FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Generalized and Fractional Hypertree Decompositions

From Theory To Practice

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

DI Wolfgang Fischl, BSc
Matrikelnummer 00602106

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler

Diese Dissertation haben begutachtet:

Gerhard Friedrich	Francesco Scarcello

Wien, 5. Dezember 2018

Wolfgang Fischl

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Generalized and Fractional Hypertree Decompositions

From Theory To Practice

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

DI Wolfgang Fischl, BSc
Registration Number 00602106

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler

The dissertation has been reviewed by:

Gerhard Friedrich

Francesco Scarcello

Vienna, 5th December, 2018

Wolfgang Fischl

Erklärung zur Verfassung der Arbeit

DI Wolfgang Fischl, BSc
Setzgasse 32, 7072 Mörbisch am See

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

Wien, 5. Dezember 2018

Wolfgang Fischl

Acknowledgements

In the first place I want to thank my advisor Reinhard Pichler for his guidance and support during the time I was working on this thesis. I am also very glad that Georg Gottlob helped with ideas and inspiration.

Furthermore, I am grateful for the support of his team at DBAI. In particular, I want to thank all my co-authors, Shqiponja Ahmetaj, Davide M. Longo, Emanuel Sallinger, Mantas Simkus and Sebastian Skritek for the joint work on our papers and for being such good comrades. I am also thankful for having Theresa Csar as my friendly, most of the time quiet, desk neighbour.

Last, but most importantly, my gratitude goes to my friends and my family. To my parents Helga and Martin, who have always encouraged me to pursue my personal goals and made my academic studies possible. To Kerstin, for her patience during the days in which it has not been easy with me. Especially, in situations where academia was more important to me than anything else, I have always encountered tolerance for my work.

Dankeschön.

This research is supported by the Vienna Science and Technology Fund (WWTF), project ICT12-15 and by the Austrian Science Fund (FWF):P25207, P25518, P30930 and Y698.

Kurzfassung

Hyperbaumzerlegungen (hypertree decompositions), verallgemeinerte Hyperbaumzerlungen (generalized hypertree decompositions) und auch fraktionelle Hyperbaumzerlegungen (fractional hypertree decompositions) sind Methoden zum Zerlegen von Hypergraphen, die sehr erfolgreich für das Beantworten von konjunktiven Anfragen (conjunctive queries) und für das Lösen von Bedingungserfüllungsproblemen (constraint satisfaction problems) verwendet werden. Jedem Hypergraph H kann ein Breitenmaß für jede dieser Methoden zugeordnet werden: seine Hyperbaumbreite ($hw(H)$), seine verallgemeinerte Hyperbaumbreite ($ghw(H)$) und seine fraktionelle Hyperbaumbreite ($fhw(H)$). Es ist allgemein bekannt, dass es in polynomieller Zeit möglich ist zu entscheiden, ob $hw(H) \leq k$ für ein fixes k gilt. Im Gegensatz dazu ist das Entscheidungsproblem $ghw \leq k$ für jedes $k \geq 3$ NP-vollständig. Die Komplexitätsklasse des Entscheidungsproblems $fhw \leq k$ für ein fixes k ist seit über einem Jahrzehnt unbekannt.

Wir zeigen in dieser Arbeit, dass dieses Entscheidungsproblem ($fhw \leq k$) für jedes $k \geq 2$ NP-vollständig ist. Mit Hilfe derselben Ideen beweisen wir außerdem die NP-Vollständigkeit des Entscheidungsproblems $ghw \leq k$ für $k = 2$.

Danach suchen wir nach Restriktionen, um die effiziente Berechenbarkeit der verallgemeinerten und fraktionellen Hyperbaumbreite eines Hypergraphen zu ermöglichen. Im Speziellen untersuchen wir Klassen von Hypergraphen, die die beschränkte Kantenschnitteigenschaft (bounded edge intersection property), die beschränkte Multikantenschnitteigenschaft (bounded multi-edge intersection property (BMIP)), die beschränkte Gradeigenschaft (bounded degree property (BDP)), und die beschränkte VC-Dimensions-Eigenschaft besitzen.

Die theoretischen Ergebnisse zu Hyperbaumzerlegungen rücken auch immer mehr in den Fokus von praxisorientierten Projekten. Aus diesem Grund wird eine Sammlung von Hypergraphen sowie eine Bibliothek zur Analyse und zur Zerlegung von eben solchen benötigt.

Wir erfüllen diese Anforderungen in dem wir (i) konkrete Implementierungen von Hyperbaumzerlegungsalgorithmen, (ii) eine neue, umfassende Benchmark von praxisrelevanten Hypergraphen, und (iii) HyperBench, ein neues Webinterface zum Abrufen unserer Benchmarks und unserer Resultate, bereitstellen. Weiters beschreiben wir eine Reihe von Analysen die mit Hilfe dieser Infrastruktur durchgeführt wurden.

Abstract

Hypertree decompositions, as well as the more powerful generalized hypertree decompositions (GHDs), and the yet more general fractional hypertree decompositions (FHD) are hypergraph decomposition methods successfully used for answering conjunctive queries and for solving constraint satisfaction problems. Every hypergraph H has a width relative to each of these decomposition methods: its hypertree width $hw(H)$, its generalized hypertree width $ghw(H)$, and its fractional hypertree width $fhw(H)$, respectively. It is known that $hw(H) \leq k$ can be checked in polynomial time for fixed k, while checking $ghw(H) \leq k$ is NP-complete for $k \geq 3$. The complexity of checking $fhw(H) \leq k$ for a fixed k has been open for over a decade.

We settle this open problem by showing that checking $fhw(H) \leq k$ is NP-complete, even for $k = 2$. The same construction allows us to prove also the NP-completeness of checking $ghw(H) \leq k$ for $k = 2$.

After proving these hardness results, we investigate meaningful restrictions, for which checking for bounded *ghw* and *fhw* is easy. In particular, we study classes of hypergraphs that enjoy the bounded edge-intersection property (BIP), the more general bounded multi-edge intersection property (BMIP), the bounded degree property (BDP) and the bounded VC-dimension.

Given the increasing interest in using such decomposition methods in practice, a publicly accessible repository of decomposition software, as well as a large set of benchmarks, and a web-accessible workbench for inserting, analysing, and retrieving hypergraphs are called for.

We address this need by providing (i) concrete implementations of hypergraph decompositions (including new practical algorithms), (ii) a new, comprehensive benchmark of hypergraphs stemming from disparate CQ and CSP collections, and (iii) HyperBench, our new web-interface for accessing the benchmark and the results of our analyses. In addition, we describe a number of actual experiments we carried out with this new infrastructure.

Contents

CHAPTER 1

Introduction

Conjunctive Queries (CQs) are the most basic and arguably the most important class of queries in the database world. They correspond to select-project-join queries in relational algebra and select-from-where queries in SQL. As such they are a prime target in database research. For this work, we restrict ourselves to the problem of Boolean CQ Evaluation:

Boolean CQ Evaluation	
Input:	A CQ Q over a relational schema $\mathbf{R}$, instance I for $\mathbf{R}$
Output:	Is there at least one solution of Q over I?

Closely related to Boolean CQ Evaluation is the problem of solving *Constraint Satisfaction Problems (CSPs)*. Solving CSPs belongs to the most fundamental problems in Artificial Intelligence. There one asks whether a set of variables can be assigned values, such that all constraints of the CSP are satisfied.

Formally, the two problems, evaluating a CQ and solving a CSP, are essentially the same. They correspond to evaluating a first-order formulae using $\{\exists, \wedge\}$ but disallowing $\{\forall, \vee, \neg\}$ as connectives over a set of finite relations. In practice, CQs have often fewer conjuncts (query atoms) and larger relations, while CSPs have more conjuncts but smaller relations. Unfortunately, these problems are well-known to be NP-complete (query complexity) [24].

As it is the case for most intractable problems, there has been an intensive search for tractable fragments. One such fragment are *acyclic* CQs. A CQ is acyclic if there is an appropriate arrangement of the query atoms in form of a tree. A similar notion, *tree decompositions*, has been developed for graph-based NP-hard problems. Such problems become tractable for classes of instances whose corresponding graphs have bounded *treewidth*. However, the structure of CQ and CSP instances is better described using a

hypergraph rather than a graph. For such, treewidth does not generalize acyclicity, i.e. there are acyclic hypergraphs which have unbounded treewidth.

A *hypergraph* $H = (V(H), E(H))$ corresponding to a CQ (or CSP) Q has as vertex set $V(H)$ the set of variables occurring in Q and for every atom in Q, $E(H)$ contains a hyperedge consisting of all variables occurring in this atom. For this work, and from now on, we shall mainly talk about hypergraphs with the understanding that all our results are equally applicable to CQs and CSPs.

For hypergraphs various decomposition methods have been developed, in particular, *hypertree decompositions (HDs)* [51], the more general *generalized hypertree decompositions (GHDs)* [51], and the yet more general *fractional hypertree decompositions (FHDs)* [58], and corresponding notions of width of a hypergraph H have been defined: the *hypertree width* $hw(H)$, the *generalized hypertree width* $ghw(H)$, and the *fractional hypertree width* $fhw(H)$, where for every hypergraph H, $fhw(H) \leq ghw(H) \leq hw(H)$ holds. Definitions are given in Chapter 2. A number of highly relevant hypergraph-based problems such as Boolean CQ Evaluation and CSP solving become tractable for classes of instances of bounded hw, ghw, or, fhw. For each of the mentioned types of hypergraph decompositions it would thus be most useful to be able to recognize for each constant k whether a given hypergraph H has corresponding width at most k, and if so, to compute such a corresponding decomposition. More formally, for *decomposition* $\in$ {HD, GHD, FHD} and $k > 0$, the main focus of this work is to consider research questions related to the following family of problems:

Check(*decomposition*, k)	
Input:	hypergraph $H = (V, E)$;
Output:	*decomposition* of H of width $\leq k$ if it exists and answer 'no' otherwise.

In the first part of this thesis — Chapter 4 — we will investigate the Check(*decomposition*, k) problem from a theoretical view, whereas in the second part — Chapter 5 — we will answer research questions related to hypergraph decompositions in practice. We will now outline the research questions tackled as part of this thesis.

1.1 Research Questions

The Check(HD,k) problem is known to be in PTime [51] and the Check(GHD,k) problem is known to be NP-complete for $k = 3$ and above[53]. However, little has been known so far about Check(FHD, k). In fact, this has been a long standing open problem. In their 2006 paper, Grohe and Marx state [57]: "It remains an important open question whether there is a polynomial-time algorithm that determines (or approximates) the fractional hypertree width and constructs a corresponding decomposition." The 2014 journal version still mentions this as an open problem and it is conjectured that the

problem might be NP-hard [58]. The open problem is restated in [94], where further evidence for the hardness of the problem is given by showing that "it is not expressible in monadic second-order logic whether a hypergraph has bounded (fractional, generalized) hypertree width". We will tackle this open problem here:

Research Question 1: Is Check(FHD, k) tractable?

Let us now turn to generalized hypertree decompositions. In [51] the complexity of Check(GHD, k) was stated as an open problem. In [53], it was shown that testing $ghw(H) \leq k$ is NP-complete for $k \geq 3$ and, therefore, Check(GHD, k) is NP-complete for $k \geq 3$. For $k = 1$ the problem is trivially tractable because $ghw(H) = 1$ just means H is acyclic. However the case $k = 2$ has been left open. This case is quite interesting, because it was observed that the majority of practical queries from various benchmarks that are not acyclic have $ghw = 2$ [21, 36], and that a decomposition in such cases can be very helpful. Our second research question is to finally settle the complexity of Check(GHD, k) completely.

Research Question 2: Is Check(GHD, 2) tractable?

For those problems which are known to be intractable, for example, Check(GHD, k) for $k \geq 3$, and for those others that will turn out to be intractable, we would like to find large islands of tractability that correspond to meaningful restrictions of the input hypergraph instances. Ideally, such restrictions should fulfil two main criteria: (i) they need to be *realistic* in the sense that they apply to a large number of CQs and/or CSPs in real-life applications, and (ii) they need to be *non-trivial* in the sense that the restriction itself does not already imply bounded *hw*, *ghw*, or *fhw*. Trivial restrictions would be, for example, hypergraph acyclicity or bounded treewidth. Hence, our third research problem is as follows:

Research Question 3: Are there realistic, non-trivial restrictions on hypergraphs which entail the tractability of the Check(*decomp*,k) problem for *decomp* $\in$ {GHD, FHD}?

Where we do not achieve PTime algorithms for the precise computation of a decomposition of optimal width, we would like to find tractable methods for achieving good approximations. Note that for GHDs, the problem of approximations is solved, since $ghw(H) \leq 3 \cdot hw(H) + 1$ holds for every hypergraph H [5]. In contrast, for FHDs, the best known polynomial-time approximation is cubic. More precisely, in [74], a polynomial-time algorithm is presented which, given a hypergraph H with $fhw(H) = k$, computes an FHD of width $\mathcal{O}(k^3)$. We would like to find meaningful restrictions that guarantee significantly tighter approximations in polynomial time. This leads to the fourth research question:

Research Question 4: Are there realistic, non-trivial restrictions on hypergraphs which allow for a PTime computation of good approximations of $fhw(k)$?

So far we have only asked questions related to theoretical properties of hypergraph decompositions. Nevertheless, hypergraph decompositions have also found their way into commercial database systems such as LogicBlox [7, 78, 13, 64, 65] and advanced research prototypes such as EmptyHeaded [1, 3, 93, 79]. They have also been successfully used in the CSP area [6, 61, 68].

Even though such practical applications exist, many questions have remained open. For example, since all algorithms for answering CQs and solving CSPs using HDs, GHDs, or FHDs depend exponentially on the hw, ghw, or fhw, resp., it is important to have instances of "low" hw, ghw, or fhw (say ≤ 5). The goal of the second part of this thesis is to investigate whether real-world CQs and CSPs have exactly this aforementioned property.

Recently, this has been studied for millions of real-world CQs posed at various SPARQL endpoints [80, 21]. Their results show that real-world CQs with atoms of arity ≤ 3 (SPARQL queries have at most 3 variables per atom) have very low hw: the overwhelming majority is acyclic (and therefore $hw = ghw = fhw = 1$); and almost all of the rest have $hw = 2$ (and therefore also $ghw = 2$). It is, however, not clear if CQs with arbitrary arity and CSPs also have low hw, ghw, or fhw, say ≤ 5. Ghionna et al. [43] gave a positive answer to this question for a small set of TPC-H benchmark queries. Hence in the next research question, we aim to significantly extend this collection of CQs and also add CSPs from different areas.

Research Question 5: Is there a comprehensive, easily extensible benchmark of hypergraphs corresponding to CQs or CSPs for the analysis of hypergraph decomposition algorithms?

Clearly, having an answer to Research Question 5, we can use the existing system for solving the Check(HD,k) problem [54, 86] to investigate the hw of the hypergraphs in the newly created benchmark. With these results we will answer our next research question.

Research Question 6: Is the hypertree width of real-world CQs and CSPs small enough (say ≤ 5) to allow for efficient evaluation of CQs of arbitrary arity and of CSPs?

Clearly, since $fhw(H) \leq ghw(H) \leq hw(H)$ holds for any hypergraphs, an answer to above question also implies that $fhw(H) \leq 5$ and $ghw \leq 5$. Since, as mentioned above, answering CQs and solving CSPs depend exponentially on these width measures, having lower ghw and/or fhw might improve practical applications that depend on such decompositions.

Unfortunately, apart from exhaustive search over all possible decomposition trees, no implementations for solving the Check(GHD,k) has been reported yet. For solving the Check(FHD,k) problem an interesting approach has been very recently presented in [35]. There satisfiability modulo theories solving (SMT-solving) has been applied to the Check(FHD, k) problem. Another approach to solve the Check(GHD,k) and/or Check(FHD,k) problem is to apply the results of Research Questions 3 and 4. First, having real-world instances at hand, we want to verify if the restrictions found are indeed realistic:

Research Question 7: Do many instances have the properties that entail the tractability of the Check(GHD,k) and Check(FHD,k) problem?

Second, we want to turn the theoretical tractability using these restrictions into practical algorithms. Unfortunately, many times those do not immediately translate. This has also been the case for the first algorithm solving the Check(HD,k) problem. Only a highly optimized implementation called `det-k-decomp` allowed for an efficient algorithm to solve the Check(HD,k) problem [54]. We will extend this algorithm to implement a system that uses the restrictions of Research Questions 3 and 4 to solve the Check(*decomp*,k) problem for *decomp* $\in$ {GHD,FHD}, which then gives an answer to the next research questions.

Research Question 8: Does the tractable fragment of the Check(*decomp*,k) problem for *decomp* $\in$ {GHD,FHD} indeed allow for efficient algorithms that work well in practice?

Research Question 9: What is the generalized and/or fractional hypertree width of real-world CQs and CSPs?

Finally, our last question revolves around the differences of the different widths. For instance, we know that *hw* is always greater than or equal to the *ghw* and bounded from above by $3 \cdot ghw + 1$. Since, as mentioned above, answering CQs and solving CSPs depend exponentially on these width measures, *ghw* might seem by far better than *hw*. Hence, a natural question is if the two width measures typically do differ by as much as factor 3. Furthermore, we will also integrate the results of [36, 72] to also compare the *fhw* with *hw* and *ghw* and then answer our last question:

Research Question 10: By how much do the different width measures, *hw*, *ghw*, and *fhw*, differ on real-world CQ and CSP instances?

1.2 Main Results

First of all, we have investigated the above mentioned open problem concerning the recognizability of $\mathit{fhw} \leq k$ for fixed k. Our initial hope was to find a simple adaptation of the NP-hardness proof in [53] for recognizing $ghw(H) \leq k$, for $k \geq 3$. Unfortunately, this proof dramatically fails for the fractional case. In fact, the hypergraph-gadgets in that proof are such that both "yes" and "no" instances may yield the same *fhw*. However, via crucial modifications, including the introduction of novel gadgets, we succeed to construct a reduction from 3SAT that allows us to control the *fhw* of the resulting hypergraphs such that those hypergraphs arising from "yes" 3SAT instances have $\mathit{fhw}(H) = 2$ and those arising from "no" instances have $\mathit{fhw}(H) > 2$. Surprisingly, thanks to our new gadgets, the resulting proof is actually significantly simpler than the NP-hardness proof for recognizing $ghw(H) \leq k$ in [53]. We thus obtain the following result:

> **Main Result 1:** Deciding $\mathit{fhw}(H) \leq 2$ for hypergraphs H is NP-complete and, therefore, Check(FHD, k) is intractable even for $k = 2$.

This result can be extended to the NP-hardness of recognizing $\mathit{fhw}(H) \leq k$ for arbitrarily large k. Moreover, the same construction can be used to prove that recognizing ghw ≤ 2 is also NP-hard, thus killing two birds with one stone.

> **Main Result 2:** Deciding $ghw(H) \leq 2$ for hypergraphs H is NP-complete and, therefore, Check(GHD, 2) is intractable even for $k = 2$.

The Main Results 1 and 2 are presented in Section 4.1. These results close some smouldering open problems with bad news. We thus further concentrate on Research Questions 3 and 4 in order to obtain positive results for restricted hypergraph classes.

We first study GHDs, where we succeed to identify very general, realistic, and non-trivial restrictions that make the Check(GHD, k) problem tractable. These results are based on new insights about the differences of GHDs and HDs and the introduction of a novel technique for expanding a hypergraph H to an edge-augmented hypergraph H' s.t. the width k GHDs of H correspond precisely to the width k HDs of H'. The crux here is to find restrictions under which only a polynomial number of edges needs to be added to H to obtain H'. The HDs of H' can then be computed in polynomial time.

In particular, we concentrate on the *bounded intersection property (BIP)*, which, for a class $\mathscr{C}$ of hypergraphs requires that for some constant i, for each pair of distinct edges e_1 and e_2 of each hypergraph $H \in \mathscr{C}$, $|e_1 \cap e_2| \leq i$. We also look at its generalization, the *bounded multi-intersection property (BMIP)*, which, informally, requires for a class $\mathscr{C}$ of hypergraphs that for some constant c and some constant i any intersection of c distinct hyperedges of each hypergraph $H \in \mathscr{C}$ has at most i elements. We obtain the following good news, which are presented in Section 4.2.

Main Result 3: For classes of hypergraphs fulfilling the BIP or BMIP, for every constant k, the problem Check(GHD, k) is tractable. Tractability holds even for classes $\mathscr{C}$ of hypergraphs where for some constant c all intersections of c distinct edges of every $H \in \mathscr{C}$ of size n have $\mathcal{O}(\log n)$ elements. Our complexity analysis reveals that the problem Check(GHD, k) is fixed-parameter tractable w.r.t. parameter i of the BIP.

The tractability proofs for BIP and BMIP do not directly carry over to FHDs. We thus consider the degree d of a hypergraph $H = (V(H), E(H))$, which is defined as the maximum number of hyperedges in which a vertex occurs, i.e., $d = \max_{v \in V(H)} |\{e \in E(H) \mid v \in E(H)\}|$. We say that a class $\mathscr{C}$ of hypergraphs has the *bounded degree property (BDP)*, if there exists $d \geq 1$, such that every hypergraph $H \in \mathscr{C}$ has degree $\leq d$. We obtain the following result, which is presented in Section 4.3.

Main Result 4: For classes of hypergraphs fulfilling the BDP and for every constant k, the problem Check(FHD, k) is tractable.

To get yet bigger tractable classes, we also consider approximations of an optimal FHD. Towards this goal, we study the *fhw* in case of the BIP and we establish an interesting connection between the BMIP and the Vapnik–Chervonenkis dimension (VC-dimension) of hypergraphs. Our research, presented in Section 4.4 is summarized as follows.

Main Result 5: For rather general, realistic, and non-trivial hypergraph restrictions, there exist PTime algorithms that, for hypergraphs H with $\mathit{fhw}(H) = k$, where k is a constant, produce FHDs whose widths are significantly smaller than the best previously known approximation. In particular, the BIP allows us to compute in polynomial time an FHD whose width is $\leq k + \epsilon$ for arbitrarily chosen constant $\epsilon > 0$. The BMIP or bounded VC-dimension allow us to compute in polynomial time an FHD whose width is $\mathcal{O}(k \log k)$.

We then turn our attention also to the optimization problem of fractional hypertree width, i.e., given a hypergraph H, determine $\mathit{fhw}(H)$ and find an FHD of width $\mathit{fhw}(H)$. All our algorithms for the Check(FHD, k) problem have a runtime exponential in the desired width k. Hence, even with the restrictions to the BIP or BMIP we cannot expect an efficient approximation of *fhw* if *fhw* can become arbitrarily large. We will therefore study the following K-Bounded-FHW-Optimization problem for constant $K \geq 1$:

K-Bounded-FHW-Optimization		
Input:	hypergraph $H = (V, E)$;	
Output:	if $\mathit{fhw}(H) \leq K$:	find an FHD $\mathcal{F}$ of H with minimum width;
	otherwise:	answer "$\mathit{fhw}(H) > K$".

For this bounded version of the optimization problem, we will prove the following result:

> **Main Result 6:** There exists a polynomial time approximation scheme (PTAS; for details see Section 4.4) for the K-Bounded-FHW-Optimization problem in case of the BIP for any fixed $K \geq 1$.

These last two results answer Research Question 4 and hence conclude Chapter 4 — "From Theory" — of this thesis. Now, we turn our attention to the computation of the *hw*, *ghw*, and *fhw* of real-world CQs and CSPs in Chapter 5 — "To Practice". In order to get an idea of the *hw*, *ghw*, and *fhw* of various instances of hypergraphs, we gathered real-world and also randomly generated CQs and CSPs. We translated those to hypergraphs and made them publicly available on our HyperBench webpage[1] (see Section 5.1). Hence, as our first more practice oriented contribution:

> **Main Result 7:** We created *HyperBench*, a comprehensive hypergraph benchmark of initially over 3,000 hypergraphs.

This benchmark is exposed by a web interface, which allows the user to retrieve the hypergraphs or groups of hypergraphs together with a broad spectrum of properties of these hypergraphs, such as lower/upper bounds on *hw*, (multi-)intersection width, degree, etc. The insights of Section 5.2 on these properties can be summarized as follows.

> **Main Result 8:** The hypertree width of real-world CQs and CSPs is indeed small enough for most instances. In particular, 1,849 (60%) out of 3,070 instances have $hw \leq 5$.

In Section 5.3 we report on tests with a large number of known CQ and CSP benchmarks and it turns out that a very large number of instances coming from real-life applications enjoy the BIP and a yet more overwhelming number enjoys the BMIP for very low constants c and i.

> **Main Result 9:** The restrictions to BIP, BMIP, BDP, and bounded VC-dimension are indeed realistic for real-world instances of CQs and CSPs (see Section 5.3).

We then extend the software for HD computation from [54] to also solve the Check(GHD,k) problem. For a given hypergraph H, our system first computes the intersection width of H and then applies the *ghw*-algorithm from Section 4.2, which is parameterized by the intersection width. We present in Section 5.4 two different algorithms for this problem.

[1] `http://hyperbench.dbai.tuwien.ac.at`

Main Result 10: The efficient algorithms `GlobalBIP` and `LocalBIP` solve the Check(GHD,k) problem and work reasonably well in practice.

However, all algorithms, also the HD-algorithm from [54], are often only able to give us upper bounds on the real hw and ghw, i.e. for many instances we have that $hw \leq k$ or $ghw \leq k$, but not that $hw > k$ or $ghw > k$. This leaves us with many dark spots on our hw and/or ghw landscape. We therefore make use of a new idea called *balanced separators* to classify in particular "no"-instances of the Check(GHD,k) problem, which immediately entails a "no"-instance of the Check(HD,k) problem.

Main Result 11: For most (60%) CQ and CSP instances $hw = ghw$ of small width ≤ 6. In only 16 out of 1,406 cases, for which we determined the exact hw and ghw, we found an improvement of the width by 1.

