v of S find all neighbors of v in S by traversing its adjacency list once. Hence, $T(k, G)$, the time needed to compute the objective function is $O(k \cdot \Delta)$ in this case. Plugging both bounds into the time bound given by Theorem 6.3, we obtain the following.

Proposition 6.1. For any fixed μ, $0 < \mu < 1$, we can determine in $O((e(\Delta-1))^{\Delta/\mu} \cdot \Delta^3/\mu^2 \cdot n)$ time whether G contains a connected k-vertex μ-clique. Herein, Δ is the maximum degree of G.

For the general case, in which the solution S may be disconnected, we may use the running time given in Theorem 6.4 instead, since the objective function "number of edges in a graph" is component-linear.

Theorem 6.5. For any fixed μ, $0 < \mu < 1$, μ-CLIQUE can be solved in time $O((4.2 \cdot (\Delta-1))^{\Delta/\mu} \cdot \Delta^3/\mu^2 \cdot n)$, reporting a yes-instance as a no-instance with probability at most $1/e$. Herein, Δ is the maximum degree in the input graph.

6.3.3. Parameter *h*-index

We now describe how to extend our fixed-parameter results to also hold for the parameter h-index of the input graph. In many practical applications the h-index is much smaller than the maximum degree. For example, social and biological networks have few so-called hubs, that is, vertices of very high degree, and many low-degree vertices. In this case, the h-index is a smaller parameter than the maximum degree Δ.

Algorithm description. The main idea of the algorithm is as follows. Let H be the set of the h vertices with degree at least h, and assume that S is a vertex set of size k such that $G[S]$ is a μ-clique. First, by trying all 2^h subsets of H, guess the set H_S of vertices that are in $S \cap H$. Fix one such set H_S. It remains to determine which vertices of $V \setminus H$ belong to S. The number of edges in S depends on the number of edges between $S \setminus H_S$ and H_S and on the number of edges between vertices of H_S. Hence, our goal in the following is simply to find a subgraph of $V \setminus H$ that maximizes this number.

Accordingly, we compute for every vertex $v \in V \setminus H$ the number $\deg_{H_s}(v)$ of neighbors of v in H_S. Define the value $\phi(S')$, $S' \subseteq V \setminus H$, of a subgraph $G[S']$ of the graph $G[V \setminus H]$ to be

$$\phi(S') := |E(G[S'])| + \sum_{v \in S'} \deg_{H_s}(v).$$

The task is now to find a vertex set $S' \subseteq V \setminus H$ of size $k - |H_S|$ that maximizes $\phi(S')$. The overall maximum number of edges for any subgraph with k vertices that contains H_S is then this value plus the number of edges in $G[H_S]$. The overall optimum solution is simply the maximum among all possible choices of H_S.

Running time. The running time of this algorithm can be upper bounded as follows. We try 2^h different possibilities for H_S. For each possibility, we first compute $\deg_{H_s}(v)$ for each vertex in $V \setminus H$ which can be performed in $O(h \cdot n)$ time since all vertices in $V \setminus H$ have degree at most h. Then, we solve FIXED-CARDINALITY OPTIMIZATION with ϕ as defined above. Clearly, ϕ is component $+$-linear. Furthermore, by Lemma 6.5 we have $k < 2h/\mu$ and thus also $k - |H_S| < 2h/\mu$. Since the maximum degree in $G[V \setminus H]$ is h, solving each instance of FIXED-CARDINALITY OPTIMIZATION can thus be performed in $O((4.2 \cdot (h-1))^{2h/\mu} \cdot h^2/\mu^2 \cdot n)$ time.

Altogether, we obtain the following.

Theorem 6.6. μ-CLIQUE can be solved in $O(2^h \cdot (4.2 \cdot (h-1))^{2h/\mu} \cdot h^2/\mu^2 \cdot n)$ time, reporting a yes-instance as no-instance with probability at most $1/e$ where h is the h-index of the input graph.

6.3.4. Parameters degeneracy and vertex deletion

Our final application of the generic algorithm will lead to a fixed-parameter tractability result for DENSEST k-SUBGRAPH parameterized by the combined parameter degeneracy d and $\ell := n - k$. As we will show in the next section, it is not possible to achieve fixed-parameter tractability for either d or ℓ alone. Hence, it is interesting to study their combination. Recall that in μ-CLIQUE we fix some constant minimum density μ of the sought graph. This is necessary to upper bound the maximum value of k and, ultimately, obtain feasible running time bounds. For the combined parameter (d, ℓ) this constraint can be dropped leading to an algorithm for DENSEST k-SUBGRAPH. The algorithm is mainly based on the following observation.

Lemma 6.6. Let $G = (V, E)$ be a graph and let $S \subseteq V$ such that $|S| = k$ and $G[S]$ has maximum density among all k-vertex subgraphs. Then, there is no vertex in $V \setminus S$ that has degree at least $\ell + d$, where $\ell = n - k$.

Proof. Assume that there is a vertex v of degree at least $\ell + d$ in $V \setminus S$. Since v has at most $\ell - 1$ neighbors in $V \setminus S$, it has at least $d + 1$ neighbors in S. However, because G has degeneracy d, there is a vertex u of degree at most d in $G[S]$. Thus, $G[(S \setminus \{u\}) \cup \{v\}]$ is a graph with at least one edge more than $G[S]$. This contradicts the fact that $G[S]$ is densest possible. □

Algorithm description. We can regard DENSEST k-SUBGRAPH as the problem of deleting a set of exactly ℓ vertices while removing the least possible number of edges. In other words, we aim to solve a minimization variant of FIXED-CARDINALITY OPTIMIZATION where ϕ is defined as

$$\phi(S) := \left(\sum_{v \in S} \deg(v)\right) - |E(G[S])|.$$

We can translate this easily into a maximization variant by changing the sign of the function and adding a normalizing term of n to the contribution of each vertex. Formally, we aim to solve FIXED-CARDINALITY OPTIMIZATION with

$$\phi(S) := \left(\sum_{v \in S} n - \deg(v)\right) + |E(G[S])|.$$

Note that this objective function is component $+$-linear: The first part of the sum is independent of the edges in the subgraph and the second part is simply a sum of the edges. Hence, if a set S consists of two connected components, then the value of $\phi(S)$ is the sum of the objective values of the two, that is, ϕ is component sub-$+$. Further, if S and T are disjoint, then $\phi(S \cup T)$ is the sum of the two objective values plus the number of edges going between S and T in G meaning that ϕ is super-$+$. Consequently, we can apply Theorem 6.4 to the corresponding problem.

By Lemma 6.6 we can focus on the subgraph G' of G that contains only the vertices of degree at most $\ell + d$. Computing G and computing the value of $n - \deg(v)$ for each vertex of G' can be performed in $O(m)$ time. Further, the solution size is constrained to be exactly ℓ. Finally, evaluating $\phi(S)$ for a vertex set in S can be performed in $O((\ell + d) \cdot |S|)$ time. Altogether, this results in the following.

Theorem 6.7. DENSEST k-SUBGRAPH can be solved in $O((4.2 \cdot (\ell + d - 1))^{\ell} \cdot \ell^2 \cdot n + m)$ time, reporting a yes-instance as no-instance with probability at most $1/e$, where $\ell := n - k$ and d is the degeneracy of the input graph.

6.4. W[1]-hardness for parameters degeneracy and solution size

In this section, we present a parameterized reduction that shows the limits of the approach presented above. That is, we show that we cannot replace the h-index by the smaller parameter degeneracy in Theorem 6.6, retaining fixed-parameter tractability. In combination with W[1]-hardness of μ-CLIQUE with respect to the dual parameter $\ell := n - k$, it also follows that it is not possible to drop the degeneracy or ℓ from the parameterization in Theorem 6.7.

To present the corresponding reduction we need to construct a gadget graph of a given density and some further properties as follows.

Lemma 6.7. Given four positive integers a, b, c, and d, where $a < b$ and $d \le c(c-1)/2$, we can construct in poly(a, b, c, d) time a graph G such that

- G is $2(a-1)$-connected, has maximum degree at most $2a$, 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.

The proof is similar to a proof by Guo et al. [Guo+11, Lemma 2]. We make a small tweak to obtain bounded maximum degree.

Proof of Lemma 6.7. 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

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

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

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

it holds that $a \ge m/n > a-1$. Thus, we can first add $(a-1)n$ edges to G in polynomial time such that G is $2(a-1)$-connected and has maximum degree at most $2(a-$

1) [Har62]. Then, we add the remaining at most n edges to G such that they allow for a partition into two matchings, increasing the maximum degree by at most two.

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. □

Next, we give the promised reduction.

Theorem 6.8. For any fixed μ, $0 < \mu < 1$, μ-CLIQUE is W[1]-hard parameterized by (d, k), where d denotes the degeneracy of the input graph.

Proof. We reduce from CLIQUE parameterized by the size s of the sought clique.

CLIQUE
Input: An undirected graph G and a nonnegative integer s.
Question: Does G have a s-vertex clique as a subgraph?

Construction. Let $(G = (V, E), s)$ be an instance of CLIQUE. From (G, s) we construct an equivalent instance of μ-CLIQUE, $0 < \mu < 1$, as follows. First, replace every edge $\{u, v\} \in E$ by a length-two path, that is, remove $\{u, v\}$ from the graph, insert a new vertex $s_{\{u,v\}}$, and make it adjacent to u and v. In the following, let $S := \{s_e \mid e \in E\}$ denote the set of added vertices, and let G_1 denote the graph constructed in this way. Next, we make a useful observation and then continue the description of the construction.

The graph G_1 has the following property.

An induced subgraph $G_1[K]$ of G_1 on $s + \binom{s}{2}$ vertices has at most $2 \cdot \binom{s}{2}$ edges. In case of equality, $K \cap V$ induces a clique on s vertices in G.

This can be shown as follows. Let $K_1 := K \cap V$ denote the vertices of K that are also vertices of G, and let $K_2 := K \setminus K_1$ denote the other vertices of K. By construction, every vertex $v \in K_2$ has in $G_1[K]$ degree at most two. Furthermore, every vertex of K_1 has in $G_1[K]$ only neighbors that are in K_2 and vice versa. Consequently, the number of edges in $G_1[K]$ is at most $2 \cdot |K_2|$. Assume that $G_1[K]$ has more than $2 \cdot \binom{s}{2}$ edges. Then, $|K_2| > \binom{s}{2}$ and thus $|K_1| < s$. In the following, let $x := s - |K_1| = |K_2| - \binom{s}{2}$ denote the number of "excess" vertices from K_2. Clearly, there are at most $\binom{s-x}{2}$ vertices in K_2 that have two neighbors in K_1. Hence, the total number of edges in $G_1[K]$ is at most $2\cdot\binom{s-x}{2}+\left(x+\binom{s}{2}-\binom{s-x}{2}\right)=\binom{s-x}{2}+x+\binom{s}{2}$. A simple calculation shows that for all x with $0\le x<s-2$ this number is decreasing with increasing x. In case $x\ge s-2$, the number of edges in G is clearly at most $\binom{s}{2}+s-1$. In summary, this shows that no subgraph of G_1 on $s+\binom{s}{2}$ vertices can have more than $2\cdot\binom{s}{2}$ edges. It also follows that $2\cdot\binom{s}{2}$ edges can only be achieved in case $|K_1|\ge s$, and since the number of edges is at most $2\cdot|K_2|$ this implies $|K_1|=s$. Finally, this means that each pair of vertices in K_1 has a common neighbor in K_2. By construction, K_1 thus is a clique in G.

To conclude our construction, we add a gadget graph as described in Lemma 6.7: Let $\mu=a/b$ and without loss of generality, let $a\ge 3$. Then, we add the gadget graph H such that H is $2(a-1)$-connected, $2(a-1)\ge 4$, and adding $s+\binom{s}{2}$ vertices and $2\binom{s}{2}$ edges makes H have density exactly μ. Furthermore, H can be constructed in time polynomial in $a,b,s+\binom{s}{2}$, and $2\binom{s}{2}$. Let G^* be the disjoint union of G_1 and H. Set the instance of μ-Clique to $(G^*,s+\binom{s}{2}+|V(H)|)$.

Degeneracy. Since G_1 has degeneracy at most two, and the gadget graph H can be constructed in poly(s) time (observe that we can assume a and b to be constants), the degeneracy of the constructed graph is upper bounded by some polynomial in s.

Correctness. First observe the following. If G has a clique C on s vertices, then H together with $C\cup S_C$ forms a μ-clique in G^*, where $S_C=\{s_{\{u,v\}}\mid u,v\in C\}$. This follows by the definition of H. For the reverse direction, we prove that we can assume that every μ-clique M in G^* on at least $s+\binom{s}{2}+|V(H)|$ vertices contains every vertex of H. If this is true, then, it follows that $|M\cap V(G_1)|=s+\binom{s}{2}$, and, hence, $M\cap V(G)$ induces a clique on s vertices in G because of the property of G_1 we have shown above. Assume that M does not contain some i vertices of H. Then, consider the vertices in $S\cap M$. Each of these vertices has degree at most two and the removal of them makes $M\cap V(G_1)$ an independent set. Thus, we may remove the vertices in $S\cap M$ from M, and then, if $|S\cap M|<i$, remove some further vertices in $M\cap V(G_1)$ from M, and add all the missing vertices of H to M. The removal implies losing at most $2i$ edges, but in adding the missing vertices of H we gain at least $2i$ edges, since, by construction, H is 4-connected. Thus, the correctness of the construction follows. □

From the above reduction, we also obtain a lower bound on the running time of algorithms for μ-Clique. This bound is based on the Exponential Time Hypothesis (see Section 1.1.3).

Theorem 6.9. For any fixed μ, $0 < \mu \leq 1$, μ-CLIQUE cannot be solved in time $2^{o(\Delta)} \cdot \text{poly}(n)$ unless the Exponential Time Hypothesis fails. Herein, Δ is the maximum degree of the input graph.

Proof. Unless the Exponential Time Hypothesis fails, CLIQUE does not have algorithms with running time $2^{o(n)}$ (see Section 1.1.3). We observe that the reduction used in Theorem 6.8 produces instances with maximum degree at most cn, where n is the number of vertices in the CLIQUE instance and c is a constant. Thus, if there is an algorithm with the claimed running time for μ-CLIQUE, there is also a subexponential algorithm for CLIQUE and the Exponential Time Hypothesis fails.

To observe the upper bound on the maximum degree, consider the graph G_1 in the proof of Theorem 6.8. Recall that G is the graph in the CLIQUE instance. Every vertex in G_1 that stems from G has degree at most $n = |V|$ and all remaining vertices have degree two. Now consider the gadget graph H that we added to G_1 to obtain G^* and its parameter a. Since μ is a constant, also a is a constant without loss of generality. Thus, since every vertex in the gadget graph H has degree at most $2a$, the maximum degree in H is bounded by a constant. Thus, the maximum degree in G^* is bounded by cn for a large-enough constant c. □

Clearly, Theorem 6.9 also excludes algorithms with running time $2^{o(h)}$ where h is the h-index of the input graph. We remark that the number of vertices in the produced instance can only be upper bounded by a quadratic polynomial in the number of vertices of the clique instance. Hence, while that would be an interesting result, the exclusion of subexponential algorithms for μ-CLIQUE with respect to the number of vertices and $0 < \mu < 1$ does not follow from Theorem 6.8.

With a slight adaption of the reduction behind Theorem 6.8, we can also exclude polynomial-size problem kernels with respect to the parameters maximum degree and h-index.

Theorem 6.10. Unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, for any fixed μ, $0 < \mu < 1$, there are no polynomial-size problem kernels for μ-CLIQUE with respect to either maximum degree or h-index.

Proof. It suffices to prove the statement for the larger maximum degree parameter. For this, we observe that the reduction used in the proof of Theorem 6.8 implies a simple or-cross-composition from CLIQUE into μ-CLIQUE parameterized by maximum degree. (Recall the definition of or-cross-compositions and their consequences from Section 1.1.3.)

Let several instances of CLIQUE be given and without loss of generality, assume that each instance asks for a k'-vertex clique.[14] Merge the instances into one instance of CLIQUE by taking the disjoint union of the graphs. It is clear that this graph contains a clique of given number of vertices if and only if one of its connected components does. Then apply the reduction used in Theorem 6.8 to the resulting graph. To obtain that this procedure is a cross-composition, it remains to show that the maximum degree in the created instance is upper bounded by a polynomial in the maximum size of the input instances. This follows since the reduction used for Theorem 6.8 does not merge any connected components and the introduced gadget graph has size polynomial in k'. Thus, there is an or-cross-composition from CLIQUE into μ-CLIQUE parameterized by the maximum degree. □

6.5. Concluding remarks

We note that Stahl [Sta13] implemented an algorithm for finding a maximum-size μ-clique based on Theorem 6.5 as his Bachelor's project and we subsequently further developed this implementation (see Komusiewicz, Sorge, and Stahl [KSS15]). While the resulting algorithm was competitive with a branch-and-bound algorithm by Pajouh, Miao, and Balasundaram [PMB14], the running times were still often prohibitively large. We believe that this is owed to the weak structure of μ-cliques, allowing only for rather weak data reduction rules and solution size upper bounds. This leads to only weak (heuristic) pruning of the search tree.

There are many possibilities for future research on the computational problems considered in chapter. Obviously, it would be interesting to improve the presented algorithms. For the case of FIXED-CARDINALITY OPTIMIZATION, a running time of $\Delta^{o(k)} \cdot \text{poly}(n)$ is unlikely: CLIQUE is a special case of this problem and such a running time would imply an $n^{o(k)}$-time algorithm for CLIQUE which contradicts the Exponential Time Hypothesis (see Section 1.1.3). Hence, one could focus on improving the constants in the base of the exponential function here.

For μ-CLIQUE, it would be interesting to obtain a running time of $2^{O(\Delta/\mu)} \cdot \text{poly}(n)$ or to show that such a running time is unlikely with respect to some complexity-theoretic assumptions. Since $k \in O(\Delta/\mu)$ by Lemma 6.5, this corresponds to single-exponential running time with respect to the solution size. It would also be interesting to obtain polynomial-size Turing kernels (see Section 1.1.3) for μ-CLIQUE

[14] If an instance asks for a smaller clique, simply add a new vertex and connect it to all other vertices of this instance.

and any of the considered parameters. Also, is there a better polynomial-time algorithm for μ-CLIQUE on planar graphs than the trivial brute-force XP-algorithm with respect to the degeneracy parameter? Finally, a further restriction that can be made in the area of community detection is to upper bound the size of the neighborhood of the μ-cliques [IIO05; II09; Kom+09]. This represents the intuitive definition of communities as internally dense, but externally sparse graphs. Efficient algorithms exploiting such bounds would be interesting and also practically relevant.

Chapter 7

Highly connected subgraphs

This chapter is based on “Finding Highly Connected Subgraphs” by Falk Hüffner, Christian Komusiewicz, and Manuel Sorge (Proceedings of the 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM ’15) [HKS15]).

7.1. Introduction

The *edge connectivity* of a connected graph is the size of its smallest cut, that is, a set of edges whose removal separates the graph into at least two connected components. An n-vertex graph is *highly connected* if its edge connectivity is larger than $n/2$. An equivalent characterization is that a graph is highly connected if each vertex has degree at least $\lfloor n/2 \rfloor + 1$ (this follows from Chartrand [Cha66, Theorem 1]). Highly connected graphs have been used, for example, by Hartuv et al. [Har+99] to partition a DNA similarity network into clusters, and by Pržulj, Wigle, and Jurisica [PWJ04] to find protein complexes, functional units of proteins, in a cell. Intuitively, highly connected graphs are well-suited to model clusters in similarity networks because similarity is approximately transitive and because we expect that the connectivity of the clusters rises with their size.

We study exact algorithms for the following NP-complete problem:

> HIGHLY CONNECTED SUBGRAPH
> *Input:* An undirected graph $G = (V, E)$ and a nonnegative integer k.
> *Question:* Is there a vertex set $S \subseteq V$ of size exactly k such that $G[S]$ is highly connected?

Note that the set S should have size *exactly* k as opposed to size *at least* k. This is owed to highly connected graphs not being hereditary. That is, it is not necessarily the case that each subgraph of a highly connected graph is also highly connected. Thus, a graph may harbor a k-vertex highly-connected graph, no $k+1$-vertex highly connected graph but still a $k+2$-vertex highly connected graph. With this in mind, demanding S to contain precisely k vertices yields a problem definition that is easier to handle.

In addition to the natural application in analyzing complex networks [Har+00; PWJ04], a weighted variant of HIGHLY CONNECTED SUBGRAPH, in which we have edge and vertex weights and want to find a large-weight highly connected graph, also occurs as a subproblem in an algorithm for partitioning graphs into highly connected components [Hüf+14]. The algorithm is based on an integer linear program with the column generation paradigm, in which successively new variables are introduced that improve the objective function. The weighted variant of HIGHLY CONNECTED SUBGRAPH is equivalent to finding such a new variable.

Since HIGHLY CONNECTED SUBGRAPH is NP-complete (Theorem 7.1 below), to better understand its complexity we explore the "parameter ecology" [FJR13; Nie10; KN12; Har14] of this problem. That is, we determine the parameterized complexity status with respect to several distinct parameters, deriving fixed-parameter algorithms or giving lower bounds. From a practical point of view, we aim to find parameters that offer a favorable tradeoff between good performance guarantees and small parameter values.

The parameters we examine here are as follows.

- The number n of vertices in G.
- The solution size k.
- The number $\ell := n - k$ of vertices to delete to obtain the solution. That is, the dual parameter to the solution size.
- The *degeneracy* of G, the smallest integer d such that each subgraph of G contains at least one vertex of degree at most d.
- The h*-index* of G, the largest integer h such that G contains at least h vertices of degree at least h.
- An upper bound γ on the number of edges between the highly connected graph $G[S]$ and the remaining vertices. We also call γ the *edge isolation* parameter.
- An upper bound α on the number of edges not contained in the highly connected graph $G[S]$. We also call α the *edge deletion* parameter.

There are several relations between these parameters, shown in Figure 7.1. We expect that the parameters n, $n - k$, Δ, and α are not very small in practice. Hence, to obtain promising algorithms we would need performance guarantees that grow very weakly with respect to to these parameters. However, even for the "large" parameters n and $n - k$, we obtain hardness results, that is, W[2]-hardness with respect to $n - k$ and that there is no subexponential-time algorithm with respect to n unless the Exponential Time Hypothesis fails (see Section 1.1.3 for background on the Exponential Time Hypothesis). In contrast, for parameter α we obtain a subexponential-time fixed-parameter algorithm which might be feasible to use even for larger values of α with further algorithm engineering.

The parameters γ, k, h, and d seem to harbor more practical potential: Edge isolation parameters have been used to reasonable success in the past [IIO05; II09; Kom+09]. It is also plausible that the solution size k is rather small in comparison to the input graph size. The h-index h and degeneracy d are generally small in biological and social networks, for example [ES12; ELS13] (see also Section 6.1).

Our contribution. We now list our results in more detail, going from the hardest parameters to the easiest, corresponding roughly to going from small expected parameter values to large ones. Table 7.1 gives an overview on the results. For the

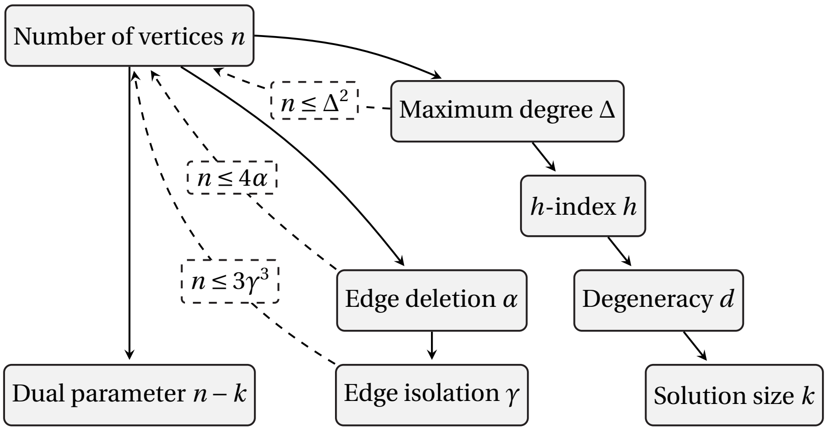


Figure 7.1.: Hasse diagram of the parameters and their boundedness relation in yes-instances of HIGHLY CONNECTED SUBGRAPH. A solid arc means that the target is upper bounded by the source in all yes-instances. A dashed arc means that the target is upper bounded by the source in all yes-instances resulting from the polynomial-time preprocessing routines given in Sections 7.4.1 to 7.4.3.

parameter $\ell := n - k$ (the number of vertices to delete to obtain a highly connected subgraph), we obtain a hardness result: HIGHLY CONNECTED SUBGRAPH admits a trivial $n^{O(\ell)}$-time algorithm, but is W[2]-hard with respect to ℓ (Theorem 7.1). For the number k of vertices in the solution, fixed-parameter tractability is also unlikely since we obtain W[1]-hardness even if we additionally consider the degeneracy of G as a parameter (Theorem 7.2). Considering the number of vertices n, we can clearly solve the problem in $2^n \cdot n^{O(1)}$ time. We show that unless the Exponential Time Hypothesis fails, this cannot be improved to $2^{o(n)} \cdot n^{O(1)}$ time (Theorem 7.3). If the parameter is the number γ of edges between $G[S]$ and the remaining vertices, then the problem can be solved in $O(4^{\gamma} n^2)$ time (Theorem 7.5). Finally, if we consider the number α of edges to delete to obtain a highly connected subgraph (plus singleton vertices), we obtain an $2^{O(\sqrt{\alpha}\log\alpha)} + O(\alpha^2 nm)$-time algorithm (Theorem 7.8). Thus, contrary to the parameter n, a subexponential running time can be achieved.

Known results and related work. The algorithm by Hartuv and Shamir [HS00] partitions a graph heuristically into highly connected components; another algo-

Param.	Result	Reference
$n-k$	$n^{n-k}\,\mathrm{poly}(n)$	trivial
	W[2]-hard	Theorem 7.1
k, d	$2^d \cdot n^{d+O(1)}$ (note $k \le 2d+2$)	Proposition 7.1
	W[1]-hard	Theorem 7.2
Δ	$O((e(\Delta-1))^{2\Delta} \cdot (\Delta k)^2 \cdot n)$	Proposition 7.2
h	$O((6 \cdot (h-1))^{2h} \cdot (hk)^2 \cdot n)$ (randomized)	Theorem 7.4
n	$2^n \cdot \mathrm{poly}(n)$	trivial
	No $2^{o(n)}$-time algorithm	Theorem 7.3
γ	$O(4^\gamma n^2)$	Theorem 7.5
	$3\gamma^3$-vertex Turing kernel	Theorem 7.6
	No poly. kernel	trivial
α	$2^{O(\sqrt{\alpha}\log\alpha)} + O(\alpha^2 nm)$	Theorem 7.8
	4α-vertex kernel	Theorem 7.7

Table 7.1.: Summary of our complexity results for HIGHLY CONNECTED SUBGRAPH. The running time lower bound for n is based on the Exponential Time Hypothesis, and the lower bound on the problem kernel size for γ is based on the assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP}/\mathsf{poly}$.

rithm tries to explicitly minimize the number of edges outside of these components [Hüf+14]. Highly connected graphs can be seen as *clique relaxation* [Kos05; BP13; PYB13; Kom16], that is, a graph class that has properties similar to cliques, without being as restrictive. The definition of highly connected graphs is similar to *0.5-quasi-complete graphs* [MIH99], that is, graphs where every vertex has degree at least $(n-1)/2$. These graphs are also referred to as *(degree-based) 0.5-quasi-cliques* [LW08]. The difference to highly connected graphs seems minor, however, 0.5-quasi-complete graphs may have separators of size one whereas highly-connected graphs may not. Recently, also the problem of finding subgraphs with high *vertex* connectivity has been examined from a classical complexity point of view [Ver+14].

To the best of our knowledge, Ito, Iwama, and Osumi [IIO05] were the first to consider a formal notion of isolation in the context of dense subgraph identification. Our notion of isolation differs from the previous ones, as we count the total size of the cut $(S, V \setminus S)$, whereas previous definitions count the size of $(S, V \setminus S)$ divided by the size of S [IIO05; II09] or the minimum of the number of outgoing edges per vertex [Kom+09]. Clearly, our isolation parameter is larger than both of the other two. It easily follows from our Theorem 7.3 that HIGHLY CONNECTED SUBGRAPH is NP-complete if we require at least one vertex in the solution which has zero outgoing edges. The parameterization by average number of outgoing edges per vertex is left as an open question. Blisnetz and Karpov [BK] recently studied the partitioning into highly connected components and finding highly connected subgraphs. Providing a tighter analysis of our previously published algorithm [HKS15], they independently observed that HIGHLY CONNECTED SUBGRAPH can be solved in $2^{O(\sqrt{\alpha}\log\alpha)} \cdot \mathrm{poly}(n)$ where α is the number of edges not in the highly connected subgraph. We note that, here, we additionally provide a general analysis of the approach used in the algorithm, which leads us to believe that it cannot yield substantially better running times.

Outline of this chapter. Before giving our results, we describe a special notation for cuts in Section 7.2 that we use throughout this chapter. In Section 7.3 we give our hardness results and in Section 7.4 we describe our algorithms. We conclude in Section 7.5.

7.2. Specific preliminaries

In this chapter, it will be useful to speak of a cut, a set of edges whose removal disconnects a graph, in terms of a vertex bipartition. Hence, we use the following definition which is slightly different from the one in Section 1.1.1. A *cut* in a graph $G = (V, E)$ is a vertex bipartition (A, B), that is, $A \cap B = \emptyset$ and $A \cup B = V$. The *cut edges* are the edges with one end in A and the other in B. The *size* of a cut is the number of its cut edges.

7.3. Hardness results

We now give our hardness results with respect to the vertex deletion parameter ℓ (Section 7.3.1), the solution size and degeneracy combined (Section 7.3.2), and the number of vertices (Section 7.3.3).

7.3.1. Parameter vertex deletion number

For finding large cliques in a graph, one successful approach is to use fixed-parameter algorithms for the parameter "number of vertices in the graph that are not in the clique" [IIO05; Kom+09]. We show that such fixed-parameter algorithms are unlikely for HIGHLY CONNECTED SUBGRAPH.

Theorem 7.1. HIGHLY CONNECTED SUBGRAPH is NP-complete and W[2]-hard with respect to $\ell := n - k$.

Proof. We present a reduction from HITTING SET which is NP-complete [GJ79] and W[2]-hard with respect to k [DF99]:

> HITTING SET
> *Input:* A hypergraph $\mathcal{H}$ and a nonnegative integer k.
> *Question:* Is there a set $H \subseteq V(\mathcal{H})$ of size at most k such that $S \cap H \neq \emptyset$ for each hyperedge S in $\mathcal{H}$?

Given an instance $(\mathcal{H}, k)$ of HITTING SET, we construct an instance $(G = (V, E), k')$ of HIGHLY CONNECTED SUBGRAPH as follows. Denote $U = V(\mathcal{H})$, $n = |U|$, and $\mathcal{E}(\mathcal{H}) = \{F_1, \ldots, F_m\}$, and assume without loss of generality that $U = \{1, \ldots, n\}$ and $k < n - 1$). Initially, set $V := U \cup V_F$ where $V_F := \{f_i \mid 1 \leq i \leq m\}$, that is, create one vertex for each element and each set of the HITTING SET instance. Next, make each vertex f_i adjacent to all vertices $u \in U \setminus F_i$, that is, to the vertices corresponding to elements *not* in F_i. This will encode the HITTING SET instance. Now, add three cliques V_X, V_Y, and V_Z to G, where V_X has size $k+1$, V_Y has size n, and V_Z has size m. These three cliques will enforce that at least k vertices are deleted, and that some of the deleted vertices are contained in U. The purpose of the edges between U and V_F is to make sure that for each f_i at least one deleted vertex from U is not a neighbor of f_i (and thus it is contained in F_i). To achieve these properties, add the following edges.

First, add all edges between the vertices in U, V_Y, and V_Z, that is, make $U \cup V_Y \cup V_Z$ a clique. Furthermore, add all possible edges between V_X and V_F and make each vertex in V_X adjacent to exactly $n-k+1$ vertices of V_Y. When adding these edges, ensure that every vertex in V_Y has at least one neighbor in V_X. Furthermore, add all edges between V_F and V_Z and make each vertex $f_i \in V_S$ adjacent to $|F_i|-1$ vertices in V_Y.

To complete the construction, set $k' := |V|-k$. Note that this implies $\ell = k$ (recall that ℓ is the number of vertices that are not in the sought highly connected subgraph). Before we show the correctness of the reduction, observe the following about the degrees of the vertices in G:

- Each vertex in $U \cup V_Y \cup V_Z$ has degree at least $m+2n-1$.
- Each vertex in V_X has degree exactly $m+n+1$ in G.
- Each vertex in V_F has degree exactly $m+n+k$ in G.

It remains to show that

> $(\mathcal{H}, k)$ is a yes-instance of HITTING SET $\Leftrightarrow$ (G, k') is a yes-instance of HIGHLY CONNECTED SUBGRAPH.

($\Rightarrow$): Let $H \subseteq U$ be a size-k hitting set. We show that $G-H$ is highly connected. Note that $|V|-k = 2(m+n)+1$ and thus $G-H$ is highly connected if all its vertices have degree at least $m+n+1$. Since $k<n$, all vertices in $U \cup V_Y \cup V_Z$ have degree at least $m+n+1$. Furthermore, each vertex in V_X has degree at least $m+n+1$ in $G-H$ since there are no edges between U and V_X. Finally, every vertex $f_i \in V_S$ has degree at least $m+n+1$: Since H is a hitting set, there is one vertex in H that is not adjacent to f_i. Thus, the degree of f_i is at least $m+n+k-(k-1) = m+n+1$.

($\Leftarrow$): Let H be a vertex set of size k such that $G-H$ is highly connected. First, note that $|V_X| = k+1$ which implies that there is at least one vertex $v \in V_X$ that is not deleted. Since $G-H$ is highly connected this implies that v has more than $\lfloor(|V|-|H|)/2\rfloor = m+n+1$ neighbors in $G-H$. Since $|H| = k$, we have $N(v) \cap H = \emptyset$, implying that $V_X \cap H = \emptyset$. This means that all vertices in H are nonadjacent to *all* vertices in V_X, and thus $H \subseteq U \cup V_Z$. We show that $H \cap U$ is a hitting set for $\mathcal{H}$. Assume that this is not the case; then there is one vertex in $f_i \in V_F$ such that all deleted vertices are adjacent to f_i since f_i is adjacent to all vertices in $(U \cup V_Z) \setminus F_i$. This vertex has degree $m+n$ in $G-H$. This contradicts the fact that $G-H$ is highly connected.

Since the above reduction is a parameterized polynomial-time reduction, this shows that HIGHLY CONNECTED SUBGRAPH is NP-complete and W[2]-hard with respect to ℓ. □

Algorithm 1: Improved XP-algorithm for degeneracy.

Input: A graph G of degeneracy d and a nonnegative integer k.
Output: A k-vertex highly connected graph $G[S]$ if there is any.

```
1 while $V(G) \neq \emptyset$ do
2     $v \leftarrow$ a vertex of minimum degree in $G$
3     foreach $S_1 \subseteq N(v)$ do
4         foreach $S_2 \subseteq V(G) \setminus N[v]$ such that $|S_2| \leq d$ and $|S_2| = k - |S_1|$ do
5             $H \leftarrow G[\{v\} \cup S_1 \cup S_2]$
6             if $H$ is highly connected then return $H$
7     $G \leftarrow G - v$
```

7.3.2. Parameters solution size and degeneracy

A graph has degeneracy d if every subgraph contains at least one vertex that has degree at most d. In many graphs from real-world applications, the degeneracy of a graph is very small compared to the input graph size [ELS13]. For yes-instances of HIGHLY CONNECTED SUBGRAPH, the degeneracy d of the input graph has to satisfy $d \geq \lfloor k/2 \rfloor + 1$ (recall that k is the number of vertices in the desired highly connected graph). Therefore, HIGHLY CONNECTED SUBGRAPH is polynomial-time solvable if the input graph has constant degeneracy: trying all subgraphs with $k \leq 2d-1$ vertices decides the problem in $n^{2d} \cdot n^{O(1)}$ time. This can be improved to the following running time.

Proposition 7.1. HIGHLY CONNECTED SUBGRAPH can be solved in $2^d \cdot n^{d+O(1)}$ time where d is the degeneracy of G.

Proof. The algorithm is given in Algorithm 1. It iteratively picks a vertex v of minimum degree (that is, of degree at most d), checks all possibilities for a highly connected graphs $G[S]$ containing v and then removes v from the graph.

To see that the algorithm is correct, it suffices to show that in Line 4 all possibilities for the vertices $S_2 := S \setminus N[v]$ of the desired highly connected graph $G[S]$, $S \supseteq S_1$, are checked: This holds true because $S \setminus N[v]$ contains at most $k/2 \leq d$ vertices, as otherwise v could not have degree at least $\lfloor k/2 \rfloor + 1$ in $G[S]$. The running time is $2^d \cdot n^{d+O(1)}$. □

Unfortunately, if we regard the degeneracy as a parameter instead of a constant, we obtain hardness, even if additionally the solution size k is a parameter.

Theorem 7.2. HIGHLY CONNECTED SUBGRAPH parameterized by the combined parameter (d,k), where d is the degeneracy of G, is W[1]-hard.

Proof. We reduce from the classic CLIQUE problem, which is W[1]-hard with respect to the solution size k [DF99].

> CLIQUE
> *Input:* An undirected graph G and a nonnegative integer k.
> *Question:* Does G have a k-vertex complete subgraph (a clique) as a subgraph?

Given an instance (G,k) of CLIQUE, produce an instance (G',k') of HIGHLY CONNECTED SUBGRAPH as follows. First, subdivide each edge $e = \{u,w\}$ in G, that is, remove e, insert a new *edge vertex* v^e and make v^e adjacent to both u and w. Call the set of edge vertices V_E. Next, add a clique with vertex set $X = \{x_1,\dots,x_\ell\}$ where $\ell = \binom{k}{2} + 3k$. Then, choose $\binom{k}{2} + 2k - 1$ arbitrary vertices in X, denote this set by X_1 and make each v^e adjacent to each vertex in X_1. Furthermore, let $X_2 := X \setminus X_1$ and make each $v \in V$ adjacent to the $k+1$ vertices in X_2 and to $\binom{k}{2} + 1$ further arbitrary vertices of X_1, finishing the construction of G'. Note that each $v \in V$ is adjacent to $\binom{k}{2} + k + 2$ vertices in X. The construction of the instance is completed by setting the size of the desired highly connected subgraph to $k' := 2 \cdot \binom{k}{2} + 4k$.

Note that the graph G' is $\binom{k}{2} + 3k$-degenerate, which can be seen by the following order of vertex removals: First, remove the vertices from V_E; these have degree $\binom{k}{2} + 2k + 1$ in the graph. Then, remove the vertices from V, these have degree $\binom{k}{2} + k + 2$ in $G' - V_E$. Finally, the remaining set X has size $\binom{k}{2} + 3k$, thus each vertex in $G - (V_E \cup V)$ has degree $\binom{k}{2} + 3k - 1$. The overall degeneracy of G' follows.

We now show the equivalence of the constructed instances, that is,

> (G,k) is a yes-instance of CLIQUE $\Leftrightarrow$ (G',k') is a yes-instance of HIGHLY CONNECTED SUBGRAPH.

($\Rightarrow$): Let $K \subseteq V$ be a clique of order k in G. Then, the set $K' := K \cup \{v^e \mid e \subseteq K\} \cup X$ induces a highly connected subgraph of order k' in G', which can be seen as follows. First, K' has size $k + \binom{k}{2} + \binom{k}{2} + 3k = k'$. Second, each vertex in $G'[K']$ has degree at least $k'/2+1 = \binom{k}{2}+2k+1$: The vertices in X have at least $|X|-1 = \binom{k}{2}+3k-1 > k'/2+1$ neighbors in $G'[K']$ since X is a clique. The vertices in K have exactly $\binom{k}{2} + k + 2$ neighbors in X plus $k-1$ neighbors in $K' \setminus (K \cup X)$ since they are adjacent to the $k-1$ vertices corresponding to the edges incident with them in $G[K]$. Finally, each remaining vertex corresponds to an edge in $G[K]$ and thus it has two neighbors in K. Since it has $\binom{k}{2}+2k-1$ neighbors in X it has exactly $k'/2+1$ neighbors in $G'[K']$.

($\Leftarrow$): Let K' be a vertex set of size k' such that $G'[K']$ is highly connected and let $y_1 = |V \cap K'|$ and $y_2 = |V_E \cap K'|$. Note that by the size of X, K' contains at least $\binom{k}{2}+k$ vertices from $V' \setminus X = V \cup V_E$, that is, $y_1 + y_2 \geq \binom{k}{2} + k$. Furthermore, note that if it contains a vertex v^e from V_E, then it contains all neighbors of v^e in G', since v^e has degree exactly $k'/2+1$ in G'. Note that by the above $K^* = K' \cap (V_E \cup V)$ directly corresponds to a subgraph of G: each vertex in V_E corresponds to an edge in G and both endpoints of this edge are in K'. Furthermore, since each vertex in $K' \cap V$ has at least $k-1$ neighbors in $K' \setminus X$, the corresponding graph has degree at least $k-1$. Finally, since K^* corresponds to a graph, we have $y_2 \leq \binom{y_1}{2}$. Since $y_1 + y_2 \geq \binom{k}{2}+k$ this implies $y_1 \geq k$. In the remainder of the proof we show $y_1 = k$ which directly implies that $K' \cap V$ is a clique in G since then K^* corresponds to a graph with k vertices and minimum degree $k-1$.

Assume towards a contradiction $y_1 \geq k+1$. We prove that we may also assume that $X \subseteq K'$. If this is not the case, then any vertex from $X \setminus K'$ belongs to X_2: we have that $K' \cap V_E \neq \emptyset$, because otherwise each $v \in V$ would have at most $\binom{k}{2}+3k$ neighbors in K', and all vertices of X_1 are in K' because all neighbors of $K' \cap V_E$ are in K'. Hence, there is a vertex $x \in X_2 \setminus K'$. By construction, this vertex is a neighbor of all vertices in $K' \cap X$ and of all vertices in $K' \cap V$. We can thus pick an arbitrary vertex $v^e \in K' \cap V_E$, remove v^e from K' and add x to K'. In the graph that is induced by the modified K', the degree of every vertex except x has increased or remains the same: the removed vertex v^e has only neighbors in X and in V and x is in G' adjacent to all vertices in X and to all vertices in V. Furthermore, x has also degree at least $k'/2+1$ since it is adjacent to the $\binom{k}{2}+2k-1$ vertices in X_1 and to the at least $y_1 \geq k+1$ vertices in $V \cap K'$.

Note that this replacement can be performed without decreasing the minimum degree in $G'[K']$ below $k'/2+1$ until $X \subseteq K'$. Hence, if $y_1 \geq k+1$, then we can also assume that $X \subseteq K'$. But then $y_2 < \binom{k}{2}$. This implies that there is at least one vertex $v \in K' \cap V$ that has less than $k-1$ neighbors in $K' \cap V_E$. Since v has only $\binom{k}{2}+k+2$ neighbors in X, it has less than $k'/2+1$ neighbors in $G'[K']$. This contradicts the assumption that $G'[K']$ is highly connected.

Hence, if $G'[K']$ is highly connected, then $y_1 = k$. By the discussion above, K^* corresponds to a subgraph of G with k vertices and $\binom{k}{2}$ edges, that is, a clique of order k. □

7.3.3. Parameter number of vertices

A trivial algorithm for HIGHLY CONNECTED SUBGRAPH is to enumerate all vertex subsets of size k and to check for each subset whether it induces a highly connected graph. This algorithm has running time $O(2^n \cdot m)$. We now show by a reduction from CLIQUE that a running time improvement to $2^{o(n)} \cdot \text{poly}(n)$ is unlikely. The idea of the reduction is to make the solution size k large, and to add to the CLIQUE instance some new graph that is so large that, in the resulting instance of HIGHLY CONNECTED SUBGRAPH, every highly connected graph of size k must contain this new graph. The remaining vertices must form a clique in order to have sufficiently high degree. The following combinatorial lemmas are used in our construction.

Lemma 7.1. For any $\ell \in \mathbb{N}$, the edges of the complete graph K_{2^ℓ} can be partitioned into $2^\ell - 1$ perfect matchings. Moreover, there is such a partition that includes two perfect matchings that together contain a spanning tree of K_{2^ℓ}, and such a partition can be computed in polynomial time in 2^ℓ.

Proof. The proof is by induction on ℓ. Clearly, the single edge of K_{2^1} can be partitioned into a perfect matching in polynomial time. Now assume that $\ell > 1$ and that the statement holds for all $\ell' < \ell$. Partition the edges of K_{2^ℓ} into three sets in the following manner. Two sets A, B of the partition are induced by two vertex-disjoint subgraphs K_A, K_B of K_{2^ℓ} each isomorphic to $K_{2^{\ell-1}}$. The third set C contains the edges with exactly one vertex from both of the subgraphs. By induction, we can compute edge-partitions into perfect matchings of K_A, K_B; call the partitions $\mathcal{P}_A, \mathcal{P}_B$. Now join $\mathcal{P}_A, \mathcal{P}_B$ into a partition $\mathcal{P}_{A\cup B}$ of $A \cup B$ by iteratively taking the union of an arbitrary part from $\mathcal{P}_A$ and an arbitrary part from $\mathcal{P}_B$ and deleting the parts from $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively. Note that the obtained partition $\mathcal{P}_{A\cup B}$ of $A \cup B$ contains $2^{\ell-1} - 1$ parts. Next, compute a partition $\mathcal{P}_C$ of C as follows. Denote

$$\begin{aligned} V(K_A) &=: \{v_0, \ldots, v_{2^{\ell-1}-1}\}, \text{ and} \\ V(K_B) &=: \{u_0, \ldots, u_{2^{\ell-1}-1}\}. \end{aligned}$$

Then,

$$\begin{aligned} E_j &:= \{\{v_i, u_{i+j}\} \mid i \in \{0, \ldots, 2^{\ell-1} - 1\}\}, \text{ and} \\ \mathcal{P}_C &:= \{E_j \mid j \in \{0, \ldots, 2^{\ell-1} - 1\}\}, \end{aligned}$$

where indices are taken modulo $2^{\ell-1}$. Clearly, $\mathcal{P}_C$ can be computed in polynomial time. Note that $\mathcal{P}_C$ is a partition of C into perfect matchings, and $|\mathcal{P}_C| = 2^{\ell-1}$. Finally, take the union $\mathscr{P}_{A\cup B} \cup \mathscr{P}_C$. Note that this is a partition of $E(K_{2^\ell})$ into perfect matchings and $|\mathscr{P}_{A\cup B} \cup \mathscr{P}_C| = 2^\ell - 1$.

For the running time, let $2^{c\cdot(\ell-1)}$ be an upper bound on the time needed to compute the edge partitions $\mathscr{P}_A$ and $\mathscr{P}_B$, and let $2^{c'\cdot(\ell-1)}$ be an upper bound on the time needed to compute $\mathscr{P}_C \cup \mathscr{P}_{A\cup B}$ from $\mathscr{P}_A$ and $\mathscr{P}_B$ where $c, c' \geq 2$ are universal constants. Then an overall time upper bound for the above procedure is

$$2^{c\cdot(\ell-1)+1} + 2^{c'\cdot(\ell-1)} \;=\; 2^{c\cdot\ell-c+1} + 2^{c'\cdot\ell-c'} \;\leq\; 2^{\max\{c,c'\}\cdot\ell+2-\min\{c,c'\}} \;\leq\; 2^{\max\{c,c'\}\cdot\ell},$$

which is polynomial in 2^ℓ.

Let us now slightly modify the step of computing E_j in order to obtain two matchings that span K_{2^ℓ}. Fix an arbitrary matching $M \in \mathscr{P}_{A\cup B}$. Rename the vertices in P_A and P_B in such a way that

$$\begin{aligned} M \;=\; &\{\{v_0, v_1\}, \{v_2, v_3\}, \ldots, \{v_{2^{2\ell-1}-2}, v_{2^{2\ell-1}-1}\}, \\ &\{u_1, u_2\}, \{u_3, v_4\}, \ldots, \{u_{2^{2\ell-1}-1}, u_0\}\}. \end{aligned}$$

Then $M \cup E_0$ induces a connected graph containing all vertices of K_{2^ℓ}. The renaming of the vertices can be performed in linear time, thus the running time increases only by a constant factor. □

Lemma 7.2. For any integer $\ell \in \mathbb{N}$, $\ell \geq 3$, there are two edge-disjoint Hamiltonian cycles in $K_{2\ell}$, computable in polynomial time.

Proof. Using a result of Bondy and Chvátal [BC76], we describe a polynomial-time algorithm that computes the two cycles. First, fix an arbitrary permutation of the vertex set which directly defines a Hamiltonian cycle. Remove this cycle from the graph. The resulting degree of each vertex is $2\ell - 1 - 2$. We now use the following result of Bondy and Chvátal [BC76]: If the closure of a graph is a complete graph, then this graph has a Hamiltonian cycle which can be computed in polynomial time. Herein, the *closure* of a graph of order n is defined as the graph obtained as follows. Add to G all edges $\{u, v\}$ for which we have $\deg(u) + \deg(v) \geq n$, do the same for the resulting graph, and iterate as long as at least one edge is added. Since the sum of two vertex degrees in our graph is at least $2(2\ell - 3) \geq 2\ell$, the closure of the resulting graph is $K_{2\ell}$ again. Thus, in polynomial time we can compute another Hamiltonian cycle in the remaining graph which is edge-disjoint from the first cycle. □

With the above two lemmas at hand, we can show that we can build a graph whose construction will actually be the main part of our reduction. The following lemma shows that we can efficiently construct the graph which we need to add to the CLIQUE instance.

Lemma 7.3. Let $a, b \in \mathbb{N}$ be two nonnegative integers such that a is even, $b-3 \geq 8$, $b-3$ is a power of two, and $a-2 \geq 2b$. There is a graph $G = (X \cup W, E)$ on the disjoint vertex sets X and W, such that

(i) $G[X]$ is connected,
(ii) $|X| = a-2$, $|W| = a-b+1$,
(iii) $N_G(X) \setminus X = W$,
(iv) each vertex in X has degree a, and each vertex in W has degree $a-b$.

Moreover, G can be constructed in time polynomial in a.

Proof. Begin with $G = (X \cup W, E)$ where $X \cup W$ is an independent set and where X and W have the prescribed sizes. Next, let $X_1, X_2 \subseteq X$ be arbitrary such that $|X_1| = |X_2| = a-b+1$ and $|X_1 \cap X_2|$ is of minimum size. That is, $|X_1 \cap X_2| = a-2b+4$, and $|X_1 \setminus X_2| = |X_2 \setminus X_1| = b-3$ (note that $a-2 \geq 2b$, hence, $|X_1 \cap X_2| \geq 6$). We add a set of matchings to G that saturate W in order to bring up the degree of the vertices in W to $a-b$. We first add a perfect matching to $G[W]$ (note that $a-b+1$ is even, because a is even and $b-3$ is a power of two). Then we add $a-b-1$ matchings that saturate W to $G[W \cup X]$; these matchings are divided evenly into matchings saturating X_1 and matchings saturating X_2. More formally, denote $W =: \{w_0, \ldots, w_{a-b}\}$, and $X =: \{x_0, \ldots, x_{a-3}\}$ such that $X_1 = \{x_0, \ldots, x_{a-b}\}$, and $X_2 = \{x_{b-3}, \ldots, x_{a-3}\}$. Let $i \in \{0, \ldots, a-b-2\}$. We define

$$M_i := \begin{cases} \{\{w_j, x_{(j+i) \bmod (a-b)}\} \mid j \in \{0, \ldots, a-b\}\}, & \text{if } i \text{ is even, and} \\ \{\{w_j, x_{b-3+((j+i+1) \bmod (a-b))}\} \mid j \in \{0, \ldots, a-b\}\}, & \text{otherwise.} \end{cases}$$

That is, if i is even, then M_i is a perfect matching in $G[W \cup X_1]$ and, otherwise M_i is a perfect matching in $G[W \cup X_2]$. Note that there is an even number of matchings M_i. Furthermore, $M_i, M_{i'}$, $i \neq i'$, are disjoint: this is clear if both i and i' are even, or both are odd. If i is odd and i' is even, then w_j is matched to some v_k with $k \equiv j \bmod 2$ in M_i whereas $M_{i'}$ matches w_j to some v_k with $k \not\equiv j \bmod 2$, because both $b-3$ and $i+1$ are even, and $a-b$ is odd. Hence, we may add all M_i, $i \in \{0, \ldots, a-b-2\}$, to G. Now each vertex in W has degree $a-b$, as required, and it remains to mend the degrees of vertices in X. Note that each vertex in $X_1 \cap X_2$ has degree $a-b-1$, and each vertex in $X \setminus (X_1 \cap X_2)$ has degree $(a-b-1)/2$. We divide $X_1 \cap X_2$ into two equal-sized parts A, B; note that this is possible, because $|X_1 \cap X_2| = a-2b+4$ is even and $a-2b+4 \geq 6$ since $a-2 \geq 2b$. We make each vertex in A adjacent to each vertex in $X_1 \setminus X_2$, and each vertex in B adjacent to each vertex in $X_2 \setminus X_1$. Now each vertex in $X_1 \cap X_2$ has degree $a-b-1+b-3 = a-4$. Using Lemma 7.2 we add four perfect matchings to $G[X_1 \cap X_2]$ (recall that $|X_1 \cap X_2| \geq 6$).

It remains to lift the degree of the vertices in $X \setminus (X_1 \cap X_2)$ which currently have degree $(a-b+1)/2+(a-2b+4)/2 = a-3b/2+5/2$. Note that $2(b-3)$ is a power of two. Now we use Lemma 7.1 to add $3b/2-5/2 \leq 2(b-3)-1$ perfect matchings to $G[X \setminus (X_1 \cap X_2)]$, including two perfect matchings that make $G[X \setminus (X_1 \cap X_2)]$ connected. Note that, indeed, $3b/2-5/2 \leq 2(b-3)-1$, because $b-3 \geq 8$. Hence, the degree of each vertex in $X \setminus (X_1 \cap X_2)$ is now a. Note also that $G[X]$ is connected. □

We say that the graph G described in the above Lemma 7.3 is an (a,b)-*equalizer* and the vertices in W are its *ports*.

Lemma 7.4. There is a many-one reduction from CLIQUE to HIGHLY CONNECTED SUBGRAPH that runs in polynomial time. Furthermore, the number of vertices in the HIGHLY CONNECTED SUBGRAPH instance is linear in the number of vertices in the CLIQUE instance.

Proof. Let (G,p) represent an instance of CLIQUE and denote $|V(G)| = n$. Without loss of generality, assume that $p-3 \geq 8$ and $p-3$ is a power of two. Otherwise, repeatedly add a universal vertex and increase p by one until $p-3 \geq 8$ and $p-3$ is a power of two. Note that this at most doubles p. Furthermore, assume that $n-1 \geq p$; otherwise, solve the instance in polynomial time.

We construct the instance (G',k) of HIGHLY CONNECTED SUBGRAPH where $k = 4n-1$. Note that the minimum degree in a highly connected graph with k vertices is $2n$. Graph G' is constructed as follows. First, copy G into G'. Then add a vertex-disjoint $(2n,p)$-equalizer. By Lemma 7.3, a $(2n,p)$-equalizer exists and is computable in polynomial time, because, by choice of (G,p), $2n$ is even, $p-3 \geq 8$ is a power of two, and $2n-2 \geq 2p$. Denote the ports of the equalizer by W and its remaining vertices by X. Add an edge between each port and each vertex in $V(G)$; this finishes the construction. The graph G' has less than $5n$ vertices since the $(2n,p)$-equalizer has less than $4n$ vertices. It remains to show the equivalence of the instances, that is,

$$(G,p) \text{ is a yes-instance} \Leftrightarrow (G', k = 4n-1) \text{ is a yes-instance.}$$

(⇒): Let $G[S]$ be a clique of order p in G. Then, $G'[S \cup X \cup W]$ is highly connected: Each vertex in S is adjacent to $p-1$ vertices in S and to $2n-p+1$ vertices in W. Hence, each vertex in S has $2n$ neighbors in $S \cup X \cup W$, as required. Each port has $2n-p$ neighbors in $X \cup W$ and p neighbors in S. Finally, each vertex in X has $2n$ neighbors in $X \cup W$.

(⇐): Let $G'[S]$ be a highly connected graph of order k in G'. There are at most n vertices in $V(G) \cap S$, thus there is at least one vertex in $S \cap X$. Since $G'[X]$ is connected and each vertex in X has degree exactly $2n$ (the minimum degree in $G'[S]$), we have $X \subseteq S$. Furthermore, since $\{v \mid X \cap N(v) \neq \emptyset\} \setminus X = W$, also $W \subseteq S$, leaving $4n-1-|X|-|W| = p$ vertices in $S \cap V(G)$. Since $N_{G'}(V(G)) \setminus V(G) = W$ and $|W| = 2n-p+1$, each vertex in $S \cap V(G)$ has at least $p-1$ neighbors in $S \cap V(G)$. Thus $G[S \cap V(G)]$ is a clique. □

Lemma 7.4 directly connects the running time of algorithms for HIGHLY CONNECTED SUBGRAPH with respect to the number n of vertices with the Exponential Time Hypothesis (see Section 1.1.3). That is, any $2^{o(n)} \cdot n^{O(1)}$-time algorithm would, via Lemma 7.4, imply also such an algorithm for CLIQUE, which would contradict the Exponential Time Hypothesis. Hence, we have the following.

Theorem 7.3. HIGHLY CONNECTED SUBGRAPH does not admit a $2^{o(n)} \cdot n^{O(1)}$-time algorithm unless the Exponential Time Hypothesis fails.

7.4. Algorithms

We now give our fixed-parameter algorithms and data reduction rules pertaining to the maximum degree, h-index (Section 7.4.1), number of edges outgoing from the solution (Section 7.4.2), and number of edges not in the solution (Section 7.4.3).

7.4.1. Parameters maximum degree and h-index

It is easy to see that HIGHLY CONNECTED SUBGRAPH is fixed-parameter tractable with respect to the maximum degree Δ in the input graph: Since highly connected graphs have diameter two [HS00, Theorem 1], the solution must be contained in the *closed two-neighborhood* $N_2[v]$ of some vertex v, that is, in the set of vertices which have distance at most two to v. Furthermore, the solution size k fulfills $k \leq 2\Delta$ by the definition of highly connectedness. Thus, simply enumerating all closed two-neighborhoods, enumerating all size $k \leq 2\Delta$ subsets of them, and testing whether one of these subsets induces a highly connected graph gives a $\Delta^{4\Delta} \cdot \text{poly}(n)$-time algorithm.

We obtain a slightly faster algorithm if we formulate HIGHLY CONNECTED SUBGRAPH as a FIXED-CARDINALITY OPTIMIZATION problem and employ Theorem 6.3 which harbors an algorithm for maximizing objective functions over k-vertex connected subgraphs. To do this, we define the objective function $\phi(S)$ to be the minimum degree in $G[S]$, which can be evaluated in $O(\Delta k)$-time for any $S \subseteq V(G)$ of size k. Hence, Theorem 6.3 implies the following, where we use the fact that $k \leq 2\Delta$.

Proposition 7.2. HIGHLY CONNECTED SUBGRAPH can be solved in $O((e(\Delta-1))^{2\Delta}\cdot\Delta k^2\cdot n)$-time, where Δ is the maximum degree in G.

In a similar fashion, we can apply Theorem 6.4 to HIGHLY CONNECTED SUBGRAPH parameterized by the h-index of the input graph G. Recall that the h-index of G is the largest integer h such that there are at least h vertices of degree h.

Theorem 7.4. HIGHLY CONNECTED SUBGRAPH admits a randomized algorithm that runs in $O((6\cdot(h-1))^{2h}\cdot(hk)^2\cdot n)$ time and reports a yes-instance as a no-instance with probability at most $1/e$. Herein, h is the h-index of the input graph.

Proof. Let G denote the input graph. Clearly, the h-index of G can be computed in polynomial time. Moreover, along with the h-index we can compute a partition of $V(G)$ into H and W such that $|H|\le h$ and each vertex in $G[W]$ has degree at most h. We guess the intersection S_H of the vertex set S of the desired highly connected graph with the set H by trying all possibilities. Then we aim to find the intersection S_W of S with W via Theorem 6.4. (Note that $G[S_W]$ might not be connected.) To do this, we label each vertex v in $G[W]$ by its number ℓ_v of neighbors in S_H. We define the objective function $\phi(T):=\min_{v\in T}\deg_{G[T]}(v)+\ell_v$. Clearly, if there is a highly connected graph $G[S]$ with $S\cap H=S_H$, then we may assume that $S\cap W=S_W$ where S_W maximizes $\phi(S_W)$. It is not hard to check that ϕ is component $+$-linear (see Definition 6.3). Thus, it follows from Theorem 6.4 that there is a $O((4.2\cdot(h-1))^{k'-1}\cdot(hk)^2\cdot n)$-time algorithm to find $S_W\subseteq W$ with $|S_W|=k'=k-|S_H|$ that maximizes $\phi(S_W)$. Finally, we check whether $G[S_H\cup S_W]$ is highly connected and output $S_H\cup S_W$ if that holds true.

Clearly, if there is a highly connected graph in G, then the above algorithm finds one. The running time is $O(2^h(4.2\cdot(h-1))^{2h}\cdot(hk)^2\cdot n)$ since $k'\le 2\Delta\le 2h$. Since $\sqrt{2}\cdot 4.2<6$, this implies the claimed running time bound. □

7.4.2. Parameter edge isolation number

In this section, we present a single-exponential fixed-parameter algorithm and data reduction rules for the number γ of edges between the desired highly connected subgraph $G[S]$ and the remaining graph. In this case, S is called "γ-isolated". More formally, let $G=(V,E)$ be a graph. We call a set $S\subseteq V$ *γ-isolated* in G if $(S,V\setminus S)$ is a cut of size at most γ. We omit the clause "in G" if the graph is clear from the context. This definition of isolation leads to the following problem formulation.

ISOLATED HIGHLY CONNECTED SUBGRAPH
Input: An undirected graph $G = (V, E)$ and nonnegative integers k and γ.
Question: Is there a k-vertex γ-isolated highly connected subgraph contained in G?

The notion of isolation is not only motivated from an algorithmic point of view but also from the application. Ideally, communities in a network have fewer connections to the rest of the network [PYB13]. Thus, putting an additional constraint on the number of outgoing edges may yield better communities than merely demanding high edge connectivity.

In the following, it will be useful to consider an augmented version of ISOLATED HIGHLY CONNECTED SUBGRAPH: we place integer labels on the vertices which imply that these vertices are harder to isolate. We thus additionally equip each instance of ISOLATED HIGHLY CONNECTED SUBGRAPH with a labeling $f\colon V \to \mathbb{N}$ and we call $V' \subseteq V$ *γ-isolated under f* if there are at most $\gamma - \sum_{v\in V'} f(v)$ edges between V' and $V \setminus V'$ in G. Without loss of generality, assume $k \geq 2$ in the following.

The algorithm first performs three data reduction rules. The first simple rule removes connected components that are too small.

Rule 7.1. Remove all connected components with less than k vertices from G.

The next rule finds connected components that are either trivial solutions or cannot contain any solution since proper subgraphs violate the isolation condition.

Rule 7.2. If there is a connected component $C = (V', E')$ of G that has minimum cut size at least $\gamma + 1$, then accept if C is highly connected, $|V'| = k$, and V' is γ-isolated under f. Otherwise (if C's minimum cut has size at least $\gamma + 1$ but C is not highly connected, $|V'| \neq k$, or V' is not γ-isolated under f), remove C from G.

Correctness proof for Rule 7.2. The rule is clearly correct if it accepts. If the rule removes C, then C has a minimum cut of size at least $\gamma + 1$. Thus, for every induced subgraph $C[S]$ of C that does not contain all of its vertices, set S is not γ-isolated. Hence, no subgraph of C is a solution and we can safely remove C. □

Rule 7.3. If G has a connected component C with a minimum cut (A, B) of size at most $k/2$, then do the following. For each $v \in A$ redefine $f(v) := f(v) + |N(v) \cap B|$ and for each $v \in B$ redefine $f(v) := f(v) + |N(v) \cap A|$. Then, delete all edges between A and B.

Correctness proof for Rule 7.3. Any k-vertex subgraph of C with nonempty intersection with both sides of (A,B) is not highly connected as it has a minimum cut of size at most $k/2$. Hence, any highly connected induced subgraph $C[S]$ of C is either contained in $C[A]$ or in $C[B]$. If S is γ-isolated under f in G, then it is also γ-isolated under the modified f in the modified graph (and vice-versa) by the way we have redefined f. □

Exhaustive application of these rules yields the following relation between the values γ and $k/2$.

Lemma 7.5. If Rules 7.1 to 7.3 are not applicable, then $\gamma > k/2$.

Proof. Assume the contrary. Each connected component has a minimum cut cutting at least one edge because Rule 7.1 is not applicable and $k \geq 2$. Further, each connected component has a cut of size at most γ because Rule 7.2 is not applicable. By assumption, $\gamma \leq k/2$ and hence, each connected component has a cut of size at most $k/2$ which contradicts the inapplicability of Rule 7.3. □

As shown by the following lemma, the reduction rules can be applied efficiently.

Lemma 7.6. Rules 7.1 to 7.3 can be exhaustively applied in $O((kn+\gamma)nm)$ time.

Proof. We first apply Rule 7.3 exhaustively. To this end, we determine for each connected component of G whether it contains a minimum cut of size at most $k/2$ by fixing an arbitrary vertex v and for each vertex u running $k/2+1$ rounds of the Ford-Fulkerson algorithm to find a flow from v to u, where every edge has unit capacity. If at some round the flow does not increase, then we find a corresponding cut by considering the strongly connected components in the residual graph and we apply Rule 7.3. We repeat the procedure if it was applicable.

Finding the connected components in the residual graph, and a round of the Ford-Fulkerson algorithm can both be implemented to run in $O(n+m)$ time (recall that we use unit capacities). Hence, if $C_1,\ldots,C_\ell$ are the connected components of G, then one iteration of Rule 7.3 takes $\sum_{i=1}^{\ell} O(k|C_i|(|C_i|+|E(G[C_i])|)) = O(knm)$ time. The overall number of iterations of applying Rule 7.3 is at most n since the number of connected components increases by one in each iteration, thus the overall running time for exhaustively applying Rule 7.3 is $O(kn^2m)$.

Then we apply Rule 7.2 by computing for each connected component C whether its minimum cuts have size at least γ. If this is true, then we check whether C is highly connected and compute the sum of the f-values of each vertex to decide whether the vertex set of C is γ-isolated. By the same arguments as above, the computation of the minimum cut size can be performed in $O(\gamma nm)$ time overall. The summation of the f-values can be performed in $O(n+m)$ time overall and for all connected components we can decide in $O(n+m)$ time whether they are highly connected. Hence, Rule 7.2 can be exhaustively applied in $O(\gamma nm)$ time.

Finally, we check whether Rule 7.1 is applicable which can be easily performed in $O(n+m)$ time. Altogether, we arrive at the claimed running time bound. □

Using the above, we can now present the promised fixed-parameter algorithm.

Theorem 7.5. There is an $O(4^{\gamma}n^2+(kn+\gamma)nm)$-time algorithm for ISOLATED HIGHLY CONNECTED SUBGRAPH.

Proof. We first reduce the instance with respect to Rules 7.1 to 7.3. By Lemma 7.6 this can be done in $O((kn+\gamma)nm)$ time. Next, we guess one vertex v that is in the solution S (by branching into n cases according to the n vertices). We start with $S' := \{v\}$ and try to extend S' to a solution. More precisely, we choose a vertex v' from the neighborhood of S' (that is, from $\bigcup_{u\in S'} N(u)\setminus S'$), and branch into two cases: add v' to S', or exclude v', that is, delete v' and increase $f(u)$ by one for all $u \in N(v')$. In the first case, we increase $|S'|$ by one. In the second case, we increase $\sum_{u\in S'} f(u)$ by at least one. Branching is performed until $|S'| = k$, or $\sum_{u\in S'} f(u)$ exceeds γ, or the neighborhood of S' is empty. When $|S'|$ reaches k, we check whether S' is highly connected and γ-isolated under f. If this is the case, we have found a solution. Otherwise, when $\sum_{u\in S'} f(u)$ exceeds γ or no branching is possible because the neighborhood of S' is empty, then we abort the branch; in this case, clearly no superset of S' can be a solution. The height of the search tree is upper bounded by $k+\gamma$, and each branch can be executed in $O(n)$ time, yielding a running time bound of $O(n\cdot 2^{k+\gamma}\cdot n)$.

We now distinguish two cases: $k \le \gamma$ and $k > \gamma$. In the first case $2^{k+\gamma} \le 4^{\gamma}$, as required. If $k > \gamma$, then there is at least one vertex in S that has no neighbors outside of S. Thus, instead of $S' = \{v\}$, we can start with $S' := \{v\} \cup N(v)$. Since v has more than $k/2$ neighbors in S, we have $|S'| > k/2+1$, and thus there are less than $k/2$ branches of adding a vertex. By Lemma 7.5, $2^{k/2+\gamma} \le 4^{\gamma}$. □

Data reduction analysis

We now present a way of analyzing the presented data reduction rules by giving a Turing kernelization for ISOLATED HIGHLY CONNECTED SUBGRAPH parameterized by γ. Informally, a Turing kernelization is a reduction of the input instance of a parameterized problem to many instances of the same problem which are small measured in the parameter. Then the solution to the original input instance can be computed by solving the small problem instances separately. For a formal definition see Section 1.1.3.

To motivate the Turing kernelization result we first observe that ISOLATED HIGHLY CONNECTED SUBGRAPH does not admit a polynomial problem kernel. We can see this via a trivial or-cross-composition: The disjoint union of a set of graphs has an isolated highly connected subgraph if and only if at least one of the graphs has one. Hence, ISOLATED HIGHLY CONNECTED SUBGRAPH has an or-cross-composition, implying the following (see Section 1.1.3).

Proposition 7.3. ISOLATED HIGHLY CONNECTED SUBGRAPH does not admit a polynomial-size problem kernel with respect to γ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

We now describe the Turing kernelization algorithm in detail. It consists of

1. applying Rules 7.2 and 7.3 exhaustively (wherefore we need the mapping $f(v)$ indicating that v is harder to isolate as above),
2. removing vertices of high degree,
3. carving out small subgraphs that contain a solution, if any, and
4. removing the labeling $f(v)$.

The final step is only used for the analysis (as we have to produce instances of ISOLATED HIGHLY CONNECTED SUBGRAPH again); in practice, since the labeling $f(v)$ is not hard to deal with, one would only carry out the first three steps.

Step 1. Given an instance of ISOLATED HIGHLY CONNECTED SUBGRAPH, we first reduce to the augmented version of ISOLATED HIGHLY CONNECTED SUBGRAPH in which we introduce the vertex labeling f. Then we apply Rules 7.2 and 7.3 exhaustively.

Step 2. Apply the following reduction rule which removes high-degree vertices.

Rule 7.4. Let (G, k, γ) be an instance of ISOLATED HIGHLY CONNECTED SUBGRAPH that is reduced with respect to Rules 7.2 and 7.3. If G contains a vertex v of degree at least $3\gamma - f(v)$, then remove v from G, and for each $u \in N(v)$ increase $f(u)$ by one.

Correctness proof for Rule 7.4. Since G is reduced with respect to Rules 7.2 and 7.3, we have $\gamma > k/2$ (due to Lemma 7.5). Thus, v has less than 2γ neighbors in any solution $G[S]$. This implies that v has more than $\gamma - f(u)$ neighbors in $V \setminus S$. Consequently, if $v \in S$, then S is not γ-isolated under f. Hence, v is not contained in any solution. □

Clearly, Rule 7.4 can be carried out in linear time.

Step 3. Given an instance (G, k, γ) of ISOLATED HIGHLY CONNECTED SUBGRAPH with mapping f, we now construct at most n instances of ISOLATED HIGHLY CONNECTED SUBGRAPH (with different mappings) that have $O(\gamma^3)$ vertices each. The original instance is a yes-instance if and only if one of these instances is a yes-instance. The idea is to exploit the fact that highly connected graphs have diameter two [HS00, Theorem 1]. Thus, to find highly connected graphs, it is sufficient to explore the two-neighborhood of each vertex. More precisely, the instances are constructed as follows.

For each vertex $v \in V$, construct the instance (G'_v, k, γ) with mapping f_v. Graph $G'_v := G[N_2[v]]$ where $N_2[v]$ is the set of all vertices that have distance at most two from v (including v). To retain the information about isolation for subgraphs of $G[N_2[v]]$, for each vertex $u \in N_2[v]$, we define $g(u) = |N(u) \setminus N[v]|$. Using this, for each vertex $u \in N_2[v]$ we define mapping $f_v(u) = f|_{N_2[v]}(u) + g(u)$ where $f|_{N_2[v]}(u)$ is f restricted to $N_2[v]$. It is not hard to prove the following.

Lemma 7.7. Let $S \subseteq V(G)$. Vertex set S induces a k-vertex highly connected graph in G and S is γ-isolated in G under f if and only if there is a vertex $v \in V(G)$ such that S induces a k-vertex highly connected graph in G_v and S is γ-isolated in G_v under f_v.

Step 4. Finally, we remove the mapping f_v from each instance (G'_v, k, γ) by adding degree-one neighbors, obtaining an instance (G_v, k, γ). Herein, to obtain graph G_v from G'_v, for every u of G_v add $f_v(u)$ new vertices and make them adjacent to u. In this way, we obtain at most n instances (G_v, k, γ) of ISOLATED HIGHLY CONNECTED SUBGRAPH. Since $k > 1$ without loss of generality, and because degree-one vertices can never be in a k-vertex highly connected graph, we have the following.

Lemma 7.8. Let $v \in V(G)$ and $S \subseteq V(G'_v)$. Vertex set S induces a k-vertex highly connected graph in G'_v and S is γ-isolated in G'_v under f if and only if S induces a k-vertex highly connected graph in G_v and S is γ-isolated in G_v.

Using the correctness of Rules 7.2 to 7.4 together with Lemmas 7.7 and 7.8, we obtain the following.

Corollary 7.1. Let q be an instance of ISOLATED HIGHLY CONNECTED SUBGRAPH. Instance q is yes if and only if one of the instances is yes that result from applying Steps 1 to 4 to q.

Thus, to obtain a Turing kernelization, it remains to show that we can query our size-bounded oracle for each of resulting instances (G_v, k, γ), that is, that each G_v has bounded size.

Lemma 7.9. Let (G_v, k, γ) be an instance resulting from applying Steps 1 to 4 to an instance of ISOLATED HIGHLY CONNECTED SUBGRAPH. Then G_v has less than $(3\gamma)^3$ vertices and less than $3\gamma^4$ edges.

Proof. Let f be the mapping and G be the graph after Step 2. Since G is reduced with respect to Rule 7.4. The maximum degree in G is at most $3\gamma - 1 - f(u)$ which implies $|N_2[v]| < 1 + 3\gamma - 1 + (3\gamma - 1)^2$. This is also the number of vertices in the graph G'_v of the instance (G'_v, k, γ) constructed in Step 3. For each vertex in $N_2[v]$ we then add further degree-one neighbors in Step 4, but the difference between the degree of u in G'_v and G_v is exactly $f(u)$. Thus, each vertex in G_v has degree at most $3\gamma-1$. Consequently, graph G_v has at most $(1+3\gamma-1+(3\gamma-1)^2)\cdot(3\gamma-1) < (3\gamma)^3$ vertices. Moreover, as each vertex in G_v has degree at most $3\gamma - 1$, graph G_v has also less than $(3\gamma)^4$ edges. □

Combining Corollary 7.1 and Lemma 7.9 leads to the following.

Theorem 7.6. ISOLATED HIGHLY CONNECTED SUBGRAPH admits a Turing kernelization of size $O(\gamma^4)$ which has less than $(3\gamma)^3$ vertices. The running time is $O((kn+\gamma)nm)$.