The theoretical tractability results for the Check(FHD,k) problem in Sections 4.3 and 4.4 cannot be carried over as easily as the results for the Check(GHD,k) problem. Nevertheless, in Section 5.5 we present an algorithm which solves the Check(FHD,k) problem in PTime given that the arity of the hypergraph is bounded by a constant. This restriction is highly relevant since with the emergence of the Semantic Web lots of queries are nowadays formulated in the SPARQL query language. There all atoms have arity 3, which makes our new algorithm highly applicable.

Main Result 12: The new algorithm k, r`-rank-frac-decomp` solves the Check(FHD,k) for instances of low arity, e.g. SPARQL queries.

Our benchmark contains 70 SPARQL queries and a total of 1,041 other hypergraphs having arity ≤ 5. For these we run the new k, r`-rank-frac-decomp` algorithm.

Main Result 13: For instances with low arity (≤ 5) it holds in 265 (10%) out of 1,111 cases that $fhw < hw$.

We close Chapter 5 in Section 5.6 with a discussion on the differences between hw, ghw, and fhw. As we have already noticed in Main Result 11, we have only found 16 (0.5%) out of 3,070 hypergraphs where $ghw < hw$. For the case of fhw we found 1,271 (41%) out of 3,070 hypergraphs where $fhw < hw$.

1.3 Structure of the Work

In Chapter 2 we introduce basic notions and notations used throughout the thesis. We also review elementary results to be be used in the later chapters.

Chapter 3 discusses the state-of-the-art related to hypertree decompositions. In particular, we review fundamental results on hypertree decompositions and present most recent findings on decomposition methods. Furthermore, we relate several more practical works to our empirical findings.

The main part of the thesis starts with the theoretical aspects of hypertree decompositions in Chapter 4. In particular, we answer Research Questions 1-4. First, we show the NP-completeness of the Check(FHD,k) and Check(GHD,k) problem for $k = 2$ (Main Results 1 and 2). We then prove tractability of the Check(GHD,k) problem for instances having the BMIP (Main Result 3). Furthermore, we prove tractability of the Check(FHD,k) problem for instances having the BDP (Main Result 4) and polynomial approximability for instances having the BIP or BMIP (Main Results 5 and 6).

The practical aspects of hypertree decompositions are then investigated in Chapter 5, where we answer Research Questions 5-10. First, we present the HyperBench benchmark, a collection of over 3,000 hypergraphs, that can be used for the analysis of hypertree decomposition algorithms (Main Result 7). Then, we analyse the hypertree width of these hypergraphs (Main Result 8) and report on different non-trivial hypergraph properties (Main Result 9). Given that the BIP is indeed small for most instances, we develop two algorithms that solve the Check(GHD,k) problem for exactly such instances (Main Result 10). With a new approach we are then able to determine the exact hw for more than half of the hypergraphs in our benchmark (Main Result 11). Finally, we present an algorithm that solves the Check(FHD,k) problem for instances of low arity (Main Result 12 and 13) and compare the different width measures.

We close this thesis in Chapter 6 with a short summary. We also discuss open issues and give an outlook towards future work.

This thesis is based on the following publications:

[36] Wolfgang Fischl, Georg Gottlob, Davide M. Longo and Reinhard Pichler. HyperBench: A Benchmark and Tool for Hypergraphs and Empirical Findings. In *Proceedings of PODS 2019*. To appear.

[38] Wolfgang Fischl, Georg Gottlob and Reinhard Pichler. General and Fractional Hypertree Decompositions: Hard and Easy Cases. In *Proceedings of PODS 2018*, pages 17-32. ACM, 2018.

[37] Wolfgang Fischl, Georg Gottlob, Reinhard Pichler. General and Fractional Hypertree Decompositions: Hard and Easy Cases (Full version). *CoRR*, abs/1611.01090, 2016. `http://arxiv.org/abs/1611.01090`.

CHAPTER 2

Preliminaries

2.1 Hypergraphs

A *hypergraph* is a pair $H = (V(H), E(H))$, consisting of a set $V(H)$ of *vertices* and a set $E(H)$ of *hyperedges* (or, simply *edges*), which are non-empty subsets of $V(H)$. We assume that hypergraphs do not have isolated vertices, i.e. for each $v \in V(H)$, there is at least one edge $e \in E(H)$, s.t. $v \in e$. For a set $C \subseteq V(H)$, we define $\mathit{edges}(C) = \{e \in E(H) \mid e \cap C \neq \emptyset\}$ and for a set $S \subseteq E(H)$, we define $V(S) = \{v \in e \mid e \in S\}$. Actually, for a set S of edges, it is convenient to write $\bigcup S$ (and $\bigcap S$, respectively) to denote the set of vertices obtained by taking the union (or the intersection, respectively) of the edges in S. Hence, we can write $V(S)$ simply as $\bigcup S$.

For a hypergraph H and a set $C \subseteq V(H)$, we say that a pair of vertices $v_1, v_2 \in V(H)$ is $[C]$-adjacent if there exists an edge $e \in E(H)$ such that $\{v_1, v_2\} \subseteq (e \setminus C)$. A $[C]$-path π from v to v' consists of a sequence $v = v_0, \ldots, v_h = v'$ of vertices and a sequence of edges $e_0, \ldots, e_{h-1}$ $(h \geq 0)$ such that $\{v_i, v_{i+1}\} \subseteq (e_i \setminus C)$, for each $i \in [0 \ldots h-1]$. We denote by $V(\pi)$ the set of vertices occurring in the sequence $v_0, \ldots, v_h$. Likewise, we denote by $\mathit{edges}(\pi)$ the set of edges occurring in the sequence $e_0, \ldots, e_{h-1}$. A set $W \subseteq V(H)$ of vertices is $[C]$-connected if $\forall v, v' \in W$ there is a $[C]$-path from v to v'. A $[C]$-component is a maximal $[C]$-connected, non-empty set of vertices $W \subseteq V(H) \setminus C$.

2.2 (Fractional) Edge Covers

Let $H = (V(H), E(H))$ be a hypergraph and consider (edge-weight) functions $\lambda\colon E(H) \to \{0, 1\}$ and $\gamma\colon E(H) \to [0, 1]$. For $\theta \in \{\lambda, \gamma\}$, we denote by $B(\theta)$ the set of all vertices *covered* by θ:

$$B(\theta) = \left\{ v \in V(H) \mid \sum_{e \in E(H), v \in e} \theta(e) \geq 1 \right\}$$

The weight of such a function θ is defined as

$$\mathit{weight}(\theta) = \sum_{e \in E(H)} \theta(e).$$

Following [51], we will sometimes consider λ as a set with $\lambda \subseteq E(H)$ (i.e., the set of edges e with $\lambda(e) = 1$) and the weight of λ as the cardinality of this set. However, for the sake of a uniform treatment with function γ, we shall prefer to treat λ as a function.

Definition 2.1. An *edge cover* of a hypergraph H is a function $\lambda : E(H) \to \{0, 1\}$ such that $V(H) = B(\lambda)$. The *edge cover number* $\rho(H)$ is the minimum weight of all edge covers of H. ◁

Note that the edge cover number can be calculated by the following integer linear program (ILP).

$$\begin{aligned} \text{minimize:} \quad & \sum_{e \in E(H)} \lambda(e) \\ \text{subject to:} \quad & \sum_{e \in E(H), v \in e} \lambda(e) \geq 1, \quad \text{for all } v \in V(H) \\ & \lambda(e) \in \{0, 1\} \qquad \text{for all } e \in E(H) \end{aligned}$$

By substituting all $\lambda(e)$ by $\gamma(e)$ and by relaxing the last condition of the ILP above to $\gamma(e) \geq 0$, we arrive at the linear program (LP) for computing the fractional edge cover number to be defined next. Note that even though our weight function is defined to take values between 0 and 1, we do not need to add $\gamma(e) \leq 1$ as a constraint, because implicitly by the minimization itself the weight on an edge for an edge cover is never greater than 1. Also note that now the program above is an LP, which (in contrast to an ILP) can be solved in PTime.

Definition 2.2. A *fractional edge cover* of a hypergraph $H = (V(H), E(H))$ is a function $\gamma : E(H) \to [0, 1]$ such that $V(H) = B(\gamma)$. The *fractional edge cover number* $\rho^*(H)$ of H is the minimum weight of all fractional edge covers of H. We write $\mathit{supp}(\gamma)$ to denote the *support* of γ, i.e., $\mathit{supp}(\gamma) := \{e \in E(H) \mid \gamma(e) > 0\}$. ◁

Clearly, we have $\rho^*(H) \leq \rho(H)$ for every hypergraph H, and $\rho^*(H)$ can be much smaller than $\rho(H)$. However, below we give an example, which is important for our proof of Theorem 4.1 and where $\rho^*(H)$ and $\rho(H)$ coincide.

Lemma 2.1. *Let K_{2n} be a clique of size $2n$. Then the equalities $\rho(K_{2n}) = \rho^*(K_{2n}) = n$ hold.*

Proof. Since we have to cover each vertex with weight ≥ 1, the total weight on the vertices of the graph is $\geq 2n$. As the weight of each edge adds to the weight of at most 2 vertices, we need at least weight n on the edges to achieve $\geq 2n$ weight on the vertices. On the other hand, we can use n edges each with weight 1 to cover $2n$ vertices. Hence, in total, we get $n \leq \rho^*(K_{2n}) \leq \rho(K_{2n}) \leq n$. □

2.3 HDs, GHDs, and FHDs

We now define three types of hypergraph decompositions:

Definition 2.3. A *generalized hypertree decomposition* (GHD) of a hypergraph $H = (V(H), E(H))$ is a tuple $\left\langle T, (B_u)_{u \in N(T)}, (\lambda_u)_{u \in N(T)} \right\rangle$, such that $T = (N(T), E(T))$ is a rooted tree, for each $u \in T$ the set of vertices $B_u \subseteq V(H)$ and the following conditions hold:

(1) for each $e \in E(H)$, there is a node $u \in N(T)$ with $e \subseteq B_u$;

(2) for each $v \in V(H)$, the set $\{u \in N(T) \mid v \in B_u\}$ is connected in T;

(3) for each $u \in N(T)$, λ_u is a function $\lambda_u \colon E(H) \to \{0, 1\}$ with $B_u \subseteq B(\lambda_u)$. ◁

Let us clarify some notational conventions used throughout this work. To avoid confusion, we will consequently refer to the elements in $V(H)$ as *vertices* (of the hypergraph) and to the elements in $N(T)$ as the *nodes* of T (of the decomposition). Now consider a decomposition $\mathcal{G}$ with tree structure T. For a node u in T, we write T_u to denote the subtree of T rooted at u. By slight abuse of notation, we will often write $u' \in T_u$ to denote that u' is a node in the subtree T_u of T. Moreover, we define $V(T_u) := \bigcup_{u' \in T_u} B_{u'}$ and, for a set $V' \subseteq V(H)$, we define $\mathit{nodes}(V') = \{u \in T \mid B_u \cap V' \neq \emptyset\}$. If we want to make explicit the decomposition $\mathcal{G}$, we also write $\mathit{nodes}(V', \mathcal{G})$ synonymously with $\mathit{nodes}(V')$. By further overloading the *nodes* operator, we also write $\mathit{nodes}(T_u)$ or $\mathit{nodes}(T_u, \mathcal{G})$ to denote the nodes in a subtree T_u of T, i.e., $\mathit{nodes}(T_u) = \mathit{nodes}(T_u, \mathcal{G}) = \{v \mid v \in T_u\}$.

Definition 2.4. A *hypertree decomposition* (HD) of a hypergraph $H = (V(H), E(H))$ is a GHD, which in addition also satisfies the following condition:

(4) for each $u \in N(T)$, $V(T_u) \cap B(\lambda_u) \subseteq B_u$ ◁

Definition 2.5. A *fractional hypertree decomposition* (FHD) [58] of a hypergraph $H = (V(H), E(H))$ is a tuple $\left\langle T, (B_u)_{u \in N(T)}, (\gamma)_{u \in N(T)} \right\rangle$, where conditions (1) and (2) of Definition 2.3 plus condition (3') hold:

(3') for each $u \in N(T)$, γ_u is a function $\gamma_u : E(H) \to [0, 1]$ with $B_u \subseteq B(\gamma_u)$. ◁

The width of a GHD, HD, or FHD is the maximum weight of the functions λ_u or γ_u, respectively, over all nodes u in T. Moreover, the generalized hypertree width, hypertree width, and fractional hypertree width of H (denoted $\mathit{ghw}(H)$, $\mathit{hw}(H)$, $\mathit{fhw}(H)$) is the minimum width over all GHDs, HDs, and FHDs of H, respectively. Condition (2) is called the "connectedness condition", and condition (4) is referred to as "special condition" [51]. The set B_u is often referred to as the "bag" at node u. Note that, strictly speaking, only HDs require that the underlying tree T be rooted. For the sake of a uniform treatment we

assume that also the tree underlying a GHD or an FHD is rooted (with the understanding that the root is arbitrarily chosen).

We now recall two fundamental properties of the various notions of decompositions and width.

Lemma 2.2. *Let H be a hypergraph and let H' be a vertex induced subhypergraph of H, then $hw(H') \leq hw(H)$, $ghw(H') \leq ghw(H)$, and $\mathit{fhw}(H') \leq \mathit{fhw}(H)$ hold.*

Lemma 2.3. *Let H be a hypergraph. If H has a subhypergraph H' such that H' is a clique, then every HD, GHD, or FHD of H has a node u such that $V(H') \subseteq B_u$.*

Strictly speaking, Lemma 2.3 is a well-known property of tree decompositions – independently of the λ- or γ-label.

2.4 Normal Form for FHDs

In Sections 4.3 and 4.4, we will make use of the fractional normal form (FNF), which generalizes the normal form of HDs from [51]. We will show here, that we can transform any FHD into fractional normal form. This transformation follows closely the transformation of HDs into normal form given in [51]. We will first carry over the definition of the normal form of HDs to the fractional setting.

Definition 2.6. An FHD $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ of a hypergraph H is in *fractional normal form (FNF)* if for each node $r \in T$, and for each child s of r, the following conditions hold:

1. there is *exactly one* $[B_r]$-component C_r such that $V(T_s) = C_r \cup (B_r \cap B_s)$ holds;
2. $B_s \cap C_r \neq \emptyset$, where C_r is the $[B_r]$-component satisfying Condition 1;
3. $B(\gamma_s) \cap B_r \subseteq B_s$. ◁

We now carry over several properties of the normal form from [51] to our FNF defined above. An inspection of the corresponding proofs in [51] reveals that these properties hold with minor modifications also in the fractional case. We thus state the following results below without explicitly "translating" the proofs of [51] to the fractional setting.

Note that [51] deals with HDs and, therefore, in all decompositions considered there, the special condition holds. However, for all properties of the normal form carried over from HDs to FHDs in Lemmas 2.4 and 2.5 below, the special condition is not needed.

Lemma 2.4 (Lemma 5.2 from [51])**.** *Consider an arbitrary FHD $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ of a hypergraph H. Let r be a node in T, let s be a child of r and let C be a $[B_r]$-component of H such that $C \cap V(T_s) \neq \emptyset$. Then, $\mathit{nodes}(C, \mathcal{F}) \subseteq \mathit{nodes}(T_s)$.*

Lemma 2.5 (Lemma 5.3 from [51]). *Consider an arbitrary FHD $\mathcal{F} = \langle T, (B_u)_{u\in T}, (\gamma_u)_{u\in T}\rangle$ of a hypergraph H. Let r be a node in T and let $U \subseteq V(H) \setminus B_r$ such that U is $[B_r]$-connected. Then $nodes(U, \mathcal{F})$ induces a (connected) subtree of T.*

We will now show that any FHD $\mathcal{F}$ of width k can be transformed into an FHD of width k in fractional normal form.

Theorem 2.1 (Theorem 5.4 from [51]). *For every FHD $\mathcal{F}$ of a hypergraph H with $width(\mathcal{F}) = k$ there exists an FHD $\mathcal{F}^+$ of H in FNF with $width(\mathcal{F}^+) = k$.*

Remark. The crucial part of the transformation into normal form is to ensure Conditions 1 and 2. Here, the proof of Theorem 5.4 from [51] can be taken over literally because it only makes use of the tree structure of the decomposition, the bags and the connectedness condition. Ensuring also Condition 3 of our FNF is easy, because we may always extend B_s by nodes from $B(\gamma_s) \cap B_r$ without violating the connectedness condition. To make these steps more clear, we will give a detailed proof of Theorem 2.1 here. This proof follows closely the proof of Theorem 5.4 from [51]. Note that it does not change if we consider GHDs.

Proof. Let $\mathcal{F} = \langle T, (B_u)_{u\in T}, (\gamma_u)_{u\in T}\rangle$ be an arbitrary FHD of H of width k. We show how to transform $\mathcal{F}$ into an FHD $\mathcal{F}^+$ of width k in fractional normal form. The proof follows closely the proof of Theorem 5.4 from [51].

Assume that there exist two nodes r and s such that s is a child of r, and s violates any condition of Definition 2.6. If s satisfies Condition (1), but violates Condition (2), then $B_s \subseteq B_r$ holds. In this case, simply eliminate node s from the tree T and attach all children of s to r. It is immediate to see that this transformation preserves all conditions of Definition 2.5.

Now assume that T_s does not meet Condition (1) of Definition 2.6, and let $C_1, \ldots, C_h$ be all the $[r]$-components containing some vertex occurring in $V(T_s)$. Hence, $V(T_s) \subseteq \left(\bigcup_{i=1}^{h} C_i \cup B_r\right)$. For each $[r]$-component C_i $(1 \leq i \leq h)$, consider the set of nodes $nodes(C_i, \mathcal{F})$. By Lemma 2.5, $nodes(C_i, \mathcal{F})$ induces a subtree of T, and by Lemma 2.4, $nodes(C_i, \mathcal{F}) \subseteq nodes(T_s)$. Hence $nodes(C_i, \mathcal{F})$ induces in fact a subtree of T_s.

For each node $n \in nodes(C_i, \mathcal{F})$ define a new node $u_n^{C_i}$, and let $\gamma_{u_n^{C_i}} = \gamma_n$ and $B_{u_n^{C_i}} = B_n \cap (C_i \cup B_r)$. Note that $B_{u_n^{C_i}} \neq \emptyset$, because by definition of $nodes(C_i, \mathcal{F})$, B_n contains some vertex belonging to C_i. Let $N_i = \{u_n^{C_i} \mid n \in nodes(C_i, \mathcal{F})\}$ and, for any C_i $(1 \leq i \leq h)$, let T_i denote the (directed) graph (N_i, E_i) such that $u_p^{C_i}$ is a child of $u_q^{C_i}$ if and only if p is a child of q in T. T_i is clearly isomorphic to the subtree of T_s induced by $nodes(C_i, \mathcal{F})$, hence T_i is a tree as well.

Now transform the FHD $\mathcal{F}$ as follows: Delete every node in $nodes(T_s)$ from T, and attach to r every tree T_i for $1 \leq i \leq h$. Intuitively, we replace the subtree T_s by a set of trees $\{T_1, \ldots, T_h\}$. By construction, T_i contains a node $u_n^{C_i}$ for each node n belonging

to $\mathit{nodes}(C_i, \mathcal{F})$ $(1 \leq i \leq h)$. Then, if we let $\mathit{children}(r)$ denote the set of children of r in the new tree T obtained after the transformation above, it holds that for any $s' \in \mathit{children}(r)$, there exists an $[r]$-component C of H such that $\mathit{nodes}(T_{s'}) = \mathit{nodes}(C, \mathcal{F})$, and $V(T_{s'}) \subseteq (C \cup B_r)$.

Furthermore, it is easy to verify that all the conditions of Definition 2.5 are preserved during this transformation. As a consequence, Condition (2) of Definition 2.5 immediately entails that $(V(T_{s'}) \cap B_r) \subseteq B_{s'}$. Hence, $V(T_{s'}) = C \cup (B_{s'} \cap B_r)$. Thus, any child of r satisfies both Condition (1) and Condition (2) of Definition 2.6.

Now assume that some node $u \in \mathit{children}(r)$ violates Condition (3) of Definition 2.6. Then we add to B_u the set of vertices $B(\gamma_u) \cap B_r$. Because vertices in B_r induce connected subtrees of T, and B_r does not contain any vertices occurring in some $[r]$-component, this further transformation never invalidates any other condition. Moreover, for this transformation, we only use already covered vertices of γ_u and therefore do not change the width of the FHD.

Note that the root of T cannot violate any of the normal form conditions, because it has no parent in T. Moreover, the transformations above never change the parent r of a violating node s. Thus, if we apply such a transformation to the children of the root of T, and iterate the process on the new children of the root of T, and so on, we eventually get a new FHD $\mathcal{F}^+ = \langle T^+, (B_u)_{u \in T^+}, (\gamma_u)_{u \in T^+} \rangle$ of H in fractional normal form. □

CHAPTER 3

State of the Art

3.1 Structural Decomposition Methods

Ever since the NP-completeness of Boolean CQ Evaluation was settled in [24], tractable fragments have been looked for. The most relevant tractable fragments for us are those based on *structural decomposition methods*. There the underlying structure of BCQs (and also CSPs), which are captured by hypergraphs, is decomposed into a tree structure. An overview of the structural decomposition methods presented in this section is given in Table 3.1.

For each of the methods listed in Table 3.1 we are going to discuss three fundamental properties: (P1) *tractable recognizability*, which allows us to efficiently compute such a decomposition, (P2) *tractable query answering*, which allows us to efficiently answer a query using such a decomposition, and (P3) *generalization of acyclicity*, which means that acyclic queries have bounded width. A ✓ indicates in Table 3.1 that the property

	(P1)	(P2)	(P3)
submodular width	?	✗	✓
fractional hypertree width	✗(Sec. 4.1)	✓	✓
generalized hypertree width	✗	✓	✓
subset based widths	✓	✓	✓
hypertree width	✓	✓	✓
query width	✗	✓	✓
α-acyclicity	✓	✓	✓
tree width	✓	✓	✗
fractional edge cover	✓	✓	✗

Table 3.1: Overview of different widths having the properties (P1)-(P3) listed below

holds, a ✗ that the property does not hold, and a **?** that it is not known whether the property holds. Generalizations of α-acyclicity are stacked on top of each other. As we will see, tree width and fractional edge covers do not generalize α-acyclicity.

For uniformity with the definitions introduced in Chapter 1, we will deviate in this section from the standard definitions given in the literature and formulate equivalent ones in the notation introduced here.

Acyclicity. One of the first concepts in database theory that allows for efficient evaluation of CQs is *acyclicity*, which was introduced in [14] and, independently, in [45]. A hypergraph H is *acyclic* if it has a join tree $JT(H)$ [18].

Definition 3.1. A join tree of a hypergraph $H = (V(H), E(H))$ is a tuple $JT(H) = \langle T, (\lambda_u)u \in T\rangle$, s.t. $T = (N(T), E(T))$ is a rooted tree and the following conditions hold:

(1) for each $u \in N(T)$, λ_u is a function $\lambda_u \colon E(H) \to \{0, 1\}$ with $\mathit{weight}(\lambda_u) = 1$

(2) for each $e \in E(H)$, there is a node $u \in N(T)$ with $e = B(\lambda_u)$;

(3) for each $v \in V(H)$, the set $\{u \in N(T) \mid v \in B(\lambda_u)\}$ is connected in T. ◁

Intuitively, a join tree arranges all hyperedges of a hypergraph H, s.t. the nodes where a vertex $X \in V(H)$ occurs induce a connected subtree of T. This notion of acyclicity is the most general one and known as α-acyclicity [33]. The Check(JT) problem can be solved in linear time [91]. The problem of Boolean CQ Evaluation can be solved for acyclic queries using Yannakakis' algorithm in time $\mathcal{O}(m \cdot r \cdot \log r)$ [97], where m is the number of query atoms and r the size of the largest database relation relevant to the query.

It is these two properties:

tractable recognizability and (P1)

tractable query-answering (P2)

that made acyclicity a so-called (accessible) "island of tractability" for the query answering problem [66] — and for equivalent problems, such as conjunctive query containment, constraint satisfaction problems, the homomorphism problem and so on [49, 46].

Unfortunately, many real-world queries (and also CSPs) are not acyclic. Therefore efforts have been made to weaken the notion of acyclicity while still retaining the two previously mentioned important properties. For all these efforts the degree of acyclicity of a hypergraph is often referred to as its *width*.

Tree Decompositions. A well-known notion that generalizes acyclicity on graphs is *bounded treewidth*, which is based on *tree decompositions* [82]. Tree decompositions can be straightforwardly defined for hypergraphs.

Definition 3.2. A *tree decomposition* of a hypergraph $H = (V(H), E(H))$ is a tuple $TD(H) = \langle T, B_{u \in T} \rangle$, s.t. $T = (N(T), E(T))$ is a rooted tree, for each $u \in T$ there is a set $B_u \subseteq V(H)$ and the following conditions hold:

(1) for each $e \in E(H)$, there is a node $u \in N(T)$ with $e \subseteq B_u$;

(2) for each $v \in V(H)$, the set $\{u \in N(T) \mid v \in B_u\}$ is connected in T.

The *width*(TD) of a tree decomposition is $\max_{u \in T} |B_u| - 1$, i.e. the size of the largest bag minus 1 and the *treewidth* $tw(H)$ of a hypergraph H is the minimum width over all its tree decompositions. ◁

The Check(TD,k) problem can be solved in linear time [20], hence tree decompositions can be recognized efficiently. Tree decompositions also allow for tractable query answering. The Boolean CQ Evaluation problem can be solved for queries having treewidth bounded by some constant k in time $\mathcal{O}(m' \cdot D^{k+1} \cdot \log D)$ [46], where m' is the number of vertices of the decomposition tree T, and D is the number of distinct values in the database. Therefore, tree decompositions have been successfully used for conjunctive query answering (or, equivalently, CSP-solving) [29, 25, 67].