Proof. It only remains to prove the running time bound. Step 1 takes $O((kn+\gamma)nm)$ time which follows from Lemma 7.6. Clearly, Rule 7.4 can be carried out in $O(n+m)$ time, which is the only computation needed for Step 2. For each vertex v, the subgraph G'_v from Step 3 can be computed in linear time via a modified breadth-first search routine to determine $N_2[v]$, and then one pass over all edges of G to insert the required edges into G'_v. As every vertex $v \in G'_v$ has $f(v)$ neighbors in the input graph, also adding the degree-one neighbors to G_v in Step 4 can be carried out in linear time. Thus, Steps 2 to 4 need $O(n(n+m))$ time which is dominated by Step 1. □

7.4.3. Parameter edge deletion number

We now show that there is a subexponential-time fixed-parameter algorithm for HIGHLY CONNECTED SUBGRAPH with respect to α, the number of edges we are allowed to delete in order to obtain a highly connected graph of order k. Subsequently, we give a slight modification of the algorithm and a more precise analysis which in combination yield an improved running time. The algorithm is a search tree algorithm which branches on whether or not a given vertex is part of the highly connected graph. Repeated application of two reduction rules (similar to Rules 7.2 and 7.3 above) ensures that the branches are effective in reducing the remaining search space. To give a precise presentation of the branching step and the reduction rules, we define the problem with an additional *seed* S, a set of vertices which have to be in the highly connected graph.

SEEDED HIGHLY CONNECTED EDGE DELETION
Input: An undirected graph $G = (V, E)$, a vertex set $S \subseteq V$, and nonnegative integers k and α.
Question: Is there a set $E' \subseteq E$ of at most α edges such that $G - E'$ consists only of degree-zero vertices and a $(k + |S|)$-vertex highly connected subgraph containing S?

For $S = \emptyset$ we obtain the plain edge deletion problem. The reduction rules are as follows.

Rule 7.5. If there is a connected component $C = (V', E')$ of G that has minimum cut size at least $\alpha + 1$, then accept if C is highly connected, $S \subseteq V'$, $|V' \setminus S| = k$, and the remaining connected components of G contain at most α edges. Otherwise reject.

Correctness proof for Rule 7.5. The rule is clearly correct if it accepts. If it rejects, then the instance is a no-instance: If there is a highly connected graph in $G[V \setminus V']$, then the $\alpha + 1$ or more edges of C are not in this graph. Thus, any solution is contained in C. Hence, if the number of edges in the remaining components is more than α or if $S \setminus V' \neq \emptyset$, then the instance is a no-instance. Otherwise, either C is not highly connected or $|V' \setminus S| \neq k$. In both cases a highly connected graph of order $|S| + k$ that is contained in C has less than $|V'|$ vertices and thus it needs to be cut from the rest of C. This needs at least $\alpha + 1$ edges. Consequently, there is no solution and the instance is a no-instance. □

Rule 7.6. If there is a connected component of G that has a minimum cut of size at most $(k + |S|)/2$, then delete all cut edges and reduce α by their number.

Correctness proof for Rule 7.6. Every subgraph G' of G with non-empty intersection with both "sides" of the minimum cut has a cut of size $(k+|S|)/2$. Thus, if G' has order $(k+|S|)$, then it is not highly connected. Hence, for each deleted edge at least one of its endpoints is not in any solution. □

Similarly to the edge isolation parameter, after using the reduction rules k, $|S|$, and α are related as follows.

Lemma 7.10. If Rules 7.5 and 7.6 are not applicable, then $\alpha > (k+|S|)/2$.

Proof. Assume Rules 7.5 and 7.6 are not applicable but $\alpha \leq (k+|S|)/2$. Without loss of generality we may assume that there are no connected components consisting of singleton vertices. Otherwise, simply remove them. Hence, each connected component has a minimum cut cutting at least one edge. Further, there is a connected component with minimum cut of size at most α because Rule 7.5 is not applicable. By assumption $\alpha \leq (k+|S|)/2$ and, hence, there is a connected component with a minimum cut of size at most $(k+|S|)/2$ which contradicts the inapplicability of Rule 7.6. □

The running time of the rules can be upper bounded in a similar way as it was done in Section 7.4.2.

Lemma 7.11. Rules 7.5 and 7.6 are exhaustively applicable in $O(\alpha^2 nm)$ time.

Proof. We first decide for each connected component of G whether it contains a minimum cut of size at most $\alpha+1$ by fixing an arbitrary vertex v and for each vertex u running $\alpha+2$ rounds of the Ford-Fulkerson algorithm to find a flow from v to u with unit edge capacities. If at some round the flow does not increase, then we find a corresponding cut by considering the strongly connected components in the residual graph and apply the rules. We iterate the procedure if Rule 7.6 was applicable.

Both rules, finding the connected components in the residual graph, and a round of Ford-Fulkerson can each be implemented to run in $O(n+m)$ time due to the unit edge capacities. Hence, if $C_1,\ldots,C_\ell$ are the connected components of G, then one iteration takes

$$\sum_{i=1}^{\ell} O(\alpha|C_i|(|C_i|+|E(G[C_i])|)) = O(\alpha nm)$$

time. Since if Rule 7.5 applies then we are finished, and if Rule 7.6 applies then both the number of connected components increases and α decreases, the whole procedure takes $O(\min\{n,\alpha\}\alpha nm)$ time. □

Exhaustively applying the reduction rules lets us upper bound the number of the remaining vertices linearly in α. This will be useful in the search tree algorithm below.

Rule 7.7. If Rules 7.5 and 7.6 have been applied to the instance, and the resulting graph G contains at least $2\alpha + 4\alpha/k$ vertices or at least $\binom{2\alpha}{2} + \alpha$ edges, then reject.

Correctness of Rule 7.7. Assume that $(G = (V, E), k, \alpha)$ is a yes-instance of SEEDED HIGHLY CONNECTED EDGE DELETION to which Rules 7.5 and 7.6 have been applied exhaustively and let $E' \subseteq E$ be of minimum size such that $G - E'$ consists of degree-zero vertices and a highly connected subgraph $G[S']$ such that $S \subseteq S'$ and $|S'| = |S| + k$. We upper bound the number of vertices and edges in G. We first upper bound $|V \setminus S'|$. Because the instance is reduced with respect to Rule 7.6, the graph G has minimum vertex degree $k/2$. Hence, the number of edges incident with at least one vertex in $V \setminus S'$ is at least $|V \setminus S'| \cdot k/4$. This number is at most α and thus $|V \setminus S'| \cdot k/4 \leq \alpha$ which implies $|V \setminus S'| \leq 4\alpha/k$. Thus G contains at most $k+|S|+4\alpha/k$ vertices. By Lemma 7.10, $\alpha > (k+|S|)/2$. This implies $|S'| < 2\alpha$ and thus also that the number of edges within a solution $G[S']$ is less than $\binom{2\alpha}{2}$. Hence, every instance with at least $2\alpha + 4\alpha/k$ vertices or $\binom{2\alpha}{2} + \alpha$ edges is a no-instance, meaning that Rule 7.7 is correct. □

As a side-result, the above Rule 7.7 directly yields a problem kernel for SEEDED HIGHLY CONNECTED EDGE DELETION.

Theorem 7.7. SEEDED HIGHLY CONNECTED EDGE DELETION admits a problem kernel with at most $2\alpha+4\alpha/k$ vertices and $\binom{2\alpha}{2}+\alpha$ edges computable in $O(\alpha^2 nm)$ time.

The final ingredient in our subexponential-time search tree algorithm is the following simple branching rule. It simply takes a vertex and branches on whether or not it should be added to the seed S for the desired highly connected graph.

Branching rule 7.1. If $\alpha+k \geq 0$, then choose an arbitrary vertex $v \in V \setminus S$ and branch into the cases of adding v to S or removing v from G. That is, create the instances $I_1 = (G, S \cup \{v\}, k-1, \alpha)$ and $I_2 = (G-v, S, k, \alpha - \deg_G(v))$. Accept if I_1 or I_2 is accepted.

It is clear that Branching rule 7.1 is correct. We now describe the complete algorithm and bound its running time. Intuitively, in the beginning, the reduction rules ensure that the minimum degree remains large and, hence, Branching rule 7.1 remains effective. Later on, when k becomes small compared to α, it is more beneficial to simply try each k-vertex subset of the current graph. This is efficient, because the reduction rules ensure that that the vertex set is bounded in terms of α.

Algorithm 2: Subexponential fixed-parameter algorithm for the number α of deleted edges.

Input: A graph G, $S \subseteq V(G)$ and two nonnegative integers α and k, forming an instance of SEEDED HIGHLY CONNECTED EDGE DELETION.

Output: A $(|S|+k)$-vertex highly connected subgraph of G such that there are at most α edges of G not in this subgraph, if there is any such subgraph.

```
1 Apply Rules 7.5 and 7.6 exhaustively
2 Apply Rule 7.7 if possible
3 if $k \leq 2\sqrt{\alpha}$ then
4     foreach $V' \subseteq V(G) \setminus S$ such that $|V'| = k$ do
5         if $G[S \cup V']$ is highly connected then
6             Accept if $G$ contains at most $\alpha$ edges that are not contained in $S \cup V'$
7 else
8     Apply Branching rule 7.1 and recurse on the two created instances
```

Theorem 7.8. There is an $O(2^{4\cdot\alpha^{3/4}} + \alpha^2 nm)$-time algorithm for HIGHLY CONNECTED EDGE DELETION.

Proof. Algorithm description. The procedure is shown in Algorithm 2. Algorithm 2 first applies the kernelization from Theorem 7.7. Then, it searches for a solution by brute force if $k \leq 2\sqrt{\alpha}$. Otherwise, it applies Branching rule 7.1. From the correctness of the data reduction and branching rules it is clear that this algorithm finds a solution if there is one.

Running time. Note that in each recursive call of Algorithm 2, except the first one, the input instance has $O(\alpha)$ vertices according to Theorem 7.7. Thus applying Rules 7.5 and 7.6 in a call of Algorithm 2 amounts to $O(\alpha^5)$ time by Lemma 7.11, except in the first call, where we spend $O(\alpha^2 nm)$ time. Clearly, these running times dominate the one for Rule 7.7.

Next, in each recursive call we may have to check whether $S \cup V'$ is highly connected for all k-vertex subsets V'. This is done only after Rules 7.5 and 7.6 have been exhaustively applied and only if $k \leq 2\sqrt{\alpha}$. Thus, the graph G is of order at most $2\alpha + 4\alpha/k \leq 4\alpha$ (note that $k \geq 2$ without loss of generality). Hence, testing the subgraphs amounts to $O((4\alpha)^{2\sqrt{\alpha}+2})$ time. In total, the time spent per search tree node is $O((4\alpha)^{\max\{5, 2\sqrt{\alpha}+2\}})$.

Now let us upper bound the number of leaves C of the search tree. Note that the total number of search tree nodes is within a constant factor of C. For an instance $I = (G, S, k, \alpha)$ of SEEDED HIGHLY CONNECTED EDGE DELETION, define the value $\mu(I) := k + \alpha$ in the root of the search tree, after applying Rules 7.5 and 7.6. Then, $\mu(I) \le 3\alpha$ by Lemma 7.10.

Let $C(\mu(I))$ denote a nondecreasing upper bound on the number of leaves that a search tree with a root with value $\mu(I)$ can have. Whenever we apply Branching rule 7.1, μ is reduced by a certain amount. More precisely, $C(\mu(I))$ satisfies $C(\mu(I)) \le C(\mu(I_1)) + C(\mu(I_2))$ (herein, I_1 and I_2 are the instances resulting from an application of Branching rule 7.1 to instance I) and we define $C(0) = 1$. Further, since Rule 7.6 is not applicable, $\deg_G(v) \ge (|S| + k)/2 \ge k/2 \ge \sqrt{\alpha}$ in the application of Branching rule 7.1. This implies $C(\mu(I)) \le C(\mu(I) - 1) + C(\mu(I) - \sqrt{\alpha})$. Hence $C(\mu(I))$ is upper bounded by the number of paths in $\mathbb{R}^2$ from the origin to some point (x, y) that take only steps $(1,0)$ or $(0, \sqrt{\alpha})$, where $x + y = \mu(I)$. Scaling the y-axis by a factor of $1/\sqrt{\alpha}$, computing $C(\mu(I))$ reduces to the problem of counting such paths from the origin to some (x, y') taking only steps $(1,0)$ or $(0,1)$ such that $x + \sqrt{\alpha}y' = \mu(I)$. We now upper bound the number of these paths.

The number of $(0,1)$ steps is at most $3\sqrt{\alpha}$. If the path contains i $(0,1)$-steps, then the total number of steps in the path is $i + 3\alpha - \sqrt{\alpha}i$. Hence, there are $\binom{i+3\alpha-\sqrt{\alpha}i}{i}$ paths with exactly i steps $(0,1)$. This implies $C(\mu(I)) \le \sum_{i=0}^{3\sqrt{\alpha}} \binom{i+3\alpha-\sqrt{\alpha}i}{i}$. To bound this number we use the fact that $\binom{a+b}{a} \le 2^{2\sqrt{ab}}$ [Fom+14, Lemma 9]. Hence

$$C(\mu(I)) \le \sum_{i=0}^{3\sqrt{\alpha}} 2^{2\sqrt{i\cdot(3\alpha-\sqrt{\alpha}i)}}.$$

Consider the derivative $f(i)$ of $2\sqrt{i \cdot (3\alpha - \sqrt{\alpha}i)}$ with respect to i. We have

$$f(i) = \frac{\sqrt{\alpha}(3\sqrt{\alpha} - 2i)}{\sqrt{\sqrt{\alpha}i(3\sqrt{\alpha} - i)}}.$$

Inspecting $f(i)$ shows that $\sqrt{i \cdot (3\alpha - \sqrt{\alpha}i)}$ is maximized over $0 \le i \le 3\sqrt{\alpha}$ if $i = 3\sqrt{\alpha}/2$. This gives $C(\mu(I)) \le 3\sqrt{\alpha} \cdot 2^{3\sqrt{\alpha\sqrt{\alpha}}}$. Finally,

$$3\sqrt{\alpha} \cdot 2^{3\sqrt{\alpha\sqrt{\alpha}}} \cdot (4\alpha)^{\max\{5, 2\sqrt{\alpha}+2\}} \in O(2^{4\alpha^{3/4}}),$$

giving the overall running time bound of $O(2^{4\alpha^{3/4}} + \alpha^2 nm)$. □

Although the presented algorithm is a subexponential-time algorithm with relatively small constants in the exponential functions, it is unclear whether it can be useful in practice. This is because the parameter α is likely to be large in real-world instances. In theory, the algorithm seems, however, preferable to the simple $2^n \cdot \text{poly}(n)$-time branching algorithm which simply branches on a vertex into the cases to take it into the solution or remove it from the graph. The reason is that the instances that we encounter in practice are likely to be sparse, for example, for social networks. That is, the overall number of edges is upper bounded by $c \cdot n$ for some small constant c. This implies that $\alpha \le c \cdot n$ for these instances, meaning that the exponential term $2^{4\alpha^{3/4}}$ is smaller than 2^n, even for moderate values of α and n.

In practice, rather than using brute force in Algorithm 2, it would be prudent to use another more sophisticated solution strategy. It is then important to set the break-point in Line 3 in such a way that the running time of the solution strategy (or brute force) and the running time used up by Branching rule 7.1 are balanced. There is also a theoretical way to analyze this tradeoff, which we point out now.

Instead of switching from the branching routine to the brute-force routine on the problem kernel when $k \le 2\sqrt{\alpha}$, there is a more favorable break-point which better balances branching and brute force. This subsequently leads to the improved exponential term. Generally, we can compute a break-point to achieve a running time $2^{O(\alpha/\phi)} + O(\alpha^2 nm)$, for every function ϕ that satisfies certain conditions shown below. Herein, ϕ' denotes the derivative of ϕ with respect to α.

Theorem 7.9. There is an $2^{O(\alpha/\phi)} + O(\alpha^2 nm)$-time algorithm for HIGHLY CONNECTED EDGE DELETION for every positive, nondecreasing, unbounded, and two-times differentiable function $\phi = \phi(\alpha)$ that is computable within the same time bound and fulfills $\phi^2 \in o(\alpha/\log\alpha)$ and $\phi' \in \Omega(1/\sqrt{\alpha\log\alpha})$.

Putting $\phi(\alpha) = \sqrt{\alpha}/\log\alpha$ we obtain the following.

Corollary 7.2. There is an $2^{O(\sqrt{\alpha}\log\alpha)} + O(\alpha^2 nm)$-time algorithm for HIGHLY CONNECTED EDGE DELETION.

In the remainder of this section, we prove Theorem 7.9.

Proof of Theorem 7.9. Algorithm description. We use Algorithm 2 with the following modifications: The algorithm takes as an additional input an integer $\hat{c}$, the break-point mentioned above, which we specify below. In Line 3, the condition in the if clause is $k \le \hat{c}$ instead of $k \le 2\sqrt{\alpha}$. The remaining pseudo code is identical. Using the correctness of the reduction and branching rules, the algorithm is correct.

Running time. Let $c \in \mathbb{R}$ be arbitrary such that $c > 1$. We derive a sufficient condition on the relationship of c and $\hat{c}$ such that the running time of the algorithm is $O(2^{\log(c)\cdot\alpha} \cdot (3\alpha)^{\max\{5,\hat{c}+2\}} + \alpha^2 nm)$. Herein and in the following, log denotes the logarithm to base 2. We first bound the number of search tree leaves in terms of $\mu(I) = \log(c)\cdot(k+\alpha)$.

Let $L(\mu(I))$ be a nondecreasing upper bound on the number of leaves that a search tree with a root with value $\mu(I)$ can have. That is, $L(\mu(I))$ satisfies $L(\mu(I)) \le L(\mu(I_1)) + L(\mu(I_2))$ (herein, I_1 and I_2 are the instances resulting from an application of Branching rule 7.1 to instance I) and we define $L(0) = 1$. Further, since Rule 7.6 is not applicable, $\deg_G(v) \ge (|S|+k)/2 \ge k/2 \ge \hat{c}/2$ and this implies $L(\mu(I)) \le L(\mu(I) - \log c) + L(\mu(I) - \log(c)\cdot\hat{c}/2)$.

Let us derive values of $\hat{c}$ for which we have $L(\mu(I)) \le 2^{\mu(I)}$, that is

$$2^{\mu(I)-\log c} + 2^{\mu(I)-\log(c)\cdot\hat{c}/2} \le 2^{\mu(I)}.$$

Dividing by $2^{\mu(I)}$ gives

$$\begin{aligned} 2^{-\log c} + 2^{-\log(c)\cdot\hat{c}/2} &\le 1 \\ \frac{1}{c} + \frac{1}{c^{\hat{c}/2}} &\le 1 \\ \frac{1}{c^{\hat{c}/2}} &\le \frac{c-1}{c} \\ c^{\hat{c}/2} &\ge \frac{c}{c-1} \\ \hat{c}/2 &\ge \log_c \frac{c}{c-1}. \end{aligned}$$

Hence, choosing $\hat{c} = \lceil 2\log_c(c/(c-1)) \rceil$ gives $L(\mu(I)) \le 2^{\mu(I)}$ and applying $\alpha > k/2$ from Lemma 7.10 we have $L(\mu(I)) \le 2^{\log(c)\cdot 3\alpha}$. The overall number of nodes in the search tree is of the same order.

To upper bound the running time, it remains to find the running time of processing each search tree node. Note that in each search tree node, except the root, the input is a kernelized instance and hence contains at most $2\alpha + 4\alpha/k$ vertices. As, for our purposes, it suffices to find an upper bound of $O(\alpha^{\max\{5,\hat{c}+2\}})$ time for expanding a search tree node (except for the root), we may assume without loss of generality that $k \ge 5$ and hence $2\alpha + 4\alpha/k \le 3\alpha$. Thus applying Rules 7.5 and 7.6 amounts to $O(\alpha^5)$ time except in the root, where it is $O(\alpha^2 nm)$ time. If $k \le \hat{c}$, then we have additional running time of $O((3\alpha)^{\hat{c}} \cdot \alpha^2)$ to test each k-vertex subgraph. Overall, the algorithm thus runs in $O(2^{\log c\cdot 3\alpha} \cdot (3\alpha)^{\max\{5,\hat{c}+2\}} + \alpha^2 nm)$ time.

It remains to find a suitable $c > 1$ and, in turn, a suitable integer $\hat{c}$. For this, we set $c = c(\alpha) = 2^{1/\phi(\alpha)}$. This is a valid choice for c since $c(\alpha)$ is greater than 1 for all nontrivial instances. Using the above analysis of the running time we obtain a bound of $\mathrm{O}(2^{3\alpha/\phi} \cdot (3\alpha)^{\max\{5,\hat{c}+2\}} + \alpha^2 nm + A)$ where A is the time needed to compute $\hat{c}$ and

$$\begin{aligned}
\hat{c} \;\geq\; 2\log_c \frac{c}{c-1} \;&=\; \frac{2\log\frac{2^{1/\phi}}{2^{1/\phi}-1}}{\log 2^{1/\phi}} \;=\; 2\phi \cdot \log \frac{2^{1/\phi}}{2^{1/\phi}-1} \\
&=\; 2\phi \cdot \left(\frac{1}{\phi} - \log\left(2^{1/\phi} - 1\right) \right) \;=\; 2 + 2\phi \cdot \log \frac{1}{2^{1/\phi}-1}.
\end{aligned}$$

Note that, if $\hat{c} \log \alpha \in \mathrm{O}(\alpha/\phi)$ and A is reasonably small, then the time bound for the algorithm is $\mathrm{O}(2^{\mathrm{O}(\alpha/\phi)} + \alpha^2 nm)$. We claim that ϕ is such that our choice of c and $\hat{c}$ indeed imply that $\hat{c} \log \alpha \in \mathrm{O}(\alpha/\phi)$, that is

$$\lim_{\alpha\to\infty} \frac{\log(\alpha) \cdot \phi \cdot f(\alpha,\phi)}{\frac{\alpha}{\phi}} \;=\; \lim_{\alpha\to\infty} \frac{f(\alpha,\phi)}{\frac{\alpha}{\phi^2 \log \alpha}} < \infty \tag{7.1}$$

where $f(\alpha,\phi) = \log \frac{1}{2^{1/\phi}-1}$. Let us apply L'Hôpital's rule to prove Inequality (7.1). For this, we need that $\lim_{\alpha\to\infty} f(\alpha,\phi) = \infty$, which is fulfilled since ϕ is nondecreasing and unbounded, and we need that $\lim_{\alpha\to\infty} \frac{\alpha}{\phi^2 \log \alpha} = \infty$, which is also fulfilled, because $\phi^2 \in \mathrm{o}(\alpha/\log \alpha)$. Assuming both conditions hold, we may apply L'Hôpital's rule to obtain a limit equivalent to the one in Inequality (7.1) (if it exists) as follows. Below ln denotes the natural logarithm and recall that for a function g we denote by g' its derivative with respect to α.

$$\begin{aligned}
\lim_{\alpha\to\infty} \frac{f'(\alpha,\phi)}{\left(\frac{\alpha}{\phi^2 \log \alpha}\right)'} \;&=\; \lim_{\alpha\to\infty} \frac{(2^{1/\phi}-1)\frac{-1}{(2^{1/\phi}-1)^2} \cdot \frac{-2^{1/\phi}\phi' \ln 2}{\phi^2}}{\frac{1}{\phi^2} \cdot \left(\frac{1}{\log \alpha} - \frac{1}{\ln 2 \log^2 \alpha}\right) - \frac{2\phi'}{\phi^3} \cdot \frac{\alpha}{\log \alpha}} \\
=\; \lim_{\alpha\to\infty} \frac{\log(\alpha) \cdot 2^{1/\phi} \cdot \phi' \ln 2}{(2^{1/\phi}-1) \cdot \left(1 - \frac{1}{\ln 2 \log \alpha} - \frac{2\phi'\alpha}{\phi}\right)} \;&=\; \lim_{\alpha\to\infty} \frac{\frac{\log(\alpha) \cdot 2^{1/\phi} \cdot \phi' \ln 2}{1 - \frac{1}{\ln 2 \log \alpha} - \frac{2\phi'\alpha}{\phi}}}{2^{1/\phi}-1}
\end{aligned} \tag{7.2}$$

Note that, in the above, we have used the fact that ϕ is differentiable for $\alpha > 0$. We want to apply L'Hôpital's rule again to eliminate the factor $2^{1/\phi} - 1$. For this, we have $\lim_{\alpha\to\infty} 2^{1/\phi} - 1 = 0$ since we assumed ϕ to be positive, nondecreasing, and unbounded. In addition, $\log(\alpha)\cdot 2^{1/\phi}\cdot\phi'\ln 2 \in \mathrm{O}(\phi'\log\alpha)$ and since we assumed $\phi^2 \in \mathrm{o}(\alpha/\log\alpha)$ we have

$$\left|1-\frac{1}{\ln 2\log\alpha}-\frac{2\phi'\alpha}{\phi}\right| \in \Omega\left(\phi'\sqrt{\alpha\log\alpha}\right). \tag{7.3}$$

Hence, also

$$\lim_{\alpha\to\infty}\frac{\log(\alpha)\cdot 2^{1/\phi}\cdot\phi'\ln 2}{1-\frac{1}{\ln 2\log\alpha}-\frac{2\phi'\alpha}{\phi}} = 0$$

and we can apply L'Hôpital's rule to Equation (7.2). We obtain the following where we let $h(\alpha,\phi)=1-1/(\ln 2\log\alpha)-2\phi'\alpha/\phi$ and where we use the fact that ϕ' is differentiable for $\alpha>0$.

$$\lim_{\alpha\to\infty}\frac{\left(\log(\alpha)\cdot 2^{1/\phi}\cdot\phi'\ln 2\right)'}{(2^{1/\phi}-1)'\cdot h(\alpha,\phi)}-\frac{\log(\alpha)\cdot 2^{1/\phi}\cdot\phi'\ln 2\cdot h(\alpha,\phi)'}{(2^{1/\phi}-1)'\cdot h(\alpha,\phi)^2}$$

$$= \lim_{\alpha\to\infty}\frac{\ln 2\left(\left(\phi''\log\alpha+\frac{\phi'}{\alpha\ln 2}\right)\cdot 2^{1/\phi}+\log(\alpha)\cdot\phi'\cdot\frac{-2^{1/\phi}\phi'\ln 2}{\phi^2}\right)}{\frac{-2^{1/\phi}\phi'\ln 2}{\phi^2}\cdot h(\alpha,\phi)}$$
$$-\frac{\log(\alpha)\cdot 2^{1/\phi}\cdot\phi'\ln 2\cdot\left(\frac{-1}{\alpha\log^2(\alpha)\ln 2}-\frac{2(\phi'+\phi''\alpha-(\phi')^2\alpha)}{\phi^2}\right)}{\frac{-2^{1/\phi}\phi'\ln 2}{\phi^2}\cdot h(\alpha,\phi)^2}$$

$$= \lim_{\alpha\to\infty}\frac{\ln 2\left(\left(\phi''\log\alpha+\frac{\phi'}{\alpha\ln 2}\right)\cdot 2^{1/\phi}\right)}{\frac{-2^{1/\phi}\phi'\ln 2}{\phi^2}\cdot h(\alpha,\phi)}+\frac{\log(\alpha)\cdot\phi'\cdot\ln 2}{h(\alpha,\phi)}$$
$$+\frac{\log(\alpha)\phi^2}{h(\alpha,\phi)^2}\cdot\left(\frac{1}{\alpha\log^2(\alpha)\ln 2}+\frac{2(\phi'+\phi''\alpha-(\phi')^2\alpha)}{\phi^2}\right)$$

Using Equation (7.3) summand two vanishes and it remains to show the following.

$$\lim_{\alpha\to\infty}\frac{1}{h(\alpha,\phi)}\cdot\left(\frac{\phi^2\phi''\log\alpha}{\phi'}+\frac{\phi^2}{\alpha\ln 2}\right)$$
$$+\frac{1}{h(\alpha,\phi)^2}\cdot\left(\frac{\phi^2}{\alpha\log(\alpha)\ln 2}+2\log(\alpha)\cdot(\phi'+\phi''\alpha-(\phi')^2\alpha)\right) < \infty$$

Using $\phi^2 \in \mathrm{o}(\alpha/\log\alpha)$, the above simplifies further.

$$\lim_{\alpha\to\infty} \frac{\phi^2\phi''\log\alpha}{h(\alpha,\phi)\phi'} + \frac{2\log(\alpha)\cdot(\phi'+\phi''\alpha-(\phi')^2\alpha)}{h(\alpha,\phi)^2} < \infty \tag{7.4}$$

We need to upper bound ϕ' and ϕ''. First, since $\phi^2 \in \mathrm{o}(\alpha/\log\alpha)$ we have

$$\begin{aligned}\phi' &\in \mathrm{o}\left(\left(\sqrt{\frac{\alpha}{\log\alpha}}\right)'\right) = \mathrm{o}\left(\frac{\frac{\sqrt{\log\alpha}}{2\sqrt{\alpha}} - \frac{\sqrt{\alpha}}{2\sqrt{\log\alpha}}\cdot\frac{1}{\alpha\ln 2}}{\log\alpha}\right) = \mathrm{o}\left(\frac{\log\alpha}{\sqrt{\alpha}\log^{3/2}\alpha}\right)\\ &= \mathrm{o}\left(\frac{1}{\sqrt{\alpha\log\alpha}}\right).\end{aligned} \tag{7.5}$$

Similarly $\phi'' \in \mathrm{o}(g(\phi,\alpha))$, where

$$\begin{aligned}g(\phi,\alpha) &= \left(\sqrt{\frac{\alpha}{\log\alpha}}\right)'' = \frac{\frac{2\sqrt{\alpha}\log^{3/2}\alpha}{\alpha\ln 2} - \left(\log\alpha - \frac{1}{\ln 2}\right)\cdot\left(\frac{2\log^{3/2}\alpha}{2\sqrt{\alpha}} + \frac{3\sqrt{\alpha}\sqrt{\log\alpha}}{\alpha\ln 2}\right)}{4\alpha\log^3\alpha}\\ &= \frac{\frac{2}{\ln(2)\cdot\log\alpha} - \left(1-\frac{1}{\ln(2)\cdot\log\alpha}\right)\cdot\left(1+\frac{3}{\ln(2)\cdot\log\alpha}\right)}{4\alpha\sqrt{\alpha\log\alpha}} \in \mathrm{O}\left(\frac{1}{\alpha\sqrt{\alpha\log\alpha}}\right).\end{aligned} \tag{7.6}$$

Let us now prove Equation (7.4) by considering both summands individually. Employing Equation (7.5) and then Equation (7.3), we see that the first summand is

$$\frac{\phi^2\phi''\log\alpha}{h(\alpha,\phi)\phi'} \in \mathrm{O}\left(\frac{\alpha}{\alpha\sqrt{\alpha\log\alpha}\cdot h(\alpha,\phi)\cdot\phi'}\right) \subseteq \mathrm{O}\left(\frac{1}{\alpha\log(\alpha)\cdot(\phi')^2}\right)$$

which turns to a constant as α grows, because we assumed that $\phi' \in \Omega(1/\sqrt{\alpha\log\alpha})$. The second summand of Equation (7.4) is upper bounded as follows.

$$\frac{2\log(\alpha)\cdot(\phi'+\phi''\alpha-(\phi')^2\alpha)}{h(\alpha,\phi)^2} \in \mathrm{O}\left(\frac{\log\alpha\cdot\left(\frac{1}{\sqrt{\alpha\log\alpha}}+\frac{\alpha}{\alpha\sqrt{\alpha\log\alpha}}\right) - \frac{\alpha}{\alpha\log\alpha}}{\alpha\log(\alpha)\cdot(\phi')^2}\right)$$

Using again $\phi' \in \Omega(1/\sqrt{\alpha\log\alpha})$ also this summand turns to some constant for growing α.

Hence, we have proved that for all $\phi = \phi(\alpha) > 0$ ($\alpha > 0$) that are nondecreasing and unbounded, that fulfill $\phi^2 \in o(\alpha/\log\alpha)$, that are two-times differentiable for $\alpha > 0$, and that fulfill $\phi' \in \Omega(1/\sqrt{\alpha\log\alpha})$ we may choose $\hat{c}$ such that $\hat{c}\log\alpha \in O(\alpha/\phi)$. Hence yielding a running time of $O(2^{O(\alpha/\phi)} + \alpha^2 nm + A)$ for our algorithm where A is the time needed to compute $\hat{c}$. Finally, if ϕ is efficiently computable, there also is an efficiently computable function in $O(\alpha/(\phi\log\alpha))$ computing $\hat{c}$ from α. □

7.5. Concluding remarks

In this chapter, we aimed to better understand the parameterized complexity of HIGHLY CONNECTED SUBGRAPH. With respect to the parameters maximum degree Δ, h-index h, and degeneracy d the results are similar to the μ-CLIQUE problem: We obtain tractability for Δ and h, but intractability with respect to d even when combined with the solution size k. The tractability result for the h-index is promising for practice, but the running time must be improved. We conjecture, however, that single exponential, that is, $c^h \cdot \text{poly}(n)$ running time for some constant c cannot be achieved. The intractability result for $d + k$ is disappointing, as these parameters are likely to be small in real-world instances. Perhaps it is possible to add another small parameter to the parameter combination and still obtain a useful algorithm, or to give fixed-parameter approximation algorithms. (Although we believe that this is unlikely due to the fact that highly connected subgraphs are not hereditary.)

Regarding subexponential-time algorithms, we showed that there is no such algorithm with respect to the number n of vertices but that there is one with $2^{O(\sqrt{\alpha}\log\alpha)} + O(\alpha^2 nm)$ running time, where α is the number of edges not in the solution. From the fact that the best theoretical upper bound of α with respect to n that we can give is of the order $\alpha \leq \binom{n}{2}$, it seems that the trivial $2^n \cdot \text{poly}(n)$-time algorithm might be superior to the subexponential one with respect to α. However, in practice we expect that α is upper bounded by cn for some small constant c, making the one with respect to α preferable in many cases. It is interesting to note that the analysis of the subexponential-time algorithm applies more generally to any problem parameterized by two parameters κ, ν that fulfills the following conditions. It has a polynomial problem kernel with respect to κ, an $n^{O(\nu)}$-time algorithm, and a search-tree algorithm such that the depth of search tree is upper bounded linearly in κ and the branching vector is $(1, \Omega(\nu))$. Then we obtain a subexponential algorithm with respect to κ. It would be interesting to search for other applications of this result. We also note that the analysis of our algorithm could still be improved.

The practically most relevant of our results is perhaps the single-exponential algorithm with respect to the number γ of edges between the desired highly connected graph and the remaining vertices. Indeed, preliminary experiments on subnetworks of the yeast protein interaction network identified by Bruckner, Hüffner, and Komusiewicz [BHK15] show promise: in more than 80 % of the instances the largest (isolated) highly connected subgraph can be determined within an hour. Moreover, the corresponding algorithm shows improvement over a straightforward enumeration-based algorithm that loosely corresponds to an enhanced version of Proposition 7.2. Finally, the isolation parameter may also be a useful tool for practitioners searching for separated communities.

Part III

Be robust!

Chapter 8

Vector connectivity sets

This chapter is based on “On Kernelization and Approximation for the Vector Connectivity Problem” by Stefan Kratsch and Manuel Sorge (Algorithmica [KS16]).

8.1. Introduction

In this chapter, we study VECTOR CONNECTIVITY, a problem related to a variant of the DOMINATING SET problem: In VECTOR DOMINATION we are given a graph G and a *demand* $\lambda(v)$ for each vertex $v \in V(G)$, and we are asked to find a minimum-size vertex subset $S \subseteq V(G)$ such that for each vertex $v \in V(G) \setminus S$, there are $\lambda(v)$ vertex-disjoint paths from v to S, each of which has length one [HPV]. Herein and in the following, a set of vertex-disjoint paths from v to S means a set of paths, each of which goes from v to a vertex in S, such that each pair of the paths is vertex-disjoint except for sharing v as an endpoint. In VECTOR CONNECTIVITY, we do not require the length of the paths to be one. Formally, the problem is defined as follows.

VECTOR CONNECTIVITY
Input: An undirected graph G, nonnegative integers k and d, and a demand function $\lambda\colon V \to \{0,\dots,d\}$.
Question: Is there a set $S \subseteq V(G)$ of at most k vertices such that for each vertex $v \in V(G) \setminus S$ there are $\lambda(v)$ vertex-disjoint paths from v to S?

We also say that the value $\lambda(v)$ is the *demand* of vertex v. We say that $S \subseteq V$ is a *vector connectivity set* for (G,λ) if for each vertex in $V(G) \setminus S$ there are $\lambda(v)$ vertex-disjoint paths form v to S.

VECTOR CONNECTIVITY is NP-complete [CMR15] and was introduced by Boros et al. [Bor+14] to study connectivity and domination constraints. There are three natural applications in which this problem arises.

The first application is related to installing servers in computer networks to serve users. The placement of the servers shall be so that the service is robust against failure in the network. More precisely, each user has a requirement on the robustness, its demand, meaning that she requires a certain number of physically independent ways to connect to the service. Modeling the possible server housing and user positions as vertices, and their physical connections as edges, we arrive at a graph model of this problem. Herein, a minimum-size vector connectivity set represents the locations of servers to install to serve all users with their corresponding robustness conditions. Instead of robustness constraints, the demands of the users could also model requirements on a minimum throughput value.

Similarly, one can think of the graph as a street network in which we want to place warehouses. From the warehouses we can then supply goods to clients [CMR15]. As before, the demands model constraints on the robustness of the supply, which could play a role in critical infrastructure in regions struck by natural disasters, for example. In this way, Vector Connectivity can also be seen as a variant of the Facility Location problem [HD02] in which the costs of serving demand are negligible, but instead redundancy is required.

Cicalese, Milanič, and Rizzi [CMR15] also noted an application related to information propagation in social networks. (Information propagation was also one of the motivations for Boros et al. [Bor+14] to initiate the study of Vector Connectivity.) Imagine an advertiser trying to reach individuals by a viral marketing campaign. However, the target individuals are skeptical, tracing the information that they get from their peers back to their sources. Each individual has a certain integer threshold t, and she only believes the information to be true, if she can trace it back to via t independent paths to distinct sources. Hence, modeling the social network as a graph, the advertiser can use a vector connectivity set to convince the target individuals that her product is valuable.