However, while it is the case that acyclic graphs have $tw = 1$, it is not the case that acyclic hypergraphs have $tw = 1$. For example, consider the family $(H_n)n \in N$ of hypergraphs having n vertices $\{x_1, \ldots, x_n\}$ and one hyperedge $R(x_1, \ldots, x_n)$. Clearly, for each H_n in this family of hypergraphs even though acyclic, it holds that $tw(H_n) = n - 1$. Therefore, the family $(H_n)n \in N$ of hypergraphs has unbounded treewidth. Hence, a third property is desirable for generalizations of hypergraph acyclicity:

Acyclic hypergraphs have bounded width (P3)

Query Decompositions. A notion that truly generalizes join trees is *query decomposition* [25]. There instead of having each node of a join tree labelled by exactly one hyperedge, we allow for each node to be labelled by multiple hyperedges.

Definition 3.3. A *query decomposition* of a hypergraph $H = (V(H), E(H))$ is a tuple $QD(H) = \langle T, (\lambda_u)_{u \in T} \rangle$, s.t. $T = (N(T), E(T))$ is a rooted tree and the following conditions hold:

(1) for each $u \in N(T)$, λ_u is a function $\lambda_u \colon E(H) \to \{0, 1\}$

(2) for each $e \in E(H)$, there is a node $u \in N(T)$ with $e \subseteq B(\lambda_u)$;

(3) for each $v \in V(H)$, the set $\{u \in N(T) \mid v \in B(\lambda_u)\}$ is connected in T.

The *width*(QD) of a query decomposition is $k = \max_{u \in T} weight(\lambda_u)$, i.e. the weight of the largest edge cover and the *querywidth* $qw(H)$ of a hypergraph H is the minimum width over all its query decompositions. ◁

It is easy to see that join trees are exactly those query decompositions with $width = 1$ and hence, acyclic hypergraphs have $qw = 1$. Additionally, queries having bounded qw can be evaluated efficiently in polynomial time by using a modified version of Yannakakis' algorithm [25]. However, the Check(QD,k) problem is NP-complete [51], and therefore property (P1) is not fulfilled.

(Generalized) Hypertree Decompositions. This is solved by combining the notions of tree decompositions (a bag B_u for vertices) with query decompositions (a cover λ_u for the vertices), which leads to the notions of (generalized) hypertree decompositions [51, 52], see Definitions 2.3 and 2.4. Queries having (generalized) hypertree width k can be answered efficiently in time $\mathcal{O}(n \cdot r^k \cdot \log r)$, where n is the number of variables in the query and r the size of the largest database relation. However, as already mentioned in Section 1.1, the Check(GHD,k) problem is NP-complete [53], whereas the Check(HD,k) problem is tractable [51] and a k-width hypertree decomposition can be computed in time $\mathcal{O}(m^{2k}n^2)$, where m is the number of atoms and v the number of variables in the query [47]. Hence, so far, hypertree width is the most relevant generalization of acyclicity, since it can be recognized efficiently. Additionally, queries (and, equivalently, CSPs) of bounded hypertreewidth can be answered (resp. solved) efficiently in combined complexity.

Improving the width. Since the hypertree width k is in the exponent of the query evaluation algorithm, one might ask whether there is a more general notion of hypertree width that can still be recognized efficiently. In [53] the framework of *subset based decompositions* was presented. The basic idea is to also allow subedges for covering the bags of a tree decomposition. In particular, if we allow all subsets of the hyperedges of H, the notions of *ghw* and *hw* coincide. However, this set might be exponential in the size of the hypergraph. Therefore, based on this idea, decomposition methods were proposed that restrict the number of subedges added to the hypergraph. This includes: *component decomposition* [53], *spread-cut decompositions* [27], and *greedy hypertree decompositions* [55, 56].

In order to make the Check(GHD,k) and Check(FHD,k) problem tractable we take in Sections 4.2-4.4 a different approach than the above mentioned methods. Instead of restricting the decomposition method, we restrict the hypergraphs to have certain properties. For example, for the case of bounded arity (maximum edge size), using the results on greedy hypertree decompositions, it is shown in [56] that the Check(GHD,k) problem is fixed-parameter tractable when parameterized by the arity.

Fractional edge cover and fractional hypertree decompositions. A more radical approach for improving the width was proposed in [10, 58]. There it was shown that the size of the output of a query can be bounded by $r^{\rho^*(H)}$, where r is the size of the largest relation in the database and $\rho^*(H)$ is the fractional edge cover number of the hypergraph H corresponding to the query. Clearly, calculating the fractional edge cover number can be done in polynomial time, the output bound also gives us an algorithm to efficiently solve the query answering problem, but there are acyclic queries that have unbounded fractional edge cover number [58]. For example, consider the class of hypergraphs that

only have disjoint edges. The (generalized) hypertree width of this class is 1, whereas the fractional edge cover number is unbounded [58].

Therefore, in [58] the authors proposed fractional hypertree decompositions, see Definition 2.5. The query answering problem still can be solved in polynomial time, and it now indeed generalizes acyclicity, but, as we will see in Section 4.1, the Check(FHD,k) problem can not be solved efficiently. However, [74] proposes a polynomial time algorithm which, given a hypergraph H with $fhw(H) = k$, computes an FHD of width $\mathcal{O}(k^3)$.

Submodular width. The most general notion of query acyclicity is *submodular width* (*sw*) [75], which in fact also generalizes fractional hypertree width. Still, this notion has two weaknesses. First, unlike for hypertree decompositions, tractable query answering can not be guaranteed. Instead, in [75] Marx shows fixed-parameter tractability in case of bounded submodular width, where the hypergraph is used as parameter. In addition, the decompositions used there are not only dependent on the hypergraph of the query but also on the actual database used. Second, it is not clear how to recognise hypergraphs of low submodular width.

Summary. In this section we have presented results on different structural decomposition methods for hypergraphs most relevant to our work. As it is depicted in Table 3.1, all different width notions that generalize acyclicity can be put into relationship to each other. It is well known, that for all hypergraphs H the following sequence of inequalities holds:

$$sw(H) \leq fhw(H) \leq ghw(H) \leq hw(H) \leq qw(H).$$

Still, many different other methods exist, e.g. *Biconnected Components* [39], Hinge Decompositions [60], *tree projections* [83], and so on. A thorough comparison would go beyond the scope of this work. More details can be found in [48, 50, 46].

3.2 Empirical Results on Hypertree Decompositions

We will now discuss several types of works that are highly relevant to the experimental evaluation in Chapter 5. The work most closely related is the implementation of HD computation by the `DetKDecomp` program reported in [54], which all our algorithms are built upon. The first implementation of an algorithm to compute a k-width HD was called `opt-`k`-decomp` [47, 71, 86]. Both algorithms are exact and may require too much time when dealing with large instances. Therefore several heuristic algorithms have been proposed [84, 90, 77]. Some of them are based on finding tree decompositions (e.g., [28, 19]), where then the bags are covered with as few hyperedges as possible [30]. These methods actually compute GHDs.

The same algorithms via tree decompositions can also be used for finding FHDs of small width. However, to the best of our knowledge, no actual implementations have been proposed yet. A different novel approach is taken in [35]. There an efficient encoding of the check-problem for FHDs to SMT (SAT modulo Theory) was presented. For

the empirical evaluation of the resulting algorithm the hypergraphs of the Hyperbench benchmark, to be presented in Section 5.1, were used.

The second important input to our work comes from the various sources [8, 15, 16, 17, 42, 54, 69, 62, 92] which we took our CQs and CSPs from. A detailed description of those can be found in Section 5.1. Most of those CQs and CSPs have not been analysed with respect to hypertree width before.

To the best of our knowledge, Ghionna et al. [43] presented the first systematic study of HDs of benchmark CQs from TPC-H. There they primarily wanted to find out to what extent HDs can actually speed up query evaluation. They achieved very positive results in this respect, which have recently been confirmed by the work of Perelman et al. [79], Tu et al. [93] and Aberger et al. [2, 1] on query evaluation using FHDs. As a side result, Ghionna et al. also detected that CQs tend to have low hypertree width (a finding which was later confirmed in [21, 80] and also in our study). In a pioneering effort, Bonifati, Martens, and Timm [21] have recently analysed an unprecedented, massive amount of queries: they investigated 180,653,910 queries from (not openly available) query logs of several popular SPARQL endpoints. After elimination of duplicate queries, there were still 56,164,661 queries left, out of which 26,157,880 queries were in fact CQs. The authors thus significantly extend previous work by Picalausa and Vansummeren [80], who analysed 3,130,177 SPARQL queries posed by humans and software robots at the DBPedia SPARQL endpoint. The focus in [80] is on structural properties of SPARQL queries such as keywords used and variable structure in optional patterns. There is one paragraph devoted to CQs, where it is noted that 99.99% of ca. 2 million CQs considered in [80] are acyclic.

Many of the CQs (over 15 million) analysed in [21] have arity 2 (here we consider the maximum arity of all atoms in a CQ as the arity of the query), which means that all triples in such a SPARQL query have a constant at the predicate-position. Bonifati et al. made several interesting observations concerning the shape of these graph-like queries. For instance, they detected that exactly one of these queries has $tw = 3$, while all others have $tw \leq 2$ (and hence $hw \leq 2$). As far as the CQs of arity 3 are concerned (for CQs expressed as SPARQL queries, this is the maximum arity achievable), among many characteristics, also the hypertree width was computed by using the original `DetKDecomp` program from [54]. Out of 6,959,510 CQs of arity 3, only 86 (i.e. 0.01‰) turned out to have $hw = 2$ and 8 queries had ¸$hw = 3$, while all other CQs of arity 3 are acyclic.

For the analysis of CSPs, much less work has been done. Although it has been shown that exploiting (hyper-) tree decompositions may significantly improve the performance of CSP solving [6, 61, 63, 68], a systematic study on the (generalized) hypertree width of CSP instances has only been carried out by few works [54, 68, 87].

To the best of our knowledge, we are the first to analyse the *hw*, *ghw*, and *fhw* of ca. 3,000 CSP instances, where most of these instances have not been studied in this respect before.

CHAPTER 4

From Theory

This chapter constitutes the main part of this thesis. We will answer Research Questions 1-4 and provide some new theoretical insights to generalized and fractional hypertree decomposition. First, we will show in Section 4.1 NP-hardness for the Check(GHD,k) and Check(FHD,k) problem for $k = 2$. We will then introduce classes of hypergraphs for which the problems Check(GHD,k) (in Section 4.2) and Check(FHD,k) (in Sections 4.3 and 4.4) can be solved or at least approximated in polynomial time.

4.1 NP-hardness of Check(*decomp*, k)

The main result in this section is the NP-hardness of Check($decomp, k$) with $decomp \in$ {GHD, FHD} and $k = 2$. At the core of the NP-hardness proof is the construction of a hypergraph H with certain properties. The gadget in Figure 4.1 will play an integral part of this construction.

Lemma 4.1. *Let M_1, M_2 be disjoint sets and $M = M_1 \cup M_2$. Let $H = (V(H), E(H))$ be a hypergraph and $H_0 = (V_0, E_A \cup E_B \cup E_C)$ a subhypergraph of H with $V_0 =$*

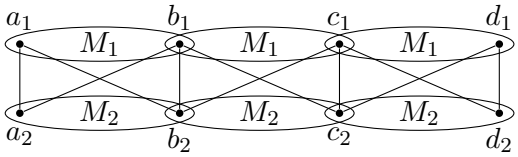

Figure 4.1: Basic structure of H_0 in Lemma 4.1

$\{a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2\} \cup M$ *and*

$$E_A = \{\{a_1, b_1\} \cup M_1, \{a_2, b_2\} \cup M_2, \{a_1, b_2\}, \{a_2, b_1\}, \{a_1, a_2\}\}$$
$$E_B = \{\{b_1, c_1\} \cup M_1, \{b_2, c_2\} \cup M_2, \{b_1, c_2\}, \{b_2, c_1\}, \{b_1, b_2\}, \{c_1, c_2\}\}$$
$$E_C = \{\{c_1, d_1\} \cup M_1, \{c_2, d_2\} \cup M_2, \{c_1, d_2\}, \{c_2, d_1\}, \{d_1, d_2\}\}$$

where no element from the set $R = \{a_2, b_1, b_2, c_1, c_2, d_1, d_2\}$ *occurs in any edge of* $E(H) \setminus (E_A \cup E_B \cup E_C)$*. Then, every FHD* $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ *of width* ≤ 2 *of* H *has nodes* u_A, u_B, u_C *s.t.:*

- $\{a_1, a_2, b_1, b_2\} \subseteq B_{u_A} \subseteq M \cup \{a_1, a_2, b_1, b_2\}$
- $B_{u_B} = \{b_1, b_2, c_1, c_2\} \cup M$*,*
- $\{c_1, c_2, d_1, d_2\} \subseteq B_{u_C} \subseteq M \cup \{c_1, c_2, d_1, d_2\}$*, and*
- u_B *is on the path from* u_A *to* u_C*.*

Proof. Consider an arbitrary FHD $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ of width ≤ 2 of H. Observe that a_1, a_2, b_1, and b_2 form a clique of size 4. Hence, by Lemma 2.3, there is a node u_A in $\mathcal{F}$, such that $\{a_1, a_2, b_1, b_2\} \subseteq B_{u_A}$. It remains to show that also $B_{u_A} \subseteq M \cup \{a_1, a_2, b_1, b_2\}$ holds. To this end, we use a similar reasoning as in the proof of Lemma 2.1: to cover each vertex in $\{a_1, a_2, b_1, b_2\}$, we have to put weight ≥ 1 on each of these 4 vertices. By assumption, the only edges containing 2 out of these 4 vertices are the edges in $E_A \cup \{\{b_1, b_2\}\}$. All other edges in $E(H)$ contain at most 1 out of these 4 vertices. Hence, in order to cover $\{a_1, a_2, b_1, b_2\}$ with weight ≤ 2, we are only allowed to put non-zero weight on the edges in $E_A \cup \{\{b_1, b_2\}\}$. It follows, that $B_{u_A} \subseteq M \cup \{a_1, a_2, b_1, b_2\}$ indeed holds.

Analogously, for the cliques b_1, b_2, c_1, c_2 and c_1, c_2, d_1, d_2, there must exist nodes u_B and u_C in $\mathcal{F}$ with $\{b_1, b_2, c_1, c_2\} \subseteq B_{u_B} \subseteq M \cup \{b_1, b_2, c_1, c_2\}$ and $\{c_1, c_2, d_1, d_2\} \subseteq B_{u_C} \subseteq M \cup \{c_1, c_2, d_1, d_2\}$.

It remains to show that u_B is on the path from u_A to u_C and $B_{u_B} = \{b_1, b_2, c_1, c_2\} \cup M$ holds. We first show that u_B is on the path between u_A and u_C. Suppose to the contrary that it is not. We distinguish two cases. First, assume that u_A is on the path between u_B and u_C. Then, by connectedness, $\{c_1, c_2\} \subseteq B_{u_A}$, which contradicts the property $B_{u_A} \subseteq M \cup \{a_1, a_2, b_1, b_2\}$ shown above. Second, assume u_C is on the path between u_A and u_B. In this case, we have $\{b_1, b_2\} \subseteq B_{u_C}$, which contradicts the property $B_{u_C} \subseteq M \cup \{c_1, c_2, d_1, d_2\}$ shown above.

We now show that also $B_{u_B} = \{b_1, b_2, c_1, c_2\} \cup M$ holds. Since we have already established $\{b_1, b_2, c_1, c_2\} \subseteq B_{u_B} \subseteq M \cup \{b_1, b_2, c_1, c_2\}$, it suffices to show $M \subseteq B_{u_B}$. First, let T'_a be the subgraph of T induced by $\mathit{nodes}(\{a_1, a_2\}, \mathcal{F})$ and let T'_d be the subgraph of T induced by $\mathit{nodes}(\{d_1, d_2\}, \mathcal{F})$. We show that each of the subgraphs T'_a and T'_d is connected (i.e., a subtree of T) and that the two subtrees are disjoint. The connectedness

is immediate: by the connectedness condition, each of $nodes(\{a_1\}, \mathcal{F})$, $nodes(\{a_2\}, \mathcal{F})$, $nodes(\{d_1\}, \mathcal{F})$, and $nodes(\{d_2\}, \mathcal{F})$ is connected. Moreover, since H contains an edge $\{a_1, a_2\}$ (resp. $\{d_1, d_2\}$), the two subtrees induced by $nodes(\{a_1\}, \mathcal{F})$, $nodes(\{a_2\}, \mathcal{F})$ (resp. $nodes(\{d_1\}, \mathcal{F})$, $nodes(\{d_2\}, \mathcal{F})$) must be connected, hence T'_a and T'_d are subtrees of T. It remains to show that T'_a and T'_d are disjoint.

Suppose to the contrary that there exists a node u which is both in T'_a and in T'_d, i.e., $a_i, d_j \in B_u$ for some $i \in \{1, 2\}$ and $j \in \{1, 2\}$. We claim that u must be on the path between u_A and u_C. Suppose it is not. This means that either u_A is on the path between u and u_C or u_C is on the path between u and u_A. In the first case, B_{u_A} has to contain d_j by the connectedness condition, which we have already ruled out above. In the second case, B_{u_C} has to contain a_i, which we have also ruled out above. Hence, u is indeed on the path between u_A and u_C. We have already shown above that also u_B is on the path between u_A and u_C. Hence, there are two cases depending on how u and u_B are arranged on the path between u_A and u_C. First, assume u is on the path between u_A and u_B. In this case, B_{u_B} also contains d_j, which we have ruled out above. Second, assume u is on the path between u_B and u_C. Then B_{u_B} has to contain a_i, which we have also ruled out above. Thus, there can be no node u in T with $a_i, d_j \in B_u$ for some i, j and therefore the subtrees T'_a and T'_d are disjoint and connected by a path containing u_B.

As every edge must be covered, there are nodes in T'_a that cover $\{a_1, b_1\} \cup M_1$ and $\{a_2, b_2\} \cup M_2$, respectively. Hence, the subtree T'_a covers $M = M_1 \cup M_2$, i.e., $M \subseteq \bigcup_{u \in T'_a} B_u$. Likewise, T'_d covers M. Since both subtrees are disjoint and u_B is on the path between them, by the connectedness condition, we have $M \subseteq B_{u_B}$. □

Theorem 4.1. *The* CHECK*(decomp, k) problem is* NP*-complete for decomp* $\in \{$*GHD, FHD*$\}$ *and* $k = 2$.

Proof. The problem is clearly in NP: guess a tree decomposition and check in polynomial time for each node u whether $\rho(B_u) \leq 2$ or $\rho^*(B_u) \leq 2$, respectively, holds. The NP-hardness is proved by a reduction from 3SAT. Before presenting this reduction, we first introduce some useful notation.

Notation. For $i, j \geq 1$, we denote $\{1, \ldots, i\} \times \{1, \ldots, j\}$ by $[i; j]$. For each $p \in [i; j]$, we denote by $p \oplus 1$ ($p \ominus 1$) the successor (predecessor) of p in the usual lexicographic order on pairs, that is, the order $(1, 1), \ldots, (1, j), (2, 1), \ldots, (i, 1), \ldots, (i, j)$. We refer to the first element $(1, 1)$ as min and to the last element (i, j) as max. We denote by $[i; j]^-$ the set $[i; j] \setminus \{\max\}$, i.e. $[i; j]$ without the last element.

Now let $\varphi = \bigwedge_{j=1}^{m}(L_j^1 \vee L_j^2 \vee L_j^3)$ be an arbitrary instance of 3SAT with m clauses and variables $x_1, \ldots, x_n$. From this we will construct a hypergraph $H = (V(H), E(H))$, which consists of two copies H_0, H'_0 of the (sub-)hypergraph H_0 of Lemma 4.1 plus additional edges connecting H_0 and H'_0. We use the sets $Y = \{y_1, \ldots, y_n\}$ and $Y' = \{y'_1, \ldots, y'_n\}$ to encode the truth values of the variables of φ. We denote by Y_l (Y'_l) the set $Y \setminus \{y_l\}$ ($Y' \setminus \{y'_l\}$). Furthermore, we use the sets $A = \{a_p \mid p \in [2n + 3; m]\}$ and $A' = \{a'_p \mid p \in$

$[2n+3; m]\}$, and we define the following subsets of A and A', respectively:

$$A_p = \{a_{\min}, \ldots, a_p\} \qquad \overline{A_p} = \{a_p, \ldots, a_{\max}\}$$
$$A'_p = \{a'_{\min}, \ldots, a'_p\} \qquad \overline{A'_p} = \{a'_p, \ldots, a'_{\max}\}$$

In addition, we will use another set S of elements, that controls and restricts the ways in which edges are combined in a possible FHD or GHD. Such a decomposition will have, implied by Lemma 4.1, two nodes u_B and u'_B such that $S \subseteq B_{u_B}$ and $S \subseteq B_{u'_B}$. From this, we will reason on the path connecting u_B and u'_B.

The concrete set S used in our construction of H is obtained as follows. Let $Q = [2n+3; m] \cup \{(0,1), (0,0), (1,0)\}$, hence Q is an extension of the set $[2n+3; m]$ with special elements $(0,1), (0,0), (1,0)$. Then we define the set S as $S = Q \times \{1,2,3\}$.

The elements in S are pairs, which we denote as $(q \mid k)$. The values $q \in Q$ are themselves pairs of integers (i, j). Intuitively, q indicates the position of a node on the "long" path π in the desired FHD or GHD. The integer k refers to a literal in the j-th clause. We will write the wildcard $*$ to indicate that a component in some element of S can take an arbitrary value. For example, $(\min \mid *)$ denotes the set of tuples $(q \mid k)$ where $q = \min = (1,1)$ and k can take an arbitrary value in $\{1,2,3\}$. We will denote by S_p the set $(p \mid *)$. For instance, $(\min \mid *)$ will be denoted as $S_{\min}$. Further, for $p \in [2n+3; m]$ and $k \in \{1,2,3\}$, we define singletons $S_p^k = \{(p \mid k)\}$.

Problem reduction. Let $\varphi = \bigwedge_{j=1}^{m}(L_j^1 \vee L_j^2 \vee L_j^3)$ be an arbitrary instance of 3SAT with m clauses and variables $x_1, \ldots, x_n$. From this we construct a hypergraph $H = (V(H), E(H))$, that is, an instance of Check$(decomp, k)$ with $decomp \in \{\text{GHD, FHD}\}$ and $k = 2$.

We start by defining the vertex set $V(H)$:

$$V(H) = S \cup A \cup A' \cup Y \cup Y' \cup \{z_1, z_2\} \cup \{a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2, a'_1, a'_2, b'_1, b'_2, c'_1, c'_2, d'_1, d'_2\}.$$

The edges of H are defined in 3 steps. First, we take two copies of the subhypergraph H_0 used in Lemma 4.1:

- Let $H_0 = (V_0, E_0)$ be the hypergraph of Lemma 4.1 with $V_0 = \{a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2\} \cup M_1 \cup M_2$ and $E_0 = E_A \cup E_B \cup E_C$, where we set $M_1 = S \setminus S_{(0,1)} \cup \{z_1\}$ and $M_2 = Y \cup S_{(0,1)} \cup \{z_2\}$.
- Let $H'_0 = (V'_0, E'_0)$ be the corresponding hypergraph, with $V'_0 = \{a'_1, a'_2, b'_1, b'_2, c'_1, c'_2, d'_1, d'_2\} \cup M'_1 \cup M'_2$ and E'_A, E'_B, E'_C are the primed versions of the edge sets $M'_1 = S \setminus S_{(1,0)} \cup \{z_1\}$ and $M'_2 = Y' \cup S_{(1,0)} \cup \{z_2\}$.

In the second step, we define the edges which (as we will see) enforce the existence of a "long" path π between the nodes covering H_0 and the nodes covering H'_0 in any FHD of width ≤ 2.

- $e_p = A'_p \cup \overline{A_p}$, for $p \in [2n+3; m]^-$,
- $e_{y_i} = \{y_i, y'_i\}$, for $1 \leq i \leq n$,
- For $p = (i, j) \in [2n+3; m]^-$ and $k \in \{1, 2, 3\}$:

$$e_p^{k,0} = \begin{cases} \overline{A_p} \cup (S \setminus S_p^k) \cup Y \cup \{z_1\} & \text{if } L_j^k = x_l \\ \overline{A_p} \cup (S \setminus S_p^k) \cup Y_l \cup \{z_1\} & \text{if } L_j^k = \neg x_l, \end{cases}$$

$$e_p^{k,1} = \begin{cases} A'_p \cup S_p^k \cup Y'_l \cup \{z_2\} & \text{if } L_j^k = x_l \\ A'_p \cup S_p^k \cup Y' \cup \{z_2\} & \text{if } L_j^k = \neg x_l. \end{cases}$$

Finally, we need edges that connect H_0 and H'_0 with the above edges covered by the nodes of the "long" path π in a GHD or FHD:

- $e^0_{(0,0)} = \{a_1\} \cup A \cup S \setminus S_{(0,0)} \cup Y \cup \{z_1\}$
- $e^1_{(0,0)} = S_{(0,0)} \cup Y' \cup \{z_2\}$
- $e^0_{\max} = S \setminus S_{\max} \cup Y \cup \{z_1\}$
- $e^1_{\max} = \{a'_1\} \cup A' \cup S_{\max} \cup Y' \cup \{z_2\}$

This concludes the construction of the hypergraph H. Before we prove the correctness of the problem reduction, we give an example that will help to illustrate the intuition underlying this construction.