Our contribution. We analyze the parameterized complexity of Vector Connectivity with respect to the solution size k, mainly from a data reduction point of view. We provide a randomized fixed-parameter algorithm for Vector Connectivity parameterized by the solution size k (Section 8.4). We prove that, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, Vector Connectivity does not admit a polynomial problem kernel with respect to k and, in fact, even with respect to $k + d$, where d is the maximum demand (Section 8.7). However, the variant Vector d-Connectivity where the maximum demand d is a fixed constant *does* admit a vertex-linear problem kernel with respect to k.

Our analysis of the problem starts with a data reduction rule stating that we can safely "forget" the demand of $r := \lambda(v)$ at a vertex v if v has vertex-disjoint paths to r vertices each of demand at least r (Section 8.3). Basically, the reason is that these vertex-disjoint paths prove that, if we can serve the demand of all remaining vertices, then we can also serve the demand of v. After exhaustive application of the corresponding data reduction rule, all remaining vertices of demand r have separators of size at most $r - 1$, separating them from other vertices of demand at least r. By analyzing these separators we then show that any yes-instance of Vector d-Connectivity has at most d^2k vertices with nonzero demand; the corresponding upper bound for Vector Connectivity is $k^3 + k$.

In Section 8.4 we prove that Vector d-Connectivity admits a vertex-linear problem kernel with respect to k. Our method is to identify regions in the input graph which are separated from the rest of the graph by small (constant-size) separators and to replace these regions by a constant-size gadget. The regions are all anchored in one of the vertices with nonzero demand and we show that all the remaining vertices can be removed using the so-called torso operation. After this, combining the above upper bound d^2k on the number of vertices with nonzero demand with an upper bound $\phi(d)$ on the number of regions for each nonzero demand vertex, there remain only $\phi(d)d^2k$ vertices. The function $\phi(d)$ is exponential in d and we know that it must be superpolynomial in d since it would otherwise contradict our lower bound on the kernel size (Section 8.7) which rules out size polynomial in $k+d$.

The replacement of the regions is done by giving an explicit description of the properties of each region and finding a constant-size replacement with the same properties. Hence, we can indeed construct the kernelization algorithm, albeit the replacements are found using brute force.

A somewhat simpler, non-constructive *existence proof* of a linear-vertex problem kernelization for VECTOR d-CONNECTIVITY can be pieced together from recent work on meta kernelization on sparse graph classes [Fom+13]. We explain some details in Section 8.5 and highlight differences to our approach. The existence proof also relies on replacing regions of the input graph by constant-size gadgets. However, it does not imply a way to construct these gadgets, which, in comparison, our explicit description of their properties does. We also give a direct proof for the number of such regions rather than relying on a known argument for upper bounding the number of connected subgraphs of bounded size and bounded neighborhood size (via the two-families theorem of Bollobás, see Jukna [Juk01]). This avoids an exponential dependency of the kernel size on the size of the gadgets.

Known results and related work. The study of the VECTOR CONNECTIVITY problem was initiated by Boros et al. [Bor+14] who gave polynomial-time algorithms for trees, cographs, and split graphs. Moreover, they obtained an $(\ln n+2)$-factor approximation algorithm for the general case. Cicalese, Milanič, and Rizzi [CMR15] continued the study of VECTOR CONNECTIVITY and among other results proved that the optimization version, in which we seek to minimize the size of the vector connectivity set, is APX-hard (and NP-hard) on general graphs, even when all demands are upper bounded by four.

With regard to approximation, we mention that it is possible to augment the reduction rules presented in Section 8.3 and use them as a crucial ingredient in polynomial-time approximation algorithms for the optimization variants of VECTOR d-CONNECTIVITY and VECTOR CONNECTIVITY in which we want to minimize the vector connectivity sets. In this way, one obtains a factor d approximation algorithm for VECTOR d-CONNECTIVITY and a factor opt approximation algorithm for

VECTOR CONNECTIVITY, where opt denotes the minimum size of a vector connectivity set [KS16].

In a talk during a Dagstuhl seminar[15] Martin Milanič asked whether VECTOR CONNECTIVITY is fixed-parameter tractable with respect to the solution size k. This was answered affirmatively by Daniel Lokshtanov[16] based on an extension of the Randomized Contractions technique [Chi+16]. Our fixed-parameter tractability result is different from his approach. We instead rely on a matroid intersection algorithm of Marx [Mar09].

Outline of this chapter. Before giving the kernelization and fixed-parameter tractability results, in Section 8.2 we give some basic facts on vertex-disjoint paths and related notions of separators which we tacitly use later on. In Section 8.3 we then describe how to reduce the number of vertices with nonzero demand. We use the corresponding reduction rules in our fixed-parameter tractability result for VECTOR CONNECTIVITY with respect to k in Section 8.4 and in the vertex-linear kernelization in Section 8.6. An outline of the kernelization argument and an alternative non-constructive proof sketch for the kernelization is given in Section 8.5. We rule out polynomial problem kernels for VECTOR CONNECTIVITY under the premise that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$ in Section 8.7. We conclude in Section 8.8.

8.2. Specific preliminaries

We now introduce some notation and definitions and note some observations that we use later on. They are related to vertex-disjoint paths, special kinds of separators that we use, submodular functions, and so-called closest sets.

We sometimes need to check whether there are a given number of vertex-disjoint paths between two vertices.

Proposition 8.1. Given a graph G with n vertices and m edges, $s, t \in V(G)$, and $\ell \in \mathbb{N}$, in $O(\ell(n+m))$ time we can check whether there are ℓ internally vertex-disjoint paths between s and t in the graph G.

Proof sketch. We check for the existence of paths above by by using a folklore reduction to finding a flow of value ℓ in a flow network [Ski08]: Create a flow network by introducing two vertices v_{in} and v_{out} for each vertex $v \in V(G)$, and also an arc

[15] Dagstuhl Seminar 14071 on "Graph Modification Problems", February 9–14, 2014.
[16] Unpublished result.

$(v_{\text{in}}, v_{\text{out}})$ of capacity one. Then, for each edge $\{u, v\}$, add two arcs $(u_{\text{out}}, v_{\text{in}}), (v_{\text{out}}, u_{\text{in}})$ with capacity infinity. One can check that there is a flow of value ℓ between s_{out} and t_{in} if and only if the desired ℓ vertex-disjoint paths between s and t exist. We can check in $O(\ell(n+m))$-time whether there is a flow of value ℓ between s_{out} and t_{in} by running ℓ flow-augmentation rounds of the Ford-Fulkerson algorithm. Thus, within the same time, we can test whether there are ℓ vertex disjoint paths between s and t in G. □

Due to the nature of the VECTOR CONNECTIVITY problem we also frequently want to know whether there are $\ell \in \mathbb{N}$ vertex-disjoint paths from a vertex v to some vertex set S. This can be reduced to finding ℓ internally vertex-disjoint paths as follows. First, introduce $\ell - 1$ copies of v with the same neighborhood. Then introduce a new vertex s adjacent to all copies of v and introduce a new vertex t adjacent to all vertices in S. In the resulting graph there are ℓ internally vertex-disjoint paths between s and t if and only if there are ℓ vertex-disjoint paths between v and S. Thus, by Proposition 8.1 we can check this in $O(\ell(n+m))$ time.

Corollary 8.1. Given a graph G with n vertices and m edges, $v \in V(G)$, $S \subseteq V(G) \setminus \{v\}$, and $\ell \in \mathbb{N}$, in $O(\ell(n+m))$ time we can check whether there are ℓ vertex-disjoint paths from v to S in G.

The natural counterpart to a set of vertex-disjoint paths from v to S, in the spirit of Menger's theorem, is a *v,S-separator*, that is, a vertex subset $C \subseteq V(G) \setminus \{v\}$ such that in $G - C$ no vertex of S is reachable from v. Note that a v,S-separator C does not necessarily have the property that $G - C$ has more connected components than G, contrary to our definition of ordinary separators. The maximum number of vertex-disjoint paths from v to S equals the minimum size of a v,S-separator. Minimal separators can be thought of as the neighborhood of one of the connected components in the rest of the graph. In this regard, several proofs use the function $f\colon 2^V \to \mathbb{N}\colon U \mapsto |N(U)|$, which is well-known to be *submodular*, that is, for all $X, Y \subseteq V$ it holds that

$$f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y).$$

So-called *closest sets* will be used frequently; these occur naturally in separator problems but appear to have no generalized name. We define a vertex set C to be *closest to v* if C is the unique v,C-separator of size at most $|C|$, where v,C-separator is in the above sense. As an example, if C is a minimum s,t-separator that, among such separators, has the smallest connected component for s in $G - C$, then C is also closest. The following proposition captures some properties of closest sets, mostly specialized to v, S-separators (see also Kratsch and Wahlström [KW12]).

Proposition 8.2. Let $G = (V, E)$, let $v \in V$, and let $S \subseteq V \setminus \{v\}$. The following holds.

(i) If C is a closest v, S-separator and X the connected component of v in $G - C$, then for every set $X' \subseteq X$ with $v \in X' \subsetneq X$ we have $|N(X')| > |N(X)|$.

(ii) The minimum v, S-separator C minimizing the size of the connected component of v in $G - C$ is unique. The set C is also the unique minimum v, S-separator closest to v.

(iii) If C_1 and C_2 are minimum v, S-separators and C_1 is closest to v, then the connected component of v in $G - C_1$ is fully contained in the connected component of v in $G - C_2$.

(iv) If $C \subseteq V \setminus \{v\}$ is closest to v, then so is every subset C' of C.

Proof. Let $G = (V, E)$, let $v \in V$, and let $S \subseteq V \setminus \{v\}$. We give simple proofs for all claims; some of them use the submodularity of $f \colon 2^V \to \mathbb{N} \colon U \mapsto |N(U)|$.

First, we prove Statement (i). Note that $N(X) = C$ as, otherwise, $|N(X)| < |C|$ and $N(X)$ is a v, C-separator, contradicting the closeness of C. Let $X' \subseteq X$ with $v \in X' \subsetneq X$. Clearly $X' \cap C \subseteq X \cap C = \emptyset$, making $N(X')$ a v, C-separator. Since $X' \subsetneq X$ and X is connected, it follows that $N(X') \cap X \neq \emptyset$ and, thus, that $N(X') \neq N(X)$. Since C is closest to v, we have $|N(X')| > |C| = |N(X)|$ because, otherwise, $N(X')$ would be a v, C-separator of size at most $|C|$ but different from $C = N(X)$. Thus, Statement (i) holds.

Next, we prove Statement (ii). Assume that both C_1 and C_2 are minimum v, S-separators that furthermore minimize the size of the connected component of v in $G - C_i$; let X_i denote those components. Note that $N(X_i) = C_i$ since C_i is a minimum v, S-separator. We have

$$f(X_1) + f(X_2) \;\geq\; f(X_1 \cap X_2) + f(X_1 \cup X_2).$$

Clearly, $(X_1 \cup X_2) \cap S = \emptyset$, implying that $N(X_1 \cup X_2)$ is a v, S-separator. It follows that $|N(X_1) \cup N(X_2)| = f(X_1 \cup X_2) \geq f(X_2)$, which implies that $f(X_1) \geq f(X_1 \cap X_2)$. It is easy to see that $N(X_1 \cap X_2)$ is also a v, S-separator of size at most $f(X_1) = |C_1|$ and, if $C_1 \neq C_2$, then $X_1 \neq X_2$ and $X_1 \cap X_2 \subsetneq X_1$; a contradiction to X_1 being a minimum-size connected component. Thus, we have $C_1 = C_2$, proving the first part of Statement (ii).

For the second part, assume first that C_3 is a v, C_1-separator of size at most $|C_1|$ and let X_3 be the connected component of v in $G - C_3$. Since every path from v to S contains a vertex of C_1, each such path must also contain a vertex of C_3 (to separate v from C_1). Thus C_3 is also a v,S-separator, but then C_3 is also minimum (as $|C_3| \leq |C_1|$). Note that, since C_3 is a v,C_1-separator we have $X_3 \cap C_1 = \emptyset$. It follows that $X_3 \subseteq X_1$. If $X_3 \subsetneq X_1$, then C_3 would violate the choice of C_1 as minimum v,S-separator with minimum component size for v. But then we have $X_3 = X_1$ and $C_1 = C_3$, which proves closeness of C_1 to v.

Finally, assume that C_4 is another minimum v,S-separator closest to v; we want to show $C_1 = C_4$. If X_1 and X_4 are the corresponding connected components, then $C_i = N(X_i)$. By submodularity of f and the same arguments as above we find that $|N(X_1 \cap X_4)| \leq |C_1|$. If $X_1 \nsubseteq X_4$, then $v \in X_1 \cap X_4 \subsetneq X_1$ which, by Statement (i), implies $|N(X_1 \cap X_4)| > |N(X_1)| = |C_1|$; a contradiction. Else, if $X_1 \subseteq X_4$, then $C_1 = N(X_1)$ is a v,C_4-separator of size $|C_1| = |C_4|$; this implies $C_1 = C_4$ by closeness of C_4. Thus, Statement (ii) holds.

Now we prove Statement (iii). Let C_1 and C_2 be minimum v,S-separators and let C_1 be closest to v. Let X_i be the connected component of v in $G - C_i$. Note that $N(X_i) = C_i$, since C_i is minimum. Assume, for the sake of contradiction, that $X_1 \nsubseteq X_2$, implying that $X_1 \cap X_2 \subsetneq X_1$. By Statement (i) we have $f(X_1 \cap X_2) = |N(X_1 \cap X_2)| > |C_1| = |C_2| = f(X_2)$. Because

$$f(X_1) + f(X_2) \;\geq\; f(X_1 \cap X_2) + f(X_1 \cup X_2),$$

we have $|N(X_1 \cup X_2)| < f(X_1) = |C_1| = |C_2|$. However, $N(X_1 \cup X_2)$ is also a v,S-separator; this contradicts C_1 and C_2 being minimum v,S-separators. Thus, also Statement (iii) holds.

Finally, we prove Statement (iv). Let C be closest to v and let $C' \subseteq C$. If C' is not closest to v, then there is a v,C'-separator C'' of size at most $|C'|$ and with $C'' \neq C'$. Consider $\hat{C} = C'' \cup (C \setminus C')$ and note that $|\hat{C}| \leq |C''| + |C \setminus C'| \leq |C'| + |C \setminus C'| = |C|$. Observe that $\hat{C}$ is a v,C-separator since $C'' \subseteq \hat{C}$ separates v from C', and $C \setminus C'$ is contained in $\hat{C}$. Thus, $|\hat{C}| < |C|$ would contradict C being closest to v because it implies that $\hat{C} \neq C$ in addition to $\hat{C}$ being a v,C-separator of size at most $|C|$. Thus, $|\hat{C}| = |C|$ and by closeness of C to v, we have $\hat{C} = C$. The latter can hold only if $C'' \supseteq C'$ because $\hat{C} = C'' \cup (C \setminus C')$, but then $C'' = C'$ because $|C''| \leq |C'|$, contradicting $C'' \neq C'$. Thus C' is indeed closest to v. □

8.3. Reducing the number of vertices with nonzero demand

In this section we introduce a data reduction rule for Vector Connectivity that reduces the total demand. We prove that the reduction rule does not affect the solution space, which makes it applicable not only for kernelization but also for approximation and heuristics. For use in an approximation algorithm, see Kratsch and Sorge [KS16]. In Section 8.6, we will use the reduction rule in our polynomial kernelization for VECTOR d-CONNECTIVITY parameterized by k, and in Section 8.4 we apply the reduction rule in a fixed-parameter algorithm for VECTOR CONNECTIVITY parameterized by k.

Intuitively, if a vertex v has many vertex-disjoint paths to other vertices with at least the same demand, then, satisfying these other vertices will automatically satisfy v. To make this formal, let G be a graph, $\lambda\colon V(G) \to \{0,\dots,d\}$ a demand function, and $v \in V(G)$. Denote by $X(v)$ the set of vertices with demand at least $\lambda(v)$, that is, $X(v) = \{u \in V(G) \mid (u \neq v) \wedge (\lambda(u) \geq \lambda(v))\}$.

Rule 8.1. Let (G,λ,k) be an instance of VECTOR CONNECTIVITY. If there is a vertex $v \in V(G)$ with at least $\lambda(v)$ vertex-disjoint paths from v to $X(v)$, then set the demand $\lambda(v)$ of v to zero.

We prove that the rule does not affect the space of feasible solutions for every instance. Intuitively, the fact that there are many vertex disjoint paths from v to other vertices of at least the same demand, and the assumption that there is a vector connectivity set for all the remaining vertices, imply that there cannot be a small separator that separates v from the vector connectivity set.

Lemma 8.1. Let (G,λ,k) be an instance of VECTOR CONNECTIVITY and let (G,λ',k) be the instance obtained via a single application of Rule 8.1. For every $S \subseteq V(G)$ it holds that S is a solution for (G,λ,k) if and only if S is a solution for (G,λ',k).

Proof. Let v denote the vertex whose demand was set to zero by Rule 8.1. Clearly, $\lambda(u) = \lambda'(u)$ for all vertices $u \in V(G) \setminus \{v\}$, and $\lambda'(v) = 0$, that is, $\lambda(u) \geq \lambda'(u)$ for all vertices $u \in V(G)$. Hence, it suffices to show that, if S fulfills all demands according to λ', then S fulfills also all demands according to λ. This in turn comes down to proving that S fulfills the demand of r at v assuming that it fulfills all demands according to λ'. If $v \in S$, then the demand of v is trivially fulfilled so henceforth assume that $v \notin S$.

Set $r := \lambda(v)$. Let $w_1,\dots,w_r \in X(v)$ denote vertices different from v with demand each at least r such that there exist r vertex-disjoint paths from v to $\{w_1,\dots,w_r\}$, that is, a single path to each w_i. Such vertices exist because Rule 8.1 was applicable to v.

Assume for contradiction that S does not satisfy the demand of r at v (recall that $v \notin S$, by assumption), that is, that there are no r vertex-disjoint paths from v to S

that overlap only in v. It follows directly that there is a v, S-separator C of size at most $r-1$. (Recall that C may contain vertices of S but not the vertex v.) Let R denote the connected component of v in $G-C$. Then the following holds for each vertex $w_i \in \{w_1, \dots, w_r\}$:

- If $w_i \in S$, then $w_i \notin R$: Otherwise, we would have $S \cap R \supseteq \{w_i\} \neq \emptyset$ contradicting the fact that v can reach no vertex of S in $G-C$.
- If $w_i \notin S$, then $w_i \notin R$: Since S fulfills demands according to λ' there must be at least r vertex-disjoint paths from w_i to S that overlap only in w_i. However, since $w_i \in R$ the set C is also a w_i, S-separator; a contradiction since C has size less than r.

Thus, no vertex $w_1, \dots, w_r$ is contained in R. This, however, implies that C separates v from $\{w_1, \dots, w_r\}$, contradicting the fact that there are r vertex-disjoint paths from v to $\{w_1, \dots, w_r\}$ that overlap only in v. It follows that no such v, S-separator C can exist, and, hence, that there are at least $r = \lambda(v)$ vertex-disjoint paths from v to S, as claimed. Thus, S fulfills the demand of r at v and hence, fulfills all demands according to λ. (Recall that the converse holds trivially since $\lambda(u) \geq \lambda'(u)$ for all vertices $u \in V(G)$.) □

We have established that applications of Rule 8.1 do not affect the solution space of an instance while reducing the number of vertices with nonzero demand.

Lemma 8.2. Rule 8.1 can be exhaustively applied in $O(dn(n+m))$ time, where d is the largest demand of any vertex, n is the number of vertices in the input graph, and m is its number of edges.

Proof. We check once for every vertex v with nonzero demand, whether Rule 8.1 applies to v and, if so, do apply it. Since the set of vertices with nonzero demand only shrinks in this process, Rule 8.1 cannot become applicable to a vertex for which we have already determined that it is not applicable. Using Corollary 8.1 we can check applicability in $O(d(n+m))$ time, yielding the claimed running time bound. □

To analyze the impact of Rule 8.1 we will now upper bound the number of vertices with nonzero demand in an exhaustively reduced instance in terms of the optimum solution size opt and the maximum demand d. To this end, we need the following technical lemma about the structure of reduced instances as well as some notation. If (G, λ, k) is reduced with respect to Rule 8.1, then for each vertex v with demand $r = \lambda(v) \geq 1$ there is a separator C of size at most $r-1$ that separates v from all other vertices with demand at least r. We fix for each vertex v with demand at least one a vertex set C, denoted $C(v)$, by picking the unique closest minimum v, D_v-separator where $D_v = \{u \in V \setminus \{v\} \mid \lambda(u) \geq \lambda(v)\}$. Furthermore, for such vertices v, let $R(v)$ denote the connected component of v in $G - C(v)$.

Intuitively, every solution S must intersect every set $R(v)$ since $|C(v)| < \lambda(v)$. The following lemma shows implicitly that Rule 8.1 limits the amount of overlap of the sets $R(v)$.

Lemma 8.3. Let (G, λ, k) be reduced under Rule 8.1. Let $u, v \in V(G)$ be distinct vertices with $\lambda(u) = \lambda(v) \geq 1$. If $R(u) \cap R(v) \neq \emptyset$, then $u \in C(v)$ or $v \in C(u)$.

Proof. Assume for contradiction that we have u, v with $\lambda(u) = \lambda(v) \geq 1$ and with $R(u) \cap R(v) \neq \emptyset$ and $u \notin C(v)$ and $v \notin C(u)$. We will show that this implies that at least one of $C(u)$ and $C(v)$ is not a closest minimum separator, giving a contradiction. By definition of the separators $C(u)$ and $C(v)$ as separating u and v, respectively, from all other vertices of at least the same demand, we have $u \notin R(v)$ and $v \notin R(u)$; furthermore $u \notin C(u)$ and $v \notin C(v)$, by definition.

Let $C = C(u) \cup C(v)$ and note that $u, v \notin C$. Let I, J denote the connected components of u, v in $G - C$. Note that $I \subseteq R(u)$ since $C \supseteq C(u)$ and, thus, $v \notin I$. Similarly we have $u \notin J$ and thus I and J are two different connected components in $G - C$. As the next step, we show that

$$|C(u)| + |C(v)| \geq |N(I)| + |N(J)|. \tag{8.1}$$

To this end, let us first note that $N(I) \cup N(J) \subseteq C = C(u) \cup C(v)$ by definition of I and J as connected components of $G - C$. Thus, every vertex p that appears in exactly one of $N(I)$ and $N(J)$ contributes value one to the right-hand side of Inequality (8.1) and at least value one to the left-hand side since it must be contained in $C(u) \cup C(v)$. Now, for vertices $p \in N(I) \cap N(J)$ we see that they contribute value two to the right-hand side of Inequality (8.1). Note that each such vertex is contained in a path from u to v whose other interior vertices are disjoint from $C = C(u) \cup C(v)$. Thus, p must be contained in both $C(u)$ and $C(v)$ since otherwise the corresponding set would fail to separate u from v (or vice versa), which is required since $\lambda(u) = \lambda(v)$. Therefore, if any vertex contributes a value of two to the right-hand side, then it also contributes two to the left-hand side. This establishes Inequality (8.1).

Now, from Inequality (8.1) we immediately get that at least one of $|N(I)| \leq |C(u)|$ or $|N(J)| \leq |C(v)|$ must hold. Due to symmetry we may assume $|N(I)| \leq |C(u)|$ without loss of generality. Recall that $u \in I$. Furthermore, we can see that $I \subsetneq R(u)$ by using the fact that $R(u) \cap R(v) \neq \emptyset$: Let $q \in R(u) \cap R(v) \subseteq R(u)$. If $q \in I$, then u can reach q in $G - C$ but, in particular, also in $G - C(v)$. By definition of $R(v)$ and using $q \in R(v)$, we know that v can reach q in $G - C(v)$, implying that there is a path from u to v in $G - C(v)$ (since there is a walk through q), violating the fact that $C(v)$ in particular separates v from u. Thus $q \notin I$ and since $I \subseteq R(u)$ follows from $C \supseteq C(u)$, we get that $I \subsetneq R(u)$. Since $|N(I)| \leq |C(u)|$ we find that $N(I)$ is of at most the same size as $C(u)$ but with a smaller connected component I for u, contradicting the fact that $C(u)$ is the unique minimum closest set that separates u from all other vertices of demand at least $\lambda(u)$. This completes the proof of the lemma. □

Now, we can give the promised upper bound on the number of vertices with nonzero demand.

Lemma 8.4. Let (G, λ, k) be an instance of VECTOR CONNECTIVITY that is reduced with respect to Rule 8.1 and let opt denote the minimum size of a vector connectivity set $S \subseteq V$ for this instance. Then there are at most $d^2\mathsf{opt}$ vertices with nonzero demand in G.

Proof. For analysis, let $S \subseteq V$ denote an arbitrary vector connectivity set of size opt, that is, such that every v with $\lambda(v) \geq 1$ has $v \in S$ or there are $\lambda(v)$ vertex-disjoint paths from v to S that overlap only in v. We will prove that for all $r \in \{1, \ldots, d\}$ there are at most $2r - 1$ vertices of demand r in G (according to λ). Fix some $r \in \{1, \ldots, d\}$ and let D_r denote the set of vertices with demand exactly r. For each $v \in D_r$ the vector connectivity set S must contain at least one vertex of $R(v)$ since $C(v) = N(R(v))$ has size at most $r - 1$. (Otherwise, $C(v)$ would be a v, S-separator of size less than r.) Fix some vertex $p \in S$ and let $v_1, \ldots, v_\ell$ denote all vertices of demand r that have $p \in R(v_i)$. We will prove that $\ell \leq (2r - 1)$ and $|D_r| \leq \mathsf{opt}(2r - 1)$.

At most $r - 1$ vertices are contained in $C(v_i)$ for every $i \in \{1, \ldots, \ell\}$. Thus, on the one hand, the total size $\sum |C(v_i)|$ of these sets is at most $(r - 1)\ell$. On the other hand, for every pair v_i, v_j with $1 \leq i < j \leq \ell$ we know that $v_i \in C(v_j)$ or $v_j \in C(v_i)$ by Lemma 8.3, since $R(v_i) \cap R(v_j) \supseteq \{p\} \neq \emptyset$. Thus, every pair contributes at least a value of one to the total size of the sets $C(v_i)$. (To see that different pairs have disjoint contributions note that, for example, $v_i \in C(v_j)$ uniquely defines pair v_i, v_j.) We get the following inequality:

$$(r - 1)\ell \geq \sum_{i=1}^{\ell} |C(v_i)| \geq \binom{\ell}{2}.$$

Thus, $\frac{1}{2}\ell(\ell-1) \leq (r-1)\ell$, implying that $\ell \leq 2r-1$. Since there are exactly opt choices for $p \in S$ and every set $R(v)$ for $v \in D_r$ must be intersected by S, we get an upper bound

$$\mathsf{opt} \cdot \ell \;\leq\; \mathsf{opt}(2r-1)$$

for the size of D_r. If we sum this over all choices of $r \in \{1, \dots, d\}$ we get an upper bound

$$\sum_{r=1}^{d} \mathsf{opt}(2r-1) \;=\; \mathsf{opt} \sum_{r=1}^{d} 2r-1 \;=\; \mathsf{opt} \cdot d^2$$

for the number of vertices with nonzero demand. This completes the proof. □

Lemma 8.4 directly implies a data reduction rule for VECTOR d-CONNECTIVITY and VECTOR CONNECTIVITY: If there are more than d^2k vertices with nonzero demand, then opt must exceed k and we can safely reject the instance.

Rule 8.2. Let (G, λ, k) be reduced with respect to Rule 8.1, with $\lambda\colon V(G) \to \{0, \dots, d\}$. If there are more than d^2k vertices of nonzero demand, then return no.

We can also derive that, in yes-instances that are reduced with respect to Rule 8.2, there are at most $k^3 + k$ vertices with nonzero demand: There can be at most k vertices of demand greater than k since those must be in the vector connectivity set. Additionally, if the size opt of the minimum vector connectivity set fulfills $\mathsf{opt} \leq k$, then there are at most $d^2\mathsf{opt} \leq k^3$ vertices of demand at most $d = k$, for a total of $k^3 + k$.

Rule 8.2 will be used in our kernelization in Section 8.6. The upper bound $k^3 + k$ on the number of demand vertices for VECTOR CONNECTIVITY will be used in the fixed-parameter algorithm with respect to k in Section 8.4.

8.4. A fixed-parameter algorithm for small vector connectivity sets

In this section we present a randomized fixed-parameter algorithm for VECTOR CONNECTIVITY parameterized by k. Recall that the data reduction rules in Section 8.3 allow us to reduce the number of vertices with nonzero demand to at most $k^3 + k$ (or to safely reject). Based on this we give a randomized algorithm building on a randomized fixed-parameter algorithm of Marx [Mar09] for intersection of linear matroids. Intuitively, each matroid will correspond to one vertex v with nonzero demand and represent the candidates for vector connectivity sets "from v's point of view". A vector connectivity set is then found as a vertex set S for which all vertices with nonzero demand agree that it is a candidate, that is, S is in the intersection of all corresponding matroids. The randomization in the algorithm comes mainly from the need to construct a representation for the required matroids. Before describing the algorithm more precisely, we need the following notation and algorithmic results related to matroids.

Matroids. A *matroid* is a tuple $(U, \mathcal{I})$ where U is a set, called the *ground set*, and $\mathcal{I}$ is a family of subsets of U, called the *independent sets* which satisfy the following three conditions.

- $\emptyset \in \mathcal{I}$.
- If $I \in \mathcal{I}$, then for every $J \subseteq I$ also $J \in \mathcal{I}$.
- If $I, J \in \mathcal{I}$, and $|I| > |J|$ then there is $u \in I \setminus J$ such that $J \cup \{u\} \in \mathcal{I}$.

A *representation* of a matroid $(U, \mathcal{I})$ is a matrix M over some field F such that U one-to-one corresponds to the columns in M and a set $I \subseteq U$ is independent, that is, $I \in \mathcal{I}$, if and only if the columns corresponding to I in M are linearly independent. A matroid is said to be *linear* if it has a representation.

The following definitions of basis, dual, and contraction are provided only for completeness. In the proof, we only use the property of contractions given in Proposition 8.3 and the fact that contractions are computable in polynomial time. A *basis* of a matroid is a maximal independent set. Note that every matroid can be defined by its set of bases. The *dual* of a matroid $\mathcal{M}$, denoted by $\mathcal{M}^*$, is the matroid which has the following set of bases

$$\{B \subseteq U \mid U \setminus B \text{ is a basis of } \mathcal{M}\}.$$

Let $X \subseteq U$. *Deleting* X from a matroid $\mathcal{M} = (U, \mathcal{I})$ results in the matroid

$$\mathcal{M} \setminus X \;:=\; (U \setminus X, \{I \subseteq U \setminus X \mid I \in \mathcal{I}\}).$$

The *contraction* of a matroid $\mathcal{M}$ by X is the matroid $\mathcal{M}/X := (\mathcal{M}^* \setminus X)^*$, that is, the dual of the deletion of X from the dual of $\mathcal{M}$. As mentioned, we only use the following property of contractions which follows from Oxley [Oxl11, Proposition 3.1.7].

Proposition 8.3. Let $\mathcal{M} = (U, \mathcal{I})$ be a matroid and $X \subseteq U$. If $\mathcal{M}$ has an independent set that contains X, then

$$\mathcal{M}/X = (U \setminus X, \{I \subseteq U \setminus X \mid I \cup X \in \mathcal{I}\}).$$

Marx [Mar09] showed that, given the representation of a matroid, the representation of the contraction or deletion by a subset of the ground set can be computed in polynomial time. For more on matroids, see Oxley [Oxl11]. Marx [Mar09] gave the following algorithm for computing independent sets in the intersection of several matroids. Our fixed-parameter tractability proof is built on this algorithm.

Theorem 8.1 (Marx [Mar09])**.** Let $\mathcal{M}_1, \ldots, \mathcal{M}_\ell$ be linear matroids given via their representations over the same field and let $k, p \in \mathbb{N}$ be nonnegative integers, and let r be the longest length in bits of one of the matroid representations. There is a computable function ϕ and a randomized algorithm with running time $\phi(k, \ell)\,\mathrm{poly}(r, p)$ with the following property. If there is a cardinality-k set that is independent in each matroid $\mathcal{M}_1, \ldots, \mathcal{M}_\ell$, then the algorithm returns such a set with probability at least $1 - 1/2^p$, and it returns no otherwise.

We are here concerned only with the following special type of matroids. Let G be a graph and $S \subseteq V(G)$ be a vertex subset. A vertex subset $T \subseteq V(G)$ is *linked* to S if there are $|T|$ pairwise vertex-disjoint paths from S to T in G. Perfect [Per68] showed that, for a fixed set $S \subseteq V(G)$, the sets T linked to S in G form a matroid, that is,

$$\mathcal{G} = (V(G), \{T \subseteq V(G) \mid T \text{ is linked to } S \text{ in } G\})$$

is a matroid. Such a matroid $\mathcal{G}$ is called *gammoid*; we say that it is the *gammoid induced by G and S*, and we call the vertices in S the *sources* of $\mathcal{G}$. From Perfect's result it also follows that every gammoid is linear. Marx [Mar09, Theorem 5.4] proved, moreover, that there is a randomized polynomial-time algorithm that, given a graph G and a vertex subset $S \subseteq V(G)$, finds a representation for the gammoid induced by G and S in the following sense.

Theorem 8.2 (Perfect [Per68] and Marx [Mar09])**.** Let G be a graph with n vertices and m edges, $S \subseteq V(G)$ a vertex subset, $\mathcal{G}$ the gammoid induced by G and S, and $p \in \mathbb{N}$ a nonnegative integer. There is a randomized algorithm with running time $\mathrm{poly}(p, n, m)$ that constructs a representation of a matroid $\mathcal{M}$ such that

- each independent set in $\mathcal{M}$ is independent in $\mathcal{G}$, and
- all independent sets in $\mathcal{G}$ are independent in $\mathcal{M}$ with probability at least $1 - 1/2^p$.

That is, the algorithm computes a representation of the gammoid $\mathcal{G}$ with high probability.

Intuition. Before making the fixed-parameter algorithm for VECTOR CONNECTIVITY fully formal, we describe the basic idea with slightly more precision. For each vertex v with nonzero demand, the feasible vector connectivity sets from v's point of view are supersets of the sets "linked" to v in the input graph G, that is, sets from which there are $\lambda(v)$ vertex-disjoint paths to v. By introducing $\lambda(v)-1$ copies of v as sources in a gammoid, the vertices that v connects to in a vector connectivity set are indeed linked to the sources, hence, are independent sets in the gammoid. To extend these sets to possible vector connectivity sets of size k we introduce $k-\lambda(v)$ universal dummy vertices as sources into the gammoid. This results in a gammoid for v in which all possible vector connectivity sets from v's point of view are independent sets. We repeat this process for all vertices with nonzero demand. Using the fact that, after exhaustively applying Rule 8.2, there are at most k^3 vertices with nonzero demand, we obtain at most k^3 gammoids. We find representations for the gammoids using the algorithm in Theorem 8.2 and then find the vector connectivity set in the intersection of all these gammoids using Marx' algorithm from Theorem 8.1.

Theorem 8.3. Let $p, n \in \mathbb{N}$. There is a randomized algorithm with running time $\phi(k)\cdot$ poly(p,n) that, given an instance of VECTOR CONNECTIVITY with n vertices,

- returns a vector connectivity set of size k with probability at least $1-1/2^p$ if the instance is yes,
- returns no otherwise.

Proof. We assume that the input instance (G,λ,k) is already reduced to at most k^3+k vertices with nonzero demand (otherwise, apply Rules 8.1 and 8.2). Denote the set of demand vertices by $D := \{v \in V(G) \mid \lambda(v) \geq 1\}$. Clearly, if the instance is yes, then there exists a solution of size *exactly* k (barring the case that $|V(G)| < k$, which would be trivial).

Algorithm description. As a first step, we guess the intersection of a solution S^* of size k with the set D; there are at most $(k^3+k)^k$ choices for $S_0 = D \cap S^*$. Note that all vertices of demand exceeding k must be contained in S_0 for S^* to be a solution (we ignore S_0 if this is not true).

For each $v \in D \setminus S_0$, we construct a matroid $\mathcal{M}_v$ over $V' = V \setminus D$ such that any set $S_1 \subseteq V'$ of size at most $k-|S_0|$ is independent in $\mathcal{M}_v$ if and only if v has $\lambda(v)$ vertex-disjoint paths to $S_0 \cup S_1$. Concretely, the matroid $\mathcal{M}_v$ with ground set V' is constructed as follows.

- Build a graph G_v by first adding to G additional $c-1$ copies of v, named $v_2,\dots,v_c$, where $c = \lambda(v)$. We use $v_1 := v$ for convenience. Second, add $r =$

$k-c$ universal vertices $w_1, \dots, w_r$, that is, each vertex $w_1, \dots, w_r$ has neighborhood $V \cup \{v_2, \dots, v_c\}$.

- Let $\mathcal{M}''_v$ denote the gammoid induced by the graph G_v and the source set $T = \{v_1, \dots, v_c, w_1, \dots, w_r\}$. Recall that the independent sets of a gammoid are exactly those subsets I of the ground set that have $|I|$ vertex-disjoint paths from the sources to I. Compute a representation for this gammoid using Theorem 8.2. We will consider the error probability parameter below.

- Obtain $\mathcal{M}'_v$ from $\mathcal{M}''_v$ by deleting $(T \cup D) \setminus S_0$. That is, $\mathcal{M}'_v$ has ground set $V' \cup S_0$ and has as independent sets precisely those independent sets of $\mathcal{M}''_v$ that are disjoint from $(T \cup D) \setminus S_0$.

- Create $\mathcal{M}_v$ from $\mathcal{M}'_v$ by contracting S_0, that is, $\mathcal{M}_v = \mathcal{M}'_v / S_0$. Note that the ground set of $\mathcal{M}_v$ is V'. Moreover, by Proposition 8.3, if S_0 is independent in $\mathcal{M}'_v$, then any I is independent in $\mathcal{M}_v$ if and only if $S_0 \cup I$ is independent in $\mathcal{M}'_v$.

Use Theorem 8.1 to search for a set I^* of size $k - |S_0|$ that is independent in each matroid $\mathcal{M}_v$ for $v \in D \setminus S_0$. (Analyzing the underlying algorithm of Theorem 8.2, it is not hard to check that we can assume that all matroids $\mathcal{M}_v$ are represented over the same field.) If a set I^* is found, then test whether $S_0 \cup I^*$ is a vector connectivity set for (G, λ, k) by appropriate polynomial-time flow computations. If this is the case, then return the solution $S_0 \cup I^*$. Otherwise, if $S_0 \cup I^*$ is not a solution or if no set I^* was found, then try the next set S_0, or answer no if no set S_0 is left to check.

It remains to prove that the algorithm is correct and to analyze its success probability Sand running time.

Correctness. Clearly, if (G, λ, k) is a no-instance, then the algorithm will always answer no as all possible solutions $S_0 \cup I^*$ are tested for feasibility.

Assume now that (G, λ, k) is yes, let S^* be a solution of size k, and consider the iteration of the algorithm in which $S_0 = D \cap S^*$. Assume that both algorithms underlying Theorems 8.1 and 8.2 do return the representation of the gammoid and the independent set if there is any. We treat the success probability below.

We prove that $S^* \setminus S_0$ is independent in each $\mathcal{M}_v$. Note that $S^* \setminus S_0 \subseteq V'$. Pick an arbitrary $v \in D \setminus S_0$. We have that there are $c = \lambda(v)$ paths from v to S^* in G that are vertex-disjoint. Thus, by giving each path a private copy of v, we get c (fully) vertex-disjoint paths in G_v, one path from each vertex in $\{v_1, \dots, v_c\}$ to S^*. We get additional $r = k - c$ paths from $\{w_1, \dots, w_r\}$ to the remaining vertices of S^* since $S^* \subseteq V' \cup S_0 \subseteq N(w_i)$. Thus, the set S^* is independent in each gammoid $\mathcal{M}''_v$.

Therefore, in each $\mathcal{M}'_v$ also S^* is independent because $S^* \cap ((T \cup D) \setminus S_0) = \emptyset$. By Proposition 8.3, this implies that in $\mathcal{M}_v$ (obtained from $\mathcal{M}'_v$ by contraction of S_0) the set $S^* \setminus S_0$ is independent and has size $k - |S_0|$. Thus, the algorithm underlying Theorem 8.1 will find *some* set I of size $k - |S_0|$ that is independent in all matroids $\mathcal{M}_v$ for $v \in D \setminus S_0$.

We claim that $I \cup S_0$ is a vector connectivity set for (G, λ, k). Let $v \in D \setminus S_0$. We know that I is independent in $\mathcal{M}_v$ and, thus, $S := I \cup S_0$ is independent in $\mathcal{M}'_v$ by Proposition 8.3. Clearly, S is also independent in $\mathcal{M}''_v$. Thus, in G_v there are $|S| = k$ paths from T to S. This entails $c = \lambda(v)$ vertex-disjoint paths from $\{v_1, \ldots, v_c\}$ to S that each contain no further vertex of T since $|T| = k$. By construction of G_v, we directly get $\lambda(v)$ paths from v to S in G that are vertex-disjoint except for overlap in v. Thus, S satisfies the demand of each $v \in D \setminus S_0$. Since $S \supseteq S_0$, we see that S satisfies all demands. Thus, the algorithm returns a feasible solution, as required.

Success probability. We now set the success probability parameters p_1, p_2 of the algorithms underlying Theorems 8.1 and 8.2 so to achieve an overall success probability of at least $1 - 1/2^p$. It is only crucial that in the iteration of the algorithm in which $S_0 = S^* \cap D$ all the at most $k^3 + k$ gammoid representations are computed successfully and that then the matroid intersection algorithm successfully finds a suitable independent set. Setting $p_1 := p_2 := q$ and $k' := k^3 + k + 1$, the probability that all these algorithms succeed is $(1 - 1/2^q)^{k'}$. We would like this probability to be at least $1 - 1/2^p$; let us find a q that satisfies this. Taking the 2^qth power and root and the $(2^p - 1)$th power and root, respectively, we get

$$\left(\left(1 - \frac{1}{2^q}\right)^{2^q}\right)^{\frac{k'}{2^q}} \geq \left(\left(1 - \frac{1}{2^p}\right)^{2^p - 1}\right)^{\frac{1}{2^p - 1}}. \tag{8.2}$$

Using the fact that for all $x > 0$ we have

$$\left(1 - \frac{1}{x+1}\right)^x < \frac{1}{e} < \left(1 - \frac{1}{x}\right)^x,$$

we see that Inequality (8.2) is implied by

$$e^{\frac{-k'}{2^q}} \geq e^{\frac{-1}{2^p - 1}}.$$

Taking the natural logarithm and rearranging terms, we get $2^q \geq (2^p - 1)k'$. Hence, it suffices to set $q = p + \log k'$. (Note that q is polynomial in p and k.)

Running time. The algorithm from Theorem 8.1 for finding a set of size k' that is independent in ℓ matroids is a fixed-parameter algorithm with respect to $k'+\ell$. We have $k' \leq k$ and $\ell \leq |D| \leq k^3 + k$ in all iterations of our algorithm and there are at most $(k^3 + k)^k$ iterations. Combining this with the fact that the error parameter q is polynomial in p and k, and with the polynomial running time for computing the matroid representation (Theorem 8.2), we obtain overall fixed-parameter running time with respect to k. □

Concerning a more precise running time bound for the algorithm in Theorem 8.3, the best upper bound we can give is in $2^{\Omega(k^4)} \cdot \Omega(pn^2)$. Hence, it does not seem worthwhile to implement the algorithm without further improvements. However, Theorem 8.3 does give a good indication that there is no strong lower bound on the running time of practical algorithms for Vector Connectivity with small k.

8.5. Kernelization outline and non-constructive argument

We now sketch an argument that shows that there is a vertex-linear kernelization for Vector d-Connectivity. Recall that, herein, we treat the maximum demand d as a problem-specific constant. This section also serves as an outline for our kernelization argument in Section 8.6.

Given an instance of Vector d-Connectivity, we proceed in three steps.

1. Identify a family $\mathcal{X}$ of vertex sets such that
   a) the union $\bigcup\mathcal{X}$ of all sets in $\mathcal{X}$ contains the optimal solution without loss of generality,
   b) for each $X \in X$, the size of $N(X)$ is upper bounded by a constant,
   c) all sets $X \in \mathcal{X}$ can be enumerated in polynomial time, and
   d) if we assume that each $X \in \mathcal{X}$ has constant size, then $|\mathcal{X}|$ is bounded from above by a linear function in k (and exponential in d).
2. For each $X \in \mathcal{X}$ such that $|X|$ is larger than some constant, replace $G[X]$ by an equivalent but smaller, constant-size gadget.
3. Remove all vertices not contained in $\bigcup_{X\in\mathcal{X}}(X \cup N(X))$ from G.

The family $\mathcal{X}$ in Step 1 is based on Cicalese, Milanič, and Rizzi's characterization of vector connectivity sets in terms of hitting sets of a special hypergraph [CMR15]. For Step 2 we use a generalized notion of a so-called protrusion replacement algorithm by Fomin et al. [Fom+13]. Finally, for Step 3, we use the so-called torso operation which removes the superfluous vertices, but keeps the connections that they provide intact. Note that these three steps indeed imply a vertex-linear kernelization for Vector d-Connectivity: After Step 3 only vertices in $\bigcup_{X\in\mathcal{X}}(X \cup N(X))$ remain, each $X \in \mathcal{X}$ has constant size after Step 2 has been carried out and each $N(X)$, $X \in \mathcal{X}$, has constant size by Condition (1b) on $\mathcal{X}$. Finally, the number of sets in $\mathcal{X}$ is upper bounded linearly in the solution size k (Condition (1d) on $\mathcal{X}$), yielding the overall vertex-linear upper bound. We now explain the three steps in more detail.

Step 1: The family $\mathcal{X}$. If the neighborhood of $X \subseteq V(G)$ of the input graph G is smaller than the largest demand of a vertex $v \in X$, then every solution must select at least one vertex in X to satisfy v (by Menger's Theorem). We now introduce notation for a set family $\mathcal{X}$ that contains all such sets but additionally restrict it to (inclusion-wise) *minimal* sets X where the demand of some vertex in X exceeds $|N(X)|$. We use the minimality condition later in Section 8.6. We then state the result of Cicalese, Milanič, and Rizzi [CMR15] using our notation.

Definition 8.1 ($\mathcal{X}(G,\lambda)$)**.** Let $G=(V,E)$ and let $\lambda\colon V\to\mathbb{N}$. The *family* $\mathcal{X}(G,\lambda)$ contains all *minimal* sets $X \subseteq V$ such that

(i) $G[X]$ is connected and
(ii) there is a vertex $v \in X$ with $\lambda(v) > |N(X)|$.

Using this notation, we adapt Proposition 1 by Cicalese, Milanič, and Rizzi [CMR15] to obtain the following.

Proposition 8.4 (Cicalese, Milanič, and Rizzi [CMR15])**.** Let G be a graph and let $\lambda\colon V(G)\to\mathbb{N}$ be a demand function. Let $\mathcal{X} := \mathcal{X}(G,\lambda)$. Then every set $S \subseteq V(G)$ is a vector connectivity set for (G,λ) if and only if it is a hitting set for $\mathcal{X}$, that is, it has a nonempty intersection with each $X \in \mathcal{X}$.

Proof. Cicalese, Milanič, and Rizzi [CMR15] proved Proposition 8.4 without the minimality restriction, allowing for a larger family, say $\mathcal{X}^+ \supseteq \mathcal{X}$. Clearly, hitting sets for $\mathcal{X}^+$ are also hitting sets for $\mathcal{X}$. Conversely, since $\mathcal{X}^+ \setminus \mathcal{X}$ contains only supersets of sets in $\mathcal{X}$, hitting sets for $\mathcal{X}$ are also hitting sets for $\mathcal{X}^+$. □

Note that for the general case of VECTOR CONNECTIVITY with unrestricted demands the size of $\mathcal{X}(G,\lambda)$ can be exponential in $|V(G)|$; for VECTOR d-CONNECTIVITY there is a straightforward upper bound of $|\mathcal{X}| = O(|V(G)|^d)$ since $|N(X)| \le d-1$. However, even for VECTOR d-CONNECTIVITY, the sets $X \in \mathcal{X}$ are not necessarily small and, thus, it is not prudent to take a hitting set approach for the kernelization.

From Proposition 8.4 it directly follows that family $\mathcal{X}$ fulfills Condition (1a), that is, $\bigcup\mathcal{X}$ contains the optimal solution without loss of generality. Furthermore, since $|N(X)| \le d$, Condition (1b) is fulfilled and we can enumerate all $X \in \mathcal{X}$ in polynomial time (Condition (1c)), by enumerating all possibilities for $N(X)$ and checking the demands in the connected components of $G - N(X)$. To upper bound $|\mathcal{X}|$ in the case that each $X \in \mathcal{X}$ has constant size (Condition (1d)), we use Rule 8.2 and the fact that yes-instances reduced with respect to Rule 8.2 contain at most d^2k vertices with nonzero demand. Using the connectivity of X and the fact that each $X \in \mathcal{X}$ contains at least one vertex with nonzero demand, we can enumerate the sets X using a search tree procedure: We start with $X = \{v\}$ for a vertex v with nonzero demand, pick a vertex $u \in N_G(X)$ and branch into two cases: to add u to X or to fix u outside of X. In the first case, X grows (recall that it has size upper bounded by a constant, say c) and in the second case, the vertices separating X from the rest of the graph grows (recall that, by Definition 8.1, there are at most d of these vertices). Thus, there are overall at most $2^{c+d}d^2k$ sets $X \in \mathcal{X}$. This bound can be improved slightly by using the two-families theorem of Bollobás, see Jukna [Juk01].

In comparison, in Section 8.6 we consider a set family $\mathcal{Y}$ that is different from $\mathcal{X}$ (but contains supersets of all those sets) and works as well in the kernelization argument. We obtain an explicit upper bound $d^2k \cdot 2^{d^3+d}$ on the number of sets in $\mathcal{Y}$, that is, the upper bound does not depend on the constant size-bound c on the sets in the family.

Step 2: Replacing $G[X]$. We now need a way to successively replace $G[X]$, where $X \in \mathcal{X}$ and $|X|$ is larger than some constant, by an equivalent smaller, constant-size gadget. Such a procedure is one of the main ingredients in so-called meta kernelization algorithms, where the aim is to prove polynomial problem kernels for all problems in general sparse graph classes that can be expressed in a certain way [Bod+16; Fom+10; Kim+15].

For the meta kernelization approach to work, however, we need an algorithm to analyze $G[X]$ in order to find out by what kind of gadget we should replace it. A basic assumption that underlies many results in meta kernelization is that the relevant $G[X]$ have bounded treewidth. In general, if $G[X]$ has bounded treewidth and $|N_G(X)|$ is upper bounded by a constant, then X is called a *protrusion*. However, on the face of it, for VECTOR d-CONNECTIVITY there is no reason to assume small treewidth and we also do not restrict the input graphs.[17] Vertex-linear kernels for problems on general graphs are much more rare than the wealth of such kernels on sparse input graphs.

Arguably, the most crucial properties of a protrusion are the small boundary and the fact that we can efficiently compute subproblems on the protrusion; in principle, we do not need bounded treewidth. Intuitively, we essentially want to know the best solution value for each choice of interaction of a global solution for the whole graph with the boundary of the protrusion. Thus, Fomin et al. [Fom+13] defined a relaxed variant of protrusions by insisting only on a small boundary and efficient algorithms for solving subproblems rather than demanding bounded treewidth. The efficient algorithm that they demand comes down to an XP-algorithm with respect to the solution size, which clearly exists for VECTOR d-CONNECTIVITY. Hence, there is an efficient algorithm for analyzing the graph $G[X]$.

To determine the constant-size gadget by which to replace $G[X]$, Fomin et al. further demand that the treated problem, VECTOR d-CONNECTIVITY in our case, has finite integer index and is monotone. Being monotone basically means that a partial solution for $G[X]$ can be extended to a solution for the whole graph, and finite integer index means that there is only a finite number of classes of graphs $G[X]$ such that the graphs in each class interact in the same way with the rest of the graph. Monotonicity can directly be checked for VECTOR d-CONNECTIVITY and finite integer index can be proved by using techniques of Bodlaender et al. [Bod+16]. Thus, combining this with the techniques of Fomin et al. [Fom+13], we can analyze the graph $G[X]$ and find an equivalent, constant-size replacement efficiently.

The resulting algorithm, however, contains as hard-wired constants the possible replacements for $G[X]$. These are guaranteed to exist and to be finitely many by the finite integer index property; however, we are not aware of a method in the literature to *compute* the possible replacements.[18] In contrast, in Section 8.6 we provide a constructive description of what constitutes gadget graphs equivalent to $G[X]$. This

[17] There might be a way to obtain bounded treewidth for $G[X]$ using the irrelevant vertices technique [RS12], which we did not explore, however.

[18] The proof for finite integer index by Bodlaender et al. [Bod+16] constructs for every instance of $G[X]$ a short string by which we can identify the equivalence class of $G[X]$ with respect to the possible interactions with the rest of the graph. However, in absence of bounded treewidth we are not aware of a method to either compute this string efficiently, or to construct an automaton that recognizes the equivalence classes from which we can read off minimal replacements for an instance in a given equivalence class.

enables us to give a single algorithm that works for all values of d based on maximum flow computations, rather than requiring for each value of d an algorithm with hard-wired representative gadgets.

Step 3: Removing vertices not in $\bigcup_X (X \cup N(X))$. In contrast to the two previous steps, this is rather simple: By Proposition 8.4 we can assume that all solution vertices are contained in $\bigcup \mathcal{X}$. Moreover, each vertex with nonzero demand is contained in $\bigcup \mathcal{X}$. That is, the only purpose of the remaining vertices outside of $\bigcup \mathcal{X}$ is to provide paths that connect vertices in $\bigcup \mathcal{X}$. Hence, we can remove each connected component C in $G - \bigcup \mathcal{X}$ from G and make its neighborhood $N(C)$ a clique in G. This is known as the torso operation, see Marx, O'Sullivan, and Razgon [MOR13], for example. We give a formal correctness proof in Section 8.6.

In summary, the above shows that there is a vertex-linear kernelization algorithm for VECTOR d-CONNECTIVITY. However, the proof is not constructive, and the upper bound on the kernel size depends exponentially on the (unknown) size of the replacement gadgets for $G[X]$. In Section 8.6 we provide a proof based on the same outline that circumvents both these drawbacks.

8.6. Vertex-linear kernelization for constant demand

In this section we give a vertex-linear kernelization for VECTOR d-CONNECTIVITY parameterized by k. Recall that, herein, we treat the upper bound d on the demand of each vertex as a problem-specific constant. We will leverage Rules 8.1 and 8.2 throughout this section. Hence, we will, sometimes tacitly, assume that all instances of VECTOR d-CONNECTIVITY are reduced with respect to these rules.

The kernelization follows a similar outline as the one in Section 8.5, but differs in two key points. First, we do not use the family $\mathcal{X}(G, \lambda)$ from Definition 8.1 directly; in fact, we not even materialize any set in $\mathcal{X}$ but use them only for analysis. We instead use a family $\mathcal{Y} = \mathcal{Y}(G, \lambda)$ which contains larger sets. The advantage is that the cardinality of $\mathcal{Y}$ is smaller, leading to a upper bound on $|\mathcal{Y}|$ that does not depend exponentially on the maximum size of the sets in $\mathcal{Y}$ as was the case for the family $\mathcal{X}$.

Second, rather than relying on the finite integer index property of VECTOR d-CONNECTIVITY in order to replace the $G[Y]$, $Y \in \mathcal{Y}$, by constant-size gadgets, we characterize their interaction with the rest of the graph directly, which yields an algorithm that can analyze the $G[Y]$ and find a constant-size replacement gadget. The characterization can also be seen as a direct way to prove finite integer index for VECTOR d-CONNECTIVITY; of course, we get the stronger result that includes polynomial-time checkable equivalence classes.

This section comprises three parts. In Section 8.6.1, we describe the family $\mathcal{Y}$ and give an upper bound on its cardinality. Then, in Section 8.6.2 we give a characterization of the $G[Y]$ and a reduction rule that replaces them by constant-size gadgets. Finally, in Section 8.6.3, we complete the kernelization algorithm and prove that it yields a vertex-linear problem kernel for VECTOR d-CONNECTIVITY with respect to the solution size k.

8.6.1. The family $\mathcal{Y}$

As the first step towards the kernelization, we prove that instances (G,λ,k) of VECTOR d-CONNECTIVITY that are reduced with respect to Rule 8.1 have the property that every set $X \in \mathcal{X}(G,\lambda)$ contains at most d^3 vertices with nonzero demand. For ease of presentation we define $D(G,\lambda) := \{v \in V(G) \mid \lambda(v) \geq 1\}$, and use the shorthand $D = D(G,\lambda)$ whenever G and λ are clear from context.

Lemma 8.5. For all $X \in \mathcal{X}$ we have $|X \cap D| \leq (d-1)d^2 \leq d^3$.

Proof. Recall from Section 8.3 the definition of $C(v)$ as the unique closest minimum $v, D'(v)$-separator, where $D'(v) = \{u \in V \setminus \{v\} \mid \lambda(u) \geq \lambda(v)\}$, and the definition of $R(v)$ as the connected component of v in $G - C(v)$. Fix some $r \in \{1,\ldots,d\}$, define $D_r = \{v \in D \mid \lambda(v) = r\}$, and consider the relation of $R(v)$ with X for $v \in D_r \cap X$. Note that $D_r \setminus \{v\} \subseteq D'(v)$.

If $R(v) \subsetneq X$, then this would contradict minimality of X: By reducedness with respect to Rule 8.1, we have $|C(v)| < \lambda(v)$ or else the rule would apply to v. But then $R(v)$ with $N(R(v)) = C(v)$ fulfills the conditions for being in $\mathcal{X}$, except possibly for minimality. This would prevent $X \supsetneq R(v)$ from being included in $\mathcal{X}$. Else, if $R(v) = X$, then no further vertex of D_r is in X, since $C(v)$ is also a $v, D_r \setminus \{v\}$-separator as $D_r \setminus \{v\} \subseteq D'(v)$. In this case we get, $|X \cap D_r| = 1$.

In the remaining case, there is no $v \in D_r \cap X$ with $R(v) \subseteq X$. It follows that for all $v \in D_r \cap X$ we have $R(v) \cap X \neq \emptyset$ but $R(v) \nsubseteq X$. This implies $R(v) \cap N(X) \neq \emptyset$ since both $G[X]$ and $G[R(v)]$ are connected. We will use this fact to upper bound the number of vertices with demand r in X. Let $w \in N(X)$ and let $W \subseteq D_r \cap X$ contain those vertices v of demand r whose set $R(v)$ contains w; each $R(v)$ must contain at least one vertex in $N(X)$. Thus, for any two vertices $u, v \in W$ we find that their sets $R(u)$ and $R(v)$ have a nonempty intersection, since they share at least w. We can now repeat the same analysis as used in the proof of Lemma 8.4 to get that

$|W| \leq 2r - 1$. Over all choices of w we get an upper bound of $(d-1)(2r-1)$ vertices of demand r in X.

We showed that for each choice of $r \in \{1, \dots, d\}$ we have at most $(d-1)(2r-1)$ vertices of demand r in X. Summing over all $r \in \{1, \dots, d\}$ this yields an upper bound of $(d-1)d^2 \leq d^3$ for $|X \cap D|$, as claimed. □

To arrive at our kernelization we will later establish a reduction rule that shrinks connected subgraphs with small boundary and bounded number of demand vertices to constant size. This is akin to black box protrusion-based reduction rules, especially to Fomin et al.'s approach [Fom+13], but we give an explicit algorithm that comes down to elementary two-way flow computations. To get an explicit (linear) upper bound for the number of subproblems, we introduce a new family $\mathcal{Y}$ with larger but (as we will see) fewer sets, and apply the reduction process to graphs $G[Y]$ with $Y \in \mathcal{Y}$ instead.

Definition 8.2 ($\mathcal{Y}(G, \lambda, d)$)**.** Let $G = (V, E)$, let $d \in \mathbb{N}$, and let $\lambda \colon V \to \{0, \dots, d\}$. The family $\mathcal{Y}(G, \lambda, d)$ contains all sets $Y \subseteq V$ with the following properties.

(i) $G[Y]$ is connected.
(ii) $|Y \cap D| \leq d^3$, that is, Y contains at most d^3 vertices v with nonzero demand $\lambda(v)$.
(iii) $|N(Y)| \leq d$, that is, Y has at most d neighbors.
(iv) There is a vertex $v \in Y \cap D$, that is, $\lambda(v) \geq 1$, such that $N(Y)$ is the unique closest minimum $v, D \setminus Y$-separator.

For an instance (G, λ, d) of VECTOR d-CONNECTIVITY, we relate $\mathcal{X} = \mathcal{X}(G, \lambda)$ and $\mathcal{Y} = \mathcal{Y}(G, \lambda, d)$ by proving that every set $X \in \mathcal{X}$ is contained in at least one $Y \in \mathcal{Y}$. Intuitively, this proves that all "interesting" parts of the instance are contained in subgraphs $G[Y]$.

Lemma 8.6. Let $G = (V, E)$ a graph, $d \in \mathbb{N}$, and $\lambda \colon V \to \{0, \dots, d\}$. Let $\mathcal{X} := \mathcal{X}(G, \lambda)$ and $\mathcal{Y} := \mathcal{Y}(G, \lambda, d)$. Then for all $X \in \mathcal{X}$ there exists $Y \in \mathcal{Y}$ with $X \subseteq Y$.

Proof. Let $X \in \mathcal{X}$ and pick $v \in X$ with $\lambda(v) > |N(X)|$. Let $D_1 = D \setminus X$, that is, those vertices with nonzero demand that are not in X. Now, let $Z \subseteq V \setminus \{v\}$ the minimum v, D_1-separator that is closest to v, and let Y be the connected component of v in $G - Z$; thus $Z = N(Y)$. We claim that $X \subseteq Y$ and $Y \in \mathcal{Y}$.

First, assume for contradiction that $X \nsubseteq Y$. We use the submodularity of $f \colon 2^V \to \mathbb{N} \colon U \mapsto |N(U)|$, which implies

$$f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y). \tag{8.3}$$

Note that $D_1 \cap (X \cup Y) = (D_1 \cap X) \cup (D_1 \cap Y) = \emptyset$ and that $v \in X \cup Y$. It follows that $N(X \cup Y)$ is also a v, D_1-separator and, using that Z is minimum, we get that $f(X \cup Y) = |N(X \cup Y)| \geq |Z| = |N(Y)| = f(Y)$. Plugging this into Inequality (8.3) we obtain $f(X) \geq f(X \cap Y)$. Note that $v \in X \cap Y$ and $D_1 \cap (X \cap Y) = \emptyset$, implying that $N(X \cap Y)$ is also a v, D_1-separator of size at most $|N(X)| = f(X)$. From $X \not\subseteq Y$ we get $X \cap Y \subsetneq X$. But then $X \cap Y$ is a proper subset of X containing v and having $|N(X \cap Y)| \leq |N(X)| < \lambda(v)$. It follows that the connected component of v in $G[X \cap Y]$ also has neighborhood size (in G) less than $\lambda(v)$. This contradicts the fact that X is a minimal set fulfilling the properties of Definition 8.1, which is required for $X \in \mathcal{X}$. We conclude that, indeed, $X \subseteq Y$.

Second, let us check that Y fulfills the requirements for $Y \in \mathcal{Y} = \mathcal{Y}(G, \lambda, d)$ (as in Definition 8.2). Note that Z separates v from all vertices with nonzero demand that are not in X, since $D_1 = D \setminus X$. Thus, every vertex with nonzero demand in Y is also contained in X, that is, $D \cap Y \subseteq D \cap X$, which upper bounds their number by d^3 using Lemma 8.5. It also follows that $D \setminus X = D \setminus Y$, since $X \subseteq Y$ implies $D \cap X \subseteq D \cap Y$. Furthermore, $G[Y]$ is connected and $N(Y)$ is a minimum $v, D \setminus Y$-separator that is closest to v; note that $D_1 = D \setminus X = D \setminus Y$. Finally, since $N(X)$ is also a v, D_1-separator and $N(Y)$ is minimum, we conclude that $|N(Y)| \leq |N(X)| < \lambda(v) \leq d$. Thus, indeed, $Y \in \mathcal{Y}$ as claimed. □

We prove that the number of sets $Y \in \mathcal{Y}$ is linear in k for every fixed d. Thus, by later shrinking the size of sets in $\mathcal{Y}$ to some constant we get $O(k)$ vertices in total over sets $Y \in \mathcal{Y}$.

Lemma 8.7. Let (G, λ, k) an instance of VECTOR d-CONNECTIVITY with $\lambda \colon V(G) \to \{0, \ldots, d\}$ and let $\mathcal{Y} := \mathcal{Y}(G, \lambda, d)$. Then $|\mathcal{Y}| \leq d^2 k \cdot 2^{d^3+d}$.

Proof. We prove the lemma by giving a branching process that enumerates all sets $Y \in \mathcal{Y}$ within the leaves of its branching tree and by showing that the tree has at most $d^2 k \cdot 2^{d^3+d}$ leaves in which sets Y are found. Given $G = (V, E)$, $\lambda \colon V \to \{0, \ldots, d\}$, and $k \in \mathbb{N}$, the branching process works as follows. (Recall $D = \{v \in V \mid \lambda(v) \geq 1\}$.)

1. As a first step, *branch* on choosing one vertex $v \in D$. Recall that an instance reduced with respect to Rule 8.2 has $|D| \leq d^2 k$. Hence, this branching step incurs a factor of $|D| \leq d^2 k$ to the number of leaves and creates one node for each choice.

2. Maintain disjoint sets $D_0, D_1 \subseteq D$, starting with $D_0 := \{v\}$ and $D_1 := \emptyset$, and the minimum v, D_1-separator Z that is closest to v; initially $Z = \emptyset$. Throughout, use Y to refer to the connected component of v in $G - Z$.

3. All further *branchings* work as follows: Pick an arbitrary vertex $p \in D \setminus (D_0 \cup D_1)$ that is reachable from v in $G - Z$. *Branch* on either adding this vertex to D_0 or to D_1, creating a child node for each choice. In the branch where p is added to D_1 update the minimum v, D_1-separator Z closest to v and update the connected component Y of v in $G - Z$.

4. Terminate a branch if one of the following three *termination conditions* occurs.
   a) The size of D_0 exceeds d^3.
   b) The size of Z exceeds d.
   c) No vertex of $D \setminus (D_0 \cup D_1)$ is reachable from v in $G - Z$.

We now analyze this process. First, we show that every set of $\mathcal{Y}$ occurs as the set Y in some leaf node of the process, that is, a node to which a termination condition applies. Second, we show that the number of such leaf nodes is upper bounded by $|D| \cdot 2^{d^3+d} \le d^2 k \cdot 2^{d^3+d}$.

Each set of $\mathcal{Y}$ occurs in some leaf. We show that *every* set $Y^* \in \mathcal{Y}$ is found as the set Y of at least one leaf of the branching tree. To this end, fix an arbitrary set $Y^* \in \mathcal{Y}$ and let $Z^* := N_G(Y^*)$. Furthermore, let $D_0^* := Y^* \cap D$. Let $v \in Y^* \cap D$ such that $Z^* = N(Y^*)$ is the unique minimum, closest $v, D \setminus Y^*$-separator. In the first branching step the process can clearly pick v for its choice of vertex in D; in this case it continues with $D_0 = \{v\}$, $D_1 = \emptyset$, $Z = \emptyset$, and Y is the connected component of v in G. Consider nodes in the branching process with current sets Y, D_0, D_1, and $Z = N(Y)$ fulfilling the *requirements* that

- $Y \supseteq Y^*$ and
- $D_0 \subseteq D_0^*$ and $D_1 \cap D_0^* = \emptyset$.

Among such nodes pick one that is either a leaf or such that neither child node fulfills the requirements. Clearly, the node with $D_0 = \{v\}$, $D_1 = \emptyset$, $Z = \emptyset$, and Y equal to the component of v in G fulfills the requirements, so we can indeed always find such a node by starting at this one and following child nodes fulfilling the requirements until reaching a leaf or until both child nodes do not fulfill the requirements. We now consider these two cases individually.

Case 1: Leaf node fulfilling the requirements. If the chosen node is a leaf then one of the three termination conditions must hold. Clearly, we cannot have $|D_0| > d^3$ since that would imply $|D_0^*| > d^3$, violating the definition of $\mathcal{Y}$ and the sets therein.

Similarly, we cannot have $|Z| > d$: In this regard, note that $D_1 \cap D_0^* = \emptyset$ implies that $D_1 \subseteq D \setminus D_0^* = D \setminus (D \cap Y^*) = D \setminus Y^*$. It follows directly that Z^*, which separates v from $D \setminus Y^*$, also separates v from D_1. But then the size of Z^* is an upper bound for the minimum separator size for v, D_1-separators and $|Z| > d$ would imply $|Z^*| \geq |Z| > d$, again violating the definition of $\mathcal{Y}$.

Thus, in case of a leaf node the only remaining option is that no further vertex of $D \setminus (D_0 \cup D_1)$ is reachable from v in $G - Z = G - N(Y)$. Recall that Z is the minimum closest v, D_1-separator. By termination Z also separates v from $D \setminus (D_0 \cup D_1)$, making it a closest $v, D \setminus D_0$-separator. Since $D_0 \subseteq D_0^*$, it follows that $D \setminus Y^* = D \setminus D_0^* \subseteq D \setminus D_0$. Thus, the size of the minimum $v, D \setminus Y^*$-separator Z^* is upper bounded by $|Z|$ since Z is also a $v, D \setminus Y^*$-separator. Recall that we have $Y \supseteq Y^*$, which implies that $Z^* = N(Y^*)$ also separates v from $Z = N(Y)$. Now, because Z is closest to v, it is the unique v, Z-separator of size at most $|Z|$, which implies $Z = Z^*$ since we just derived that $|Z^*| \leq |Z|$. Thus, $Y = Y^*$ since both are identified as the connected component of v in $G - Z = G - Z^*$. We conclude that Y^* is equal to Y in the chosen node if it is a leaf.

Case 2: Internal node fulfilling the requirements. Now, let us consider the case that the chosen node is not a leaf of the branching tree. We want to check that at least one possible branch must lead us to a child node that also fulfills our restrictions; this would contradict our choice of node that is either a leaf or such that neither child node fulfills the requirements, implying that we necessarily pick a leaf node. Thus, for each $Y^* \in \mathcal{Y}$ there is a leaf of the branching tree with $Y = Y^*$. For clarity, in the following discussion we will use Y, D_0, etc. for the current node and Y', D_0', etc. for the considered child node in the branching tree.

Since we are not in a leaf in this case, the process chooses an arbitrary vertex $p \in D \setminus (D_0 \cup D_1)$ that is reachable from v in $G - Z$ to branch on. If $p \in D_0^*$, then the child node corresponding to adding p to D_0 has $D_0' = D_0 \cup \{p\} \subseteq D_0^*$ and $D_1' = D_1$. Thus, $Z' = Z$ and, hence, $Y' = Y \supseteq Y^*$. Thus, the child node fulfills all requirements; a contradiction.

Otherwise, if $p \notin D_0^*$, then $p \in D \setminus D_0^* = D \setminus Y^*$. Thus, the child node corresponding to adding p to D_1 has $D_0' = D_0 \subseteq D_0^*$ and $D_1' = D_1 \cup \{p\}$, implying that $D_1' \cap D_0^* = D_1 \cap D_0^* = \emptyset$. For the desired contradiction it remains to prove that $Y' \supseteq Y^*$ since we assumed that neither child fulfills the requirements.

Assume that $Y^* \not\subseteq Y'$. Thus, $Y^* \cap Y' \subsetneq Y^*$. We again use the submodularity of the function $f\colon 2^V \to \mathbb{N}\colon U \mapsto |N(U)|$ and get

$$f(Y^*) + f(Y') \geq f(Y^* \cap Y') + f(Y^* \cup Y'). \tag{8.4}$$

Since $v \in Y^* \cap Y' \subsetneq Y^*$ and $N(Y^*)$ is closest to v it follows that $f(Y^* \cap Y') = |N(Y^* \cap Y')| > |N(Y^*)| = f(Y^*)$, by Proposition 8.2. Plugging this into Inequality (8.4) yields $f(Y^* \cup Y') < f(Y')$; let us check that this violates the fact that $Z' = N(Y')$ is a minimum v, D'_1-separator: We have $v \in Y^* \cup Y'$ and

$$D'_1 \cap (Y^* \cup Y') \;=\; \underbrace{(D'_1 \cap Y^*)}_{\subseteq D} \cup \underbrace{(D'_1 \cap Y')}_{=\emptyset} \;=\; (D'_1 \cap Y^*) \cap D \;=\; D'_1 \cap D^*_0 \;=\; \emptyset.$$

Thus, indeed, we find that $N(Y^* \cup Y')$ is a v, D'_1-separator and we know from $f(Y^* \cup Y') < f(Y')$ that it is smaller than the assumed minimum v, D'_1-separator $Z' = N(Y')$; a contradiction.

Hence, by our choice of node that fulfills the requirements and is a leaf or neither child fulfills the requirements, we must obtain a leaf with set Y equal to Y^*.

Number of leaves containing some $Y^* \in \mathcal{Y}$. We have seen that every set $Y^* \in \mathcal{Y}$ is equal to the set Y of some leaf node fulfilling certain requirements. Furthermore, leaves with $|D_0| > d^3$ or $|Z| > d$ were shown not to correspond to any $Y \in \mathcal{Y}$. We will now analyze the number of leaf nodes with $|D_0| \le d^3$ and $|Z| \le d$.

Crucially, we show that each branching increases $|D_0| + |Z|$. For adding p to D_0 this is obvious, for adding p to D_1 we prove it next. Concretely, we prove that adding p to D_1 increases the minimum size of v, D_1-separators by at least one. (This is essentially folklore but we provide a proof for completeness.) Use $D'_1 = D_1 \cup \{p\}$ and let Z and Z' denote minimum v, D_1- and v, D'_1-separators closest to v; use Y and Y' for the components of v in $G - Z$ and $G - Z'$, respectively. We use again the submodular function $f \colon 2^V \to \mathbb{N} \colon U \to |N_G(U)|$ and obtain

$$f(Y) + f(Y') \;\ge\; f(Y \cup Y') + f(Y \cap Y'). \tag{8.5}$$

Both Z and Z' separate v from D_1. Thus, $D_1 \cap (Y \cup Y') = \emptyset$ and $N(Y \cup Y')$ is also a v, D_1-separator since it creates the component $Y \cup Y'$ for v in $G - N(Y \cup Y')$. Since Z is a minimum v, D_1-separator, we must have $f(Y \cup Y') = |N(Y \cup Y')| \ge |Z| = f(Y)$. Plugging this into Inequality (8.5) yields $f(Y \cap Y') \le f(Y')$. If $f(Y') = |Z'| \le |Z| = f(Y)$, that is, if adding p to D_1 does not increase the minimum size of v, D_1-separators,[19] then $f(Y \cap Y') \le |Z|$ and $N(Y \cap Y')$ is also a v, D_1-separator of size at most $|Z|$. However, as $p \notin Y'$, since Z' separates v from $D'_1 = D_1 \cup \{p\}$, the set $Y \cap Y'$ is a strict subset of Y; this is a contradiction to Z being closest to v. As a consequence,

[19]These values coincide with $f(Y)$ and $f(Y')$ since we chose $Z = N(Y)$ and $Z' = N(Y')$ as minimum v, D_1- and v, D'_1-separators.

the size of closest, minimum v, D_1-separators increases whenever we branch into adding a so far reachable vertex to D_1.