Example 4.1. Suppose that an instance of 3SAT is given by the propositional formula $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$, i.e.: we have $n = 3$ variables and $m = 2$ clauses. From this we construct a hypergraph $H = (V(H), E(H))$. First, we instantiate the sets Q, A, A', S, Y, and Y' from our problem reduction.

$$\begin{aligned} A &= \{a_{(1,1)}, a_{(1,2)}, a_{(2,1)}, a_{(2,2)}, \ldots, a_{(9,1)}, a_{(9,2)}\}, \\ A' &= \{a'_{(1,1)}, a'_{(1,2)}, a'_{(2,1)}, a'_{(2,2)}, \ldots, a'_{(9,1)}, a'_{(9,2)}\}, \\ Q &= \{(1,1), (1,2), (2,1), (2,2), \ldots, (9,1), (9,2)\} \cup \{(0,1), (0,0), (1,0)\}, \\ S &= Q \times \{1,2,3\}, \\ Y &= \{y_1, y_2, y_3\}, \\ Y' &= \{y'_1, y'_2, y'_3\}. \end{aligned}$$

According to our problem reduction, the set $V(H)$ of vertices of H is

$$\begin{aligned} V(H) = S \cup A \cup A' \cup Y \cup Y' \cup \{z_1, z_2\} \cup \\ \{a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2\} \cup \{a'_1, a'_2, b'_1, b'_2, c'_1, c'_2, d'_1, d'_2\}. \end{aligned}$$

The set $E(H)$ of edges of H is defined in several steps. First, the edges in H_0 and H_0' are defined: We thus have the subsets $E_A, E_B, E_C, E_A', E_B', E_C' \subseteq E(H)$, whose definition is based on the sets $M_1 = S \setminus S_{(0,1)} \cup \{z_1\}$, $M_2 = Y \cup S_{(0,1)} \cup \{z_2\}$, $M_1' = S \setminus S_{(1,0)} \cup \{z_1\}$, and $M_2' = Y' \cup S_{(1,0)} \cup \{z_2\}$. The definition of the edges

$$\begin{aligned}
e_p &= A_p' \cup \overline{A_p} && \text{for } p \in \{(1,1),(1,2),\dots(8,1),(8,2),(9,1)\},\\
e_{y_i} &= \{y_i, y_i'\} && \text{for } 1 \le i \le 3,\\
e^0_{(0,0)} &= \{a_1\} \cup A \cup S \setminus S_{(0,0)} \cup Y \cup \{z_1\},\\
e^1_{(0,0)} &= S_{(0,0)} \cup Y' \cup \{z_2\},\\
e^0_{(9,2)} &= S \setminus S_{(9,2)} \cup Y \cup \{z_1\}, \text{ and}\\
e^1_{(9,2)} &= \{a_1'\} \cup A' \cup S_{(9,2)} \cup Y' \cup \{z_2\}
\end{aligned}$$

is straightforward. We concentrate on the edges $e_p^{k,0}$ and $e_p^{k,1}$ for $p \in \{(1,1),(1,2),\dots(8,1),(8,2),(9,1)\}$ and $k \in \{1,2,3\}$. These edges play the key role for covering the bags of the nodes along the "long" path π in any FHD or GHD of H. This path can be thought of as being structured in 9 blocks. Consider an arbitrary $i \in \{1,\dots,9\}$. Then $e^{k,0}_{(i,1)}$ and $e^{k,1}_{(i,1)}$ encode the k-th literal of the first clause and $e^{k,0}_{(i,2)}$ and $e^{k,1}_{(i,2)}$ encode the k-th literal of the second clause (the latter is only defined for $i \le 8$). These edges are defined as follows: the edges $e^{1,0}_{(i,1)}$ and $e^{1,1}_{(i,1)}$ encode the first literal of the first clause, i.e., the positive literal x_1. We thus have

$$\begin{aligned}
e^{1,0}_{(i,1)} &= \overline{A_{(i,1)}} \cup (S \setminus S^1_{(i,1)}) \cup \{y_1, y_2, y_3\} \cup \{z_1\} \text{ and}\\
e^{1,1}_{(i,1)} &= A'_{(i,1)} \cup S^1_{(i,1)} \cup \{y_2', y_3'\} \cup \{z_2\}
\end{aligned}$$

The edges $e^{2,0}_{(i,1)}$ and $e^{2,1}_{(i,1)}$ encode the second literal of the first clause, i.e., the negative literal $\neg x_2$. Likewise, $e^{3,0}_{(i,1)}$ and $e^{3,1}_{(i,1)}$ encode the third literal of the first clause, i.e., the positive literal x_3. Hence,

$$\begin{aligned}
e^{2,0}_{(i,1)} &= \overline{A_{(i,1)}} \cup (S \setminus S^2_{(i,1)}) \cup \{y_1, y_3\} \cup \{z_1\},\\
e^{2,1}_{(i,1)} &= A'_{(i,1)} \cup S^2_{(i,1)} \cup \{y_1', y_2', y_3'\} \cup \{z_2\}\\
e^{3,0}_{(i,1)} &= \overline{A_{(i,1)}} \cup (S \setminus S^3_{(i,1)}) \cup \{y_1, y_2, y_3\} \cup \{z_1\}, \text{ and}\\
e^{3,1}_{(i,1)} &= A'_{(i,1)} \cup S^3_{(i,1)} \cup \{y_1', y_2'\} \cup \{z_2\}
\end{aligned}$$

Analogously, the edges $e^{1,0}_{(i,2)}$ and $e^{1,1}_{(i,2)}$ (encoding the first literal of the second clause, i.e., $\neg x_1$), the edges $e^{2,0}_{(i,2)}$ and $e^{2,1}_{(i,2)}$ (encoding the second literal of the second clause, i.e., x_2), and the edges $e^{3,0}_{(i,2)}$ and $e^{3,1}_{(i,2)}$ (encoding the third literal of the second clause,

i.e., $\neg x_3$) are defined as follows:

$$\begin{aligned}
e^{1,0}_{(i,2)} &= \overline{A_{(i,2)}} \cup (S \setminus S^1_{(i,2)}) \cup \{y_2, y_3\} \cup \{z_1\},\\
e^{1,1}_{(i,2)} &= A'_{(i,2)} \cup S^1_{(i,2)} \cup \{y'_1, y'_2, y'_3\} \cup \{z_2\},\\
e^{2,0}_{(i,2)} &= \overline{A_{(i,2)}} \cup (S \setminus S^2_{(i,2)}) \cup \{y_1, y_2, y_3\} \cup \{z_1\},\\
e^{2,1}_{(i,2)} &= A'_{(i,2)} \cup S^2_{(i,2)} \cup \{y'_1, y'_3\} \cup \{z_2\}\\
e^{3,0}_{(i,2)} &= \overline{A_{(i,2)}} \cup (S \setminus S^3_{(i,2)}) \cup \{y_1, y_2\} \cup \{z_1\}, \text{ and}\\
e^{3,1}_{(i,2)} &= A'_{(i,2)} \cup S^3_{(i,2)} \cup \{y'_1, y'_2, y'_3\} \cup \{z_2\}.
\end{aligned}$$

The crucial property of these pairs of edges $e^{k,0}_{(i,j)}$ and $e^{k,1}_{(i,j)}$ is that they together encode the k-th literal of the j-th clause in the following way: if the literal is of the form x_l (resp. of the form $\neg x_l$), then $e^{k,0}_{(i,j)} \cup e^{k,1}_{(i,j)}$ covers all of $Y \cup Y'$ except for y'_l (resp. except for y_l).

Formula φ in this example is clearly satisfiable, e.g., by the truth assignment σ with $\sigma(x_1) = \text{true}$ and $\sigma(x_2) = \sigma(x_3) = \text{false}$. Hence, for the problem reduction to be correct, there must exist a GHD (and thus also an FHD) of width 2 of H. In Figure 4.2, the tree structure T plus the bags $(B_t)_{t \in T}$ of such a GHD is displayed. Moreover, in Table 4.1, the precise definition of B_u and λ_u of every node $u \in T$ is given: in the column labelled B_u, the set of vertices contained in B_u for each node $u \in T$ is shown. In the column labelled λ_u, the two edges with weight 1 are shown. For the row with label $u_{p \in [2n+3; m]^-}$, the entry in the last column is $e^{k_p,0}_p, e^{k_p,1}_p$. By this we mean that, for every p, an appropriate value $k_p \in \{1, 2, 3\}$ has to be determined. It will be explained below how to find an appropriate value k_p for each p. The set Z in the bags of this GHD is defined as $Z = \{y_i \mid \sigma(x_i) = \text{true}\} \cup \{y'_i \mid \sigma(x_i) = \text{false}\}$. In this example, for the chosen truth assignment σ, we thus have $Z = \{y_1, y'_2, y'_3\}$. The bags B_t and the edge covers λ_t for each $t \in T$ are explained below.

The nodes u_C, u_B, u_A to cover the edges of the subhypergraph H_0 and the nodes u'_A, u'_B, u'_C to cover the edges of the subhypergraph H'_0 are clear by Lemma 4.1. The purpose of the nodes $u_{\min \ominus 1}$ and $u_{\max}$ is mainly to make sure that each edge $\{y_i, y'_i\}$ is covered by some bag. Recall that the set Z contains exactly one of y_i and y'_i for every i. Hence, the node $u_{\min \ominus 1}$ (resp. $u_{\max}$) covers each edge $\{y_i, y'_i\}$, such that $y'_i \in Z$ (resp. $y_i \in Z$).

We now have a closer look at the nodes $u_{(1,1)}$ to $u_{(9,1)}$ on the "long" path π. More precisely, let us look at the nodes $u_{(i,1)}$ and $u_{(i,2)}$ for some $i \in \{1, \ldots, 8\}$, i.e., the "i-th block". It will turn out that the bags at these nodes can be covered by edges from H because φ is satisfiable. Indeed, our choice of $\lambda_{u_{(i,1)}}$ and $\lambda_{u_{(i,2)}}$ is guided by the literals satisfied by the truth assignment σ, namely: for $\lambda_{u_{(i,j)}}$, we have to choose some k_j, such that the k_j-th literal in the j-th clause is true in σ. For instance, we may define $\lambda_{u_{(i,1)}}$ and $\lambda_{u_{(i,2)}}$ as follows:

$$\lambda_{u_{(i,1)}} = \{e^{1,0}_{(i,1)}, e^{1,1}_{(i,1)}\} \qquad \lambda_{u_{(i,2)}} = \{e^{3,0}_{(i,2)}, e^{3,1}_{(i,2)}\}$$

$u \in T$	B_u	λ_u
u_C	$\{d_1, d_2, c_1, c_2\} \cup Y \cup S \cup \{z_1, z_2\}$	$\{c_1, d_1\} \cup M_1$, $\{c_2, d_2\} \cup M_2$
u_B	$\{c_1, c_2, b_1, b_2\} \cup Y \cup S \cup \{z_1, z_2\}$	$\{b_1, c_1\} \cup M_1$, $\{b_2, c_2\} \cup M_2$
u_A	$\{b_1, b_2, a_1, a_2\} \cup Y \cup S \cup \{z_1, z_2\}$	$\{a_1, b_1\} \cup M_1$, $\{a_2, b_2\} \cup M_2$
$u_{\min \ominus 1}$	$\{a_1\} \cup A \cup Y \cup S \cup Z \cup \{z_1, z_2\}$	$e^0_{(0,0)}, e^1_{(0,0)}$
$u_{p \in [2n+3;m]^-}$	$A'_p \cup \overline{A_p} \cup S \cup Z \cup \{z_1, z_2\}$	$e_p^{k_p,0}, e_p^{k_p,1}$
$u_{\max}$	$\{a'_1\} \cup A' \cup Y' \cup S \cup Z \cup \{z_1, z_2\}$	$e^0_{\max}, e^1_{\max}$
u'_A	$\{a'_1, a'_2, b'_1, b'_2\} \cup Y' \cup S \cup \{z_1, z_2\}$	$\{a'_1, b'_1\} \cup M'_1$, $\{a'_2, b'_2\} \cup M'_2$
u'_B	$\{b'_1, b'_2, c'_1, c'_2\} \cup Y' \cup S \cup \{z_1, z_2\}$	$\{b'_1, c'_1\} \cup M'_1$, $\{b'_2, c'_2\} \cup M'_2$
u'_C	$\{c'_1, c'_2, d'_1, d'_2\} \cup Y' \cup S \cup \{z_1, z_2\}$	$\{c'_1, d'_1\} \cup M'_1$, $\{c'_2, d'_2\} \cup M'_2$

Table 4.1: Definition of B_u and λ_u for GHD of H.

The covers $\lambda_{u_{(i,1)}}$ and $\lambda_{u_{(i,2)}}$ were chosen because the first literal of the first clause and the third literal of the second clause are true in σ. Now let us verify that $\lambda_{u_{(i,1)}}$ and $\lambda_{u_{(i,2)}}$ are indeed covers of $B_{u_{(i,1)}}$ and $B_{u_{(i,2)}}$, respectively. By the definition of the edges $e^{k,0}_{(i,j)}, e^{k,1}_{(i,j)}$ for $j \in \{1,2\}$ and $k \in \{1,2,3\}$, it is immediate that $e^{k,0}_{(i,j)} \cup e^{k,1}_{(i,j)}$ covers $\overline{A_{(i,j)}} \cup A'_{(i,j)} \cup S \cup \{z_1, z_2\}$. The only non-trivial question is if $\lambda_{u_{(i,j)}}$ also covers Z. Recall that by definition, $(e^{1,0}_{(i,1)} \cup e^{1,1}_{(i,1)}) \supseteq (Y \cup Y') \setminus \{y'_1\}$. Our truth assignment σ sets $\sigma(x_1) =$ true. Hence, by our definition of Z, we have $y_1 \in Z$ and $y'_1 \notin Z$. This means that $e^{1,0}_{(i,1)} \cup e^{1,1}_{(i,1)}$ indeed covers Z and, hence, all of $B_{u_{(i,1)}}$. Note that we could have also chosen $\lambda_{u_{(i,1)}} = \{e^{2,0}_{(i,1)}, e^{2,1}_{(i,1)}\}$, since also the second literal of the first clause (i.e., $\neg x_2$) is true in σ. In this case, we would have $(e^{2,0}_{(i,1)} \cup e^{2,1}_{(i,1)}) \supseteq (Y \cup Y') \setminus \{y_2\}$ and Z indeed does not contain y_2. Conversely, setting $\lambda_{u_{(i,1)}} = \{e^{3,0}_{(i,1)}, e^{3,1}_{(i,1)}\}$ would fail, because in this case, $y'_3 \notin (e^{3,0}_{(i,1)} \cup e^{3,1}_{(i,1)})$ since x_3 occurs positively in the first clause. On the other hand, we have $y'_3 \in Z$ by definition of Z, because $\sigma(x_3) =$ false holds.

Checking that $\lambda_{u_{(i,2)}}$ as defined above covers Z is done analogously. Note that in the second clause, only the third literal is satisfied by σ. Hence, setting $\lambda_{u_{(i,2)}} = \{e^{3,0}_{(i,2)}, e^{3,1}_{(i,2)}\}$ is the only option to cover $B_{u_{(i,2)}}$ (in particular, to cover Z). Finally, note that σ as defined above is not the only satisfying truth assignment of φ. For instance, we could have chosen $\sigma(x_1) = \sigma(x_2) = \sigma(x_3) =$ true. In this case, we would define $Z = \{y_1, y_2, y_3\}$ and the covers $\lambda_{u_{(i,j)}}$ would have to be chosen according to an arbitrary choice of one literal per clause that is satisfied by this assignment σ. ◁

To prove the correctness of our problem reduction, we have to show the two equivalences: first, that $ghw(H) \leq 2$ if and only if φ is satisfiable and second, that $fhw(H) \leq 2$ if and only if φ is satisfiable. We prove the two directions of these equivalences separately.

Proof of the "if"-direction. First assume that φ is satisfiable. It suffices to show that then H has a GHD of width ≤ 2, because $fhw(H) \leq ghw(H)$ holds. Let σ be a

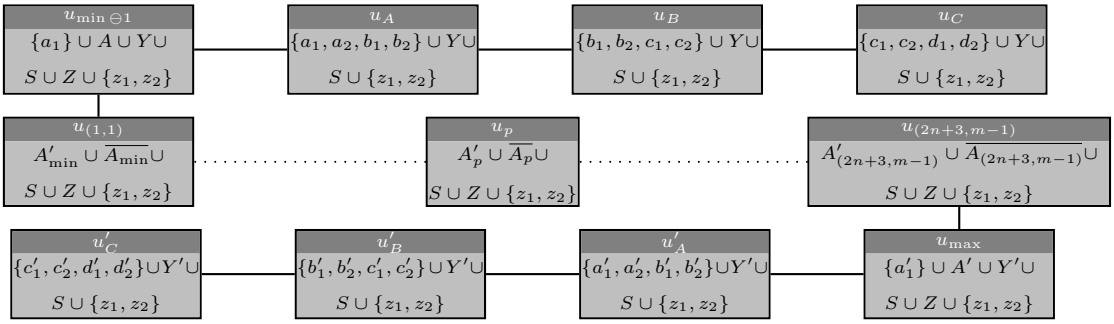

Figure 4.2: Intended path of the GHD of hypergraph H in the proof of Theorem 4.1

satisfying truth assignment. Let us fix for each $j \leq m$, some $k_j \in \{1, 2, 3\}$ such that $\sigma(L_j^{k_j}) = 1$. By l_j, we denote the index of the variable in the literal $L_j^{k_j}$, that is, $L_j^{k_j} = x_{l_j}$ or $L_j^{k_j} = \neg x_{l_j}$. For $p = (i, j)$, let k_p refer to k_j and let $L_p^{k_p}$ refer to $L_j^{k_j}$. Finally, we define Z as $Z = \{y_i \mid \sigma(x_i) = 1\} \cup \{y'_i \mid \sigma(x_i) = 0\}$.

A GHD $\mathcal{G} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ of width 2 for H is constructed as follows. T is a path u_C, u_B, u_A, $u_{\min \ominus 1}$, $u_{\min}$,..., $u_{\max}$, u'_A, u'_B, u'_C. The construction is illustrated in Figure 4.2. The precise definition of B_u and λ_u is given in Table 4.1. Clearly, the GHD has width ≤ 2. We now show that $\mathcal{G}$ is indeed a GHD of H:

(1) For each edge $e \in E$, there is a node $u \in T$, such that $e \subseteq B_u$:

- $\forall e \in E_X : e \subseteq B_{u_X}$ for all $X \in \{A, B, C\}$,
- $\forall e' \in E'_X : e' \subseteq B_{u'_X}$ for all $X \in \{A, B, C\}$,
- $e_p \subseteq B_{u_p}$ for $p \in [2n + 3; m]^-$,
- $e_{y_i} \subseteq B_{u_{\min \ominus 1}}$ (if $y'_i \in Z$) or $e_{y_i} \subseteq B_{u_{\max}}$ (if $y_i \in Z$), respectively,
- $e_p^{k,0} \subseteq B_{u_{\min \ominus 1}}$ for $p \in [2n + 3; m]^-$,
- $e_p^{k,1} \subseteq B_{u_{\max}}$ for $p \in [2n + 3; m]^-$,
- $e^0_{(0,0)} \subseteq B_{u_{\min \ominus 1}}$, $e^1_{(0,0)} \subseteq B_{u_{\max}}$,
- $e^0_{\max} \subseteq B_{u_{\min \ominus 1}}$ and $e^1_{\max} \subseteq B_{u_{\max}}$.

All of the above inclusions can be verified in Table 4.1.

(2) For each vertex $v \in V$, the set $\{u \in T \mid v \in B_u\}$ induces a connected subtree of T, which is easy to verify in Table 4.1.

(3) For each $u \in T$, $B_u \subseteq B(\lambda_u)$: the only inclusion which cannot be easily verified in Table 4.1 is $B_{u_p} \subseteq B(\lambda_{u_p})$. In fact, this is the only place in the proof where we make use of the assumption that φ is satisfiable. First, notice that the set $A'_p \cup \overline{A_p} \cup S \cup \{z_1, z_2\}$ is clearly a subset of $B(\lambda_{u_p})$. It remains to show that

$Z \subseteq B(\lambda_{u_p})$ holds for arbitrary $p \in [2n+3; m]^-$. We show this property by a case distinction on the form of $L_p^{k_p}$.

Case (1): First, assume that $L_p^{k_p} = x_{l_j}$ holds. Then $\sigma(x_{l_j}) = 1$ and, therefore, $y'_{l_j} \notin Z$. But, by definition of $e_p^{k_p,0}$ and $e_p^{k_p,1}$, vertex y'_{l_j} is the only element of $Y \cup Y'$ not contained in $B(\lambda_{u_p})$. Since $Z \subseteq (Y \cup Y')$ and $y'_{l_j} \notin Z$, we have that $Z \subseteq B(\lambda_{u_p})$.

Case (2): Now assume that $L_p^{k_p} = \neg x_{l_j}$ holds. Then $\sigma(x_{l_j}) = 0$ and, therefore, $y_{l_j} \notin Z$. But, by definition of $e_p^{k_p,0}$ and $e_p^{k_p,1}$, vertex y_{l_j} is the only element of $Y \cup Y'$ not contained in $B(\lambda_{u_p})$. Since $Z \subseteq (Y \cup Y')$ and $y_{l_j} \notin Z$, we have that $Z \subseteq B(\lambda_{u_p})$.

Two crucial lemmas. Before we prove the "only if'-direction, we define the notion of complementary edges and state two important lemmas related to this notion.

Definition 4.1. Let e and e' be two edges from the hypergraph H as defined before. We say e' is the *complementary* edge of e (or, simply, e, e' are complementary edges) whenever

- $e \cap S = S \setminus S'$ for some $S' \subseteq S$ and
- $e' \cap S = S'$. ◁

Observe that for every edge in our construction that covers $S \setminus S'$ for some $S' \subseteq S$ there is a complementary edge that covers S', for example $e_p^{k,0}$ and $e_p^{k,1}$, $e_{(0,0)}^0$ and $e_{(0,0)}^1$, and so on. In particular there is no edge that covers S completely. Moreover, consider arbitrary subsets S_1, S_2 of S, s.t. (syntactically) $S \setminus S_i$ is part of the definition of e_i for some $e_i \in E(H)$ with $i \in \{1, 2\}$. Then S_1 and S_2 are disjoint.

We now present two lemmas needed for the "only if"-direction.

Lemma 4.2. *Let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be an FHD of width ≤ 2 of the hypergraph H constructed above. For every node u with $S \cup \{z_1, z_2\} \subseteq B_u$ and every pair e, e' of complementary edges, it holds that $\gamma_u(e) = \gamma_u(e')$.*

Proof. First, we try to cover z_1 and z_2. For z_1 we have to put total weight 1 on the edges in E^0, and to cover z_2 we have to put total weight 1 on the edges in E^1, where

$$\begin{aligned} E^0 = &\{e_p^{k,0} \mid p \in [2n+3; m]^- \text{ and } 1 \leq k \leq 3\} \cup \\ &\{e_{(0,0)}^0, e_{\max}^0\} \cup \\ &\{\{a_1, b_1\} \cup M_1, \{b_1, c_1\} \cup M_1, \{c_1, d_1\} \cup M_1\} \cup \\ &\{\{a'_1, b'_1\} \cup M'_1, \{b'_1, c'_1\} \cup M'_1, \{c'_1, d'_1\} \cup M'_1\} \end{aligned}$$

$$\begin{aligned} E^1 = &\{e_p^{k,1} \mid p \in [2n+3; m]^- \text{ and } 1 \leq k \leq 3\} \cup \\ &\{e^1_{(0,0)}, e^1_{\max}\} \cup \\ &\{\{a_2, b_2\} \cup M_2, \{b_2, c_2\} \cup M_2, \{c_2, d_2\} \cup M_2\} \cup \\ &\{\{a'_2, b'_2\} \cup M'_2, \{b'_2, c'_2\} \cup M'_2, \{c'_2, d'_2\} \cup M'_2\} \end{aligned}$$

In order to also cover S with weight 2, we are only allowed to assign weights to the above edges. Let S_i be a subset of S, s.t. $S \setminus S_i \subseteq e_i^0$, where $e_i^0 \in E^0$. Suppose $\gamma_u(e_i^0) = w_i$. Still, we need to put weight 1 on the vertices in S_i. In order to do so, we can put at most weight $1 - w_i$ on the edges in $E^0 \setminus \{e_i^0\}$, which covers S_i with weight at most $1 - w_i$. The only edge in E^1 that intersects S_i is the complementary edge e_i^1 of e_i^0. Hence, we have to set $\gamma_u(e_i^1) \geq w_i$. This holds for all edges $e^1 \in E^1$. Moreover, recall that both $\sum_{e^0 \in E^0} \gamma_u(e^0) = 1$ and $\sum_{e^1 \in E^1} \gamma_u(e^1) = 1$ hold. Hence, we cannot afford to set $\gamma_u(e_i^1) > w_i$ for some i, since this would lead to $\sum_{e^1 \in E^1} \gamma_u(e^1) > 1$. We thus have $\gamma_u(e_i^0) = \gamma_u(e_i^1) = w_i$ for every $e_i^0 \in E^0$ and its complementary edge $e_i^1 \in E^1$. □

Lemma 4.3. *Let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be an FHD of width ≤ 2 of the hypergraph H constructed above and let $p \in [2n+3; m]^-$. For every node u with $S \cup A'_p \cup \overline{A_p} \cup \{z_1, z_2\} \subseteq B_u$, the only way to cover $S \cup A'_p \cup \overline{A_p} \cup \{z_1, z_2\}$ by a fractional edge cover γ of weight ≤ 2 is by putting non-zero weight exclusively on edges $e_p^{k,0}$ and $e_p^{k,1}$ with $k \in \{1, 2, 3\}$. Moreover, $\sum_{k=1}^3 \gamma(e_p^{k,0}) = 1$ and $\sum_{k=1}^3 \gamma(e_p^{k,1}) = 1$ must hold.*

Proof. As in the proof of Lemma 4.2, to cover z_1 we have to put weight 1 on the edges in E^0 and to cover z_2 we have to put weight 1 on the edges in E^1, where E^0 and E^1 are defined as in the proof of Lemma 4.2. Since we have $\mathit{width}(\mathcal{F}) \leq 2$, we have to cover $A'_p \cup \overline{A_p} \cup S$ with the weight already put on the edges in $E^0 \cup E^1$. In order to cover A'_p, we have to put weight 1 on the edges in E^1_p, where

$$E^1_p = \{e_r^{k,1} \mid r \geq p\} \cup \{e^1_{\max}\}.$$

Notice that, $E^1_p \subseteq E^1$ and therefore $\sum_{e \in E^1 \setminus E^1_p} \gamma_u(e) = 0$. Similar, in order to cover $\overline{A_p}$, we have to put weight 1 on the edges in E^0_p, where

$$E^0_p = \{e_s^{k,0} \mid s \leq p\} \cup \{e^0_{(0,0)}\}.$$

Again, since $E^0_p \subseteq E^0$, $\sum_{e \in E^0 \setminus E^0_p} \gamma_u(e) = 0$. It remains to cover $S \cup \{z_1, z_2\}$. By Lemma 4.2, in order to cover S, z_1 and z_2, we have to put the same weight w on complementary edges e and e'. The only complementary edges in the sets E^0_p and E^1_p are edges of the form $e_p^{k,0}$ and $e_p^{k,1}$ with $k \in \{1, 2, 3\}$. In total, we thus have $\sum_{k=1}^3 e_p^{k,0} = 1$ and $\sum_{k=1}^3 e_p^{k,1} = 1$. □

Proof of the "only if"-direction. It remains to show that φ is satisfiable if H has a GHD or FHD of width ≤ 2. Due to the inequality $\mathit{fhw}(H) \leq \mathit{ghw}(H)$, it suffices to show that φ is satisfiable if H has an FHD of width ≤ 2. For this, let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$

be such an FHD. Let u_A, u_B, u_C and u'_A, u'_B, u'_C be the nodes that are guaranteed by Lemma 4.1. We state several properties of the path connecting u_A and u'_A, which heavily rely on Lemmas 4.2 and 4.3.

Claim A. *The nodes u'_A, u'_B, u'_C (resp. u_A, u_B, u_C) are not on the path from u_A to u_C (resp. u'_A to u'_C).*

Proof of Claim A. We only show that none of the nodes u'_i with $i \in \{A, B, C\}$ is on the path from u_A to u_C. The other property is shown analogously. Suppose to the contrary that some u'_i is on the path from u_A to u_C. Since u_B is also on the path between u_A and u_C we distinguish two cases:

- Case (1): u'_i is on the path between u_A and u_B; then $\{b_1, b_2\} \subseteq B_{u'_i}$. This contradicts the property shown in Lemma 4.1 that u'_i cannot cover any vertices outside H'_0.
- Case (2): u'_i is on the path between u_B and u_C; then $\{c_1, c_2\} \subseteq B_{u'_i}$, which again contradicts Lemma 4.1.

Hence, the paths from u_A to u_C and from u'_A to u'_C are indeed disjoint. ◇

Claim B. *The following equality holds: $nodes(A \cup A', \mathcal{F}) \cap \{u_A, u_B, u_C, u'_A, u'_B, u'_C\} = \emptyset$.*

Proof of Claim B. Suppose to the contrary that there is a u_X (the proof for u'_X is analogous) for some $X \in \{A, B, C\}$, s.t. $u_X \in nodes(A \cup A', \mathcal{F})$; then there is some $a \in (A \cup A')$, s.t. $a \in B_{u_X}$. This contradicts the property shown in Lemma 4.1 that u_X cannot cover any vertices outside H_0. ◇

We are now interested in the sequence of nodes $\hat{u}_i$ that cover the edges $e^0_{(0,0)}, e_{\min}, e_{\min \oplus 1}, \ldots, e_{\max \ominus 1}, e_{\max}$. Before we formulate Claim C, it is convenient to introduce the following notation. To be able to refer to the edges $e^0_{(0,0)}, e_{\min}, e_{\min \oplus 1}, \ldots, e_{\max \ominus 1}, e^1_{\max}$ in a uniform way, we use $e_{\min \ominus 1}$ as synonym of $e^0_{(0,0)}$ and $e_{\max}$ as synonym of $e^1_{\max}$. We can thus define the natural order $e_{\min \ominus 1} < e_{\min} < e_{\min \oplus 1} < \cdots < e_{\max \ominus 1} < e_{\max}$ on these edges.

Claim C. *The FHD $\mathcal{F}$ has a path containing nodes $\hat{u}_1, \ldots, \hat{u}_N$ for some N, such that the edges $e_{\min \ominus 1}, e_{\min}, e_{\min \oplus 1}, \ldots, e_{\max \ominus 1}, e_{\max}$ are covered in this order. More formally, there is a mapping $f : \{\min \ominus 1, \ldots, \max\} \to \{1, \ldots, N\}$, s.t.*

- *$\hat{u}_{f(p)}$ covers e_p and*
- *if $p < p'$ then $f(p) \leq f(p')$.*

By a path containing nodes $\hat{u}_1, \ldots, \hat{u}_N$ *we mean that $\hat{u}_1$ and $\hat{u}_N$ are nodes in $\mathcal{F}$, such that the nodes $\hat{u}_2, \ldots, \hat{u}_{N-1}$ lie (in this order) on the path from $\hat{u}_1$ to $\hat{u}_N$. Of course, the path from $\hat{u}_1$ to $\hat{u}_N$ may also contain further nodes, but we are not interested in whether they cover any of the edges e_p.*

Proof of Claim C. Suppose to the contrary that no such path exists. Let $p \geq \min$ be the maximal value such that there is a path containing nodes $\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_l$, which cover $e_{\min \ominus 1}, \ldots, e_p$ in this order. Clearly, there exists a node $\hat{u}$ that covers $e_{p \oplus 1} = A'_{p \oplus 1} \cup \overline{A_{p \oplus 1}}$. We distinguish four cases:

- Case (1): $\hat{u}_1$ is on the path from $\hat{u}$ to all other nodes $\hat{u}_i$, with $1 < i \leq l$. By the connectedness condition, $\hat{u}_1$ covers A'_p. Hence, in total $\hat{u}_1$ covers $A'_p \cup A$ with $A'_p = \{a'_{\min}, \ldots, a'_p\}$ and $A = \{a_{\min}, \ldots, a_{\max}\}$. Then $\hat{u}_1$ covers all edges $e_{\min \ominus 1}, \ldots, e_p$. Therefore, the path containing nodes $\hat{u}_1$ and $\hat{u}$ covers $e_{\min \ominus 1}, \ldots, e_{p \oplus 1}$ in this order, which contradicts the maximality of p.

- Case (2): $\hat{u} = \hat{u}_1$, hence, $\hat{u}_1$ covers $A'_{p \oplus 1} \cup A$ with $A'_{p \oplus 1} = \{a'_{\min}, \ldots, a'_{p \oplus 1}\}$ and $A = \{a_{\min}, \ldots, a_{\max}\}$. Then, $\hat{u}_1$ covers all $e_{\min \ominus 1}, \ldots, e_{p \oplus 1}$, which contradicts the maximality of p.

- Case (3): $\hat{u}$ is on the path from $\hat{u}_1$ to $\hat{u}_l$ and $\hat{u} \neq \hat{u}_1$. Hence, $\hat{u}$ is between two nodes $\hat{u}_i$ and $\hat{u}_{i+1}$ for some $1 \leq i < l$ or $\hat{u} = \hat{u}_{i+1}$ for some $1 \leq i < l - 1$. The following arguments hold for both cases. Now, there is some $q \leq p$, such that e_q is covered by $\hat{u}_{i+1}$ and $e_{q \ominus 1}$ is covered by $\hat{u}_i$. Therefore, $\hat{u}$ covers $\overline{A_q}$ either by the connectedness condition (if $\hat{u}$ is between $\hat{u}_i$ and $\hat{u}_{i+1}$) or simply because $\hat{u} = \hat{u}_{i+1}$. Hence, in total, $\hat{u}$ covers $A'_{p \oplus 1} \cup \overline{A_q}$ with $A'_{p \oplus 1} = \{a'_{\min}, \ldots, a'_{p \oplus 1}\}$ and $\overline{A_q} = \{a_q, a_{q \oplus 1}, \ldots, a_p, a_{p \oplus 1}, \ldots a_{\max}\}$. Then, $\hat{u}$ covers all edges $e_q, e_{q \oplus 1}, \ldots, e_{p \oplus 1}$. Therefore, the path containing nodes $\hat{u}_1, \ldots, \hat{u}_i, \hat{u}$ covers $e_{\min \ominus 1}, \ldots, e_{p \oplus 1}$ in this order, which contradicts the maximality of p.

- Case (4): There is a u^* on the path from $\hat{u}_1$ to $\hat{u}_l$, such that the paths from $\hat{u}_1$ to $\hat{u}$ and from $\hat{u}$ to $\hat{u}_l$ go through u^* and, moreover, $u^* \neq \hat{u}_1$. Then, u^* is either between $\hat{u}_i$ and $\hat{u}_{i+1}$ for some $1 \leq i < l$ or $u^* = \hat{u}_{i+1}$ for some $1 \leq i < l - 1$. The following arguments hold for both cases. There is some $q \leq p$, such that e_q is covered by $\hat{u}_{i+1}$ and $e_{q \ominus 1}$ is covered by $\hat{u}_i$. By the connectedness condition, u^* covers

  - $A'_p = \{a'_{\min}, \ldots, a'_p\}$, since u^* is on the path from $\hat{u}$ to $\hat{u}_l$, and
  - $\overline{A}_q = \{a_q, \ldots, a_p, a_{p \oplus 1}, \ldots a_{\max}\}$, since u^* is on the path from $\hat{u}_1$ to $\hat{u}_{i+1}$ or $u^* = \hat{u}_{i+1}$.

  Then u^* covers all edges $e_q, e_{q \oplus 1}, \ldots, e_p$. Therefore, the path containing the nodes $\hat{u}_1, \ldots, \hat{u}_i$, u^*, $\hat{u}$ covers $e_{\min \ominus 1}, \ldots, e_{p \oplus 1}$ in this order, which contradicts the maximality of p. ◇

So far we have shown, that there are three disjoint paths from u_A to u_C, from u'_A to u'_C and from $\hat{u}_1$ to $\hat{u}_N$, respectively. It is easy to see, that u_A is closer to the path $\hat{u}_1, \ldots, \hat{u}_N$ than u_B and u_C, since otherwise u_B and u_C would have to cover a_1 as well, which is impossible by Lemma 4.1. The same also holds for u'_A. In the next claims we will argue that the path from u_A to u'_A goes through some node $\hat{u}$ of the path from $\hat{u}_1$ to $\hat{u}_N$. We

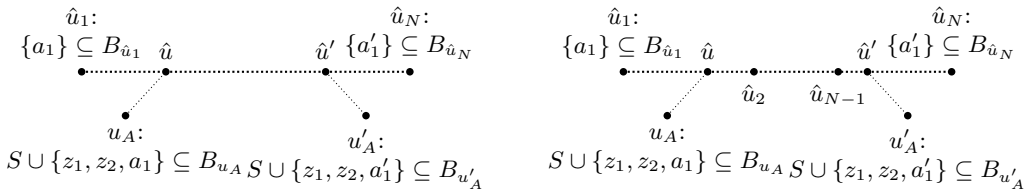

Figure 4.3: Arrangement of nodes $\hat{u}_1$, $\hat{u}$, $\hat{u}'$, and $\hat{u}_N$ from Claim E (a) and Claim G (b).

write $\pi(\hat{u}_1, \hat{u}_N)$ as a short-hand notation for the path from $\hat{u}_1$ to $\hat{u}_N$. Next, we state some important properties of $\pi(\hat{u}_1, \hat{u}_N)$ and the path from u_A to u'_A.

Claim D. *In the FHD $\mathcal{F}$ of H of width ≤ 2, the path from u_A to u'_A has non-empty intersection with $\pi(\hat{u}_1, \hat{u}_N)$.*

Proof of Claim D. Suppose to the contrary that the path from u_A to u'_A is disjoint from $\pi(\hat{u}_1, \hat{u}_N)$. We distinguish three cases:

- Case (1): u_A is on the path from u'_A to (some node in) $\pi(\hat{u}_1, \hat{u}_N)$. Then, by the connectedness condition, u_A must contain a'_1, which contradicts Lemma 4.1.
- Case (2): u'_A is on the path from u_A to $\pi(\hat{u}_1, \hat{u}_N)$. Analogously to Case (1), we get a contradiction by the fact that then u'_A must contain a_1.
- Case (3): There is a node u^* on the path from u_A to u'_A, which is closest to $\pi(\hat{u}_1, \hat{u}_N)$, i.e., u^* lies on the path from u_A to u'_A and both paths, the one connecting u_A with $\pi(\hat{u}_1, \hat{u}_N)$ and the one connecting u'_A with $\pi(\hat{u}_1, \hat{u}_N)$, go through u^*. Hence, by the connectedness condition, the bag of u^* contains $S \cup \{z_1, z_2, a_1, a'_1\}$. By Lemma 4.2, in order to cover $S \cup \{z_1, z_2\}$ with weight ≤ 2, we are only allowed to put non-zero weight on pairs of complementary edges. However, then it is impossible to achieve also weight ≥ 1 on a_1 and a'_1 at the same time. ◇

Claim E. *In the FHD $\mathcal{F}$ of H of width ≤ 2 there are two distinct nodes $\hat{u}$ and $\hat{u}'$ in the intersection of the path from u_A to u'_A with $\pi(\hat{u}_1, \hat{u}_N)$, s.t. $\hat{u}$ is the node in $\pi(\hat{u}_1, \hat{u}_N)$ closest to u_A and $\hat{u}'$ is the node in $\pi(\hat{u}_1, \hat{u}_N)$ closest to u'_A. Then, on the path $\pi(\hat{u}_1, \hat{u}_N)$, $\hat{u}$ comes before $\hat{u}'$. See Figure 4.3 (a) for a graphical illustration of the arrangement of the nodes $\hat{u}_1$, $\hat{u}$, $\hat{u}'$, and $\hat{u}_N$ on the path $\pi(\hat{u}_1, \hat{u}_N)$.*

Proof of Claim E. First, we show that $\hat{u}$ and $\hat{u}'$ are indeed distinct. Suppose towards a contradiction that they are not, i.e. $\hat{u} = \hat{u}'$. Then, by connectedness, $\hat{u}$ has to cover $S \cup \{z_1, z_2\}$, because $S \cup \{z_1, z_2\}$ is contained in B_{u_A} and in $B_{u'_A}$. Moreover, again by connectedness, $\hat{u}$ also has to cover $\{a_1, a'_1\}$, because a_1 is contained in $B_{\hat{u}_1}$ and in B_{u_A} and a'_1 is contained in $B_{\hat{u}_N}$ and in $B_{u'_A}$. As in Case (3) in the proof of Claim D, this is impossible by Lemma 4.2. Hence, $\hat{u}$ and $\hat{u}'$ are distinct.

Second, we show that, on the path from $\hat{u}_1$ to $\hat{u}_N$, the node $\hat{u}$ comes before $\hat{u}'$. Suppose to the contrary that $\hat{u}'$ comes before $\hat{u}$. Then, by the connectedness condition, $\hat{u}$ covers the following (sets of) vertices:

- a'_1, since we are assuming that $\hat{u}'$ comes before $\hat{u}$, i.e., $\hat{u}$ is on the path from $\hat{u}_N$ to u'_A;
- a_1, since $\hat{u}$ is on the path from $\hat{u}_1$ to u_A;
- $S \cup \{z_1, z_2\}$, since $\hat{u}$ is on the path from u_A to u'_A.

In total, $\hat{u}$ has to cover all vertices in $S \cup \{z_1, z_2, a_1, a'_1\}$. Again, by Lemma 4.2, this is impossible with weight ≤ 2. ◇

Claim F. *In the FHD $\mathcal{F}$ of H of width ≤ 2 the path $\pi(\hat{u}_1, \hat{u}_N)$ has at least 3 nodes $\hat{u}_i$, i.e., $N \geq 3$.*

Proof of Claim F. First, it is easy to verify that $N \geq 2$ must hold. Otherwise, a single node would have to cover $\{e_{\min \ominus 1}, e_{\min}, e_{\min \oplus 1}, \ldots, e_{\max \ominus 1}, e_{\max}\}$ and, hence, in particular, $S \cup \{z_1, z_2, a_1, a'_1\}$, which is impossible as we have already seen in Case (3) of the proof of Claim D.

It remains to prove $N \geq 3$. Suppose to the contrary that $N = 2$. By the problem reduction, hypergraph H has distinct edges $e_{\min \ominus 1}$, $e_{\min}$ and $e_{\max}$. Hence, $\hat{u}_1$ covers at least $e_{\min \ominus 1}$ and $\hat{u}_2$ covers at least $e_{\max}$. Recall from Claim E the nodes $\hat{u}$ and $\hat{u}'$, which constitute the endpoints of the intersection of the path from u_A to u'_A with the path $\pi(\hat{u}_1, \hat{u}_N)$, cf. Figure 4.3(a). Here we are assuming $N = 2$. We now show that, by the connectedness condition of FHDs, the nodes $\hat{u}$ and $\hat{u}'$ must cover certain vertices, which will lead to a contradiction by Lemma 4.2.

- vertices covered by $\hat{u}$: node $\hat{u}$ is on the path between u_A and u'_A. Hence, it covers $S \cup \{z_1, z_2\}$. Moreover, $\hat{u}$ is on the path between $\hat{u}_1$ and u_A (or even coincides with $\hat{u}_1$). Hence, it also covers a_1. In total, $\hat{u}$ covers at least $S \cup \{z_1, z_2, a_1\}$.
- vertices covered by $\hat{u}'$: node $\hat{u}'$ is on the path between u_A and u'_A. Hence, it covers $S \cup \{z_1, z_2\}$. Moreover, $\hat{u}'$ is on the path between $\hat{u}_2$ and u'_A (or even coincides with $\hat{u}_2$). Hence, it also covers a'_1. In total, $\hat{u}'$ covers at least $S \cup \{z_1, z_2, a'_1\}$.

One of the nodes $\hat{u}_1$ or $\hat{u}_2$ must also cover the edge $e_{\min}$. We inspect these 2 cases separately:

- Case (1): suppose that the edge $e_{\min}$ is covered by $\hat{u}_1$. Then, $\hat{u}_1$ covers vertex $a'_{\min}$, which is also covered by $\hat{u}_2$. Hence, also $\hat{u}$ covers $a'_{\min}$. In total, $\hat{u}$ covers $S \cup \{z_1, z_2, a_1, a'_{\min}\}$. However, by Lemma 4.2, we know that, to cover $S \cup \{z_1, z_2\}$ with

weight ≤ 2, we are only allowed to put non-zero weight on pairs of complementary edges. Hence, it is impossible to achieve also weight ≥ 1 on a_1 and on $a'_{\min}$ at the same time.

- Case (2): suppose that the edge $e_{\min}$ is covered by $\hat{u}_2$. Then, $\hat{u}_2$ covers vertex $a_{\min}$ (actually, it even covers all of A), which is also covered by $\hat{u}_1$. Hence, also $\hat{u}'$ covers $a_{\min}$. In total, $\hat{u}'$ covers $S \cup \{z_1, z_2, a'_1, a_{\min}\}$. Again, this is impossible by Lemma 4.2.

Hence, the path $\pi(\hat{u}_1, \hat{u}_N)$ indeed has at least 3 nodes $\hat{u}_i$. ◇

Claim G. *In the FHD $\mathcal{F}$ of H of width ≤ 2 all the nodes $\hat{u}_2, \ldots, \hat{u}_{N-1}$ are on the path from u_A to u'_A. For the nodes $\hat{u}$ and $\hat{u}'$ from Claim E, this means that the nodes $\hat{u}_1, \hat{u}$, $\hat{u}_2, \hat{u}_{N-1}$, $\hat{u}'$, $\hat{u}_N$ are arranged in precisely this order on the path $\pi(\hat{u}_1, \hat{u}_N)$ from $\hat{u}_1$ to $\hat{u}_N$, cf. Figure 4.3 (b). The node $\hat{u}$ may possibly coincide with $\hat{u}_1$ and $\hat{u}'$ may possibly coincide with $\hat{u}_N$.*

Proof of Claim G. We have to prove that $\hat{u}$ lies between $\hat{u}_1$ and $\hat{u}_2$ (not including $\hat{u}_2$) and $\hat{u}'$ lies between $\hat{u}_{N-1}$ and $\hat{u}_N$ (not including $\hat{u}_{N-1}$). For the first property, suppose to the contrary that $\hat{u}$ does not lie between $\hat{u}_1$ and $\hat{u}_2$ or $\hat{u} = \hat{u}_2$. This means, that there exists $i \in \{2, \ldots, N-1\}$ such that $\hat{u}$ lies between $\hat{u}_i$ and $\hat{u}_{i+1}$, including the case that $\hat{u}$ coincides with $\hat{u}_i$. Note that, by Claim E, $\hat{u}$ cannot coincide with $\hat{u}_N$, since there is yet another node $\hat{u}'$ between $\hat{u}$ and $\hat{u}_N$.

By definition of $\hat{u}_i$ and $\hat{u}_{i+1}$, there is a $p \in [2n+3; m]$, such that both $\hat{u}_i$ and $\hat{u}_{i+1}$ cover a'_p. Then, by the connectedness condition, $\hat{u}$ covers the following (sets of) vertices:

- a'_p, since $\hat{u}$ is on the path from $\hat{u}_i$ to $\hat{u}_{i+1}$ (or $\hat{u}$ coincides with $\hat{u}_i$),
- a_1, since $\hat{u}$ is on the path from $\hat{u}_1$ to u_A,
- $S \cup \{z_1, z_2\}$, since $\hat{u}$ is on the path from u_A to u'_A.

However, by Lemma 4.2, we know that, to cover $S \cup \{z_1, z_2\}$ with weight ≤ 2, we are only allowed to put non-zero weight on pairs of complementary edges. Hence, it is impossible to achieve also weight ≥ 1 on a'_p and a_1 at the same time.

It remains to show that $\hat{u}'$ lies between $\hat{u}_{N-1}$ and $\hat{u}_N$ (not including $\hat{u}_{N-1}$). Suppose to the contrary that it does not. Then, analogously to the above considerations for $\hat{u}$, it can be shown that there exists some $p \in [2n+3; m]$, such that $\hat{u}'$ covers the vertices $S \cup \{z_1, z_2, a_p, a'_1\}$. Again, this is impossible by Lemma 4.2. ◇

By Claim C, the decomposition $\mathcal{F}$ contains a path $\hat{u}_1 \cdots \hat{u}_N$ that covers the edges $e_{\min \ominus 1}, e_{\min}, e_{\min \oplus 1}, \ldots, e_{\max \ominus 1}, e_{\max}$ in this order. We next strengthen this property by showing that every node $\hat{u}_i$ covers exactly one edge e_p.

Claim H. *Each of the nodes $\hat{u}_1, \ldots, \hat{u}_N$ covers exactly one of the edges $e_{\min \ominus 1}$, $e_{\min}$, $e_{\min \oplus 1}$, $\ldots$, $e_{\max \ominus 1}$, $e_{\max}$.*

Proof of Claim H. We prove this property for the "outer nodes" $\hat{u}_1$, $\hat{u}_N$ and for the "inner nodes" $\hat{u}_2 \cdots \hat{u}_{N-1}$ separately. We start with the "outer nodes". The proof for $\hat{u}_1$ and $\hat{u}_N$ is symmetric. We thus only work out the details for $\hat{u}_1$. Suppose to the contrary that $\hat{u}_1$ not only covers $e_{\min \ominus 1}$ but also $e_{\min}$. We distinguish two cases according to the position of node $\hat{u}$ in Figure 4.3 (b):

- Case (1): $\hat{u} = \hat{u}_1$. Then, $\hat{u}_1$ has to cover the following (sets of) vertices:
  - $S \cup \{z_1, z_2\}$, since $\hat{u}$ is on the path from u_A to u'_A and we are assuming $\hat{u} = \hat{u}_1$.
  - a_1, since $\hat{u}_1$ covers $e_{\min \ominus 1}$,
  - $a'_{\min}$, since we are assuming that $\hat{u}_1$ also covers $e_{\min}$.

  By applying Lemma 4.2, we may conclude that the set $S \cup \{z_1, z_2, a_1, a'_{\min}\}$ cannot be covered by a fractional edge cover of weight ≤ 2.
- Case (2): $\hat{u} \neq \hat{u}_1$. Then $\hat{u}$ is on the path from $\hat{u}_1$ to $\hat{u}_2$. Hence, $\hat{u}$ has to cover the following (sets of) vertices:
  - $S \cup \{z_1, z_2\}$, since $\hat{u}$ is on the path from u_A to u'_A,
  - a_1, since $\hat{u}$ is on the path from u_A to $\hat{u}_1$,
  - $a'_{\min}$, since $\hat{u}$ is on the path from $\hat{u}_1$ to $\hat{u}_2$.

  As in Case (1) above, $S \cup \{z_1, z_2, a_1, a'_{\min}\}$ cannot be covered by a fractional edge cover of weight ≤ 2 due to Lemma 4.2.

It remains to consider the "inner" nodes $\hat{u}_i$ with $2 \leq i \leq N-1$. Each such $\hat{u}_i$ has to cover $S \cup \{z_1, z_2\}$ since all these nodes are on the path from u_A to u'_A by Claim G. Now suppose that $\hat{u}_i$ covers $e_p = A'_p \cup \overline{A_p}$ for some $p \in \{e_{\min}, \ldots, e_{\max \ominus 1}\}$. By Lemma 4.3, covering all of the vertices $A'_p \cup \overline{A_p} \cup S \cup \{z_1, z_2\}$ by a fractional edge cover of weight ≤ 2 requires that we put total weight 1 on the edges $e_p^{k,0}$ and total weight 1 on the edges $e_p^{k,1}$ with $k \in \{1, 2, 3\}$. However, then it is impossible to cover also $e_{p'}$ for some p' with $p' \neq p$. This concludes the proof of Claim F. ◇

We can now associate with each $\hat{u}_i$ for $1 \leq i \leq N$ the corresponding edge e_p and write u_p to denote the node that covers the edge e_p. By Claim G, we know that all of the nodes $u_{\min} \ldots, u_{\max \ominus 1}$ are on the path from u_A to u'_A. Hence, by the connectedness condition, all these nodes cover $S \cup \{z_1, z_2\}$.