Thus, every child node has $|D_0'| + |Z'| \geq |D_0| + |Z| + 1$ and, clearly, neither value decreases when branching. Thus, the process can only reach leaf nodes with $|D_0| \leq d^3$ and $|Z| \leq d$ via internal nodes where $|D_0| + |Z| \leq d^3 + d$. It follows that the number of such leaf nodes is upper bounded by $|D| \cdot 2^{d^3+d} \leq d^2 k \cdot 2^{d^3+d}$. (Once $|D_0| \geq d^3$ or $|Z| \geq d$ at most one child node can lead to such a leaf, since the other child violates the restriction on $|D_0|$ or $|Z|$; these branches are not counted.) Thus, $|\mathcal{Y}| \leq d^2 k \cdot 2^{d^3+d}$, as claimed. □

8.6.2. Reducing the size of sets in $\mathcal{Y}$

In this section, we explain and prove how to reduce the size of sets $Y \in \mathcal{Y}$ through modifications on the graph G. At a high level, this will be achieved by replacing subgraphs $G[Y]$ by "equivalent" subgraphs of bounded size. When this is done, we know that the total number of vertices in sets $Y \in \mathcal{Y}$ is $O(k)$. Since this part is somewhat technical and long, let us try to illustrate it first.

Illustration of the approach

Consider a set $Y \in \mathcal{Y}$ and its (small) neighborhood $Z := N_G(Y)$. Think of deciding whether (G, λ, k) is yes as a game between two players, Alice and Bob. Alice sees only $G[Y \cup Z]$ and wants to satisfy the demands of all vertices in Y, and Bob sees only $G - Y$ and wants to satisfy the demands of the vertices in $V \setminus Y$. To achieve a small solution the players must cooperate and exchange information about paths between vertices in Z, or between Z and vertices of a partial solution; paths that they can *provide* or that they *require*.

Since our goal is to simplify $G[Y]$, all notation is given using Alice's perspective. Crucially, we know that there are only constantly many vertices with nonzero demand in Y, which can be seen to imply that the intersection of optimal solutions with Y is upper bounded (Lemma 8.8 below). Thus, Alice can try all partial solutions $S_Y \subseteq Y$ of bounded size and determine what *facilities* each S_Y provides for Bob, and what *requirements* she has on Bob to satisfy her demand vertices using S_Y and further paths through $G - Y$.

Let us be slightly more concrete about facilities and requirements, before making them fully formal. If we fix some partial solution $S_Y \subseteq Y$, then Alice can offer (as facilities) to Bob to connect some subsets of Z to S_Y by disjoint paths in $G[Y \cup Z]$, and to, additionally, provide paths connecting certain sets of vertices in Z. There can be a large number of such options for each S_Y. Similarly, to fulfill demands in Y, Alice may need (as requirements) that Bob can provide paths from certain subsets of Z to a solution and, additionally, paths connecting sets of vertices in Z. (Note that there is some asymmetry here since Bob's part is too large to fully analyze, but this will be no problem.) Fortunately, while there may be many choices for S_Y, and many facilities and requirements relative to a single S_Y, it will turn out that the overall number of things to keep track of is bounded (in d); this will be called the *signature* of $G[Y \cup Z]$. Ultimately, we will be able to replace $G[Y \cup Z]$ by a bounded-size graph with the same signature.

Formal proof

We now make our approach formal. For convenience, let us introduce the following notation. Below we deal with sets of paths that do not necessarily share the endpoint v but that are still pairwise vertex-disjoint except for possibly sharing v as an endpoint. Call such a set of paths *v-independent.* For a graph G, a vertex v, an integer i, and two vertex subsets $A, B \subseteq V(G)$ we define a (v, i, A, B)*-constrained path packing* as a set of $i + |A|$ v-independent paths from $A \cup \{v\}$ to B in G. Herein we explicitly allow $v \notin V(G)$ if $i = 0$. If $i = 0$, then we simplify the notation and speak of (A, B)-constrained path packings instead. Note that, regardless of whether $v \in V(G)$, the paths saturate each vertex in A. Furthermore, we tacitly assume that, if $A \cap B \neq \emptyset$, then the paths corresponding to $A \cap B$ in the packing are of length zero, that is, they each comprise a single vertex.

Let us now begin with the definition of signatures. In the following, let G and λ represent the graph and demand function of an instance of VECTOR d-CONNECTIVITY. To reduce the amount of notation we also fix a set $Y \subseteq V(G)$ such that $|Y \cap D| \leq d^3$ and $|N(Y)| \leq d$. (In the kernelization procedure, the role of Y will be assumed by some set in $\mathcal{Y}$.) We denote the neighborhood $N(Y)$ by Z.

We will first take care of the requirements that Alice has. The facilities will be treated later. As mentioned above, a requirement comprises several sets of paths outside of $G[Y]$ one set of which needs to be provided by Bob (and his part of the solution) in order to satisfy the demand of a vertex $v \in D \cap Y$. To this end, we define a *satisfying connector.* In the following $Y \uplus Z$ denotes the disjoint union of the sets Y and Z.

Definition 8.3 (Satisfying connector)**.** Let H be a graph on vertex set $Y \uplus Z$, let $v \in Y$, let v have positive demand d_v, and let $S_Y \subseteq Y$ be a partial solution. A tuple (A, B, C) with $A, B, C \subseteq Z$, pairwise disjoint, is a *satisfying connector for v with respect to S_Y*

in H if either $v \in S_Y$ or there is a $(v, d_v, A, B \cup S_Y)$-constrained path packing in $H - C$. The set of all satisfying connectors for v with respect to the partial solution S_Y is denoted by $\mathsf{sat}(H, Z, d_v, S_Y, v)$.

Now we can formally define the requirement of Alice. Intuitively, Bob should provide paths that "hit" each set of satisfying connectors induced by a vertex with positive demand.

Definition 8.4 (Requirement)**.** Let H be a graph on vertex set $Y \uplus Z$, let $\lambda\colon Y \to \{0, \dots, d\}$ be a demand function on Y, and let $S_Y \subseteq Y$ be a partial solution. The *requirement* $\mathsf{req}(H, Z, \lambda, S_Y)$ is the collection

$$\{\mathsf{sat}(H, Z, \lambda(v), S_Y, v) \mid v \in D(H, \lambda) \cap Y\}.$$

After the definition of signatures we will prove that the demand of each vertex in $D(\lambda) \cap Y$ can be met if and only if the requirement of Alice is met by a suitable path packing provided by Bob.

Note that, in order to be able to replace $G[Y]$ by a different graph, we need to know which requirements it imposes for every relevant choice of the partial solution S_Y. Since we are aiming for a polynomial-time kernelization, we also need to be able to compute them in polynomial time. For this, we first upper bound the size of S_Y.

Lemma 8.8. For each vector connectivity set S of (G, λ), there is a vector connectivity set S' such that $|S'| \le |S|$ and such that $|Y \cap S'| \le d^3 + d$.

Proof. Assume that $|Y \cap S| \ge d^3 + d$. Remove all vertices in $(Y \cup Z) \cap S$ from S and add all vertices in $D \cap Y$ as well as all vertices in Z. Denote the resulting vertex set by S'. Recall that $|D \cap Y| \le d^3$ by definition of Y. Hence, $|Y \cap S'| \le |D \cap Y| + |N_G(Y)| \le d^3 + d$ and, thus, $|S'| \le |S|$. Clearly, S' satisfies all demands of vertices in Y and Z since $(D \cap Y) \cup Z \subseteq S'$. For vertices $v \in D \setminus (Y \cup Z)$, note that at most $|Z|$ paths used for reaching S from v can traverse Z, and all of those can be shortened to end in $Z \subseteq S'$; all other paths avoid Z and thereby $Y \cup Z$, implying that they still end in vertices of $S \setminus (Y \cup Z) = S' \setminus (Y \cup Z)$. □

We are almost ready to compute the requirements; crucially, we need to check whether there are suitable (v, d, A, B)-constrained path packings in polynomial time.

Lemma 8.9. Let G be a graph, $v \in V(G)$, $d \in \mathbb{N}$, and $A, B \subseteq V(G)$. It is possible to check in polynomial time whether there is a (v, d, A, B)-constrained path packing in G.

Proof sketch. We reduce the check to computing a maximum flow in a modified graph as follows. First, remove each vertex in $A \cap B$ from the graph—we can assume that they represent paths of length zero in the desired path packing. Then, add $d-1$ copies of v to G, each adjacent to all neighbors of v. Then, attach a new vertex s to each vertex in A and the copies of v, and attach a new vertex t to each vertex in B. It is not hard to check that there are $d + |A \setminus B|$ internally vertex-disjoint paths from s to t in the modified graph if and only if there is a (v, d, A, B)-constrained path packing in the original graph.

Checking whether there are enough internally vertex-disjoint s-t paths can be done in polynomial time using Proposition 8.1. □

Lemma 8.10. Let $Y \in \mathcal{Y}$, $Z = N(Y)$, and $H = G[Y \cup Z]$. The collection

$$\{\mathsf{req}(H, Z, \lambda|_Y, S_Y) \mid S_Y \subseteq Y \wedge |S_Y| \leq d^3 + d\}$$

is computable in polynomial time. Herein, $\lambda|_Y$ denotes λ restricted to Y.

Proof. By enumerating all $S_Y \subseteq Y$ of size at most $d^3 + d$ in $n^{O(d^3)}$ time, that is, polynomial time, the task reduces to computing $\mathsf{req}(H, Z, \lambda|_Y, S_Y)$ for a given S_Y. To do this, we compute the set of satisfying connectors for each vertex v in $D \cap Y$. This in turn we do by simply iterating over all tuples (A, B, C) with $A, B, C \subseteq Z$, pairwise disjoint, and we check whether it is contained in $\mathsf{sat}(H, Z, \lambda(v), S_Y, v)$. Note that $|Z| \leq d$. Hence, there are at most $2^{O(d)} \in O(1)$ different tuples to check. Thus, it remains to check whether in $H - C$ there is a $(v, \lambda(v), A, B \cup S_Y)$-constrained path packing. This can be done using Lemma 8.9. □

So far we have only talked about the requirements that Alice has on Bob. Now let us come to the facilities that Alice provides. As mentioned, a facility models the sets of paths inside of $G[Y]$ that Alice, using her part of the solution, provides to Bob so to satisfy the demand of each vertex in $D \setminus Y$. Let us first focus on vertices in $D \setminus (Y \cup Z)$.

Definition 8.5 (Provided connector)**.** Let H be a graph on vertex set $Y \uplus Z$, and let $S_Y \subseteq Y$ be a partial solution. A tuple (A, B, C) with $A, B, C \subseteq Z$, pairwise disjoint, is a *provided connector of* S_Y *in* H if there is a $(A, B \cup S_Y)$-constrained path packing in $H - C$.

We prove below that all demands of vertices in $D \setminus (Y \cup Z)$ can be met if and only if Alice provides suitable path packings to Bob. We take special care of vertices in $D \cap Z$, as they may need multiple paths into $G[Y]$.

Definition 8.6 (Provided special connector)**.** Let H be a graph on vertex set $Y \uplus Z$ and let $S_Y \subseteq Y$. A tuple (z, i, A, B, C) with $z \in Z$, $A, B, C \subseteq Z \setminus \{z\}$, pairwise disjoint, and $i \in \{0, \dots, d\}$ is a *provided special connector of* S_Y *in* H if there is a $(z, i, A, B \cup S_Y)$-constrained path packing in $H - C$.

We are now ready to give a formal definition of the facilities provided by Alice.

Definition 8.7 (Facility)**.** Let H be a graph on vertex set $Y \uplus Z$, and let $S_Y \subseteq Y$ be a partial solution. The *facility* $\mathsf{fac}(H, Z, S_Y)$ *of* S_Y *in* H is the set of all provided connectors and provided special connectors of S_Y.

Similarly to requirements, we need an efficient algorithm for computing the facilities; this basically follows from Lemma 8.9.

Lemma 8.11. Let $Y \in \mathcal{Y}$, $Z = N(Y)$, and $H = G[Y \cup Z]$. The collection

$$\{\mathsf{fac}(H, Z, S_Y) \mid S_Y \subseteq Y \wedge |S_Y| \leq d^3 + d\}$$

is computable in polynomial time.

Proof. We first enumerate all $S_Y \subseteq Y$ with $|S_Y| \leq d^3 + d$ in $n^{O(d^3)}$ time and, for each such S_Y, we compute all provided connectors and provided special connectors. This is done by iterating over all possible tuples (A, B, C) and (z, i, A, B, C) (there are at most $d^2 \cdot 2^{O(d)} \in O(1)$ of them) and checking whether they indeed are provided (special) connectors. This is done using Lemma 8.9. □

Now we can precisely define the signature of $G[Y \cup Z]$ that we mentioned earlier.

Definition 8.8 (Signature)**.** Let H be a graph on vertex set $Y \uplus Z$, let λ be a demand function on Y, and let $S_Y \subseteq Y$ be a partial solution. The *signature* of H is the set

$$\mathsf{sig}(H, Z, \lambda) := \{(|S_Y|, \mathsf{req}(H, Z, \lambda, S_Y), \mathsf{fac}(H, Z, S_Y))\},$$

where S_Y ranges over all $S_Y \subseteq Y$ such that $|S_Y| \leq d^3 + d$.

We show below that we can safely replace $G_Z[Y \cup Z]$ with an arbitrary graph $G'[Y' \cup Z]$ that has the same signature. Let us now prove that there is a graph $G'[Y' \cup Z]$ of constant size with the same signature.

Lemma 8.12. There is a polynomial-time algorithm that receives $Y \in \mathcal{Y}$, $Z = N(Y)$, $G[Y \cup Z]$, and $\lambda|_Y$ as input and computes a graph G' on vertex set $Y' \cup Z$ such that $G'[Z] = G[Z]$ and a demand function $\lambda' \colon Y' \to \{0, \dots, d\}$ such that $\mathsf{sig}(G[Y \cup Z], Z, \lambda|_Y) = \mathsf{sig}(G', Z, \lambda')$ and $|D(G', \lambda')| \leq d^3$. Moreover, the resulting G' and λ' are encoded using at most $\phi(d)$ bits and G' has at most $\phi(d)$ vertices, for some computable function ϕ depending only on d.

Proof. The algorithm is as follows. First, compute $\mathsf{sig}(G[Y \cup Z], Z, \lambda|_Y)$ in polynomial time using Lemmas 8.10 and 8.11. Then, generate all graphs G' with $G'[Z] = G[Z]$ in the order of increasing number of vertices (break the remaining ties arbitrarily). For each graph G', iterate over all possible $\lambda'\colon V(G') \setminus Z \to \{0, \dots, d\}$ and check whether $\mathsf{sig}(G[Y \cup Z], Z, \lambda|_Y) = \mathsf{sig}(G', Z, \lambda')$ as well as $|D(G', \lambda')| \le d^3$. Clearly, this procedure terminates and finds the required tuple (G', λ') because the tuple $(G[Y \cup Z], \lambda|_Y)$ witnesses its existence.

Without loss of generality, we may assume $Z = \{1, \dots, |Z|\}$. Now note that both requirements and facilities contain only set systems over Z, tuples of elements of Z, numbers in $\{1, \dots, O(d^3)\}$, and the number of these entities is upper bounded by a function of d. Hence, $\mathsf{sig}(G[Y \cup Z], Z, \lambda|_Y)$ can be encoded using at most $\psi(|Z|) \le \psi(d)$ bits, where ψ is some monotone ascending, computable function. Thus, since $\mathsf{sig}(G[Y \cup Z], Z, \lambda|_Y)$ is the only input to the procedure that finds G' and λ', the procedure terminates after at most $\phi'(\psi(d))$ steps for some computable function ϕ', meaning that (G', λ') is of size at most $\phi'(\psi(d))$. ☐

Now we can make the notion of replacing $G[Y]$ more precise; it involves the gluing operation $\circ$ of two boundaried graphs. Recall the corresponding definitions from Section 1.1. To simplify notation, we assume here that, whenever we glue two boundaried graphs, that their boundaries are equal to some vertex subset Z defined in the context, and that their boundary labelings are equal. Hence, gluing in this context means to take disjoint union, to introduce two copies of each vertex in Z, and then to identify each pair of copies of vertices in Z.

We arrive at the reduction rule aiming at reducing the size of Y.

Rule 8.3. Let (G, λ, k) be an instance of VECTOR d-CONNECTIVITY that is reduced with respect to Rules 8.1 and 8.2. Let $Y \in \mathcal{Y}$, where $\mathcal{Y}$ is as in Definition 8.2, and let $Z = N_G(Y)$. Furthermore, let (G', λ') be as in Lemma 8.12. If $|Y| > |V(G') \setminus Z|$, then replace G by $G' \circ (G - Y)$ and replace λ by $\lambda|_{V(G) \setminus Y} \cup \lambda'$.

Before proving that Rule 8.3 is correct, we need a technical lemma that shows how paths from a demand vertex to a vertex-connectivity set are split over a separator. For this, we need the following notation. A *separation* of a graph G is a tuple (T, U) of two vertex subsets $T, U \subseteq V(G)$ such that $T \cup U = V(G)$ and there is no edge between $T \setminus U$ and $U \setminus T$ in G. The *order* of a separation (T, U) is $|T \cap U|$.

Lemma 8.13. Let G be a graph, (T, U) a separation of G, $Z = T \cap U$, $S \subseteq V(G)$, $v \in V(G) \setminus S$, and $d \in \mathbb{N}$. There are d v-independent paths from v to S in G if and only if there is an integer $i \in \{0, \dots, d\}$ and a partition of $Z \setminus \{v\}$ into four vertex sets A, B, C, D such that

(i) if $v \in T \setminus U$, then $i = d$, and if $v \in U \setminus T$, then $i = 0$,
(ii) there is a $(v, i, A, B \cup (S \setminus U))$-constrained path packing in $G[T \setminus C]$, and
(iii) there is a $(v, d - i, B, A \cup (S \cap U))$-constrained path packing in $G[U \setminus D]$.

Proof. ($\Rightarrow$): Assume first that there are d v-independent paths from v to S in G and let $\mathscr{P}$ be a corresponding path packing (with overlap only in start vertex v). We may safely assume that paths in $\mathscr{P}$ have no vertices of S as internal vertices; else they could be shortened. We will select $A, B, C, D \subseteq Z \setminus \{v\}$ and $i \in \{0, \dots, d\}$ such that the path packings exist as stated in the lemma. For the purpose of getting a clear partitioning of the edges contained in Z, we show that one of the packings exists in $G[T \setminus C] - E(G[Z])$ and one of them in the remainder of the graph. Let us shorthand H for $G[T \setminus C] - E(G[Z])$.

Consider all paths in $\mathscr{P}$ as being directed from v towards S, and consider the set $\mathscr{P}_H$ of maximal, directed subpaths in H of paths in $\mathscr{P}$ such that each path in $\mathscr{P}_H$ contains at least one arc. Denote by $\vec{H}$ the directed subgraph of H induced by $\mathscr{P}_H$. That is, $\vec{H}$ contains precisely the vertices and arcs also contained in the paths in $\mathscr{P}_H$. We can now pick the sets $A, B, C, D \subseteq Z \setminus \{v\}$. The source vertices in $\vec{H}$ not equal to v form the set A. Note that all source vertices except possibly v are contained in Z as each vertex on a path in $\mathscr{P}_H$ but not in $Z \cup \{v\}$ must have a predecessor. Similarly, sink vertices in $\vec{H}$ are contained in $Z \cup S$; we put those sink vertices that are contained in $Z \setminus \{v\}$ into B. Note that, as each path in $\mathscr{P}_H$ has length at least one, there are no vertices of in- and outdegree zero and hence $A \cap B = \emptyset$. Vertices in Z that are used by paths in $\mathscr{P}_H$, but that are neither sources nor sinks, are put into D. Vertices of $Z \setminus \{v\}$ that are not on any path in $\mathscr{P}_H$ are put into C. Clearly, A, B, C, D is a partition of $Z \setminus \{v\}$. Finally, we define $i = d$ if $v \in T \setminus U$, $i = 0$ if $v \in U \setminus T$, and if $v \in Z$, then i is defined as the outdegree of v in $\vec{H}$. The condition on d and i in the lemma is clearly fulfilled. We claim that $\mathscr{P}_H$ is the desired path packing in $G[T \setminus C]$.

Showing that $\mathscr{P}_H$ is a $(v, i, A, B \cup (S \setminus U))$-constrained path packing in $G[T \setminus C]$. Clearly, H is a subgraph of $G[T \setminus C]$ and hence $\mathscr{P}_H$ is contained in $G[T \setminus C]$. Observe that, since the paths in $\mathscr{P}_H$ are vertex-disjoint (except for v) and each path has length at least one, sources and sinks in $\vec{H}$ correspond to endpoints of these paths. Combining this with our observation from above that sources and sinks in $\vec{H}$ are in $A \cup \{v\}$ and $B \cup S$, respectively, we infer that each path in $\mathscr{P}_H$ starts in either v or A, and ends in $B \cup (S \setminus U)$. By definition of A, each vertex $w \in A$ has a path in $\mathscr{P}_H$ starting in w and, furthermore, by the definition of i, there are i paths in $\mathscr{P}_H$ that start in v. Hence, $\mathscr{P}_H$ witnesses that there are $|A| + i$ paths from $A \cup \{v\}$ to $B \cup (S \setminus U)$ in H; moreover, these paths do not touch C by definition. As the paths in $\mathcal{P}$ are v-independent, so are the paths in $\mathcal{P}_H$. Hence, $\mathcal{P}_H$ is a $(v, i, A, B \cup (S \setminus U))$-constrained path packing in $G[T \setminus C]$.

Showing the existence of a $(v, d - i, B, A \cup (S \cap U))$-constrained path packing in $G[U \setminus D]$. Take the path packing $\mathcal{P}$ and define a path packing $\mathcal{P}'$ that contains all maximal subpaths of $\mathcal{P}$ in $U \setminus D$. Note that $\mathcal{P}'$ may contain paths of length zero. We consider also $\mathcal{P}'$ as a set of directed paths, each arc inheriting its direction from $\mathcal{P}$. Clearly, $\mathcal{P}'$ is contained in $G[U \setminus D]$. We claim that $\mathcal{P}'$ is also $(v, d - i, B, A \cup (S \cap U))$-constrained.

Consider the directed subgraph $\vec{G}'$ of $G[U]$ induced by $\mathcal{P}'$. Let us find the endpoints of the paths in $\mathcal{P}'$. Clearly, if $v \in U \setminus D$, then v is such an endpoint. For the remaining endpoints, first, consider a path of length zero, represented by a vertex $w \neq v$. Since each path in $\mathcal{P}$ has length at least one and since $w \neq v$, w has a predecessor u on a path in $\mathcal{P}$. Since w represents a path of length zero, $u \in (T \setminus U) \cup D$. By definition of D (since H does not contain any edges in Z), each successor of a vertex in D on a path of $\mathcal{P}$ is contained in $T \setminus U$. Hence, in fact $u \in T \setminus U$. The only vertices in U that have neighbors in $T \setminus U$ are contained in $T \cap U = Z$. This implies that $w \in Z$ and hence $w \in Z \setminus C$. Since $\mathcal{P}'$ has empty intersection with D, we moreover have $w \notin D$. Hence, only two possibilities remain: $w \in A$ and $w \in B$. Assume that $w \in A$. Since the vertices in A are sources of $\vec{H}$, this implies that there is a path starting in $w \neq v$ in $\mathcal{P}$, a contradiction. It follows that $w \in B$. Since B represents sinks in $\vec{H}$, we moreover have $w \in S$ as, otherwise, there would be a path in $\mathcal{P}$ ending in a vertex not contained in S. Thus, as also $w \in U$, each path of length zero in $\mathcal{P}'$ ends in $S \cap U$ (and starts in B).

Next, consider paths of length at least one in $\mathcal{P}'$. Since they are pairwise vertex-disjoint (except for v), their endpoints correspond to the sources and sinks in $\vec{G}'$. Let w be a sink in $\vec{G}'$ that is not contained in S. Since w is not in S, it has a successor x on a path in $\mathcal{P}$. As above, by the definition of D, each predecessor of a vertex in D on a path in $\mathcal{P}$ is contained in $T \setminus U$. Hence, in fact $x \in T \setminus U$. The only vertices in U that have neighbors in $T \setminus U$ are contained in $T \cap U = Z$. Hence, we have $w \in Z$. Observe that $w \neq v$ as, otherwise, $\mathcal{P}$ contains a cycle. The paths in $\mathcal{P}$ are vertex-disjoint, thus, w does not have any incoming arcs in $\vec{H}$, meaning that it is a source in $\vec{H}$. This implies $w \in A$ by definition of A. Thus we obtain that $\mathcal{P}'$ is a packing of paths, each of which ends in $A \cup (S \cap U)$.

It remains to prove that $\mathcal{P}'$ contains $|B| + d - i$ paths that start in $\{v\} \cup B$; their v-independence is implied by the fact that these paths are subpaths of paths in $\mathcal{P}$. We claim that there are $d - i$ paths in $\mathcal{P}'$ starting in v. First, if $v \notin U$ then $i = d$ by definition; hence, the claim is trivially true. If $v \in U \setminus T$, then $i = 0$ and, clearly, each path in $\mathcal{P}$ that starts in v induces one such path in $\mathcal{P}'$. Thus, the claim holds also in this case. Finally, if $v \in Z$, then i is the outdegree of v in $\vec{H}$. Recall that H does not contain any edge in Z. Hence, also in the final case there are $d - i$ paths in $\mathcal{P}'$ that start in v.

To find the remaining $|B|$ paths, consider a vertex $w \in B$. Note that $w \neq v$ because $v \notin B$. By definition, w is a sink in $\vec{H}$ and, since it is a part of a path in $\mathcal{P}$ reaching S, it either is contained in S or has a successor on $\mathcal{P}$ which is not contained in $T \setminus C$. In the first case, w represents a length-zero path starting in B in $\mathcal{P}'$. In the second case, w is a source in $\vec{G}'$ by the vertex-disjointness of the paths in $\mathcal{P}$. Since the choice of w is arbitrary, and since the paths in $\mathcal{P}$ are vertex-disjoint, each vertex in $B \setminus S$ is a source in $\vec{G}'$ and hence has a path in $\mathcal{P}'$ starting in this vertex. Thus, overall, there are $|B| + d - i$ paths from $\{v\} \cup B$ to $A \cup (S \cap U)$ in $G[U \setminus D]$ which are v-independent, as required. This completes the “if” part of the lemma.

($\Leftarrow$): Assume that there is a partition A, B, C, D of $Z \setminus \{v\}$ and $i \in \mathbb{N}$ as described in the lemma and fix a $(v, i, A, B \cup (S \setminus U))$-constrained path packing $\mathcal{P}_T$ in $G[T \setminus C]$ and a $(v, d-i, B, A \cup (S \cap U))$-constrained path packing $\mathcal{P}_U$ in $G[U \setminus D]$. Consider the paths in $\mathcal{P}_T$ as directed from $\{v\} \cup A$ to $B \cup (S \setminus U)$ and the paths in $\mathcal{P}_U$ as directed from $\{v\} \cup B$ to $A \cup (S \cap U)$. Let us show that there is a packing of d v-independent paths from v to S in G.

Observe that $\mathcal{P}_T$ and $\mathcal{P}_U$ may overlap only in $A \cup B \cup (\{v\} \cap Z)$, as the graphs they are contained in overlap precisely in this vertex set. Consider the directed graph induced by the union of $\mathcal{P}_T$ and $\mathcal{P}_U$. Denote by K the (weakly) connected component of this graph that contains v. By definition of $\mathcal{P}_T$ and $\mathcal{P}_U$, vertex v is a source vertex. Let us first show that v has outdegree d in K. Otherwise, v must have a successor w in either A or B in both $\mathcal{P}_T$ and $\mathcal{P}_U$. However, as the paths in $\mathcal{P}_T$ start in A and the paths in $\mathcal{P}_U$ start in B in both cases we get a contradiction. Thus, v is a source with precisely d outgoing arcs in K.

We claim that v is the only source in K. To see this, we first derive in- and outdegrees of all vertices other than v in K. Clearly, each vertex in $K - (A \cup B \cup \{v\})$ is either in S—and has indegree one and outdegree zero in this case—or has in- and outdegree exactly one. We claim that each vertex in $A \cup B$ has indegree at most one in K. This is clear for vertices in A, as only $\mathcal{P}_U$ sends paths to A. Both packings $\mathcal{P}_T$ and $\mathcal{P}_U$ may send paths to a vertex $w \in B$ in the case that $w \in S$. Then, however, w is in a path of length zero in $\mathcal{P}_U$,[20] implying that indeed each vertex in $A \cup B$ has indegree at most one in K. Now for the sake of contradiction assume that there are

[20]Recall that in a (v, i, A, B)-constrained path packing, we assume each path with endpoints in $A \cap B$ to be of length zero.

two sources in K and consider a path in the underlying undirected graph of K between these two sources. On this path, there is a vertex with indegree at least two; a contradiction. Thus, v is the only source in K.

It now suffices to show that each vertex in $A \cup B$ either has in- and outdegree one in K or is a sink contained in S. As we have derived the same for the vertices in $V(K) \setminus (A \cup B \cup \{v\})$ above, and since the sum of all indegrees equals the sum of all outdegrees in K, this then implies that there are d vertex-disjoint paths from v to S. Let thus prove that, indeed, each vertex in $A \cup B$ has either in- and outdegree one in K or is a sink contained in S.

We have shown above that each vertex in $A \cup B$ has indegree at most one in K. Since K has v as its only source, each of the vertices in $A \cup B$ has also indegree at least one in K. Since only $\mathcal{P}_T$ has paths starting in A and only $\mathcal{P}_U$ has paths starting in B, the outdegree of the vertices in $A \cup B$ is at most one in K. Now consider a sink $w \in V(K) \cap (A \cup B)$. Recall that $\mathcal{P}_T$ contains a path starting in each vertex of A. Since $A \cap (B \cup (S \setminus U)) = \emptyset$, each of these paths has length at least one. Hence, $w \in B$. Since also $\mathcal{P}_U$ has a path starting in w, it must be of length zero and thus $w \in S$ because the paths in $\mathcal{P}_U$ end in $B \cup (S \cap U)$ and $A \cap B = \emptyset$. Thus we have shown that each vertex in K is either the source v with outdegree d, has in- and outdegree exactly one, or is a sink contained in S and has indegree exactly one. This means that there are d v-independent paths from v to S in G. $\square$

We are ready to show that Rule 8.3 respects yes and no-instances.

Lemma 8.14. Rule 8.3 is correct.

Proof. We claim that an even stronger statement holds. Namely, let $G_1, G_2, \hat{G}$ be three graphs, each containing Z as a vertex subset such that $G_1[Z] = G_2[Z] = \hat{G}[Z]$. Furthermore, let $\lambda_1 \colon V(G_1) \setminus Z \to \{0, \dots, d\}$, $\lambda_2 \colon V(G_2) \setminus Z \to \{0, \dots, d\}$, and $\hat{\lambda} \colon V(\hat{G}) \to \{0, \dots, d\}$ be three demand functions, such that $|D(G_1, \lambda_1)| \leq d^3$, $|D(G_2, \lambda_2)| \leq d^3$ and such that $\operatorname{sig}(G_1, Z, \lambda_1) = \operatorname{sig}(G_2, Z, \lambda_2)$. Then $G_1 \circ \hat{G}$ has a vector connectivity set of size k with respect to $\lambda_1 \cup \hat{\lambda}$ if and only if $G_2 \circ \hat{G}$ has a vector connectivity set of size k with respect to $\lambda_2 \cup \hat{\lambda}$.

To see that our claim implies the lemma, set $G_1 := G[Y \cup Z]$, $G_2 := G'$, $\hat{G} := G - Y$ and define the demand functions λ_1, λ_2 and $\hat{\lambda}$ accordingly. Then our claim implies that $G_1 \circ \hat{G} = G[Y \cup Z] \circ (G - Y) = G$ has a vector connectivity set of size k if and only if $G_2 \circ \hat{G} = G' \circ (G - Y)$ has such a set. That is, the claim implies that Rule 8.3 is correct.

Let us prove the claim. Note that, by swapping the names of G_1 and G_2, as well as λ_1 and λ_2, it suffices to prove one direction. Assume hence that $G_1 \circ \hat{G}$ has a vector connectivity set S of size k with respect to $\lambda_1 \cup \hat{\lambda}$. Since $|D(G_1, \lambda_1)| \leq d^3$ and $|N_{G_1 \circ \hat{G}}(V(G_1) \setminus Z)| \leq |Z| \leq d$ we may apply Lemma 8.8 with $Y = V(G_1) \setminus Z$ and hence, we may assume that $S_1 := (V(G_1) \cap S) \setminus Z$ contains at most $d^3 + d$ elements. Thus, we have

$$(|S_1|, \mathsf{req}(G_1, Z, \lambda_1, S_1), \mathsf{fac}(G_1, Z, S_1)) \in \mathsf{sig}(G_1, Z, \lambda_1).$$

Since $\mathsf{sig}(G_1, Z, \lambda_1) = \mathsf{sig}(G_2, Z, \lambda_2)$, there is a set $S_2 \subseteq V(G_2) \setminus Z$ with $|S_1| = |S_2|$, $\mathsf{req}(G_1, Z, \lambda_1, S_1) = \mathsf{req}(G_2, Z, \lambda_2, S_2)$ and $\mathsf{fac}(G_1, Z, S_1) = \mathsf{fac}(G_2, Z, S_2)$. We claim that $S' := (S \setminus S_1) \cup S_2$ is a vector connectivity set for $G_2 \circ \hat{G}$ with respect to $\lambda_2 \cup \hat{\lambda}$; clearly, we have $|S| = |S'|$.

Let us check that this is true indeed. We consider vertices in $D(G_2, \lambda_2)$, $D(\hat{G}[Z], \hat{\lambda})$, and vertices in $D(\hat{G} - Z, \hat{\lambda})$ individually. Let us start with $D(G_2, \lambda_2)$. For each vertex $v_2 \in D(G_2, \lambda_2)$, there is at least one vertex $v_1 \in D(G_1, \lambda_1)$ with $\mathsf{sat}(G_1, Z, \lambda_1, S_1, v_1) = \mathsf{sat}(G_2, Z, \lambda_2(v_2), S_2, v_2)$. Let us show that the demand of v_2 is satisfied in $G_2 \circ \hat{G}$ by S'.

Consider first the case that $v_1 \in S_1$. Then (A, B, C) is a satisfying connector for v_1 in G_1 for arbitrary three sets $A, B, C \subseteq Z$, mutually disjoint. In particular $(\emptyset, \emptyset, Z)$ is a satisfying connector for v_1 and since the set of satisfying connectors for v_1 equals the one for v_2, $(\emptyset, \emptyset, Z)$ is a satisfying connector for v_2. By the definition of satisfying connector either $v_2 \in S_2$, or there is a $(v_2, \lambda_2(v_2), \emptyset, S_2)$-constrained path packing in $G_2 - Z$, meaning that there are $\lambda_2(v_2)$ v_2-independent paths from v_2 to S_2 in $G_2 - Z$. Hence, the demand of v_2 is satisfied if $v_1 \in S_1$.

Now assume that $v_1 \notin S_1$ and observe that $(V(G_1), V(\hat{G}))$ is a separation of $\hat{G} \circ G_1$. Since there are $\lambda_1(v_1)$ v_1-independent paths from v_1 to S in $\hat{G} \circ G_1$, by Lemma 8.13 and since $v_1 \in V(G_1) \setminus V(\hat{G})$, there is a partition of Z into four sets A, B, C, D such that there is a $(v_1, \lambda_1(v_1), A, B \cup (S \setminus V(\hat{G})))$-constrained path packing $\mathscr{P}_1$ in $G_1 - C$ and a $(B, A \cup (S \cap V(\hat{G})))$-constrained path packing $\hat{\mathscr{P}}$ in $\hat{G} - D$. Because $S \setminus V(\hat{G}) = S_1$, packing $\mathscr{P}_1$ is also $(v_1, \lambda_1(v_1), A, B \cup S_1)$-constrained. Thus $\mathscr{P}_1$ witnesses that $(A, B, C) \in \mathsf{sat}(G_1, Z, \lambda_1(v_1), S_1, v_1)$, meaning that also $(A, B, C) \in \mathsf{sat}(G_2, Z, \lambda_2(v_2), S_2, v_2)$. Applying the definition of satisfying connector again, we have a $(v_2, \lambda_2(v_2), A, B \cup S_2)$-constrained path packing $\mathscr{P}_2$ in $G_2 - C$. Note that $\mathscr{P}_2$ is also $(v_2, \lambda_2(v_2), A, B \cup (S' \setminus V(\hat{G}))$-constrained. Applying again Lemma 8.13, the two path packings $\mathscr{P}_2$ and $\hat{\mathscr{P}}$ thus witness that there are $\lambda(v_2)$ v_2-independent paths from v to S in $\hat{G} \circ G_2$.

Next, consider $v \in D(\hat{G} - Z, \hat{\lambda})$; there are $\hat{\lambda}(v)$ v-independent paths from v to S in $\hat{G} \circ G_1$. Applying Lemma 8.13 with the separation $(V(G_1), V(\hat{G}))$, we obtain a partition of Z into A, B, C, D (since $v \in V(\hat{G}) \setminus V(G_1)$), a $(A, B \cup S_1)$-constrained path packing $\mathscr{P}_1$ in $G_1 - C$ and a $(v, \hat{\lambda}(v), B, A \cup (S \cap V(\hat{G}))$-constrained path $\hat{\mathscr{P}}$ packing in $\hat{G} - D$. By the definition of provided connector, we have $(A, B, C) \in \mathsf{fac}(G_1, Z, S_1) =$

$\mathsf{fac}(G_2, Z, S_2)$. Thus, again by the definition of provided connector, there is a $(A, B \cup S_2)$-constrained path packing in $G_2 - C$. Applying again Lemma 8.13, the packings $\mathcal{P}_2$ and $\hat{\mathcal{P}}$ witness that there are $\hat{\lambda}(v)$ v-independent paths from v to S' in $\hat{G} \circ G_2$.