We are now ready to construct a satisfying truth assignment σ of φ. For each $i \leq 2n+3$, let X_i be the set $B_{u_{(i,1)}} \cap (Y \cup Y')$. As $Y \subseteq B_{u_A}$ and $Y' \subseteq B_{u'_A}$, the sequence $X_1 \cap Y, \ldots, X_{2n+3} \cap Y$ is non-increasing and the sequence $X_1 \cap Y', \ldots, X_{2n+3} \cap Y'$ is non-decreasing. Furthermore, as all edges $e_{y_i} = \{y_i, y'_i\}$ must be covered by some node in $\mathcal{F}$, we conclude that for each i and j, $y_j \in X_i$ or $y'_j \in X_i$. Then, there is some $s \leq 2n+2$ such that $X_s = X_{s+1}$. Furthermore, all nodes between $u_{(s,1)}$ and $u_{(s+1,1)}$ cover X_s. We

derive a truth assignment for $x_1, \ldots, x_n$ from X_s as follows. For each $l \leq n$, we set $\sigma(x_l) = 1$ if $y_l \in X_s$ and otherwise $\sigma(x_l) = 0$. Note that in the latter case $y'_l \in X_s$.

Claim I. *The constructed truth assignment σ is a model of φ.*

Proof of Claim I. We have to show that every clause $c_j = L_j^1 \vee L_j^2 \vee L_j^3$ of φ is true in σ. Choose an arbitrary $j \in \{1, \ldots, m\}$. We have to show that there exists a literal in c_j which is true in σ. To this end, we inspect the node $u_{(s,j)}$, which, by construction, lies between $u_{(s,1)}$ and $u_{(s+1,1)}$. Let $p = (s, j)$. Then we have $A'_p \cup \overline{A_p} \cup S \cup \{z_1, z_2\} \subseteq B_{u_p}$. Moreover, by the definition of X_s, we also have $X_s \subseteq B_{u_p}$. By Lemma 4.3, the only way to cover B_{u_p} with weight ≤ 2 is by using exclusively the edges $e_p^{k,0}$ and $e_p^{k,1}$ with $k \in \{1, 2, 3\}$. More specifically, we have $\sum_{k=1}^{3} \gamma_{u_p}(e_p^{k,0}) = 1$ and $\sum_{k=1}^{3} \gamma_{u_p}(e_p^{k,1}) = 1$. Therefore, $\gamma_{u_p}(e_p^{k,0}) > 0$ for some k. We distinguish two cases depending on the form of literal L_j^k:

- Case (1): First, suppose $L_j^k = x_l$. By Lemma 4.2, complementary edges must have equal weight. Hence, from $\gamma_{u_p}(e_p^{k,0}) > 0$ it follows that also $\gamma_{u_p}(e_p^{k,1}) > 0$ holds. Thus, the weight on y'_l is less than 1, which means that $y'_l \notin B(\gamma_{u_p})$ and consequently $y'_l \notin X_s$. Since this implies that $y_l \in X_s$, we indeed have that $\sigma(x_l) = 1$.
- Case (2): Conversely, suppose $L_j^k = \neg x_l$. Since $\gamma_{u_p}(e_p^{k,0}) > 0$, the weight on y_l is less than 1, which means that $y_l \notin B(\gamma_{u_p})$ and consequently $y_l \notin X_s$. Hence, we have $\sigma(x_l) = 0$.

In either case, literal L_p^k is satisfied by σ and therefore, the j-th clause c_j is satisfied by σ. Since j was arbitrarily chosen, σ indeed satisfies φ. ◇

Claim I completes the proof of Theorem 4.1. □

We conclude this section by mentioning that the above reduction is easily extended to $k + \ell$ for arbitrary $\ell \geq 1$: for integer values ℓ, simply add a clique of 2ℓ fresh vertices $v_1, \ldots, v_{2\ell}$ to H and connect each v_i with each "old" vertex in H. Now assume a rational value $\ell \geq 1$, i.e., $\ell = r/q$ for natural numbers r, q with $r > q > 0$. To achieve a rational bound $k + r/q$, we add r fresh vertices and add hyperedges $\{v_i, v_{i\oplus 1}, \ldots, v_{i\oplus(q-1)}\}$ with $i \in \{1, \ldots, r\}$ to H, where $a \oplus b$ denotes $a + b$ modulo r. Again, we connect each v_i with each "old" vertex in H. With this construction we can give NP-hardness proofs for any (fractional) $k \geq 3$. For all fractional values $k < 3$ (except for $k = 2$) different gadgets and ideas might be needed to prove NP-hardness of Check(FHD,k), which we leave for future work.

4.2 Efficient Computation of GHDs

As discussed in Chapter 1 we are interested in finding a realistic and non-trivial criterion on hypergraphs that makes the Check(GHD, k) problem tractable for fixed k. We thus

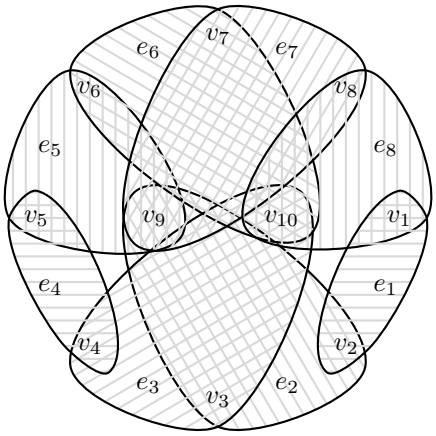

Figure 4.4: Hypergraph H_0 from Example 4.2

propose here such a simple property, namely the bounded intersection of two or more edges.

Definition 4.2. The *intersection width iwidth*(H) of a hypergraph H is the maximum cardinality of any intersection $e_1 \cap e_2$ of two distinct edges e_1 and e_2 of H. We say that a hypergraph H has the *i-bounded intersection property (i-BIP)* if *iwidth*$(H) \leq i$ holds.

Let $\mathscr{C}$ be a class of hypergraphs. We say that $\mathscr{C}$ has the *bounded intersection property (BIP)* if there exists some integer constant i such that every hypergraph H in $\mathscr{C}$ has the i-BIP. Class $\mathscr{C}$ has the *logarithmically-bounded intersection property (LogBIP)* if for each of its elements H, *iwidth*(H) is $\mathcal{O}(\log n)$, where n denotes the size of the hypergraph H. ◁

The BIP criterion is indeed non-trivial, as several well-known classes of unbounded *ghw* enjoy the 1-BIP, such as cliques and grids. Moreover, our empirical study in Section 5.3 suggests that the overwhelming number of CQs enjoys the 2-BIP (i.e., one hardly joins two relations over more than 2 attributes). To allow for a yet bigger class of hypergraphs, the BIP can be relaxed as follows.

Definition 4.3. The *c-multi-intersection width c-miwidth*(H) of a hypergraph H is the maximum cardinality of any intersection $e_1 \cap \cdots \cap e_c$ of c distinct edges $e_1, \ldots, e_c$ of H. We say that a hypergraph H has the *i-bounded c-multi-intersection property (ic-BMIP)* if *c-miwidth*$(H) \leq i$ holds.

Let $\mathscr{C}$ be a class of hypergraphs. We say that $\mathscr{C}$ has the *bounded multi-intersection property (BMIP)* if there exist constants c and i such that every hypergraph H in $\mathscr{C}$ has the ic-BMIP. Class $\mathscr{C}$ of hypergraphs has the *logarithmically-bounded multi-intersection property (LogBMIP)* if there is a constant c such that for the hypergraphs $H \in \mathscr{C}$, *c-miwidth*(H) is $\mathcal{O}(\log n)$, where n denotes the size of the hypergraph H. ◁

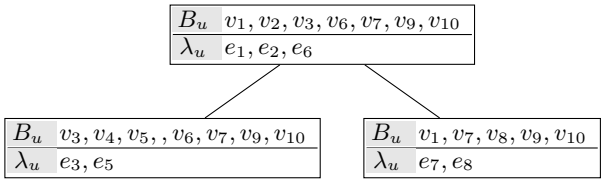

Figure 4.5: HD of hypergraph H_0 in Figure 4.4

Example 4.2. Figure 4.4 shows the hypergraph $H_0 = (V_0, E_0)$ with $ghw(H_0) = 2$ but $hw(H_0)$=3. (which is from [53], which, in turn, was inspired by work of Adler [4]). Figure 4.5 shows an HD of width 3 and Figure 4.6 shows GHDs of width 2 for the hypergraph H_0. The BIP and the 3-BMIP of H_0 is 1. Starting from c=4, the c-BMIP is 0. ◁

The LogBMIP is the most liberal restriction on classes of hypergraphs introduced in Definitions 4.2 and 4.3. The main result in this section will be that the Check(GHD, k) problem with fixed k is tractable for any class of hypergraphs satisfying this criterion.

Towards this result, first recall that the difference between HDs and GHDs lies in the "special condition" required by HDs. Assume a hypergraph $H = (V(H), E(H))$ and an arbitrary GHD $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ of H. Then $\mathcal{H}$ is not necessarily an HD, since it may contain a special condition violation (SCV), i.e.: there can exist a node u, an edge $e \in \lambda_u$ and a vertex $v \in V$, s.t. $v \in e$ (and, hence, $v \in B(\lambda_u)$), $v \notin B_u$ and $v \in V(T_u)$. Clearly, if we could be sure that $E(H)$ also contains the edge $e' = e \cap B_u$, then we would simply replace e in λ_u by e' and would thus get rid of this SCV.

Example 4.3 (Example 4.2 continued)**.** The GHDs in Figure 4.6 (a) and (b) violate the special condition in node u since the edge e_2 containing vertex v_2 is in λ_u and v_2 is in $V(T_u)$ but not in B_u. Adding

$$e_2' = e_2 \cap B_u = \{v_2, v_3, v_9\} \cap \{v_3, v_6, v_7, v_9, v_{10}\} = \{v_3, v_9\}$$

to H_0 and replacing e_2 with e_2' in λ_u would repair the SCV at node u of the GHDs in Figure 4.6. ◁

Now our goal is to define a polynomial-time computable function f which, to each hypergraph H and integer k, associates a set $f(H, k)$ of additional hyperedges such that $ghw(H) = k$ iff $hw(H') = k$ with $H = (V(H), E(H))$ and $H' = (V(H), E(H) \cup f(H, k))$. From this it follows immediately that $ghw(H)$ is computable in polynomial time. The function f is defined in such a way that $f(H, k)$ only contains subsets of hyperedges of H. Thus, f is a *subedge function* as described in [53] and a GHD of the same width can be easily obtained from any HD of H'. It is easy to see and well-known [53] that for each subedge function f, and each H and k, $ghw(H) \leq hw(H \cup f(H, k)) \leq hw(H)$. Moreover, for the "limit" subedge function f^+ where $f^+(H, k)$ consists of all possible non-empty subsets of edges of H, we have that $hw(H \cup f^+(H, k)) = ghw(H)$ [4, 53]. Of course, in

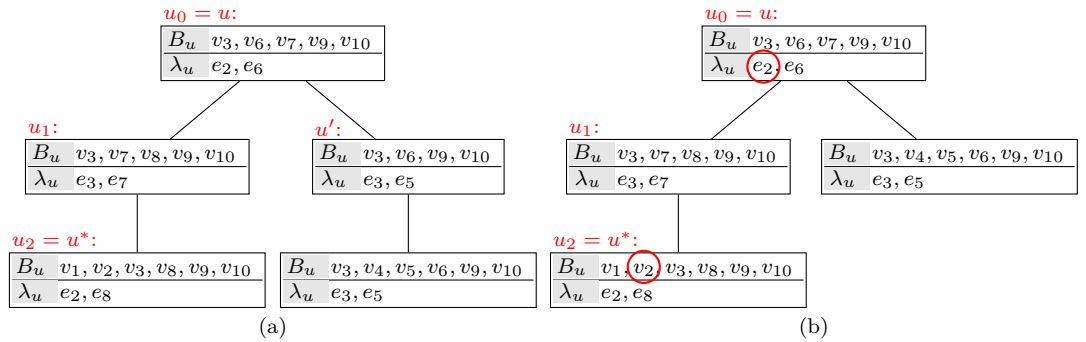

Figure 4.6: (a) non bag-maximal vs. (b) bag-maximal GHD of hypergraph H_0 in Figure 4.4

general, f^+ contains an exponential number of edges. The important point is that our function f will achieve the same, while generating a polynomial and PTime-computable set of edges only.

We start by introducing a useful property of GHDs, which we will call *bag-maximality*. Let $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ be a GHD of some hypergraph $H = (V(H), E(H))$. For each node u in T, we have $B_u \subseteq B(\lambda_u)$ by definition of GHDs and, in general, $B(\lambda_u) \setminus B_u$ may be non-empty. We observe that it is sometimes possible to take some vertices from $B(\lambda_u) \setminus B_u$ and add them to B_u without violating the connectedness condition. Of course, such an addition of vertices to B_u does not violate any of the other conditions of GHDs. Moreover, it does not increase the width.

Definition 4.4. Let $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ be a GHD of some hypergraph $H = (V(H), E(H))$. We call $\mathcal{H}$ *bag-maximal*, if for every node u in T, adding a vertex $v \in B(\lambda_u) \setminus B_u$ to B_u would violate the connectedness condition. ◁

It is easy to verify that if H has a GHD of width $\leq k$, then it also has a bag-maximal GHD of width $\leq k$.

Lemma 4.4. *For every GHD $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ of some hypergraph $H = (V(H), E(H))$, there exists a bag-maximal GHD $\mathcal{H}' = \langle T, (B'_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ of H, such that $\mathcal{H}$ and $\mathcal{H}'$ have the same width.*

Proof. Start with a GHD of width k of H. As long as there exists a node $u \in T$ and a vertex $v \in B(\lambda_u) \setminus B_u$, such that v can be added to B_u without destroying the GHD properties, select such a node u and vertex v arbitrarily and add v to B_u. By exhaustive application of this transformation, a bag-maximal GHD of width k of H is obtained. □

Example 4.4 (Example 4.3 continued)**.** Clearly, the GHD in Figure 4.6(a) violates bag-maximality in node u', since the vertices v_4 and v_5 can be added to $B_{u'}$ without violating any GHD properties. If we add v_4 and v_5 to $B_{u'}$, then bag $B_{u'}$ at node u' and

the bag at its child node are the same, which allows us to delete one of the nodes. This results in the GHD given in Figure 4.6(b), which is bag-maximal. In particular, the vertex v_2 cannot be added to B_{u_0}: indeed, adding v_2 to B_{u_0} would violate the connectedness condition, since v_2 is not in B_{u_1} but in B_{u_2}. ◁

So from now on, we will restrict ourselves w.l.o.g. to bag-maximal GHDs. Before we prove a crucial lemma, we introduce some useful notation:

Definition 4.5. Let $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ be an GHD of a hypergraph H. Moreover, let u be a node in $\mathcal{H}$ and let $e \in \lambda_u$ such that $e \setminus B_u \neq \emptyset$ holds. Let u^* denote the node closest to u, such that u^* covers e, i.e., $e \subseteq B_{u*}$. Then, we call the path $\pi = (u_0, u_1, \ldots, u_l)$ with $u_0 = u$ and $u_l = u^*$ the *critical path* of (u, e) denoted as $critp(u, e)$. ◁

Lemma 4.5. *Let $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ be a bag-maximal GHD of a hypergraph $H = (V(H), E(H))$, let $u \in T$, $e \in \lambda_u$, and $e \setminus B_u \neq \emptyset$. Let $\pi = (u_0, u_1, \ldots, u_\ell)$ with $u_0 = u$ be the critical path of (u, e). Then the following equality holds.*

$$e \cap B_u = \quad e \cap \bigcap_{i=1}^{\ell} B(\lambda_{u_i})$$

Proof. "$\subseteq$": Given that $e \subseteq B_{u_\ell}$ and by the connectedness condition, $e \cap B_u$ must be a subset of B_{u_i} for every $i \in \{1, \ldots, \ell\}$. Therefore, $e \cap B_u \subseteq e \cap \bigcap_{i=1}^{\ell} B(\lambda_{u_i})$ holds.

"$\supseteq$": Assume to the contrary that there exists some vertex $v \in e$ with $v \notin B_u$ but $v \in \bigcap_{i=1}^{\ell} B(\lambda_{u_i})$. By $e \subseteq B_{u_\ell}$, we have $v \in B_{u_\ell}$. By the connectedness condition, along the path $u_0, \ldots, u_\ell$ with $u_0 = u$, there exists $\alpha \in \{0, \ldots, \ell - 1\}$, s.t. $v \notin B_{u_\alpha}$ and $v \in B_{u_{\alpha+1}}$. However, by the assumption, $v \in \bigcap_{i=1}^{\ell} B(\lambda_{u_i})$ holds. In particular, $v \in B(\lambda_{u_\alpha})$. Hence, we could safely add v to B_{u_α} without violating the connectedness condition nor any other GHD condition. This contradicts the bag-maximality of $\mathcal{H}$. □

Example 4.5 (Example 4.3 continued)**.** Consider root node u of the GHD in Figure 4.6(b). We have $e_2 \in \lambda_u$ and $e_2 \setminus B_u = \{v_2\} \neq \emptyset$. On the other hand, e_2 is covered by u_2. Hence, the critical path of (u, e_2) is $\pi = (u, u_1, u_2)$. It is easy to verify that $e_2 \cap B_u = e_2 \cap (e_3 \cup e_7) \cap (e_8 \cup e_2) = \{v_3, v_9\}$ indeed holds. ◁

We are now ready to prove the main result of this section.

Theorem 4.2. *For every hypergraph class $\mathscr{C}$ that enjoys the LogBMIP, and for every constant $k \geq 1$, the* Check*(GHD, k) problem is tractable, i.e., given a hypergraph H, it is feasible in polynomial time to check $ghw(H) \leq k$ and, if so, to compute a GHD of width k of H.*

Proof. Let $H = (V(H), E(H))$ be an arbitrary hypergraph. Our goal is to show that there exists a polynomially bounded, polynomial-time computable set $f(H, k)$ of subedges

of H, such that $ghw(H) = k$ iff $hw(H') = k$ with $H' = (V(H), E(H) \cup f(H,k))$. By our considerations above, in order to guarantee the equivalence $ghw(H) = k$ iff $hw(H') = k$, it suffices to construct $f(H,k)$ in such a way that, in every GHD $\mathcal{G}$ of H, for every node u in $\mathcal{G}$, and every edge $e \in \lambda_u$, the set $f(H,k)$ contains the subedge $e' = e \cap B_u$.

Let $\mathcal{H} = \langle T, (B_u)_{u\in T}, (\lambda_u)_{u\in T}\rangle$ be a bag-maximal GHD of H, let $u \in T$, $e \in \lambda_u$, and $e \setminus B_u \neq \emptyset$. Let $\pi = (u_0, u_1, \ldots, u_\ell)$ with $u_0 = u$ be the critical path of (u, e). By Lemma 4.5, the equality $e \cap B_u = e \cap \bigcap_{i=1}^{\ell} B(\lambda_{u_i})$ holds. For $i \in \{1, \ldots, \ell\}$, let $\lambda_{u_i} = \{e_{i1}, \ldots, e_{ij_i}\}$ with $j_i \leq k$. Then $e \cap \bigcap_{i=1}^{\ell} B(\lambda_{u_i})$ and, therefore, also $e \cap B_u$, is of the form

$$e \cap (e_{11} \cup \cdots \cup e_{1j_1}) \cap \cdots \cap (e_{\ell 1} \cup \cdots \cup e_{\ell j_\ell}).$$

We want to construct $f(H,k)$ in such a way that it contains all possible sets $e \cap B_u$ of vertices. To this end, we proceed by a stepwise transformation of the above intersection of unions into a union of intersections via distributivity of $\cup$ and $\cap$.

For $i \in \{0, \ldots, \ell\}$, let $I_i = e \cap \bigcap_{\alpha=1}^{i} B(\lambda_{u_\alpha}) = e \cap \bigcap_{\alpha=1}^{i} (e_{\alpha 1} \cup \cdots \cup e_{\alpha j_\alpha})$. Then $I_0 = e$. For I_1 we have to distinguish two cases: if $e \in \lambda_{u_1}$, then $e \subseteq B(\lambda_{i_1})$ and, therefore, $I_1 = I_0$. If $e \notin \lambda_{u_1}$, then $I_1 = (e \cap e_{11}) \cup \cdots \cup (e \cap e_{1j_1})$. In the latter case, for computing I_2, we have to go through all sets $(e \cap e_{1\beta})$ with $\beta \in \{1, \ldots, j_1\}$ and distinguish the two cases if $\{e, e_{1\beta}\} \cap \lambda_{u_2} \neq \emptyset$ holds or not. If it holds, then we let $(e \cap e_{1\beta})$ in the disjunction of I_1 unchanged. Otherwise we replace it by $(e \cap e_{1\beta} \cap e_{21}) \cup \cdots \cup (e \cap e_{1\beta} \cap e_{2j_2})$. This splitting of intersections into unions of intersections can be iterated over all $i \in \{1, \ldots, \ell\}$ in order to arrive at $I_\ell = e \cap \bigcap_{i=1}^{\ell} B(\lambda_{u_i}) = e \cap B_u$, where I_ℓ is represented as a union of intersections.

We formalize the computation of the intersections in $I_0, \ldots, I_\ell$ by constructing the "$\bigcup\bigcap$-tree" in Algorithm 4.1 "`Union-of-Intersections-Tree`". In a loop over all $i \in \{1, \ldots, \ell\}$, we thus compute trees $\mathcal{T}_i$ such that each node p in $\mathcal{T}_i$ is labelled by a set $label(p)$ of edges. By $int(p)$ we denote the intersection of the edges in $label(p)$. The parent-child relationship between a node p and its child nodes $p_1, \ldots, p_{j_\alpha}$ corresponds to a splitting step, where the intersection $int(p)$ is replaced by the union $(int(p) \cap e_{\alpha 1}) \cup \cdots \cup (int(p) \cap e_{\alpha j_\alpha})$. It can be proved by a straightforward induction on i that, in the tree $\mathcal{T}_i$, the union of $int(p)$ over all *leaf nodes* p of $\mathcal{T}_i$ yields precisely the union-of-intersections representation of I_i.

We observe that, in the tree $\mathcal{T}_\ell$, each node has at most k child nodes. Nevertheless, $\mathcal{T}_\ell$ can become exponentially big since we have no appropriate bound on the length ℓ of the critical path. Recall, however, that we are assuming the LogBMIP, i.e., there exists a constant $c > 1$, s.t. any intersection of $\geq c$ edges of H has at most $a \log n$ elements, where a is a constant and n denotes the size of H. Now let $\mathcal{T}^*$ be the *reduced* $\bigcup\bigcap$*-tree*, which is obtained from $\mathcal{T}_\ell$ by cutting off all nodes of depth greater than $c - 1$. Clearly, $\mathcal{T}^*$ has at most k^{c-1} leaf nodes and the total number of nodes in $\mathcal{T}^*$ is bounded by $(c-1)k^{c-1}$.

The set $f(H,k)$ of subedges that we add to H will consist in all possible sets I_ℓ that we can obtain from all possible critical paths $\pi = (u_0, u_1, \ldots, u_\ell)$ in all possible bag-maximal GHDs $\mathcal{H}$ of width $\leq k$ of H. We only show that, in case of the LogBMIP, the number of

Algorithm 4.1: `Union-of-Intersections-Tree`

input : GHD $\mathcal{H}$ of H, an edge $e \in E(H)$, critical path $\pi = (u_0, \ldots, u_\ell)$ of $\mathcal{H}$
output : $\bigcup\bigcap$-tree T_ℓ

`/* Initialization: compute` (N, E) `for` T_0 `*/`
$N \leftarrow \{p\}$;
$E \leftarrow \emptyset$;
$label(p) \leftarrow \{e\}$;
$T \leftarrow (N, E)$;
`/* Compute` T_i `from` T_{i-1} `in a loop over` i `*/`
for $i \leftarrow 1$ **to** ℓ **do**
  foreach *leaf node* p *of* T **do**
    if $label(p) \cap \lambda_{u_i} = \emptyset$ **then**
      Let $\lambda_{u_i} = \{e_{i1}, \ldots, e_{ij_i}\}$;
      Create new nodes $\{p_1, \ldots, p_{j_i}\}$;
      for $\alpha \leftarrow 1$ **to** j_i **do** $label(p_\alpha) \leftarrow label(p_\alpha) \cup \{e_{i\alpha}\}$;
      $N \leftarrow N \cup \{p_1, \ldots, p_{j_i}\}$;
      $E \leftarrow E \cup \{(p, p_1), \ldots, (p, p_{j_i})\}$;
    end
  end
  $T \leftarrow (N, E)$;
end

possible sets I_ℓ is polynomially bounded. The polynomial-time computability of this set of sets is then easy to see. The set of all possible sets I_ℓ is obtained by first considering all possible reduced $\bigcup\bigcap$-trees $\mathcal{T}^*$ and then considering all sets I_ℓ that correspond to some extension $\mathcal{T}_\ell$ of $\mathcal{T}^*$.

First, let m denote the number of edges in $E(H)$, then the number of possible reduced $\bigcup\bigcap$-trees $\mathcal{T}^*$ for given H and k is bounded by $m \cdot m^{(c-1)k^{c-1}}$. This can be seen as follows: we can first construct the complete k-ary tree of depth $c-1$. Clearly, this tree has $\leq (c-1)k^{c-1}$ nodes. The root is labelled with edge e. Now we may label each other node in this tree either by a set of edges which is obtained from the label of its parent by adding one new edge (in particular, by an edge different from e) to express that such a node with such a label exists in $\mathcal{T}^*$. Or we may label a node (and consequently all its descendants) by some stop symbol $\perp$ to express that $\mathcal{T}^*$ shall not contain this node. Hence, in total, we have m choices for the initial $\bigcup\bigcap$-tree $\mathcal{T}_0$ (namely the edge e labelling the root) and $\leq m^{(c-1)k^{c-1}}$ choices to expand $\mathcal{T}_0$ to $\mathcal{T}^*$.

It remains to determine the number of possible sets I_ℓ that one can get from possible extensions $\mathcal{T}_\ell$ of $\mathcal{T}^*$. Clearly, if a leaf node in $\mathcal{T}^*$ is at depth $< c-1$, then no descendants at all of this node have been cut off. In contrast, a leaf node p in $\mathcal{T}^*$ at depth $c-1$ may be the root of a whole subtree in $\mathcal{T}_\ell$. Let $U(p)$ denote the union of the intersections

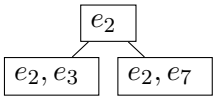

Figure 4.7: $\bigcup\bigcap$-tree of the critical path (u, u_1, u^*) of (u, e_2) in Figure 4.6(b)

represented by all leaf nodes below p. By construction of $\mathcal{T}_\ell$, $U(p) \subseteq int(p)$ holds. Moreover, by the LogBMIP, $|int(p)| \leq a \log n$ for some constant a. Hence, $U(p)$ takes one out of at most $2^{a \log n} = n^a$ possible values.

In total, an upper bound on the number of possible sets I_ℓ (and, hence, on $|f(H,k)|$) is obtained as follows: there are at most $m \cdot m^{(c-1)k^{c-1}}$ reduced trees $\mathcal{T}^*$; each such tree has at most k^{c-1} leaf nodes, and each leaf node represents at most n^a different sets of vertices. Putting all this together, we conclude that $|f(H,k)|$ is bounded by $m \cdot m^{(c-1)k^{c-1}} \cdot n^{ak^{c-1}}$ for some constant a. □

Example 4.6 (Example 4.5 continued)**.** The constructed $\bigcup\bigcap$-tree of the critical path (u, u_1, u^*) of (u, e_2) in Figure 4.6(b) is given in Figure 4.7. The intersection of unions $e_2 \cap (e_3 \cup e_7)$ is replaced by the unions of the leaf nodes $(e_2 \cap e_3) \cup (e_2 \cap e_7)$, which yields the same edge $e_2' = \{v_3, v_9\}$ as in Example 4.3. ◁

We have defined in Section 1.2 the degree d of a hypergraph H. We now consider hypergraphs of bounded degree.

Definition 4.6. We say that a hypergraph H has the *d-bounded degree property (d-BDP)* if $degree(H) \leq d$ holds.

Let $\mathscr{C}$ be a class of hypergraphs. We say that $\mathscr{C}$ has the *bounded degree property (BDP)* if there exists a constant d such that every hypergraph H in $\mathscr{C}$ has the d-BDP. ◁

The class of hypergraphs of bounded degree is an interesting special case of the class of hypergraphs enjoying the BMIP. Indeed, suppose that each vertex in a hypergraph H occurs in at most d edges for some constant d. Then the intersection of $d+1$ hyperedges is always empty. The following corollary is thus immediate.

Corollary 4.1. *For every class $\mathscr{C}$ of hypergraphs of bounded degree, for each constant k, the problem* Check*(GHD, k) is tractable.*

For the important special case of the BIP, the upper bound on $|f(H,k)|$ in the proof of Theorem 4.2, improves to $m^{k+1} \cdot 2^{k \cdot i}$. More specifically, in case of the BIP, the set $f(H,k)$ becomes

$$f(H,k) = \bigcup_{e \in E(H)} \Big(\bigcup_{e_1,\ldots,e_j \in (E(H) \setminus \{e\}), j \leq k} 2^{(e \cap (e_1 \cup \cdots \cup e_j))} \Big),$$

i.e., $f(H,k)$ contains all subsets of intersections of edges $e \in E(H)$ with unions of $\leq k$ edges of H different from e. In case of the BIP, the intersection $e \cap (e_1 \cup \cdots \cup e_j)$ has

at most $i \cdot k$ elements. Hence, $|f(H,k)| \leq m^{k+1} \cdot 2^{k \cdot i}$ holds. We thus get the following parameterized complexity result.

Theorem 4.3. *For each constant k, the* Check*(GHD, k) problem is fixed-parameter tractable w.r.t. the parameter i for hypergraphs enjoying the BIP, i.e., in this case,* Check*(GHD, k) can be solved in time $\mathcal{O}(h(i) \cdot poly(n))$, where $h(i)$ is a function depending on the intersection width i only and $poly(n)$ is a function that depends polynomially on the size n of the given hypergraph H.*

4.3 Efficient Computation of FHDs

In Section 4.2, we have shown that under certain conditions (with the BIP and BDP as most specific and the LogBMIP as most general conditions) the problem of computing a GHD of width $\leq k$ can be reduced to the problem of computing an HD of width $\leq k$. The key to this problem reduction was to add subedges which allowed us to repair all possible special condition violations (SCVs) in all possible GHDs of width $\leq k$. When trying to carry over these ideas from GHDs to FHDs, we encounter *two major challenges*: Can we repair SCVs in an FHD by ideas similar to GHDs? Does the special condition in case of FHDs allow us to extend the HD algorithm from [51] to FHDs?

As for the first challenge, recall from Theorem 4.2 that the tractability of Check(GHD, k) was achieved by adding polynomially many subedges $f(H,k)$ to a hypergraph H, such that $B_u = B(\lambda_u)$ can be enforced in every node u of a GHD of H. In other words, for $S_u = supp(\lambda_u)$, we had $B_u = \bigcup S_u$ in case of GHDs. GHDs with this property clearly satisfy the special condition. We thus reduced the Check(GHD, k) problem to the Check(HD, k) problem, which is well-known to be tractable [51]. In contrast, for FHDs, the fractional edge cover function γ_u at a node u may take any value in $[0,1]$. Therefore, $e \in supp(\gamma_u)$ (i.e., $\gamma_u(e) > 0$) does not imply $\gamma_u(e) = 1$. Hence, substantially more work will be needed to achieve $B_u = \bigcup S_u$ with $S_u = supp(\gamma_u)$ also for FHDs.

As for the second challenge, we will encounter another obstacle compared to the HD algorithm: a crucial step of the top-down construction of an HD in [51] is to "guess" $\leq k$ edges with $\lambda_u(e) = 1$ for the next node u in the HD. However, for a fractional cover γ_u, we do not have such a bound on the number of edges with non-zero weight. In fact, it is easy to exhibit a family $(H_n)_{n \in \mathbb{N}}$ of hypergraphs where it is advantageous to have unbounded $supp(\gamma_n)$ even if $(H_n)_{n \in \mathbb{N}}$ enjoys the BIP, as the following example illustrates:

Example 4.7. Consider the family $(H_n)_{n \in \mathbb{N}}$ of hypergraphs with $H_n = (V_n, E_n)$ defined as follows:

$V_n = \{v_0, v_1, \ldots, v_n\}$

$E_n = \{\{v_0, v_i\} \mid 1 \leq i \leq n\} \cup \{\{v_1, \ldots, v_n\}\}$

Clearly $iwidth(H_n) = 1$, but an optimal fractional edge cover of H_n is obtained by the following mapping γ with $supp(\gamma) = E_n$:

$\gamma(\{v_0, v_i\}) = 1/n$ for each $i \in \{1, \ldots, n\}$ and

$\gamma(\{v_1, \ldots, v_n\}) = 1 - (1/n)$

such that $\mathit{weight}(\gamma) = 2 - (1/n)$, which is optimal in this case. ◁

Nevertheless, in this section, we use the ingredients from the Check(GHD, k) problem to prove a similar (slightly weaker though) tractability result for the Check(FHD, k) problem. More specifically, we shall show that the Check(FHD, k) problem becomes tractable for fixed k, if we impose the bounded degree property. Thus, the main result of this section is:

Theorem 4.4. *For every hypergraph class $\mathscr{C}$ that has bounded degree, and for every constant $k \geq 1$, the* Check*(FHD, k) problem is tractable, i.e., given a hypergraph $H \in \mathscr{C}$, it is feasible in polynomial time to check $\mathit{fhw}(H) \leq k$ and, if so, to compute an FHD of width k of H.*

To prove this result, we tackle the two presented main challenges in reverse order. First, we will show that every hypergraph H with degree d allows for an FHD $\mathcal{F}$ with bounded support $\mathit{supp}(\gamma_u)$ at every node u of $\mathcal{F}$ (Lemma 4.6). Second, we will devise a polynomial subedge function that allows us to repair all possible SCVs of such $\mathcal{F}$ with bounded $\mathit{supp}(\gamma_u)$ at every node u (Lemma 4.11).

Bounded Support. First, we show that, for every FHD $\mathcal{F}$ of width k of a hypergraph H of degree $\leq d$, there exists an FHD $\mathcal{F}'$ of width $\leq k$ of H satisfying the following important property: for every node u in the FHD $\mathcal{F}'$, the fractional edge cover γ_u has support $\mathit{supp}(\gamma)$ bounded by a constant that depends only on k and d. For this result, we need to introduce, analogously to edge-weight functions and edge covers in Section 2.2, the notions of vertex-weight functions and vertex covers.

Definition 4.7. A vertex-weight function w for a hypergraph H assigns a weight $w(v) \geq 0$ to each vertex v of H. We say that w is a *fractional vertex cover* of H if for each edge $e \in E(H)$, $\Sigma_{v \in e} w(v) \geq 1$ holds. For a vertex-weight function w for hypergraph H, we denote by $\mathit{weight}(w)$ its total weight, i.e. $\Sigma_{v \in V(H)} w(v)$. The *fractional vertex cover number* $\tau^*(H)$ is defined as the minimum $\mathit{weight}(w)$ where w ranges over all fractional vertex covers of H. The *vertex support* $\mathit{vsupp}(w)$ of a hypergraph H under a vertex-weight function w is defined as $\mathit{vsupp}(w) = \{v \in V(H) \mid w(v) > 0\}$. ◁

For our result on bounded support, we will exploit the well-known dualities $\rho^*(H) = \tau^*(H^d)$ and $\tau^*(H) = \rho^*(H^d)$, where H^d denotes the dual of H. To make optimal use of this, we make, for the moment, several assumptions. First of all, we will assume w.l.o.g. that (1) hypergraphs have no isolated vertices and (2) no empty edges. In fact for hypergraphs with isolated vertices (empty edges), ρ^* (τ^*) would be undefined or at least not finite. Furthermore, we make the following temporary assumptions. Assume that (3) hypergraphs never have two distinct vertices of the same "edge-type" (i.e., the

two vertices occur in precisely the same edges) and (4) they never have two distinct edges of the same "vertex-type" (i.e., we exclude duplicate edges).

Assumptions (1) – (4) can be safely made. Recall that we are ultimately interested in the computation of an FHD of width $\leq k$ for given k. As mentioned above, without assumption (1), the computation of an edge-weight function and, hence, of an FHD of width $\leq k$ makes no sense. Assumption (2) does not restrict the search for a specific FHD since we would never define an edge-weight function with non-zero weight on an empty edge. As far as assumption (3) is concerned, suppose that a hypergraph H has groups of multiple vertices of identical edge-type. Then it is sufficient to consider the reduced hypergraph H^- resulting from H by "fusing" each such group to a single vertex. Obviously $\rho^*(H) = \rho^*(H^-)$, and each edge-weight function for H^- can be extended in the obvious way to an edge-weight function of the same total weight to H. Finally, assumption (4) can also be made w.l.o.g., since we can again define a reduced hypergraph H^- resulting from H by retaining only one edge from each group of identical edges. Then every edge cover of H^- is an edge cover of H. Conversely, every edge cover of H can be turned into an edge cover of H^- by assigning to each edge e in H^- the sum of the weights of e and all edges identical to e in H.

Under our above assumptions (1) – (4), for every hypergraph H, the property $H^{dd} = H$ holds and there is an obvious one-to-one correspondence between the edges (vertices) of H and the vertices (edges) of H^d. Moreover, there is an obvious one-to-one correspondence between the fractional edge covers of H and the fractional vertex covers of H^d. In particular, if there is a fractional edge cover γ for H, then its corresponding "dual" γ^d assigns to each vertex v of H^d the same weight as to the edge in H that is represented by this vertex and vice versa.

Note that if we do not make assumptions (3) and (4), then there are hypergraphs H with $H^{dd} \neq H$. For instance, consider the hypergraph H_0 with $V(H_0) = \{a, b, c\}$ and $E(H_0) = \{\, e = \{a, b, c\}\,\}$, i.e., property (3) is violated. The hypergraph H_0^d has a unique vertex e and a unique hyperedge $\{e\}$. Hence, H_0^{dd} is (isomorphic to) the hypergraph with a unique vertex a and a unique hyperedge $\{a\}$, which is clearly different from the original hypergraph H_0.

To get an upper bound on the support $supp(\gamma)$ of a fractional edge cover of a hypergraph H, we make use of the following result for fractional vertex covers. This result is due to Zoltán Füredi [40], who extended earlier results by Chung et al. [26]. Below, we appropriately reformulate Füredi's result for our purposes:

Proposition 4.1 ([40], page 152, Proposition 5.11.(iii))**.** *For every hypergraph H of rank (i.e., maximal edge size) r, and every fractional vertex cover w for H satisfying $weight(w) = \tau^*(H)$, the property $|vsupp(w)| \leq r \cdot \tau^*(H)$ holds.*

By duality, exploiting the relationship $\rho^*(H) = \tau^*(H^d)$ and by recalling that the degree of H corresponds to the rank of H^d, we immediately get the following corollary:

Corollary 4.2. *For every hypergraph H of degree d, and every fractional edge cover γ for H satisfying weight$(\gamma) = \rho^*(H)$, the property $|supp(\gamma)| \leq d \cdot \rho^*(H)$ holds.*

From now on, we no longer need to make the assumptions (3) and (4) above. In fact, Proposition 4.1 and Corollary 4.2 also hold for hypergraphs that do not fulfil these conditions as was pointed out above by our considerations on reduced hypergraphs H^-. Moreover, from now on, we exclusively concentrate on fractional *edge* covers. The excursion to fractional *vertex* covers was only needed to make use of Füredi's result reformulated in Proposition 4.1 above.

Proposition 4.1 and Corollary 4.2 state bounded support properties for the *optimal* weight functions τ^* and ρ^*. The following lemma allows us to extend the upper bound $k \cdot d$ on the support of a fractional edge cover γ of width k of a hypergraph H of degree d to the fractional edge cover γ_u in *every* node u of an FHD of width $\leq k$ of H.

Lemma 4.6. *Let H be a hypergraph of degree d and let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be an FHD of H of width k. Then there exists an FHD $\mathcal{F}' = \langle T, (B_u)_{u \in T}, (\gamma'_u)_{u \in T} \rangle$ of H of width $\leq k$ such that $\mathcal{F}$ and $\mathcal{F}'$ have exactly the same tree structure T and, for every node u in T, we have $|supp(\gamma'_u)| \leq k \cdot d$ and, $B(\gamma_u) \subseteq B(\gamma'_u)$.*

Proof. Let H and $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be as above. For each node u in T, consider the subhypergraph H_u of H where $V(H_u) = B(\gamma_u)$ and $E(H_u) = \{e \cap V(H_u) \mid e \in supp(\gamma_u)\} = \{e \cap B(\gamma_u) \mid e \in supp(\gamma_u)\}$. Note that one or more edges from $supp(\gamma_u)$ may give rise to a same edge e' of H_u, when deleting vertices $v \notin B(\gamma_u)$ from edges $e \in supp(\gamma_u)$. We call all such edges the *originators* of e' and denote the set of all originator edges for e' by $orig(e')$.

Now let $\gamma_u^{\downarrow} : E(H_u) \to (0, 1]$ be the edge-weight function which assigns each edge e' of H_u weight $\gamma_u^{\downarrow}(e') = \Sigma_{e \in orig(e')} \gamma_u(e)$, i.e., the sum of all weights by γ_u of its originators. Clearly, $\gamma_u^{\downarrow}$ is a fractional edge cover of total weight at most k for H_u. Now take an optimal fractional edge cover γ_u^* for H_u. The total weight of this cannot be greater than k either. Hence, by Corollary 4.2, $supp(\gamma_u^*) \leq k \cdot d$. Now transform γ_u^* into an edge-weight function γ'_u of the entire hypergraph H by assigning for edge e' of H_u the entire weight of e' to only one of its originators, whilst assigning weight 0 to all other originators. Clearly, the support of γ'_u is bounded by $k \cdot d$ and $B(\gamma_u) \subseteq B(\gamma'_u)$. B_u is thus covered by $B(\gamma'_u)$, and the resulting FHD $\mathcal{F}'$ has all requested properties. □

We have now tackled the first challenge of this section by showing that whenever a hypergraph H has an FHD $\mathcal{F}$ of width $\leq k$ then H also has an FHD $\mathcal{F}'$ of width $\leq k$ such that in each node u of $\mathcal{F}'$ we have $|supp(\gamma'_u)| \leq k \cdot d$. We yet have to overcome the following obstacle: in the alternating algorithm in [51] for deciding the Check(HD, k) problem, we guess at every node u of the HD a set S_u of edges with $|S_u| \leq k$ such that the edges in S_u get weight 1 by λ_u and all other edges get weight 0. Hence, we get $B(\lambda_u) = \bigcup S_u$. From this, we determine the bag $B_u \subseteq B(\lambda_u)$ via the *special condition*

recalled in Definition 2.4, which distinguishes HDs from GHDs. More specifically, let u' denote the parent of u in the hypertree decomposition and let C denote the vertices in the edges that have to be covered by some node in the subtree rooted at u. Then we may set $B_u = B(\lambda_u) \cap (B_{u'} \cup C)$.

In our case, when trying to construct a *fractional* hypertree decomposition of width $\leq k$ for a hypergraph with degree bounded by d, we know by Lemma 4.6 that we may restrict ourselves to edge-weight functions γ_u with $|supp(\gamma_u)| \leq k \cdot d$. Moreover, we can be sure that $B(\gamma_u) \subseteq \bigcup S$ with $S = supp(\gamma_u)$ holds. However, in contrast to the HD-setting studied in [51], $B(\gamma_u) = \bigcup S$ does in general *not* hold. Consequently, it is, of course, also unclear how to determine B_u.

Subedge Functions. We will now provide a solution to both problems: how to determine $B(\gamma_u)$ and how to determine B_u for each node u in an FHD? But before we do this, we define some useful notation for certain unions and intersections of families of sets.

Definition 4.8. Let S be a family of sets. We define the following further families of sets.

$\Cup S$ denotes the set-family which consists in all possible unions of an arbitrary number of sets from S. (Note that $|\Cup S| \leq 2^{|S|}$).

$\Cup_i S$ for an integer $i \geq 1$, denotes the set-family which consists in all possible unions of $\leq i$ sets from S. (Note that $|\Cup_i (S)| \leq |S|^{i+1}$).

$\Cap S$ denotes the set-family which consists in all possible intersections of an arbitrary number of sets from S. (Note that $|\Cap S| \leq 2^{|S|}$).

$\Cap_i S$ for an integer $i \geq 1$, denotes the set-family which consists in all possible intersections of $\leq i$ sets from S. (Note that $|\Cap_i S| \leq |S|^{i+1}$).

If S and S' are both families of sets, then $S \sqcap S'$ denotes the *pointwise intersection* between S and S', i.e., $S \sqcap S' = \{A \cap B \mid A \in S \text{ and } B \in S'\}$. ◁

We now establish a bound on the number of possible sets $B(\gamma)$ that can arise in a hypergraph for all possible choices of a weight function γ.

Definition 4.9. Let $\mathbb{B}(H)$ denote the set of all possible sets $B(\gamma)$ such that γ is an edge-weight function of H. For $S \subseteq E(H)$, we denote by $\mathbb{B}(S)$ the set of all possible sets $B(\gamma)$ where $\gamma(e) > 0$ if $e \in S$ and $\gamma(e) = 0$ if $e \notin S$. That is, S denotes the support of γ. ◁

Definition 4.10. An *intersection type* of a hypergraph $H = (V(H), E(H))$ (or, simply a *type*, for short), is a subset of $E(H)$. For a hypergraph H, $\mathbb{T}(H) = 2^{E(H)}$ consists of all possible types of H. For a type $t \in \mathbb{T}(H)$, we define its class $class(t) = \bigcap_{e \in t} e$ as the intersection of all edges in t. The set of all classes of H is denoted by $\mathbb{C}(H)$.

For a class $c \in \mathbb{C}(H)$ there may be more than one type t with $\mathit{class}(t) = c$. However there is only one maximal type, namely $\{e' \in E(H) \mid c \subseteq e'\}$; we denote by $\mathit{type}(c)$ this unique maximal type. ◁

Note that $\mathbb{T}(H)$ and $\mathbb{C}(H)$ depend only on H and not on any edge-weight function. Moreover, every set $B(\gamma)$, for whatever edge-weight function, must be equal to the union of some classes of H. In fact, for any particular edge-weight function γ, the set $B(\gamma)$ consists of the union of all sets $\mathit{class}(t)$ for all types t that satisfy $\gamma(t) \geq 1$ where $\gamma(t) = \Sigma_{e \in t}\gamma(e)$.
following lemma.

Lemma 4.7. *Let H be a hypergraph. Then the following properties hold:*

1. *If γ is an edge-weight function, then $B(\gamma) \in \Cup\mathbb{C}(H)$.*
2. *$\mathbb{B}(H) \subseteq \Cup\mathbb{C}(H)$.*
3. *$|\mathbb{B}(H)| \leq 2^{|\mathbb{C}(H)|} \leq 2^{|\mathbb{T}(H)|} \leq 2^{2^{|E(H)|}}$ and all three sets, $\mathbb{B}$, $\mathbb{C}$ and $\mathbb{T}$, can be computed from H in polynomial time if the cardinality of $E(H)$ is bounded by a constant.*

The above inclusion $\mathbb{B}(H) \subseteq \Cup\mathbb{C}(H)$ only gives us an *exponential* upper bound $2^{2^{|E(H)|}}$ on the number of possible sets $B(\gamma_u)$ at any node u in an FHD. However, by Lemma 4.6, we may assume w.l.o.g. that $|S_u| \leq k \cdot d$ with $S_u = \mathit{supp}(\gamma_u)$ holds for every edge-weight function γ_u of interest. Hence, we only need to consider *polynomially* many values for $\mathit{supp}(\gamma_u)$. Moreover, for each S_u, there exist only *polynomially* many possible sets $B(\gamma_u)$ with $\mathit{supp}(\gamma_u) = S_u$, i.e. $|\mathbb{B}(S_u)| \leq 2^{2^{|S_u|}}$. Hence, with Lemma 4.7, the first problem stated above is essentially solved.

It remains to find a solution to the second problem stated above, i.e., how to determine B_u for each node u in an FHD of width k? We tackle this problem by again using the idea of *subedge functions* as described in Section 4.2 for deriving tractability results for the Check(GHD, k) problem. A subedge function takes as input a hypergraph $H = (V(H), E(H))$ and produces as output a set E' of subedges of the edges in $E(H)$, such that E' is then added to $E(H)$. Clearly, adding a set E' of subedges does not change the *fhw* of H. Below, we shall define a whole family of subedge functions $h_{d,k}$, which, for fixed upper bounds d on the degree and k on the *fhw*, take a hypergraph H as input and return a polynomially bounded, polynomial-time computable set E' of subedges of $E(H)$. Adding these subedges to $E(H)$ will then allow us to define a *polynomial* upper bound on the set of all possible bags B_u at a given node u in an FHD of H.

Towards this goal, we follow a similar approach as in the proof of Theorem 4.2. There we have used the LogBMIP to devise a polynomially bounded subedge function. Here, we will restrict ourselves to hypergraphs of bounded degree.

There are mainly two issues when trying to adapt the construction from the GHD case. First, we carry over the notion of bag-maximality from GHDs to FHDs in the obvious

way: we say that an FHD $\mathcal{F}$ is *bag-maximal* if for each node u of $\mathcal{F}$, for every vertex $v \in B(\gamma_u) \setminus B_u$, adding v to B_u would violate the connectedness condition. Clearly, for every FHD $\langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$, a bag-maximal FHD $\mathcal{F}^+ = \langle T, (B_u^+), (\gamma_u) \rangle$ can be generated by adding vertices from $B(\gamma_u) \setminus B_u$ to bags B_u as long as possible. We may thus assume w.l.o.g. that our FHD $\mathcal{F}$ is bag-maximal.

For the definition of an appropriate subedge function (denoted $h_{d,k}$ to indicate that it depends on d and k), take a hypergraph H with degree bounded by $d \geq 1$ and consider an arbitrary FHD $\mathcal{F}$ of H of width $\leq k$. Let u be an arbitrary node in $\mathcal{F}$ with edge-weight function γ_u and let $e \in \mathit{supp}(\gamma_u)$ with $e \cap B(\gamma_u) \not\subseteq B_u$. As in Section 4.2, our goal is to define $h_{d,k}$ in such a way that $e' = e \cap B_u$ is contained in $h_{d,k}$ for every edge $e \in \mathit{supp}(\gamma_u)$ and every possible bag B_u in $\mathcal{F}$. As a first step towards this goal, we extend the notion of critical paths from the GHD setting to FHDs.

Definition 4.11. Let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be an FHD of a hypergraph H. Moreover, let u be a node in $\mathcal{F}$ and let $e \in \mathit{supp}(\gamma_u)$ with $e \cap B(\gamma_u) \not\subseteq B_u$. Let u^* denote the node closest to u, such that u^* covers e, i.e., $e \subseteq B_{u^*}$. Then, analogously to Definition 4.5, we call the path $\pi = (u_0, u_1, \ldots, u_l)$ with $u_0 = u$ and $u_l = u^*$ the *critical path* of (u, e) denoted as $\mathit{critp}(u, e)$. ◁

The following lemma allows us to characterize the subsets e' of e needed in the subedge function $h_{d,k}$. The proof of Lemma 4.8 can be literally translated from the proof of Lemma 4.5.