Finally, consider $v \in D(\hat{G}[Z]), \hat{\lambda})$; there are again $\hat{\lambda}(v)$ v-independent paths from v to S in $\hat{G} \circ G_1$. Now we apply Lemma 8.13 with the separation $(V(G_1), V(\hat{G}))$. This time, we obtain a partition of $Z \setminus \{v\}$ into A, B, C, D, and an integer i together with a $(v, i, A, B \cup S_1)$-constrained path packing $\mathcal{P}_1$ in $G_1 - C$ and a $(v, \hat{\lambda}(v) - i, B, A \cup (S \cap V(\hat{G})))$-constrained path packing $\hat{\mathcal{P}}$ in $\hat{G} - D$. By the definition of provided special connector, the packing $\mathcal{P}_1$ witnesses that $(v, i, A, B, C) \in \mathsf{fac}(G_1, Z, S_1) = \mathsf{fac}(G_2, Z, S_2)$. Hence, again by this definition, there is also a $(v, i, A, B \cup S_2)$-constrained path packing $\mathcal{P}_2$ in $G_2 - C$. Applying Lemma 8.13 again, the packings $\mathcal{P}_2$ and $\hat{\mathcal{P}}$ witness that there are $\hat{\lambda}(v)$ v-independent paths from v to S' in $\hat{G} \circ G_2$.

Overall we showed that each vertex v with nonzero demand $(\hat{\lambda} \cup \lambda_2)(v)$ has as many v-independent paths from v to S' in $\hat{G} \circ G_2$, meaning that S' is a vector connectivity set. Since $|S'| = |S|$, this shows that Rule 8.3 is correct. □

8.6.3. Putting things together

We can now state our kernelization procedure for instances (G, λ, k) of VECTOR d-CONNECTIVITY. The only missing piece is to argue why and how we may reduce vertices in G that are not contained in any set of $\mathcal{Y}(G, \lambda, d)$.

Theorem 8.4. VECTOR d-CONNECTIVITY has a vertex-linear polynomial kernelization with respect to k.

Proof. Given an instance (G, λ, k) of VECTOR d-CONNECTIVITY the kernelization proceeds as follows. Throughout, we refer to the current instance by (G, λ, k) and recall the use of $D = \{v \in V(G) \mid \lambda(v) \geq 1\}$.

1. Apply Rule 8.1 exhaustively and then apply Rule 8.2 (this may return answer no if we have more than $d^2 k$ demand vertices).
2. Apply Rule 8.3 once if possible.
3. Return to Step 1 if Rule 8.3 was applied.
4. Let $W := D \cup \bigcup_{Y \in \mathcal{Y}} N[Y]$. Perform the torso operation on W in G to obtain G'. That is, carry out the following steps:
   a) Start with $G' = G[W]$.

   b) For every pair $u, v \in W$, if there is a u, v-path in G with internal vertices from $V \setminus W$, then add the edge $\{u, v\}$ to G'.

5. Return (G', λ', k) as the kernelized instance, where $\lambda' = \lambda|_W$ is λ restricted to W.

Correctness. We already know that Rules 8.1 to 8.3 are correct; it remains to discuss the effect of the torso operation: Proposition 8.4 implies that minimal solutions S for (G, λ, k) are completely contained in the union of sets $X \in \mathcal{X}$, since only such vertices can contribute to S being a hitting set for $\mathcal{X}$. It follows, by Lemma 8.6, that every minimal solution S is also contained in W. Thus, if before the torso operation every vertex $v \in D$ has $\lambda(v)$ paths to S, then the same is true after the operation since there are shortcut edges for all paths with internal vertices from $V \setminus W$. The converse is more interesting.

Assume that (G, λ, k) is no and fix an arbitrary set $S \subseteq W = V(G') \subseteq V(G)$ of size at most k; we will show that S is not a solution for (G', λ', k). By assumption, S is not a solution for (G, λ, k). Thus, by Proposition 8.4, S is not a hitting set for $\mathcal{X} = \mathcal{X}(G, \lambda)$. Accordingly, fix a set $X \in \mathcal{X}$ with $S \cap X = \emptyset$. By definition of $\mathcal{X}$, let $v \in X$ with $\lambda(v) > |N(X)|$. By Lemma 8.6, there is a $Y \in \mathcal{Y}(G, \lambda, d)$ with $X \subseteq Y$. Thus, $N[X] \subseteq N[Y] \subseteq W$. If there were $\lambda(v)$ paths from v to S in G', then at least one of them avoids $N(X)$, since $|N(X)| < \lambda(v)$; let P' denote a path from v to S in G' that avoids $N(X)$. Undoing the torso operation, we get a walk P in G, with additional interval vertices from $V(G) \setminus W$. Since $(V(G) \setminus W) \cap N(X) = \emptyset$, this walk also avoids $N(X)$ and implies that X contains at least one vertex of S; a contradiction to $S \cap X = \emptyset$. Thus, no $S \subseteq V(G')$ of size at most k is a solution for (G', λ', k), implying that (G', λ', k) is no, as required.

Problem kernel size. The output graph G' has vertex set $W = D \cup \bigcup_{Y \in \mathcal{Y}} N[Y]$ where D and $\mathcal{Y}$ correspond to a fully reduced instance (G, λ, k). By Rule 8.2 we have $|D| \leq d^2 k$. By Lemma 8.7 the set $\mathcal{Y}$ contains at most $2^{d^3+d} d^2 k$ sets, each of size upper bounded by $f(d)$ for some computable function depending only on d. By Definition 8.2 the neighborhood $N(Y)$ of each set Y has size at most d. It follows that G' has $O(k)$ vertices, as claimed. Clearly, the total encoding size for an instance is $O(k^2)$ since d is a constant.

Running time. By Lemma 8.2, Rule 8.1 can be applied exhaustively in polynomial time. Clearly, Rule 8.2 can be applied in polynomial time as it only checks the number of vertices with nonzero demand. Finding one application of Rule 8.3 can be done by iterating over $Y \in \mathcal{Y}$ and applying Lemma 8.12 to each Y until we find a replacement subgraph that is strictly smaller; in total this takes polynomial time. Furthermore, repeating these steps whenever Rule 8.3 has been applied gives only a polynomial factor because each time the instance shrinks by at least one vertex. Finally, it is easy to implement the torso operation in polynomial time. □

8.7. Kernelization lower bound

In this section, we prove that VECTOR CONNECTIVITY admits no polynomial kernelization with respect to k unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. We give a reduction from HITTING SET parameterized by the number of hyperedges, which also makes a polynomial Turing kernelization unlikely (see Hermelin et al. [Her+15]). Observe that demands greater than $k+1$ can be safely replaced by demand $k+1$, hence, $d \leq k+1$ without loss of generality. Thus, the lower bound applies also to the parameter $d+k$.

Theorem 8.5. VECTOR CONNECTIVITY does not admit a polynomial problem kernel with respect to k unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ (that is, unless the polynomial hierarchy collapses).

Proof. We give a polynomial parameter transformation from HITTING SET parameterized by the number m of hyperedges to VECTOR CONNECTIVITY parameterized by k.

HITTING SET
Input: A hypergraph $\mathcal{H}$ and a nonnegative integer k.
Question: Is there a set $H \subseteq V(\mathcal{H})$ of size at most k such that $S \cap H \neq \emptyset$ for each hyperedge S in $\mathcal{H}$?

It is known that HITTING SET does not admit a polynomial problem kernel with respect to m unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, see Dom, Lokshtanov, and Saurabh [DLS14]. Together with the fact that polynomial kernelizations for NP-complete problems are preserved by polynomial-parameter transformations (see Section 1.1.3), we obtain the desired lower bound. Let $(\mathcal{H} = (U, \mathcal{E}), k)$ be an instance of HITTING SET with parameter $m = |\mathcal{E}|$. Without loss of generality, assume that $k \leq m$. Let $n := |U|$. We construct a graph G on $2(k+1)m + n$ vertices that has a vector connectivity set of size at most $k' = (k+1)m + k = O(m^2)$ if and only if $(\mathcal{H}, k)$ is yes for HITTING SET.

Construction. Start with an empty graph G. We introduce one vertex x_u to G for each element $u \in U$, and we introduce $2(k+1)$ vertices $y_{1,F}, \ldots, y_{k+1,F}, y'_{1,F}, \ldots, y'_{k+1,F}$ to G for each set $F \in \mathcal{E}$. The edges in G are defined as follows:

1. For each $i \in \{1, \ldots, k+1\}$ and $F \in \mathcal{E}$, add the edge $\{y_{i,F}, y'_{i,F}\}$.
2. For each $F \in \mathcal{E}$, each $u \in F$, and each $i \in \{1, \ldots, k+1\}$, add the edge $\{x_u, y_{i,F}\}$.
3. Add the edge between every pair of vertices in $\{y_{i,F} \mid (F \in \mathcal{E}) \wedge (i \in \{1, \ldots, k+1\})\}$.

Set the demand λ of each $y'_{i,F}$ vertex to 2 and of each $y_{i,F}$ vertex to $(k+1)m+1$; all x-vertices have demand zero. Set the budget k' to $(k+1)m+k$. This completes the construction of the Vector Connectivity instance (G, λ, k'), which can be easily performed in polynomial time.

Correctness. Assume first that (G, λ, k') is yes and let S be a vector connectivity set of size at most k'. Note that S must contain all vertices $y'_{i,F}$ since they have demand of 2 but only one neighbor (namely $y_{i,F}$). This accounts for $(k+1)m$ vertices in S; there are at most k further vertices in S. Let T contain exactly those elements $u \in U$ such that $x_u \in S$; thus $|T| \leq k$. We claim that T is a hitting set for $\mathcal{E}$, that is, T has nonempty intersection with each set in $\mathcal{E}$.

Let $F \in \mathcal{E}$; we show that $T \cap F \neq \emptyset$. Since at most k vertices in S are not y'-vertices, we can choose $i \in \{1, \ldots, k+1\}$ such that S does not contain $y_{i,F}$. Consider the set C consisting of all y-vertices other than $y_{i,F}$ as well as the vertex $y'_{i,F}$. Note that the set of neighbors of $y_{i,F}$ contains precisely all x_u with $u \in F$, all remaining y-vertices, and $y'_{i,F}$. Furthermore, x-vertices only have y-vertices as neighbors. Thus, in $G - C$ we find a connected component D containing $y_{i,F}$ and all x_u with $u \in F$ but no further vertices. Since $y_{i,F} \notin S$, there are $(k+1)m+1$ disjoint paths from $y_{i,F}$ to S. Thus, since C is has size precisely $(k+1)m$, at least one vertex in the connected component D is in the solution. Since S does not contain $y_{i,F}$ (by choice of i), we have $x_u \in S$ for some $u \in F$. Thus, $u \in T$ and $T \cap F \neq \emptyset$ indeed.

Now, assume that $(\mathcal{H}, k)$ is yes for Hitting Set and let T a hitting set of size at most k for $\mathcal{E}$. We create a vector connectivity set S by selecting all x_u with $u \in T$ as well as all y'-vertices; thus $|S| \leq k' = (k+1)m+k$. Clearly, this satisfies the demand of each y'-vertex. Consider an arbitrary vertex $y_{i,F}$ and recall that its demand is $\lambda(y_{i,F}) = (k+1)m+1$. We know that S contains at least one vertex x_u with $u \in F$ that is adjacent to $y_{i,F}$. Thus, we can find the required $(k+1)m+1$ disjoint paths from $y_{i,F}$ to S:

- We have one path $(y_{i,F}, y'_{i,F})$ and one path $(y_{i,F}, x_u)$.
- For all $(j, F') \neq (i, F)$ we get one path $(y_{i,F}, y_{j,F}, y'_{j,F})$, summing up to $(k+1)m-1$ paths.

It follows that (G, λ, k') is yes for Vector Connectivity.

Summarizing, we have given a polynomial parameter transformation to Vector Connectivity parameterized by k from Hitting Set parameterized by m, which is known not to admit a polynomial kernelization unless NP ⊆ coNP/poly [DLS14] (see also Hermelin et al. [Her+15]). Since both problems are NP-complete, a polynomial problem kernel for Vector Connectivity with respect to k would imply a polynomial problem kernel for Hitting Set(see Section 1.1.3). □

From the above reduction we also get the following corollary related to parameterizations above lower bound for Vector Connectivity: Since we have strong indication that Vector Connectivity is fixed-parameter tractable with respect to the solution size k (Theorem 8.3), it is interesting to consider smaller parameters than k. For example, the *parameterization over lower bound*, the difference between k and a lower bound ℓ on the solution size. It is not hard to check that one such lower bound on the size of each vector connectivity set is the least integer ℓ such that all vertices except at most ℓ have demand at most ℓ. However, $k - \ell$ is upper bounded by the solution size in the Hitting Set instance in the reduction in Theorem 8.5 above. Since Hitting Set is W[2]-hard with respect to the solution size, to obtain fixed-parameter tractability for parameterizations over lower bounds, we have to consider smaller lower bounds or incomparable ones.

8.8. Concluding remarks

To summarize our findings, we gave a randomized fixed-parameter tractability result for Vector Connectivity with respect to the solution size k, but also proved that Vector Connectivity does not admit a polynomial kernelization with respect to k unless NP ⊆ coNP/poly. Since demands greater than $k+1$ can be safely replaced by demand $k+1$ (because they cannot be fulfilled without putting the vertex into the solution) the lower bound extends also to parameter $k+d$, wherein d is the maximum demand. In contrast, for Vector d-Connectivity, where d is a problem-specific constant, we gave an explicit vertex-linear kernelization with at most $\phi(d) \cdot k = O(k)$ vertices; the function $\phi(d)$ is computable, but superpolynomial in d. The function $\phi(d)$ cannot be polynomial in $k+d$ unless NP ⊆ coNP/poly due to the lower bound for $k+d$.

An important ingredient of our results is Rule 8.2 that reduces the number of vertices with nonzero demand to at most $d^2 k$ (or, similarly, to at most or $k^3 + k$). This rule gave rise to the alternative fixed-parameter algorithm, the vertex-linear kernelization for VECTOR d-CONNECTIVITY with respect to k, and can also be augmented to be used in a factor opt approximation algorithm for the minimization version of VECTOR CONNECTIVITY [KS16].

An interesting technical question that we left open is to upper bound the size of the replacement graphs used in Rule 8.3, which replaces large regions in the input graph by constant-size gadgets. We know that their size is upper bounded by a function in d, however, at present we do not have an explicit bound on their size.

Given that Rule 8.2, upper bounding the number of vertices with nonzero demand, is so immensely useful in theory, it would be interesting to see how it fares in practice. We think that it is reasonable that the demand of a vertex is a small integer. Hence, it could well be that, after the reduction, the demand vertices are sequestered, drastically reducing the search space for a solution.

In this regard, an obvious question is to find an efficient, practical algorithm that produces a solution. Our fixed-parameter algorithm (Section 8.4) gives good indication that no strong running-time lower bound for such algorithms exist. However, its best running-time upper bound is $2^{\Omega(k^4)}\operatorname{poly}(n)$ and it would be desirable to achieve single-exponential, that is, $2^{O(k)}\operatorname{poly}(n)$ running time. It would also be interesting to study tractability for smaller parameters than k, for example, the difference between a lower bound on the solution size and k.

Finally, it would be interesting to generalize VECTOR CONNECTIVITY to model further realistic applications. For example, it could be worthwhile to consider directed input graphs and to upper or lower bound the length of the paths to the solution. A lower bound on the length of the paths could be useful when placing dangerous or undesirable but necessary facilities, like nuclear reactors, airports, or waste collection plants. An upper bound on the length of the paths could model latency restrictions in placing servers or the cost of serving customers when placing warehouses. Introducing such an upper bound, however, we obtain a generalization of DOMINATING SET and, hence, W[2]-hardness with respect to the solution size k. Directed graphs could be useful to model more realistic logistics problems. A simple reduction from HITTING SET shows, however, that directed VECTOR CONNECTIVITY is W[2]-hard with respect to the solution size k. In light of these simple hardness results, it would make sense to restrict the input to practically relevant instances, for example, by requiring the input graph to be planar or by considering additionally structural parameters of the input graph.

Chapter 9

Outlook

We studied the parameterized complexity of three types of NP-hard graph problems and developed algorithms for each of them as well as lower bounds on their running times. We gave directions for future research specific to each problem in the concluding remarks of the corresponding chapter. In this final chapter, we highlight some future research directions that are not specific to the individual problems.

In Part I we constructed algorithms that, given a hypergraph with m hyperedges, find in fixed-parameter running time with respect to m an r-outerplanar support or a support with feedback edge number at most a given integer f. The results in this part are foremost of theoretical interest. Thus, more research is needed to reach algorithms that are promising for practice. Herein, a repository of different hypergraph parameters and their relations would be helpful in identifying the most promising parameters to work on.

On the theoretical side, it would be interesting to identify more graph properties for which we can find supports with that property in fixed-parameter running time with respect to m, the number of hyperedges. Recall that Theorem 2.1 shows that for every graph property Π that is closed under adding degree-one vertices and for every m there is a fixed-parameter algorithm with respect to m to determine whether there is a support in Π for a given hypergraph. The existence of such algorithms and the fact that it can be made constructive in two cases suggest that a *constructive* meta-theorem to the likes of Theorem 2.1 might be possible. We conjecture that, using the same strategy as in Chapter 4 for finding r-outerplanar supports, indeed we can design fixed-parameter algorithms with respect to m to determine whether there is a Π support for all graph properties Π of bounded treewidth that are closed under adding degree-one vertices.

In Part II we gave algorithms and running time lower bounds for finding dense k-vertex subgraphs for two definitions of being dense: μ-cliques, containing at least $\mu\binom{k}{2}$ edges, and highly connected graphs, having minimum degree at least $\lfloor k/2 \rfloor + 1$. In contrast to Parts I and III, implementations were carried out [KSS15] or are underway. In future work, it would make sense to study the combination of highly connected graphs and μ-cliques into one [BHB08] and extend our algorithms to this combination. Conceptually, it seems plausible that the combination of a strong local constraint and a more lax global constraint on top is a relevant concept of a cluster. Algorithmically, this would have the advantage that we can use the strong local constraints of highly connected graphs to design data reduction rules.

In a wider scope, the graph model for communities can be enhanced: Interactions in social networks are often transient, for example, in networks that arise from email conversations. Thus, modeling interactions as edges in an ordinary graph, we lose important information about the interaction. Allowing multiple edges between two vertices and equipping each edge with a time stamp leads to so-called temporal graphs [HS12; Hol15; Mic16]. It is an interesting task to model the interactions over time that occur in a community in a temporal graph in a mathematically tangible way [Hol15]. Furthermore, it is important to carry over the theory of sparsity for ordinary graphs to their temporal equivalents and, with it, the wealth of positive algorithmic results. A starting point, which the author is currently involved in exploring, is the combination of temporal equivalents of cliques [VLM16], the degeneracy parameter, and the Bron-Kerbosch clique enumeration algorithm which is efficient on graphs of small degeneracy [ELS13].

In Part III we gave a randomized fixed-parameter algorithm and data reduction rules for placing servers in a network such that they can provide a service that is robust against failures of nodes in the network. Combining robustness constraints with density, it may make sense to require the servers to be highly connected in order to be able to effectively synchronize their databases, for example. Combining robustness constraints with sparsity, it is interesting to consider robust and sparse hypergraph supports: In one application, the supports model an interconnection network of clients that want to communicate and it is clearly desirable for such a network to be robust to network failures [CVJ13]. Chen, Vitenberg, and Jacobsen [CVJ13] modeled robustness as k-connectedness of each subgraph of the support that is induced by a hyperedge. It would be interesting to study parameterized algorithms for finding this kind of supports.

In this thesis we pursued solution graphs which shall be sparse, be dense, or be robust. We are looking forward to more results on algorithms that find solutions that shall *be large, be small, be uniform, be homogeneous, be regular, be recurrent, be recoverable, ...*

Bibliography

[AAR10] Dana Angluin, James Aspnes, and Lev Reyzin. “Inferring Social Networks from Outbreaks.” In: *Proceedings of the 21st International Conference Algorithmic Learning Theory (ALT ’10)*. Vol. 6331. Lecture Notes in Computer Science. Springer, 2010, pp. 104–118 (cit. on pp. 2, 23, 32, 36).

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009 (cit. on p. 9).

[ADF95] Karl R. Abrahamson, Rodney G. Downey, and Michael R. Fellows. “Fixed-Parameter Tractability and Completeness IV: On Completeness for W[P] and PSPACE Analogues.” In: *Annnals of Pure and Applied Logic* 73.3 (1995), pp. 235–276 (cit. on p. 17).

[Alo+10] Daniel Aloise, Sonia Cafieri, Gilles Caporossi, Pierre Hansen, Sylvain Perron, and Leo Liberti. “Column generation algorithms for exact modularity maximization in networks.” In: *Physical Review E* 82 (4 2010), pp. 1–9 (cit. on p. 101).

[ARS02] James Abello, Mauricio G. C. Resende, and Sandra Sudarsky. “Massive Quasi-Clique Detection.” In: *Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN ’02)*. Vol. 2286. Lecture Notes in Computer Science. Springer, 2002, pp. 598–612 (cit. on p. 99).

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. “Color-coding.” In: *Journal of the ACM* 42.4 (1995), pp. 844–856 (cit. on pp. 108, 118).

[BBP03] Vladimir Boginski, Sergiy Butenko, and Panos M Pardalos. “On structural properties of the market graph.” In: *Innovations in Financial and Economic Networks*. New Dimensions in Networks. Edward Elgar Publishing, 2003, pp. 29–45 (cit. on p. 97).

[BBT05] Balabhaskar Balasundaram, Sergiy Butenko, and Svyatoslav Trukhanov. "Novel Approaches for Analyzing Biological Networks." In: *Journal of Combinatorial Optimization* 10.1 (2005), pp. 23–39 (cit. on p. 116).

[BC76] J.A. Bondy and V. Chvátal. "A method in graph theory." In: *Discrete Mathematics* 15.2 (1976), pp. 111–135 (cit. on p. 147).

[Bee+83] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. "On the Desirability of Acyclic Database Schemes." In: *Journal of the ACM* 30.3 (1983), pp. 479–513 (cit. on pp. 23, 24, 34, 68, 71).

[BEN09] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization.* Princeton University Press, 2009 (cit. on p. 1).

[Bev+14a] René van Bevern, Andreas Emil Feldmann, Manuel Sorge, and Ondřej Suchý. "On the Parameterized Complexity of Computing Balanced Partitions in Graphs." In: *Theory of Computing Systems* 57 (2014), pp. 1–35 (cit. on p. xvi).

[Bev+14b] René van Bevern, Sepp Hartung, André Nichterlein, and Manuel Sorge. "Constant-factor approximations for Capacitated Arc Routing without triangle inequality." In: *Operations Research Letters* 4.4 (2014), pp. 290–292 (cit. on p. xvi).

[Bev+14c] René van Bevern, Rolf Niedermeier, Manuel Sorge, and Mathias Weller. "Complexity of Arc Routing Problems." In: *Arc Routing: Problems, Methods, and Applications.* Ed. by Ángel Corberan and Gilbert Laporte. SIAM, 2014 (cit. on p. xvi).

[Bev+15a] René van Bevern, Iyad A. Kanj, Christian Komusiewicz, Rolf Niedermeier, and Manuel Sorge. "Well-Formed Separator Sequences, with an Application to Hypergraph Drawing." In: *CoRR* abs/1507.02350 (2015) (cit. on p. xiv).

[Bev+15b] René van Bevern, Christian Komusiewicz, Rolf Niedermeier, Manuel Sorge, and Toby Walsh. "H-Index Manipulation by Merging Articles: Models, Theory, and Experiments." In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI '15).* AAAI Press, 2015, pp. 808–814 (cit. on p. xvi).

[Bev+16] René van Bevern, Iyad A. Kanj, Christian Komusiewicz, Rolf Niedermeier, and Manuel Sorge. "Twins in Subdivision Drawings of Hypergraphs." In: *24th International Symposium on Graph Drawing and Network Visualization (GD '16)*. Vol. 9801. Lecture Notes in Computer Science. Springer, 2016, pp. 67–80 (cit. on pp. xiv, 67).

[BH03] Gary D Bader and Christopher WV Hogue. "An automated method for finding molecular complexes in large protein interaction networks." In: *BMC Bioinformatics* 4.1 (2003), pp. 1–27 (cit. on p. 97).

[Bha+10] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. "Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph." In: *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC '10)*. ACM, 2010, pp. 201–210 (cit. on p. 106).

[BHB08] Mauro Brunato, Holger H. Hoos, and Roberto Battiti. "On Effectively Finding Maximal Quasi-cliques in Graphs." In: *Proceedings of the 2nd International Conference on Learning and Intelligent Optimization (LION '07)*. Vol. 5313. Lecture Notes in Computer Science. Springer, 2008, pp. 41–55 (cit. on pp. 103, 220).

[BHK15] Sharon Bruckner, Falk Hüffner, and Christian Komusiewicz. "A graph modification approach for finding core-periphery structures in protein interaction networks." In: *Algorithms for Molecular Biology* 10 (1 2015), pp. 1–13 (cit. on p. 169).

[Bie15] Therese Biedl. "On triangulating k-outerplanar graphs." In: *Discrete Applied Mathematics* 181 (2015), pp. 275–279 (cit. on p. 80).

[Bin+12] Daniel Binkele-Raible, Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. "Kernel(s) for problems with no kernel: On out-trees with many leaves." In: *ACM Transactions on Algorithms* 8.4 (2012), 38:1–38:19 (cit. on p. 15).

[BJK14] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. "Kernelization Lower Bounds by Cross-Composition." In: *SIAM Journal on Discrete Mathematics* 28.1 (2014), pp. 277–305 (cit. on p. 14).

[BK] Ivan Blisnetz and Nikolai Karpov. *Parametrisovanie algoritmui vuidelenia plotnuikh component v graphakh*. In Russian, http://mit.spbau.ru/files/Karpov.pdf (cit. on p. 140).

[BKS15] René van Bevern, Christian Komusiewicz, and Manuel Sorge. “Approximation Algorithms for Mixed, Windy, and Capacitated Arc Routing Problems.” In: *Proceedings of the 15th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS ’15)*. Vol. 48. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 130–143 (cit. on p. xvi).

[BL76] Kellogg S. Booth and George S. Lueker. “Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms.” In: *Journal of Computer and System Sciences* 13.3 (1976), pp. 335–379 (cit. on p. 26).

[BLS99] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Vol. 3. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 1999 (cit. on pp. 33, 34, 36, 66).

[Bod+09] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. “On problems without polynomial kernels.” In: *Journal of Computer and System Sciences* 75.8 (2009), pp. 423–434 (cit. on p. 14).

[Bod+16] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. “(Meta) Kernelization.” In: *Journal of the ACM* 63.5 (2016), 44:1–44:69 (cit. on pp. 193, 194).

[Bod09] Hans L. Bodlaender. “Kernelization: New Upper and Lower Bound Techniques.” In: *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC ’09)*. Vol. 5917. Lecture Notes in Computer Science. Springer, 2009, pp. 17–37 (cit. on p. 13).

[Bol06] Béla Bollobás. *The Art of Mathematics*. 1st Edition. Cambridge University Press, 2006, pp. 129–132 (cit. on pp. xiv, 107, 109, 111).

[Bon+15] Edouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Emeric Tourniaire. “Multi-parameter Analysis for Local Graph Partitioning Problems: Using Greediness for Parameterization.” In: *Algorithmica* 71.3 (2015), pp. 566–580 (cit. on pp. 107, 123, 124).

[Bor+14] Endre Boros, Pinar Heggernes, Pim van ’t Hof, and Martin Milanič. “Vector connectivity in graphs.” In: *Networks* 63.4 (2014), pp. 277–285 (cit. on pp. 2, 174–176).

[Bou+13] Nicolas Bourgeois, Aristotelis Giannakos, Giorgio Lucarelli, Ioannis Milis, and Vangelis Th. Paschos. “Exact and Approximation Algorithms for Densest k-Subgraph.” In: *Proceedings of the 7th International Workshop on Algorithms and Computation (WALCOM ’13)*. Vol. 7748. Lecture Notes in Computer Science. Springer, 2013, pp. 114–125 (cit. on p. 110).

[BP13] Balabhaskar Balasundaram and Foad Mahdavi Pajouh. “Graph Theoretic Clique Relaxations and Applications.” In: *Handbook of Combinatorial Optimization*. Springer, 2013, pp. 1559–1598 (cit. on pp. 1, 97, 98, 100, 106, 116, 139).

[Bra+11] Ulrik Brandes, Sabine Cornelsen, Barbara Pampel, and Arnaud Sallaberry. “Blocks of Hypergraphs—Applied to Hypergraphs and Outerplanarity.” In: *Proceedings of the 21st International Workshop on Combinatorial Algorithms (IWOCA ’10)*. Vol. 6460. Lecture Notes in Computer Science. Springer, 2011, pp. 201–211 (cit. on pp. 22, 24, 69, 71, 92).

[Bra+12] Ulrik Brandes, Sabine Cornelsen, Barbara Pampel, and Arnaud Sallaberry. “Path-based supports for hypergraphs.” In: *Journal of Discrete Algorithms* 14 (2012), pp. 248–261 (cit. on pp. 22, 71).

[Bra+98] Andreas Brandstädt, Feodor F. Dragan, Victor Chepoi, and Vitaly I. Voloshin. “Dually Chordal Graphs.” In: *SIAM Journal on Discrete Mathematics* 11.3 (1998), pp. 437–455 (cit. on p. 33).

[Bru+06] Maurizio Bruglieri, Matthias Ehrgott, Horst W. Hamacher, and Francesco Maffioli. “An annotated bibliography of combinatorial optimization problems with fixed cardinality constraints.” In: *Discrete Applied Mathematics* 154.9 (2006), pp. 1344–1357 (cit. on p. 109).

[BT93] Preston Briggs and Linda Torczon. “An Efficient Representation for Sparse Sets.” In: *ACM Letters on Programming Languages and Systems* 2.1-4 (1993), pp. 59–69 (cit. on p. 115).

[BTV11] C. Bujtás, Z. Tuza, and V. Voloshin. “Color-bounded hypergraphs, V: Host graphs and subdivisions.” In: *Discussiones Mathematicae Graph Theory* 31.2 (2011), pp. 223–238 (cit. on p. 23).

[BTY11] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. “Kernel bounds for disjoint cycles and disjoint paths.” In: *Theoretical Computer Science* 412.35 (2011), pp. 4570–4578 (cit. on p. 14).

[Buc+11] Kevin Buchin, Marc J. van Kreveld, Henk Meijer, Bettina Speckmann, and Kevin Verbeek. “On Planar Supports for Hypergraphs.” In: *Journal of Graph Algorithms and Applications* 15.4 (2011), pp. 533–549 (cit. on pp. 2, 22, 24, 36, 69–71, 75).

[BW88] Robert E Bixby and Donald K Wagner. “An almost linear-time algorithm for graph realization.” In: *Mathematics of Operations Research* 13.1 (1988), pp. 99–123 (cit. on pp. 26, 72).

[Cai08] Leizhen Cai. “Parameterized Complexity of Cardinality Constrained Optimization Problems.” In: *The Computer Journal* 51.1 (2008), pp. 102–121 (cit. on pp. 109, 110).

[CCC06] Leizhen Cai, Siu Man Chan, and Siu On Chan. “Random Separation: A New Method for Solving Fixed-Cardinality Optimization Problems.” In: *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC ’06)*. Vol. 4169. Lecture Notes in Computer Science. Springer, 2006, pp. 239–250 (cit. on pp. 109–111, 116, 122).

[CDS04] Vincent Conitzer, Jonathan Derryberry, and Tuomas Sandholm. “Combinatorial Auctions with Structured Item Graphs.” In: *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI ’04)*. AAAI Press / The MIT Press, 2004, pp. 212–218 (cit. on pp. 2, 22, 24, 25, 36).

[CG07] Markus Chimani and Carsten Gutwenger. “Algorithms for the Hypergraph and the Minor Crossing Number Problems.” In: *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC ’07)*. Vol. 4835. Lecture Notes in Computer Science. Springer, 2007, pp. 184–195 (cit. on p. 72).

[Cha+14] Maw-Shang Chang, Li-Hsuan Chen, Ling-Ju Hung, Peter Rossmanith, and Guan-Han Wu. “Exact algorithms for problems related to the densest k-set problem.” In: *Information Processing Letters* 114.9 (2014), pp. 510–513 (cit. on p. 110).

[Cha66] Gary Chartrand. “A Graph-Theoretic Approach to a Communications Problem.” In: *SIAM Journal on Applied Mathematics* 14.4 (1966), pp. 778–781 (cit. on pp. 99, 136).

[Che+05] Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. “Tight lower bounds for certain parameterized NP-hard problems.” In: *Information and Computation* 201.2 (2005), pp. 216–231 (cit. on p. 17).

[Che+13] Jiehua Chen, Christian Komusiewicz, Rolf Niedermeier, Manuel Sorge, Ondřej Suchý, and Mathias Weller. “Effective and Efficient Data Reduction for the Subset Interconnection Design Problem.” In: *Proceedings of the 24th International Symposium on Algorithms and Computation (ISAAC ’13)*. Vol. 8283. Lecture Notes in Computer Science. Springer, 2013, pp. 361–371 (cit. on p. xiv).

[Che+15] Jiehua Chen, Christian Komusiewicz, Rolf Niedermeier, Manuel Sorge, Ondřej Suchý, and Mathias Weller. “Polynomial-Time Data Reduction for the Subset Interconnection Design Problem.” In: *SIAM Journal on Discrete Mathematics* 29.1 (2015), pp. 1–25 (cit. on pp. xiv, 31).

[Chi+16] Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. “Designing FPT Algorithms for Cut Problems Using Randomized Contractions.” In: *SIAM Journal on Computing* 45 (4 2016), pp. 1171–1229 (cit. on p. 177).

[Cho+07] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. “Constructing scalable overlays for pub-sub with many topics.” In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC ’07)*. ACM, 2007, pp. 109–118 (cit. on pp. 2, 21, 24, 25, 32, 33, 36).

[Chu15] Julia Chuzhoy. “Excluded Grid Theorem: Improved and Simplified.” In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC ’15)*. ACM, 2015, pp. 645–654 (cit. on p. 92).

[CMR15] Ferdinando Cicalese, Martin Milanič, and Romeo Rizzi. “On the complexity of the vector connectivity problem.” In: *Theoretical Computer Science* 591 (2015), pp. 60–71 (cit. on pp. 2, 174–176, 192).

[Coo71] Stephen A. Cook. “The Complexity of Theorem-Proving Procedures.” In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC ’71)*. ACM Press, 1971, pp. 151–158 (cit. on p. 9).

[Cor+09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 3rd Edition. MIT Press, 2009 (cit. on p. 11).

[CVJ13] Chen Chen, Roman Vitenberg, and Hans-Arno Jacobsen. “Brief announcement: Constructing fault-tolerant overlay networks for topic-based publish/subscribe.” In: *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC ’13)*. ACM. 2013, pp. 184–186 (cit. on pp. 25, 220).

[Cyg+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015 (cit. on pp. 11, 16).

[DBK07] Pawan Deshpande, Regina Barzilay, and David R. Karger. "Randomized Decoding for Selection-and-Ordering Problems." In: *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL '07)*. 2007, pp. 444–451 (cit. on pp. 108, 120).

[Del14] Holger Dell. "AND-compression of NP-complete Problems: Streamlined Proof and Minor Observations." In: *Proceedings of the 9th International Symposium on Parameterized and Exact Computation (IPEC '14)*. Vol. 8894. Lecture Notes in Computer Science. Springer, 2014, pp. 184–195 (cit. on p. 14).

[DF13] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013 (cit. on pp. 11, 110).

[DF95] Rodney G. Downey and Michael R. Fellows. "Fixed-Parameter Tractability and Completeness II: On Completeness for W[1]." In: *Theoretical Computer Science* 141.1&2 (1995), pp. 109–131 (cit. on p. 110).

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999 (cit. on pp. 3, 11, 12, 29, 141, 144).

[Dic13] Leonard Eugene Dickson. "Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors." In: *American Journal of Mathematics* 35.4 (1913), pp. 413–422 (cit. on p. 29).

[Die10] Reinhard Diestel. *Graph Theory*. 4th Edition. Vol. 173. Graduate Texts in Mathematics. Springer, 2010 (cit. on p. 5).

[DK95] Ding-Zhu Du and Dean F. Kelley. "On Complexity of Subset Interconnection Designs." In: *Journal of Global Optimization* 6.2 (1995), pp. 193–205 (cit. on pp. 23, 32, 33).

[DKS14] Yann Disser, Stefan Kratsch, and Manuel Sorge. "The Minimum Feasible Tileset problem." In: *Proceedings of the 12th Workshop on Approximation and Online Algorithms (WAOA '14)*. Vol. 8952. Lecture Notes in Computer Science. Springer, 2014, pp. 144–155 (cit. on p. xvi).

[DLS14] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. "Kernelization Lower Bounds Through Colors and IDs." In: *ACM Trans. Algorithms* 11.2 (2014), 13:1–13:20. DOI: 10.1145/2650261 (cit. on pp. 215, 217).

[DM88] Ding-Zhu Du and Zevi Miller. “Matroids and subset interconnection design.” In: *SIAM Journal on Discrete Mathematics* 1.4 (1988), pp. 416–424 (cit. on pp. 2, 23, 24, 32, 36).

[Dom09] Michael Dom. “Algorithmic Aspects of the Consecutive-Ones Property.” In: *Bulletin of the EATCS* 98 (2009), pp. 27–59 (cit. on p. 26).

[Dor+10] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. “Efficient Exact Algorithms on Planar Graphs: Exploiting Sphere Cut Decompositions.” In: *Algorithmica* 58.3 (2010), pp. 790–810 (cit. on pp. xiv, 75, 79).

[Dru15] Andrew Drucker. “New Limits to Classical and Quantum Instance Compression.” In: *SIAM Journal on Computing* 44.5 (2015), pp. 1443–1479 (cit. on p. 14).

[Du86] Ding-Zhu Du. “An optimization problem on graphs.” In: *Discrete Applied Mathematics* 14.1 (1986), pp. 101–104 (cit. on pp. 25, 32, 34).

[DW12] Michael Dinitz and Gordon Wilfong. “iBGP and Constrained Connectivity.” In: *Proceedings of the 15th International Workshop Approximation Algorithms for Combinatorial Optimization Problems (APPROX ’12)*. Vol. 7408. Lecture Notes in Computer Science. Springer, 2012, pp. 122–133 (cit. on p. 26).

[Ebl+12] John D. Eblen, Charles A. Phillips, Gary L. Rogers, and Michael A. Langston. “The maximum clique enumeration problem: algorithms, applications, and implementations.” In: *BMC Bioinformatics* 13.10 (2012), pp. 1–11 (cit. on p. 98).

[EGB06] Thomas Eschbach, Wolfgang Günther, and Bernd Becker. “Orthogonal Hypergraph Drawing for Improved Visibility.” In: *Journal of Graph Algorithms and Applications* 10.2 (2006), pp. 141–157 (cit. on p. 68).

[EH66] P. Erdős and A. Hajnal. “On chromatic number of graphs and set-systems.” In: *Acta Mathematica Academiae Scientiarum Hungarica* 17.1-2 (1966), pp. 61–99 (cit. on p. 107).

[Elb15] Khaled M. Elbassioni. “A Polynomial Delay Algorithm for Generating Connected Induced Subgraphs of a Given Cardinality.” In: *Journal of Graph Algorithms and Applications* 19.1 (2015), pp. 273–280 (cit. on pp. 107, 110).

[ELS13] David Eppstein, Maarten Löffler, and Darren Strash. “Listing All Maximal Cliques in Large Sparse Real-World Graphs in Near-Optimal Time.” In: *ACM Journal of Experimental Algorithmics* 18.3 (2013), 3.1:1–3.1:21 (cit. on pp. 2, 101, 107, 137, 143, 220).

[ES12] David Eppstein and Emma S. Spiro. “The h-Index of a Graph and its Application to Dynamic Subgraph Statistics.” In: *Journal of Graph Algorithms and Applications* 16.2 (2012), pp. 543–567 (cit. on pp. 101, 107, 137).

[Fag83] Ronald Fagin. “Acyclic Database Schemes (of Various Degrees): A Painless Introduction.” In: *Proceedings of the 8th Colloquium on Trees in Algebra and Programming (CAAP ’83)*. Vol. 159. Lecture Notes in Computer Science. Springer, 1983, pp. 65–89 (cit. on p. 34).

[Fan+08] Hongbing Fan, Christian Hundt, Yu-Liang Wu, and Jason Ernst. “Algorithms and Implementation for Interconnection Graph Problem.” In: *Proceedings of the 2nd Second International Conference Combinatorial Optimization and Applications (COCOA ’08)*. Vol. 5165. Lecture Notes in Computer Science. Springer, 2008, pp. 201–210 (cit. on pp. 23, 24, 32, 35, 36, 38, 65, 66).

[Fel+09] Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. “On the parameterized complexity of multiple-interval graph problems.” In: *Theoretical Computer Science* 410.1 (2009), pp. 53–61 (cit. on p. 12).

[FFH08] J. Flower, A. Fish, and J. Howse. “Euler diagram generation.” In: *Journal of Visual Languages & Computing* 19.6 (2008), pp. 675–694 (cit. on p. 69).

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006 (cit. on p. 11).

[FJR13] Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. “Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity.” In: *European Journal of Combinatorics* 34.3 (2013), pp. 541–566 (cit. on pp. 102, 136).

[FK10] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010 (cit. on p. 16).

[Flu+15] Till Fluschnik, Stefan Kratsch, Rolf Niedermeier, and Manuel Sorge. “The Parameterized Complexity of the Minimum Shared Edges Problem.” In: *Proceedings of the 35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS ’15)*. Vol. 45. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 448–462 (cit. on p. xvi).

[Fom+10] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. “Bidimensionality and Kernels.” In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA ’10)*. SIAM, 2010, pp. 503–510 (cit. on p. 193).

[Fom+13] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. “Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs.” In: *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS ’13)*. Vol. 20. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 92–103 (cit. on pp. xv, 176, 192, 194, 197).

[Fom+14] Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. “Tight bounds for parameterized complexity of Cluster Editing with a small number of clusters.” In: *Journal of Computer and System Sciences* 80.7 (2014), pp. 1430–1447 (cit. on p. 162).

[FPK01] Uriel Feige, David Peleg, and Guy Kortsarz. “The Dense k-Subgraph Problem.” In: *Algorithmica* 29.3 (2001), pp. 410–421 (cit. on pp. 106, 110).

[Fre82] Eugene C. Freuder. “A Sufficient Condition for Backtrack-Free Search.” In: *Journal of the ACM* 29.1 (1982), pp. 24–32 (cit. on p. 107).

[Fro+16] Vincent Froese, René van Bevern, Rolf Niedermeier, and Manuel Sorge. “Exploiting Hidden Structure in Selecting Dimensions that Distinguish Vectors.” In: *Journal of Computer and System Sciences* 82.3 (2016), pp. 521–535 (cit. on p. xvi).

[FS11] Lance Fortnow and Rahul Santhanam. “Infeasibility of instance compression and succinct PCPs for NP.” In: *Journal of Computer and System Sciences* 77.1 (2011), pp. 91–106 (cit. on p. 14).

[FS97] Uriel Feige and Michael Seltser. *On the densest k-subgraph problem.* Tech. rep. The Weizmann Institute, Department of Applied Math and Computer Science, 1997 (cit. on pp. xiv, 109, 110).

[GG13] Georg Gottlob and Gianluigi Greco. “Decomposing combinatorial auctions and set packing problems.” In: *Journal of the ACM* 60.4 (2013), 24:1–24:39 (cit. on p. 25).

[GGT89] Giorgio Gallo, Michael D. Grigoriadis, and Robert Endre Tarjan. “A Fast Parametric Maximum Flow Algorithm and Applications.” In: *SIAM Journal on Computing* 18.1 (1989), pp. 30–55 (cit. on p. 110).

[GHN04] Jiong Guo, Falk Hüffner, and Rolf Niedermeier. “A Structural View on Parameterizing Problems: Distance from Triviality.” In: *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC ’04)*. Vol. 3162. Lecture Notes in Computer Science. Springer, 2004, pp. 162–173 (cit. on p. 93).

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979 (cit. on pp. 69, 141).

[GN06] Jiong Guo and Rolf Niedermeier. “Exact algorithms and applications for Tree-like Weighted Set Cover.” In: *Journal of Discrete Algorithms* 4.4 (2006), pp. 608–622 (cit. on pp. 23, 34).

[GN07] Jiong Guo and Rolf Niedermeier. “Invitation to data reduction and problem kernelization.” In: *ACM SIGACT News* 38.1 (2007), pp. 31–45 (cit. on p. 13).

[Gol88] Martin Charles Golumbic. “Algorithmic aspects of intersection graphs and representation hypergraphs.” In: *Graphs and Combinatorics* 4.1 (1988), pp. 307–321 (cit. on p. 23).

[GS83] Nathan Goodman and Oded Shmueli. “Syntactic Characterization of Tree Database Schemas.” In: *Journal of the ACM* 30.4 (1983), pp. 767–786 (cit. on p. 66).

[GT08] Qian-Ping Gu and Hisao Tamaki. “Optimal branch-decomposition of planar graphs in $O(n^3)$ Time.” In: *ACM Transactions on Algorithms* 4.3 (2008) (cit. on p. 75).

[Guo+11] Jiong Guo, Iyad A. Kanj, Christian Komusiewicz, and Johannes Uhlmann. “Editing Graphs into Disjoint Unions of Dense Clusters.” In: *Algorithmica* 61.4 (2011), pp. 949–970 (cit. on p. 129).

[Har+00] Erez Hartuv, Armin O Schmitt, Jörg Lange, Sebastian Meier-Ewert, Hans Lehrach, and Ron Shamir. “An algorithm for clustering cDNA fingerprints.” In: *Genomics* 66.3 (2000), pp. 249–256 (cit. on pp. 100, 136).

[Har+15] Sepp Hartung, Christian Komusiewicz, André Nichterlein, and Ondřej Suchý. “On structural parameterizations for the 2-club problem.” In: *Discrete Applied Mathematics* 185 (2015), pp. 79–92 (cit. on p. 103).

[Har+99] Erez Hartuv, Armin O. Schmitt, Jörg Lange, Sebastian Meier-Ewert, Hans Lehrach, and Ron Shamir. *An algorithm for clustering cDNAs for gene expression analysis*. 1999 (cit. on p. 136).

[Har14] Sepp Hartung. “Exploring Parameter Spaces in Coping with Computational Intractability.” PhD Thesis. TU Berlin, Jan. 2014 (cit. on pp. 102, 136).

[Har62] Frank Harary. “The Maximum Connectivity of a Graph.” In: *Proceedings of the National Academy of Science of the United States of America* 48.7 (1962), pp. 1142–1146 (cit. on p. 130).

[HD02] Horst W Hamacher and Zvi Drezner. *Facility Location: Applications and Theory*. Springer, 2002 (cit. on p. 175).

[Her+13] Danny Hermelin, Chien-Chung Huang, Stefan Kratsch, and Magnus Wahlström. “Parameterized Two-Player Nash Equilibrium.” In: *Algorithmica* 65.4 (2013), pp. 802–816 (cit. on p. 107).

[Her+15] Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. “A Completeness Theory for Polynomial (Turing) Kernelization.” In: *Algorithmica* 71.3 (2015), pp. 702–730 (cit. on pp. 15, 215, 217).

[HKS15] Falk Hüffner, Christian Komusiewicz, and Manuel Sorge. “Finding Highly Connected Subgraphs.” In: *Proceedings of the 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM ’15)*. Vol. 8939. Lecture Notes in Computer Science. Springer, 2015, pp. 254–265 (cit. on pp. xv, 135, 140).

[Hol+06] Klaus Holzapfel, Sven Kosub, Moritz G. Maaß, and Hanjo Täubig. “The complexity of detecting fixed-density clusters.” In: *Discrete Applied Mathematics* 154.11 (2006), pp. 1547–1562 (cit. on pp. 106, 110).

[Hol15] Petter Holme. “Modern temporal network theory: A colloquium.” In: *CoRR* abs/1508.01303 (2015) (cit. on p. 220).

[Hos+12] Jun Hosoda, Juraj Hromkovič, Taisuke Izumi, Hirotaka Ono, Monika Steinová, and Koichi Wada. “On the approximability and hardness of minimum topic connected overlay and its special instances.” In: *Theoretical Computer Science* 429 (2012), pp. 144–154 (cit. on pp. 21, 32, 35, 36, 38, 43, 65).

[Hos+15] Jun Hosoda, Juraj Hromkovič, Taisuke Izumi, Hirotaka Ono, Monika Steinová, and Koichi Wada. “Corrigendum to "On the approximability and hardness of minimum topic connected overlay and its special instances" [Theoret. Comput. Sci. 429 (2012) 144–154].” In: *Theoretical Computer Science* 562 (2015), pp. 660–661 (cit. on pp. 35, 43).

[HP91] Peter Hilton and Jean Pedersen. “Catalan Numbers, Their Generalization, and Their Uses.” In: *The Mathematical Intelligencer* 13.2 (1991), pp. 64–75 (cit. on p. 116).

[HPV] Jochen Harant, Anja Pruchnewski, and Margit Voigt. “On Dominating Sets and Independent Sets of Graphs.” In: *Combinatorics, Probability and Computing* 8 (06), pp. 547–553 (cit. on p. 174).

[HPV99] Michel Habib, Christophe Paul, and Laurent Viennot. “Partition Refinement Techniques: An Interesting Algorithmic Tool Kit.” In: *International Journal of Foundations of Computer Science* 10.2 (1999), pp. 147–170 (cit. on pp. 48, 91).

[HS00] Erez Hartuv and Ron Shamir. “A clustering algorithm based on graph connectivity.” In: *Information Processing Letters* 76.4–6 (2000), pp. 175–181 (cit. on pp. 100, 102, 138, 150, 156).

[HS12] Petter Holme and Jari Saramäki. “Temporal networks.” In: *Physics Reports* 519.3 (2012), pp. 97–125 (cit. on p. 220).

[HSP13] Wayne Hayes, Kai Sun, and Nataša Pržulj. “Graphlet-based measures are suitable for biological network comparison.” In: *Bioinformatics* 29.4 (2013), pp. 483–491 (cit. on p. 100).

[HT74] John E. Hopcroft and Robert Endre Tarjan. “Efficient Planarity Testing.” In: *Journal of the ACM* 21.4 (1974), pp. 549–568 (cit. on p. 72).

[Hüf+09] Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. “Isolation concepts for clique enumeration: Comparison and computational experiments.” In: *Theoretical Computer Science* 410.52 (2009), pp. 5384–5397 (cit. on p. 97).

[Hüf+14] Falk Hüffner, Christian Komusiewicz, Adrian Liebtrau, and Rolf Niedermeier. “Partitioning biological networks into highly connected clusters with maximum edge coverage.” In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 11.3 (2014), pp. 455–467 (cit. on pp. xv, 98, 100, 136, 139).

[HWZ08] Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. “Algorithm Engineering for Color-Coding with Applications to Signaling Pathway Detection.” In: *Algorithmica* 52.2 (2008), pp. 114–132 (cit. on pp. 108, 120).

[II09] Hiro Ito and Kazuo Iwama. “Enumeration of isolated cliques and pseudo-cliques.” In: *ACM Transactions on Algorithms* 5.4 (2009), Article 40 (cit. on pp. 134, 137, 140).

[IIO05] Hiro Ito, Kazuo Iwama, and Tsuyoshi Osumi. “Linear-Time Enumeration of Isolated Cliques.” In: *Proceedings of the 13th European Symposium on Algorithms (ESA ’05)*. Vol. 3669. Lecture Notes in Computer Science. Springer, 2005, pp. 119–130 (cit. on pp. 134, 137, 140, 141).

[IP01] Russell Impagliazzo and Ramamohan Paturi. “On the Complexity of k-SAT.” In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 367–375 (cit. on p. 16).

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. “Which Problems Have Strongly Exponential Complexity?” In: *Journal of Computer and System Sciences* 63.4 (2001), pp. 512–530 (cit. on p. 16).

[Jan17] Bart M. P. Jansen. “On Structural Parameterizations of Hitting Set: Hitting Paths in Graphs Using 2-SAT.” In: *Journal of Graph Algorithms and Applications* 21.2 (2017), pp. 219–243 (cit. on pp. 23, 34).

[JP09] Daxin Jiang and Jian Pei. “Mining frequent cross-graph quasi-cliques.” In: *ACM Transactions on Knowledge Discovery from Data* 2.4 (2009), 16:1–16:41 (cit. on p. 106).

[JP87] D. S. Johnson and H. O. Pollak. “Hypergraph planarity and the complexity of drawing Venn diagrams.” In: *Journal of Graph Theory* 11.3 (1987), pp. 309–325 (cit. on pp. 2, 22, 68, 69, 71).

[Juk01] Stasys Jukna. *Extremal Combinatorics - With Applications in Computer Science*. Texts in Theoretical Computer Science. Springer, 2001 (cit. on pp. 176, 193).

[Kan+14] Kustaa Kangas, Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. “On the Number of Connected Sets in Bounded Degree Graphs.” In: *Proceedings of the 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG ’14)*. Vol. 8747. Lecture Notes in Computer Science. Springer, 2014, pp. 336–347 (cit. on p. 109).

[Kar72] Richard M. Karp. “Reducibility Among Combinatorial Problems.” In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York*. The IBM Research Symposia Series. Plenum Press, 1972, pp. 85–103 (cit. on pp. 2, 9).

[Kho04] Subhash Khot. “Ruling Out PTAS for Graph Min-Bisection, Densest Subgraph and Bipartite Clique.” In: *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS ’04)*. IEEE Computer Society, 2004, pp. 136–145 (cit. on p. 110).

[Kim+15] Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. “Linear Kernels and Single-Exponential Algorithms Via Protrusion Decompositions.” In: *ACM Transactions on Algorithms* 12.2 (2015), 21:1–21:41 (cit. on p. 193).

[KKS08] Michael Kaufmann, Marc J. van Kreveld, and Bettina Speckmann. “Subdivision Drawings of Hypergraphs.” In: *Proceedings of the 16th International Symposium on Graph Drawing (GD ’08)*. Vol. 5417. Lecture Notes in Computer Science. Springer, 2008, pp. 396–407 (cit. on pp. 22, 68–70, 75).

[KKV04] Daniel Král, Jan Kratochvíl, and Heinz-Jürgen Voss. “Mixed hypercacti.” In: *Discrete Mathematics* 286.1-2 (2004), pp. 99–113 (cit. on p. 23).

[KMN14] Boris Klemz, Tamara Mchedlidze, and Martin Nöllenburg. “Minimum Tree Supports for Hypergraphs and Low-Concurrency Euler Diagrams.” In: *Proceedings of the 14th Scandinavian Symposium on Algorithm Theory (SWAT ’14)*. Vol. 8503. Lecture Notes in Computer Science. Springer, 2014, pp. 265–276 (cit. on pp. 22, 37, 66, 71).

[KN12] Christian Komusiewicz and Rolf Niedermeier. “New Races in Parameterized Algorithmics.” In: *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS ’12)*. Vol. 7464. Lecture Notes in Computer Science. Springer, 2012, pp. 19–30 (cit. on pp. 102, 136).

[Kom+09] Christian Komusiewicz, Falk Hüffner, Hannes Moser, and Rolf Niedermeier. “Isolation concepts for efficiently enumerating dense subgraphs.” In: *Theoretical Computer Science* 410.38–40 (2009), pp. 3640–3654 (cit. on pp. 97, 134, 137, 140, 141).

[Kom16] Christian Komusiewicz. “Multivariate Algorithmics for Finding Cohesive Subnetworks.” In: *Algorithms* 9.1 (2016), p. 21 (cit. on pp. 102, 139).

[Kos05] Sven Kosub. “Local Density.” In: *Network Analysis*. Vol. 3418. Lecture Notes in Computer Science. Springer, 2005, pp. 112–142 (cit. on pp. 97, 98, 100, 106, 139).

[Kra14] Stefan Kratsch. “Recent developments in kernelization: A survey.” In: *Bulletin of the EATCS* 113 (2014) (cit. on p. 13).

[KS03] Ephraim Korach and Michal Stern. “The clustering matroid and the optimal clustering tree.” In: *Mathematical Programming* 98.1-3 (2003), pp. 385–414 (cit. on pp. 37, 66, 71).

[KS09] Samir Khuller and Barna Saha. “On Finding Dense Subgraphs.” In: *Proceedings of 36th International Colloquium on Automata, Languages and Programming (ICALP ’09)*. Vol. 5555. Lecture Notes in Computer Science. Springer, 2009, pp. 597–608 (cit. on pp. 106, 110).

[KS11] Ján Katrenic and Ingo Schiermeyer. “Improved approximation bounds for the minimum rainbow subgraph problem.” In: *Information Processing Letters* 111.3 (2011), pp. 110–114 (cit. on p. 107).

[KS12] Christian Komusiewicz and Manuel Sorge. “Finding Dense Subgraphs of Sparse Graphs.” In: *Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC ’12)*. Vol. 7535. Lecture Notes in Computer Science. Springer, 2012, pp. 242–251 (cit. on p. xiv).

[KS15a] Christian Komusiewicz and Manuel Sorge. “An Algorithmic Framework for Fixed-Cardinality Optimization in Sparse Graphs Applied to Dense Subgraph Problems.” In: *Discrete Applied Mathematics* 193 (2015), pp. 145–161 (cit. on pp. xv, 105, 109, 110).

[KS15b] Stefan Kratsch and Manuel Sorge. “On Kernelization and Approximation for the Vector Connectivity Problem.” In: *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC ’15)*. Vol. 43. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 377–388 (cit. on p. xvi).

[KS16] Stefan Kratsch and Manuel Sorge. "On Kernelization and Approximation for the Vector Connectivity Problem." In: *Algorithmica* (2016). Online First, pp. 1–43 (cit. on pp. xvi, 173, 177, 181, 218).

[KSS15] Christian Komusiewicz, Manuel Sorge, and Kolja Stahl. "Finding Connected Subgraphs of Fixed Minimum Density: Implementation and Experiments." In: *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA '15)*. Vol. 9125. Lecture Notes in Computer Science. Springer, 2015, pp. 82–93 (cit. on pp. xv, 133, 220).

[KSV05] Antje Krause, Jens Stoye, and Martin Vingron. "Large scale hierarchical clustering of protein sequences." In: *BMC Bioinformatics* 6.1 (2005), pp. 1–12 (cit. on p. 100).

[KT96] Lefteris M. Kirousis and Dimitrios M. Thilikos. "The Linkage of a Graph." In: *SIAM Journal on Computing* 25.3 (1996), pp. 626–647 (cit. on p. 107).

[KW12] Stefan Kratsch and Magnus Wahlström. "Representative Sets and Irrelevant Vertices: New Tools for Kernelization." In: *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '12)*. IEEE Computer Society, 2012, pp. 450–459 (cit. on p. 179).

[LMS11] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. "Lower bounds based on the Exponential Time Hypothesis." In: *Bulletin of the EATCS* 105 (2011), pp. 41–72 (cit. on p. 16).

[Lok09] Daniel Lokshtanov. "New Methods in Parameterized Algorithms and Complexity." PhD Thesis. University of Bergen, Apr. 2009 (cit. on p. 15).

[LW08] Guimei Liu and Limsoon Wong. "Effective Pruning Techniques for Mining Quasi-Cliques." In: *Proceedings of the 2008 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD '08)*. Vol. 5212. Lecture Notes in Computer Science. Springer, 2008, pp. 33–49 (cit. on p. 139).

[Mäk90] Erkki Mäkinen. "How to draw a hypergraph." In: *International Journal of Computer Mathematics* 34 (1990), pp. 178–185 (cit. on pp. 22, 68, 70, 75).

[Mar09] Dániel Marx. "A parameterized view on matroid optimization problems." In: *Theoretical Computer Science* 410.44 (2009), pp. 4471–4479 (cit. on pp. 177, 185, 187).

[Mic16] Othon Michail. “An Introduction to Temporal Graphs: An Algorithmic Perspective.” In: *Internet Mathematics* 12.4 (2016), pp. 239–280 (cit. on p. 220).

[MIH99] Hideo Matsuda, Tatsuya Ishihara, and Akihiro Hashimoto. “Classifying Molecular Sequences Using a Linkage Graph With Their Pairwise Similarities.” In: *Theoretical Computer Science* 210.2 (1999), pp. 305–325 (cit. on p. 139).

[MOR13] Dániel Marx, Barry O’Sullivan, and Igor Razgon. “Finding small separators in linear time via treewidth reduction.” In: *ACM Transactions on Algorithms* 9.4 (2013), 30:1–30:35 (cit. on p. 195).

[MP15] Dániel Marx and Michał Pilipczuk. “Optimal Parameterized Algorithms for Planar Facility Location Problems Using Voronoi Diagrams.” In: *Proceedings of the 23rd European Symposium on Algorithms (ESA ’15)*. Springer, 2015, pp. 865–877 (cit. on pp. 75, 79).

[Mun00] James Munkres. *Topology*. 2nd Edition. Prentice Hall, 2000 (cit. on p. 73).

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006 (cit. on p. 11).

[Nie10] Rolf Niedermeier. “Reflections on Multivariate Algorithmics and Problem Parameterization.” In: *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS ’10)*. Vol. 5. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010, pp. 17–32 (cit. on pp. 102, 136).

[NM12] Jaroslav Nešetril and P Ossona de Mendez. *Sparsity—Graphs, Structures, and Algorithms*. Vol. 28. Algorithms and Combinatorics. Springer, 2012 (cit. on p. 1).

[OR11] Melih Onus and Andréa W. Richa. “Minimum Maximum-Degree Publish-Subscribe Overlay Network Design.” In: *IEEE/ACM Transactions on Networking* 19.5 (2011), pp. 1331–1343 (cit. on pp. 21, 25, 32, 66).

[Oxl11] James Oxley. *Matroid Theory*. Oxford University Press, 2011 (cit. on pp. 186, 187).

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994 (cit. on pp. 9, 10).

[Par+11] Brian J. Parker, Ida Moltke, Adam Roth, Stefan Washietl, Jiayu Wen, Manolis Kellis, Ronald Breaker, and Jakob Skou Pedersen. "New families of human regulatory RNA structures identified by comparative analysis of vertebrate genomes." In: *Genome Research* 21.11 (2011), pp. 1929–1943 (cit. on p. 100).

[Pat+13] Jeffrey Pattillo, Alexander Veremyev, Sergiy Butenko, and Vladimir Boginski. "On the maximum quasi-clique problem." In: *Discrete Applied Mathematics* 161.1-2 (2013), pp. 244–257 (cit. on pp. 99, 110).

[Per68] Hazel Perfect. "Applications of Menger's graph theorem." In: *Journal of Mathematical Analysis and Applications* 22.1 (1968), pp. 96–111 (cit. on p. 187).

[PMB14] Foad Mahdavi Pajouh, Zhuqi Miao, and Balabhaskar Balasundaram. "A branch-and-bound approach for maximum quasi-cliques." In: *Annals of Operations Research* 216.1 (2014), pp. 145–161 (cit. on pp. 99, 133).

[PWJ04] N. Pržulj, D.A. Wigle, and I. Jurisica. "Functional topology in a network of protein interactions." In: *Bioinformatics* 20.3 (2004), pp. 340–348 (cit. on pp. 100, 136).

[PYB13] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. "On clique relaxation models in network analysis." In: *European Journal of Operational Research* 226.1 (2013), pp. 9–18 (cit. on pp. 1, 97–100, 102, 103, 106, 139, 152).

[Rob55] Herbert Robbins. "A remark on Stirling's formula." In: *American Mathematical Monthly* (1955), pp. 26–29 (cit. on p. 17).

[RS12] Neil Robertson and Paul D. Seymour. "Graph Minors. XXII. Irrelevant vertices in linkage problems." In: *Journal of Combinatorial Theory, Series B* 102.2 (2012), pp. 530–563 (cit. on pp. 92, 194).

[SB13] Shahram Shahinpour and Sergiy Butenko. "Distance-Based Clique Relaxations in Networks: s-Clique and s-Club." In: *Proceedings of the Second International Conference on Network Analysis*. Vol. 59. Springer Proceedings in Mathematics & Statistics. Springer, 2013, pp. 149–174 (cit. on pp. 1, 102).

[Sch+12] Alexander Schäfer, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. "Parameterized computational complexity of finding small-diameter subgraphs." In: *Optimization Letters* 6.5 (2012), pp. 883–891 (cit. on pp. 15, 116).

[Sch79] Arnold Schönhage. “On the Power of Random Access Machines.” In: *Proceedings of the 6th International Colloquium on Automata, Languages and Programming (ICALP ’79)*. Vol. 71. Lecture Notes in Computer Science. Springer, 1979, pp. 520–529 (cit. on p. 10).

[Sei83] Stephen B. Seidman. “Network structure and minimum degree.” In: *Social Networks* 5 (3 1983), pp. 269–287 (cit. on pp. 57, 107).

[SF78] Stephen B. Seidman and Brian L. Foster. “A Graph-theoretic Generalization of the Clique Concept.” In: *The Journal of Mathematical Sociology* 6.1 (1978), pp. 139–154 (cit. on p. 100).

[Ski08] Steven Skiena. *The Algorithm Design Manual.* 2nd Edition. Springer, 2008 (cit. on p. 177).

[Sla78] Peter J Slater. “A characterization of SOFT hypergraphs.” In: *Canadian Mathematical Bulletin* 21.3 (1978), pp. 335–337 (cit. on pp. 34, 36).

[Sor+11] Manuel Sorge, René van Bevern, Rolf Niedermeier, and Mathias Weller. “From Few Components to an Eulerian Graph by Adding Arcs.” In: *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG ’11).* Vol. 6986. Lecture Notes in Computer Science. Springer, 2011, pp. 307–318 (cit. on p. xvi).

[Sor+12] Manuel Sorge, René van Bevern, Rolf Niedermeier, and Mathias Weller. “A new view on Rural Postman based on Eulerian Extension and Matching.” In: *Journal of Discrete Algorithms* 16 (2012), pp. 12–33 (cit. on p. xvi).

[Sor+14] Manuel Sorge, Hannes Moser, Rolf Niedermeier, and Mathias Weller. “Exploiting a Hypergraph Model for Finding Golomb Rulers.” In: *Acta Informatica* 51 (2014), pp. 449–471 (cit. on p. xvi).

[Sor13] Manuel Sorge. “A More Complicated Hardness Proof for Finding Densest Subgraphs in Bounded Degree Graphs.” In: *CoRR* abs/1306.6598 (2013) (cit. on p. xiv).

[ST94] Paul D. Seymour and Robin Thomas. “Call Routing and the Ratcatcher.” In: *Combinatorica* 14.2 (1994), pp. 217–241 (cit. on p. 75).

[Sta13] Kolja Stahl. “Algorithm Engineering für das Auffinden dichter Teilgraphen in dünnen Graphen.” Bachelor Thesis. TU Berlin, June 2013 (cit. on pp. xv, 133).

[SZ14] Hadas Shachnai and Meirav Zehavi. “Parameterized Algorithms for Graph Partitioning Problems.” In: *Proceedings of the 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG ’14)*. Vol. 8747. Lecture Notes in Computer Science. Springer, 2014, pp. 384–395 (cit. on pp. 123, 124).

[Tan89] T.-Z. Tang. “A Criterion for a Minimum Feasible Graph.” In: *Mathematica Applicata*. B (2 1989). In Chinese., pp. 21–24 (cit. on pp. 25, 34).

[Tut63] William T Tutte. “How to draw a graph.” In: *Proceedings of the London Mathematical Society* 13.3 (1963), pp. 743–768 (cit. on p. 75).

[TY84] R. Tarjan and M. Yannakakis. “Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs.” In: *SIAM Journal on Computing* 13.3 (1984), pp. 566–579 (cit. on pp. 23, 24, 36, 71).

[Ueh99] Ryuhei Uehara. *The number of connected components in graphs and its applications*. http://www.jaist.ac.jp/~uehara/ps/component.ps.gz. 1999 (cit. on p. 110).

[Ver+14] Alexander Veremyev, Oleg A. Prokopyev, Vladimir Boginski, and Eduardo L. Pasiliao. “Finding maximum subgraphs with relatively large vertex connectivity.” In: *European Journal of Operational Research* 239.2 (2014), pp. 349–362 (cit. on p. 139).

[VLM16] Jordan Viard, Matthieu Latapy, and Clémence Magnien. “Computing maximal cliques in link streams.” In: *Theoretical Computer Science* 609 (2016), pp. 245–252 (cit. on p. 220).

[VV04] Anne Verroust and Marie-Luce Viaud. “Ensuring the Drawability of Extended Euler Diagrams for up to 8 Sets.” In: *Proceedings of the Third International Conference on Diagrammatic Representation and Inference (Diagrams ’04)*. Vol. 2980. Lecture Notes in Computer Science. Springer, 2004, pp. 128–141 (cit. on p. 25).

[Wel13] Mathias Weller. “Aspects of Preprocessing Applied to Combinatorial Graph Problems.” PhD Thesis. TU Berlin, Aug. 2013 (cit. on p. 15).

[WF94] Stanley Wasserman and Katherine Faust. *Social Network Analysis*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994 (cit. on p. 97).

[Woe03] Gerhard J Woeginger. “Exact algorithms for NP-hard problems: A survey.” In: *Combinatorial Optimization–Eureka, You Shrink!* Springer, 2003, pp. 185–207 (cit. on p. 16).

[XF95] Yinfeng Xu and Xiaobing Fu. “On the minimum feasible graph for four sets.” In: *Applied Mathematics - A Journal of Chinese Universities.* B 10 (4 1995), pp. 457–462 (cit. on pp. 25, 34).

[XW05] Rui Xu and Donald Wunsch. “Survey of clustering algorithms.” In: *IEEE Transactions on Neural Networks* 16.3 (2005), pp. 645–678 (cit. on p. 98).

[Yap83] Chee-Keng Yap. “Some Consequences of Non-Uniform Conditions on Uniform Classes.” In: *Theoretical Computer Science* 26 (1983), pp. 287–300 (cit. on p. 14).

[Yu+06] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. “Predicting interactions in protein networks by completing defective cliques.” In: *Bioinformatics* 22.7 (2006), pp. 823–829 (cit. on p. 98).

[Zen+07] Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. “Out-of-core coherent closed quasi-clique mining from large dense graph databases.” In: *ACM Transactions on Database Systems* 32.2 (2007), pp. 1–40 (cit. on p. 97).

Appendix A

Problem compendium

CLIQUE
Input: An undirected graph G and a nonnegative integer k.
Question: Does G have a k-vertex complete subgraph (a clique) as a subgraph?

DENSEST k-SUBGRAPH
Input: An undirected graph $G = (V, E)$, and a nonnegative integer k.
Task: Find a vertex set $S \subseteq V$ of size exactly k such that $G[S]$ has maximum density $|E(G[S])|/\binom{|S|}{2}$.

DOMINATING SET
Input: An undirected graph G and an integer k.
Question: Is there a dominating set of size k in G?

FIXED-CARDINALITY OPTIMIZATION
Input: An undirected graph $G = (V, E)$, an objective function $\phi\colon 2^V \to \mathbb{Q}^+$, and a nonnegative integer k.
Task: Find the maximum of $\phi(S)$ over all sets $S \subseteq V$ such that $|S| = k$.

k-CNF-SATISFIABILITY
Input: A boolean formula ϕ in conjunctive normal form with at most k literals in each clause.
Question: Is there a truth assignment of the variables that makes ϕ true?

HIGHLY CONNECTED SUBGRAPH
Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.
Question: Is there a vertex set $S \subseteq V$ of size exactly k such that $G[S]$ is highly connected?

HITTING SET
Input: A hypergraph $\mathcal{H}$ and a nonnegative integer k.
Question: Is there a set $H \subseteq V(\mathcal{H})$ of size at most k such that $S \cap H \neq \emptyset$ for each hyperedge S in $\mathcal{H}$?

ISOLATED HIGHLY CONNECTED SUBGRAPH
Input: An undirected graph $G = (V, E)$ and nonnegative integers k and γ.
Question: Is there a k-vertex γ-isolated highly connected subgraph contained in G?

μ-CLIQUE
Input: An undirected 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?

MULTICOLORED CLIQUE
Input: A nonnegative integer k and an undirected graph G along with a partition of its vertex set into k independent sets.
Question: Does G have a k-vertex clique as a subgraph?

r-OUTERPLANAR SUPPORT
Input: A connected hypergraph $\mathcal{H}$ and a nonnegative integer r.
Question: Does $\mathcal{H}$ admit an r-outerplanar support?

SET MULTICOVER
Input: A universe U with covering demands $d : U \to \mathbb{N}$, a family $\mathcal{F}$ of subsets of the universe with multiplicity values $m : \mathcal{F} \to \mathbb{N}$, and a nonnegative integer $p \in \mathbb{N}$.
Question: Is there a multiset of at most p subsets from $\mathcal{F}$ that contains each $F \in \mathcal{F}$ at most $m(F)$ times, and covers each $u \in U$ with at least $d(u)$ subsets?

Subset Interconnection Design
Input: A connected hypergraph $\mathcal{H}$ and a nonnegative integer f.
Question: Does $\mathcal{H}$ admit a support with at most $|V(\mathcal{H})| - 1 + f$ edges?

Vector Connectivity
Input: An undirected graph G, nonnegative integers k and d, and a demand function $\lambda\colon V \to \{0, \ldots, d\}$.
Question: Is there a set $S \subseteq V(G)$ of at most k vertices such that for each vertex $v \in V(G) \setminus S$ there are $\lambda(v)$ vertex-disjoint paths from v to S?

Vector d-Connectivity
Input: An undirected graph G, a nonnegative integer k, and a demand function $\lambda\colon V \to \{0, \ldots, d\}$.
Question: Is there a set $S \subseteq V(G)$ of at most k vertices such that for each vertex $v \in V \setminus S$ there are $\lambda(v)$ vertex-disjoint paths from v to S?

Acknowledgments

This thesis is dedicated to my parents, Gabriele and Jörg Sorge. Without their support, it would not have been possible for me to take up the path leading to this work at all. I thank my advisor Rolf Niedermeier for guidance, ideas, encouragement, and patience. I am thankful for having had the opportunity to work in his research group and the numerous further opportunities that this work entailed.

I am grateful to my coauthors and colleagues for many ideas, opportunities to learn from them, and welcome distractions from the day-to-day work. They are, in alphabetical order, René van Bevern, Robert Bredereck, Laurent Bulteau, Jiehua Chen, Yann Disser, Andreas Emil Feldmann, Till Fluschnik, Vincent Froese, Sepp Hartung, Falk Hüffner, Iyad A. Kanj, Christian Komusiewicz, Stefan Kratsch, Hendrik Molter, Hannes Moser, André Nichterlein, Rolf Niedermeier, Kolja Stahl, Ondřej Suchý, Erik Jan van Leeuwen, Toby Walsh, Mathias Weller, and Gerhard J. Woeginger. My special thanks goes out to Christian Komusiewicz and Stefan Kratsch for many valuable discussions.

I am grateful to numerous anonymous referees of my articles who pointed out mistakes, gave interesting facts that improved the results, provided references, and made comments that improved the presentation. I especially thank Rolf Niedermeier, Hans L. Bodlaender, and Danny Hermelin who took the time to review my thesis.

I thank Jiehua Chen for the best encouragement I could dream of, and Jierong Chen for drawing the iconic pictures for each chapter.

Deutsche Forschungsgemeinschaft funded my research within the projects PABI (NI 369/7) and DAPA (NI 369/12) and Technische Universität Berlin provided my working environment.

Universitätsverlag der TU Berlin

Be sparse! Be dense! Be robust!

In this thesis we study the complexity of problems in which we are given a graph or hypergraph and want to produce a (sub)graph that satisfies certain properties. Among others, this graph shall be sparse, dense, or robust! Most considered problems herein are NP-complete and, hence, efficient algorithms for them are unlikely in general. We thus study the parameterized complexity of these problems, where we additionally measure the structure of the input or the structure of the desired output in an integer, called parameter. We aim to obtain efficient algorithms if the parameter is small. In the first part (Be sparse!), we study problems related to network design and graph drawing. We are given a collection of vertex sets (a hypergraph), and we want to find an overlay network that connects each vertex set individually. In the second part (Be dense!), we are given a graph (a social network, for example) and we want to find large cohesive subgroups. In the third and final part (Be robust!), we are given a graph (a road network, for example) and a set of customers. We want to place service points on vertices in that graph such that each customer is connected robustly to the service. In each of the parts, we give provably efficient algorithms, effective data reduction, or derive lower bounds on the running time or effectiveness.

ISBN 978-3-7983-2885-3 (print)
ISBN 978-3-7983-2886-0 (online)

ISBN 978-3-7983-2885-3

http://verlag.tu-berlin.de