Lemma 4.8. *Let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be a bag-maximal FHD of a hypergraph $H = (V(H), E(H))$, let $u \in T$, $e \in \mathit{supp}(\gamma_u)$, and $e \cap B(\gamma_u) \not\subseteq B_u$. Let $\pi = (u_0, u_1, \ldots, u_l)$ with $u_0 = u$ be the critical path of (u, e). Then the following equality holds:*

$$e \cap B_u = e \cap \bigcap_{i=1}^{l} B(\gamma_{u_i})$$

We want to define the subedge function $h_{d,k}$ such that all subedges appearing on the right-hand side of above equality. To achieve this while abstracting from the knowledge of a particular decomposition and from the knowledge of particular edge-weight functions, we will make two bold over-approximations. First, instead of considering concrete critical paths, we will consider arbitrary finite sequences $\xi = (\xi_1, \ldots, \xi_{\max(\xi)})$ of groups of $\leq k \cdot d$ edges of H, where each such group represents a potential support $\mathit{supp}(\gamma_u)$ at some potential node u of a potential FHD of H. Clearly, each effective path $\mathit{critp}(u, e)$ for any possible combination of a decomposition node u and an edge e of any possible FHD $\mathcal{F}$ of H is among these sequences. The second over-approximation we make is that instead of considering particular edge-weight functions, we will simply consider (a superset of) all possible supports of $\leq k \cdot d$ atoms, and for each such support S (a superset of) all possible sets $B(\gamma)$, i.e. $\mathbb{B}(S)$, that could possibly arise with this support. A support is simply given by a subset of $\leq k \cdot d$ edges of H. For each such support, by Lemma 4.7,

there are in fact no more than $2^{2^{k \cdot d}}$ $B(\gamma)$-sets and these are determined by unions of classes from $\mathbb{C}(H')$, where H' is the subhypergraph of H given by the support. To make this more formal, we give the following definition. Recall the notion of $\mathbb{C}$ (which denotes the set of intersections of edges contained in some type; in our case, each type consists of at most $k \cdot d$ edges) from Definition 4.10.

Definition 4.12. Let H be a hypergraph and let $\xi = (\xi_1, \ldots, \xi_{\max(\xi)})$ be an arbitrary sequence of groups of $\leq k \cdot d$ edges of H. For $i \in \{1, \ldots, \max(\xi)\}$, by slight abuse of notation, we overload the notion of $\mathbb{C}$ from Definition 4.10 as follows: we write $\mathbb{C}(\xi_i)$ to denote the set $\mathbb{C}(H^i_\xi)$, where H^i_ξ is the subhypergraph of H whose edges are the $\leq k \cdot d$ edges of ξ_i and whose vertices are precisely all vertices occurring in these edges.

Let π be a critical path of the form $\pi = (u_0, u_1, \ldots, u_l)$ of some FHD $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ of H. Suppose that each edge-weight function γ_u in $\mathcal{F}$ has $(k \cdot d)$-bounded support. Then we denote by ξ^π the sequence $\xi^\pi = (\xi_1, \ldots, \xi_l)$ with $\xi_i = \mathit{supp}(\gamma_{u_i})$ for $1 \leq i \leq l$. ◁

Our goal is to compute a set of subedges of the edges in $E(H)$ containing all sets of the form $e \cap \bigcap_{i=1}^{l} B(\gamma_{u_i})$ with $\pi = \mathit{critp}(u, e) = (u_0, u_1, \ldots, u_l)$. We may use that each γ_{u_i} has $(k \cdot d)$-bounded support. By Lemma 4.7, we know that every possible $B(\gamma_{u_i})$-set is contained in $\biguplus \mathbb{C}(H)$, i.e., every possible $B(\gamma_{u_i})$ along a critical path π can be represented as the union of classes (where each class is in turn the intersection of some edges selected from $\mathit{supp}(\gamma_{u_i})$). Hence, to obtain $e \cap \bigcap_{i=1}^{l} B(\gamma_{u_i})$, we need to compute the intersection of all unions of classes along a critical path π.

Recall that we generalize the support of edge-weight functions γ_{u_i} along a concrete critical path π in a concrete FHD $\mathcal{F}$ of H to sequences $\xi = (\xi_1, \ldots, \xi_{\max(\xi)})$, where each ξ_i is an arbitrary set of $\leq k \cdot d$ edges from H. As the crucial data structure to compute the desired intersections of unions of classes, we now define the *intersection forest* $\mathit{IF}(\xi)$. This data structure will give us a systematic way to convert the intersections of unions of classes for all possible sequences ξ into a union of intersections. Intuitively, each branch (starting at a root) in $\mathit{IF}(\xi)$ represents a possible transversal of the family $\{\mathbb{C}(\xi_i)\}_{1 \leq i \leq \max(\xi)}$ for some sequence $\xi = (\xi_1, \ldots, \xi_{\max(\xi)})$, i.e., a transversal selects one class from $\mathbb{C}(\xi_i)$ for each $i \in \{1, \ldots, \max(\xi)\}$. On every branch, we will then compute the intersection of the classes selected along this branch. Since each class is in turn an intersection of edges (namely the edges contained in some type), every branch in $\mathit{IF}(\xi)$ therefore simply yields an intersection of edges from H. Similar to the `Union-of-Intersections-Tree` algorithm presented in Section 4.2, we formalize the construction of $\mathit{IF}(\xi)$ in the Algorithm 4.2 "`Intersection-Forest`". We define $\mathit{IF}(\xi)$ as a rooted forest such that each of its nodes v is labelled by

- a set of vertices $\mathit{set}(v) \subseteq V(H)$,
- a set of levels $\mathit{levels}(v) \subseteq \{1, \ldots, \max(\xi)\}$,

Algorithm 4.2: `Intersection-Forest`

input : A sequence $\xi = (\xi_1, \ldots, \xi_{\max(\xi)})$ of groups of $\leq k \cdot d$ edges of a hypergraph H.
output : $IF(\xi)$

`/* Initialization: create a tree` T_c `for each` $c \in \mathbb{C}(\xi_1)$ `*/`
$IF(\xi) \leftarrow \emptyset$;
foreach $c \in \mathbb{C}(\xi_1)$ **do**
  $N_c \leftarrow \{v\}$; $E_c \leftarrow \emptyset$; $T_c \leftarrow (N_c, E_c)$;
  $set(v) \leftarrow c$;
  $levels(v) \leftarrow \{1\}$;
  $edges(v) \leftarrow \{e \in E(H) \mid c \subseteq e\}$;
  $mark(v) \leftarrow ok$;
  $IF(\xi) \leftarrow IF(\xi) \cup T_c$;
end
`/* Expand and update the trees as follows */`
for $i \leftarrow 2$ **to** $\max(\xi)$ **do**
  foreach *leaf node* v *of a tree* $T_v = (N_v, E_v) \in IF(\xi)$ *with* $\max(levels(v)) = i - 1$ *and* $mark(v) = ok$ **do**
    foreach $c \in \mathbb{C}(\xi_i)$ **do**
      switch $set(v) \cap c$ **do**
        case $set(v) \cap c = \emptyset$ **do** `/* Dead End */`
          $mark(v) \leftarrow fail$
        end
        case $set(v) \cap c = set(v)$ **do** `/* Passing */`
          $levels(v) = levels(v) \cup \{i\}$
        end
        case $set(v) \cap c \subsetneq set(v)$ **do** `/* Expand */`
          Create a new node v';
          $N_v \leftarrow N_v \cup \{v'\}$; $E_v \leftarrow E_c \cup \{v, v'\}$; $T_v \leftarrow (N_v, E_v)$;
          $set(v') \leftarrow set(v) \cap c$;
          $levels(v') \leftarrow \{i\}$;
          $edges(v') \leftarrow \{e \in E(H) \mid set(v') \subseteq e\}$;
          $mark(v') \leftarrow ok$;
        end
      end
    end
  end
end

- a set of edges $edges(v)$ such that $edges(v) = \{e \in E(H) \mid set(v) \subseteq e\}$;
  in other words, $set(v)$ is a *class* and $edges(v)$ is its (maximal) *type*, see Definition 4.10,
- and a mark $mark(v) \in \{ok, fail\}$.

The expansion of the trees in Algorithm 4.2 can be seen as follows:

1. *Dead End.* If for each class c of $\mathbb{C}(\xi_i)$, $set(v) \cap c = \emptyset$, then v has no children, and its mark is set to $mark(v) = fail$. Intuitively this is a dead end as it cannot be continued to yield a non-empty intersection of a transversal of the family $\{\mathbb{C}(\xi_j)\}_{1 \leq j \leq \max(\xi)}$.
2. *Passing.* For each class c of $\mathbb{C}(\xi_i)$ fulfilling $set(v) \cap c = set(v)$, insert $i+1$ into $levels(v)$. Intuitively, this makes sure the same value $set(v)$ is never repeated on a branch, and, as a consequence, every child node must have a strictly smaller $set()$-component and, thus, at least one more edge in its $edges()$-label than its parent (see also Fact 1 in Lemma 4.9 below).
3. *Expand.* For each class c of $\mathbb{C}(\xi_{i+1})$ fulfilling $set(v) \cap c \subsetneq set(v)$, create a child v' of v, and let $set(v') = set(v) \cap c$, $levels(v') = \{i+1\}$, $edges(v') = \{e \in E(H) \mid set(v') \subseteq e\}$, and $mark(v) = ok$. Note that we thus clearly have $set(v') \subsetneq set(v)$ and $edges(v) \subsetneq edges(v')$.

We now define some useful notation for talking about the intersection forest $IF(\xi)$.

Definition 4.13. Let H be a hypergraph and let $\xi = (\xi_1, \ldots, \xi_{\max(\xi)})$ be an arbitrary sequence of groups of $\leq k \cdot d$ edges of H. For $1 \leq i \leq \max(\xi)$, let $iflevel_i(\xi)$ denote the set of all nodes v of $IF(\xi)$ such that $i \in levels(v)$ and $mark(v) = ok$. We denote by $\mathbb{F}_i(\xi)$ the collection of all sets $set(v)$ where $v \in iflevel_i(\xi)$. Finally, let the *fringe of* ξ be defined as $\mathbb{F}(\xi) = \mathbb{F}_{\max(\xi)}(\xi)$. ◁

Note that, by definition of $IF(\xi)$, there cannot be any *fail* node on level $\max(\xi)$. Hence, $\mathbb{F}(\xi)$ coincides with the set of all *set*-labels at level $\max(\xi)$, i.e. the leaf nodes v of all trees in $IF(\xi)$ such that $mark(v) = ok$. We now establish some easy facts about $IF(\xi)$.

Lemma 4.9. *Let H be a hypergraph with degree d and let $\xi = \xi_1, \ldots, \xi_{\max(\xi)}$ be an arbitrary sequence of groups of $\leq k \cdot d$ edges of H. Then the intersection forest $IF(\xi)$ according to the construction in Algorithm 4.2 has the following properties:*

Fact 1. *If node v' is a child of node v in $IF(\xi)$, then $edges(v')$ must contain at least one new edge in addition to the edges already present in $edges(v)$.*

Fact 2. *The depth of $IF(\xi)$ is at most $d-1$.*

Fact 3. *Let $a = 2^{k \cdot d}$. Then $IF(\xi)$ has no more than a^{d+1} nodes and $|\mathbb{F}(\xi)| \leq a^d = 2^{d^2 \cdot k}$.*

Proof. The facts stated above can be seen as follows.

Fact 1. Note that we can only create a child node through an *Expand* operation. This requires that $set(v') = set(v) \cap c \subsetneq set(v)$ for some class $c \in \mathbb{C}(\xi_i)$. Recall that c is the intersection of all edges of $type(c)$. Hence, for $set(v')$ to shrink, $type(c)$ must contain at least one new edge not yet contained in $edges(v)$, and this edge is therefore included into $edges(v')$.

Fact 2. This follows from Fact 1 and the fact that, if H is of degree d, then any intersection of $d+1$ or more edges is empty.

Fact 3. For each sequence ξ as above, for whatever ξ_i, the inequality $|\mathbb{C}(\xi_i)| \leq a$ holds by Lemma 4.7. Hence, Fact 3 follows from the depth $d-1$ established in Fact 2 and the fact that we have at most a such trees, each with branching not larger than a.

This concludes the proof of the lemma. □

Recall that we are studying the set of arbitrary sequences $\xi = (\xi_1, \ldots, \xi_{\max(\xi)})$ of groups of $\leq k \cdot d$ edges of H because they give us a superset of possible critical paths $\pi = critp(u, e)$ in possible FHDs of H, such that each group ξ_i of edges corresponds to the support of the edge-weight function γ_{u_i} at the i-th node u_i on path π. The following lemma establishes that the intersection forests (and, in particular, the notion of $\mathbb{F}(\xi)$) introduced above indeed give us a tool to generate a superset of the set of all possible sets $\bigcap_{i=1}^{l} B(\gamma_{u_i})$ in all possible FHDs of H of width $\leq k$.

Lemma 4.10. *Let H be a hypergraph of degree $d \geq 1$ and let $\mathcal{F}$ be an FHD of H of width $\leq k$. Consider a critical path $\pi = (u_0, u_1, \ldots, u_l)$ of FHD $\mathcal{F}$, together with its associated sequence ξ^π introduced in Definition 4.12. We claim that the following relationship holds:*

$$\bigcap_{i=1}^{l} B(\gamma_{u_i}) \;\in\; \Cup\mathbb{F}(\xi^\pi)$$

Proof. We show by induction on i that, for every $i \in \{1, \ldots, l\}$, the following relationship holds:

$$\bigcap_{j=1}^{i} B(\gamma_{u_j}) \in \Cup\mathbb{F}_i(\xi^\pi)$$

Base Step. For the base case $i = 1$, recall that in the `Intersection-Forest` algorithm, the set $\mathbb{F}_1(\xi^\pi)$ is initialized to $\mathbb{C}(\xi_1)$ with $\xi_1 = supp(\gamma_{u_1})$. Moreover, by Lemma 4.7(2), $\mathbb{B}(H) \subseteq \Cup\,\mathbb{C}(H)$ and, hence, we have $\mathbb{B}(supp(\gamma_{u_1})) \subseteq \Cup\,\mathbb{C}(supp(\gamma_{u_1}))$. In total, $\mathbb{B}(supp(\gamma_{u_1})) \subseteq \Cup\mathbb{F}_1(\xi^\pi)$ indeed holds.

Inductive Step. Assume for some $i < l$ that $\bigcap_{j=1}^{i} B(\gamma_{u_j}) \in \Cup\mathbb{F}_i(\xi^\pi)$ holds. We show that the desired relationship also holds for $i+1$. Clearly, $\bigcap_{j=1}^{i+1} B(\gamma_{u_j}) = \bigcap_{j=1}^{i} B(\gamma_{u_j}) \cap B(\gamma_{u_{i+1}})$.

From this, by the induction hypothesis, together with $B(\gamma_{u_{i+1}}) \in \mathbb{B}(\xi^{\pi}_{i+1})$, and the inclusion $\mathbb{B}(\xi^{\pi}_{i+1}) \subseteq \Cup\mathbb{C}(\xi^{\pi}_{i+1})$, which holds by Lemma 4.7, we obtain:

$$\bigcap_{j=1}^{i+1} B(\gamma_{u_j}) \in \mathbb{F}_i(\xi^{\pi}) \cap \Cup\mathbb{C}(\xi^{\pi}_{i+1}).$$

By using the distributivity of $\cap$ over $\Cup$, we get:

$$\mathbb{F}_i(\xi^{\pi}) \cap \Cup\mathbb{C}(\xi^{\pi}_{i+1}) = \Cup(\mathbb{F}_i(\xi^{\pi}) \cap \mathbb{C}(\xi^{\pi}_{i+1})).$$

Moreover, by the construction of $IF(\xi^{\pi})$, we have

$$\mathbb{F}_i(\xi^{\pi}) \cap \mathbb{C}(\xi^{\pi}_{i+1}) = \mathbb{F}_{i+1}(\xi^{\pi}).$$

In fact, the *Passing* and *Expand* cases make precisely these intersections when producing level $i+1$ of $IF(\xi^{\pi})$. Therefore, in total, we obtain that

$$\bigcap_{j=1}^{i+1} B(\gamma_{u_j}) \in \Cup\mathbb{F}_{i+1}(\xi^{\pi})$$

indeed holds, which settles the inductive step. □

The desired subedge function $h_{d,k}$ therefore looks as follows:

Lemma 4.11. *Let H be a hypergraph of degree $d \geq 1$ and let $k \geq 1$. Let the subedge function $h_{d,k}$ be defined as*

$$h_{d,k}(H) = E(H) \cap (\Cup_{2^{d^2 \cdot k}} \cap_d E(H))$$

Then (for fixed constants d and k), the size of $h_{d,k}(H)$ is polynomially bounded and $h_{d,k}(H)$ can be computed in polynomial time. Moreover $h_{d,k}(H)$ contains all subedges $e \cap B_u$ of all $e \in E(H)$ for all possible bags B_u of whatever bag-maximal FHD of width $\leq k$ of H.

Proof. For any sequence ξ, each element of $\mathbb{F}(\xi)$ is the intersection of at most d edges. Moreover, by Fact 3 of Lemma 4.9, $|\mathbb{F}(\xi)| \leq a^d = 2^{d^2 \cdot k}$ holds. Therefore, for all possible sequences ξ, we have

$$\Cup\mathbb{F}(\xi) \subseteq \Cup_{2^{d^2 \cdot k}} \cap_d E(H).$$

Given that d and k are constants, the set $\Cup_{2^{d^2 \cdot k}} \cap_d E(H)$ is of polynomial size and is clearly computable in polynomial time from H. By Lemma 4.8 together with Lemma 4.10, it is then also clear that the subedge function $h_{d,k}(H)$ contains all subedges $e \cap B_u$ for all possible bags of whatever bag-maximal FHD of width $\leq k$ of H. □

Deciding the Check Problem for Hypergraphs of Bounded Degree. With the subedge function $h_{d,k}$ at hand, we have a powerful tool that will allow us to devise a polynomial-time decision procedure for the Check(HD, k) problem. Towards this goal, we first observe that adding the edges in $h_{d,k}(H)$ to a hypergraph H allows us to restrict ourselves to FHDs of a very peculiar form. This form is captured by the following definition.

Definition 4.14. An FHD $\mathcal{F} = \langle T, (B_u)_{u\in T}, (\gamma_u)_{u\in T}\rangle$ of a hypergraph H is *strict* if for every decomposition node u in T, the equality $B_u = B(\gamma_u) = \bigcup supp(\gamma_u)$ holds. ◁

Below we show that, in case of bounded degree, we can transform every FHD of width $\leq k$ into a strict FHD of width $\leq k$.

Lemma 4.12. *Let $H = (V(H), E(H))$ be a hypergraph of degree $\leq d$ and let $\mathcal{F} = \langle T, (B_u)_{u\in T}, (\gamma_u)_{u\in T}\rangle$ be an FHD of H of width $\leq k$. Suppose that H' is obtained from H by adding the edges in $h_{d,k}(H)$, i.e., $H' = (V(H'), E(H'))$ with $V(H') = V(H)$ and $E(H') = E(H) \cup h_{d,k}(H)$. Then H' admits a* strict *FHD $\mathcal{F}' = \langle T, (B_u)_{u\in T}, (\gamma'_u)_{u\in T}\rangle$ of width $\leq k$.*

Proof. Let $\mathcal{F} = \langle T, (B_u)_{u\in T}, (\gamma_u)_{u\in T}\rangle$ be an arbitrary FHD of H of width $\leq k$. W.l.o.g., assume that $\mathcal{F}$ is bag-maximal. Of course, $\mathcal{F}$ is also an FHD of H' of width $\leq k$. Let u be a node in $\mathcal{F}$ and let $e \in supp(\gamma_u)$. Suppose that $e \cap B(\gamma_u) \not\subseteq B_u$. We modify γ_u as follows: by Lemma 4.11, $E(H) \cup h_{d,k}(H)$ is guaranteed to contain the subedge $e' = e \cap B_u$ of e. Then we "replace" e in γ_u by e', i.e., we set $\gamma_u(e') := \gamma_u(e') + \gamma_u(e)$ and $\gamma_u(e) := 0$. The FHD $\mathcal{F}' = \langle T, (B_u)_{u\in T}, (\gamma'_u)_{u\in T}\rangle$ is obtained by exhaustive application of this transformation step. Clearly, such a transformation step never increases the support. Moreover, the resulting FHD $\mathcal{F}'$ is strict. □

Our strategy to devise a polynomial-time decision procedure for the Check(FHD, k) problem is to reduce it to the Check(HD, k) problem and then adapt the algorithm from [51]. However, this algorithm only finds HDs that are in *normal form*. Hence, we also need to ensure a normal form for the FHDs we are going to compute. As we have seen in Section 2.4 it is easy to carry over the normal form from [51] to the fractional setting. Indeed, it is shown in Theorem 2.1 that any arbitrary FHD can be transformed into *fractional normal form*. We now use the transformations from the proof of Theorem 2.1 to strengthen Lemma 4.12, such that FNF and bounded support can be guaranteed.

Lemma 4.13. *Let $H = (V(H), E(H))$ be a hypergraph of degree $\leq d$ and let $\mathcal{F} = \langle T, (B_u)_{u\in T}, (\gamma_u)_{u\in T}\rangle$ be an FHD of H of width $\leq k$. Suppose that H' is obtained from H by adding the edges in $h_{d,k}(H)$, i.e., $H' = (V(H'), E(H'))$ with $V(H') = V(H)$ and $E(H') = E(H) \cup h_{d,k}(H)$. Then H'admits a* strict *FHD $\mathcal{F}' = \langle T, (B_u)_{u\in T}, (\gamma'_u)_{u\in T}\rangle$* in fractional normal form *of width $\leq k$ that has $(k \cdot d)$-bounded support.*

Proof. By Lemma 4.6 there exists an FHD $\mathcal{F}_1$ of H whose supports are all bounded by $k \cdot d$. Without changing the supports, we can transform this FHD into a bag-maximal one, and we thus assume w.l.o.g. that $\mathcal{F}_1$ is bag-maximal.

Now transform $\mathcal{F}_1$ into an FHD $\mathcal{F}_2$ of width $k \cdot d$ *in FNF*, by proceeding according to the proof of Theorem 2.1 in Section 2.4, which, in turn follows closely the transformation in the proof of Theorem 5.4 in [51]. Note that this transformation preserves the support bound of $k \cdot d$. In fact, the component split made for ensuring condition 1 of FNF can never lead to a larger support, given that the bags never increase. Ensuring condition 2 results in eliminating nodes from the tree, so nothing bad can happen. Observe that condition 3, which is $B(\gamma_s) \cap B_r \subseteq B_s$ for a child node s of decomposition node r, is initially satisfied, because the initial FHD $\mathcal{F}_1$ is bag-maximal. Observe further that the splitting of a node (subtree) into several nodes (subtrees) performed to achieve condition 1 of FNF does not destroy the validity of condition 3.

Finally transform the FHD $\mathcal{F}_2$ via Lemma 4.12 into a strict FHD $\mathcal{F}' = \langle (T, (B_u)_{u \in T}, (\gamma'_u)_{u \in T} \rangle$ of H' of width $\leq k$ and with $(k \cdot d)$-bounded support. This strict FHD $\mathcal{F}'$ is still in FNF. To see this, first note that the tree structure T of the decomposition and all bags B_u remain exactly the same. Moreover, for whatever set $S \subseteq V(H)$, H and H' have exactly the same $[S]$-components. This can be seen by recalling from [51] that two vertices v_1, v_2 in a hypergraph H are $[S]$-adjacent if they are adjacent in the subhypergraph of H induced by $V(H) \setminus S$. Hence, $[S]$-adjacency remains unaltered when adding subedges.

Given that conditions 1 and 2 of FNF are only formulated in terms of B_u-bags and $[B_u]$-components – and all such bags and components are the same for $\mathcal{F}$ and $\mathcal{F}'$ – they remain valid. Condition 3, which requires that $B(\gamma'_s) \cap B_r \subseteq B_s$ for child node s of r, is now trivially satisfied, because $\mathcal{F}'$ is *strict* and, therefore, even $B(\gamma'_s) = B_s$ holds. □

The following theorem finally establishes the close connection between the Check(FHD, k) and Check(HD, k) problems for hypergraphs H of degree bounded by some constant $d \geq 1$. Recall that the edge-weight functions λ_u in an HD only assign values 0 or 1 to edges. As in [51], it is convenient to identify λ_u with a set S_u of edges, namely the edges in $E(H)$ that are assigned value 1. In other words, $S_u = \mathit{supp}(\lambda_u)$. Moreover, we can identify a set of edges S_u with the hypergraph whose set of vertices is $\bigcup S_u$ and whose set of edges is S_u. For given edge-weight function λ_u with $S_u = \mathit{supp}(\lambda_u)$, we shall write H_{λ_u} to denote this hypergraph.

Theorem 4.5. *Let H be a hypergraph whose degree is bounded by $d \geq 1$ and define H' as above, i.e., $H' = (V(H'), E(H'))$ with $E(H') = E(H) \cup h_{d,k}(H)$. Then the following statements are equivalent:*

1. *$\mathit{fhw}(H) \leq k$.*
2. *H' admits a strict hypertree decomposition (thus a query decomposition) $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ of width $\leq k \cdot d$ in normal form such that for each decom-*

position node u of $\mathcal{H}$, $\rho^*(H_{\lambda_u}) \leq k$ holds (i.e., H_{λ_u} has a fractional edge cover of weight $\leq k$).

Proof. To prove $1 \Rightarrow 2$, we use Lemma 4.13 to conclude from $\mathit{fhw}(H) \leq k$ that there exists a strict FHD $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ of H in *fractional normal form* of width $\leq k$ that has $(k \cdot d)$-bounded support. The FHD $\mathcal{F}$ can be naturally transformed into a GHD $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ by leaving the tree structure and the bags B_u unchanged and by defining λ_u as the characteristic function of $\mathit{supp}(\gamma_u)$, i.e., $\lambda_u(e) = 1$ if $e \in \mathit{supp}(\gamma_u)$ and $\lambda_u(e) = 0$ otherwise. Since $\mathcal{F}$ is strict, it follows immediately that the resulting GHD satisfies the special condition, i.e., $\mathcal{H}$ is in fact an HD. Moreover, since $\mathcal{F}$ is in (fractional) normal form, also the HD $\mathcal{H}$ is in the normal form from [51].

To see $2 \Rightarrow 1$, assume 2 holds with query decomposition $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ and assume further that, for each decomposition node u of $\mathcal{H}$, there exists a fractional edge cover γ'_u for H_{λ_u} of width $\leq k$. In particular, we thus have $B(\lambda_u) = B(\gamma'_u) = B_u$. Similarly to the proof of Lemma 4.6, we can transform each fractional edge cover γ'_u of the induced subhypergraph H_{λ_u} of H' into an edge-weight function γ_u of H by moving the weights $\gamma'_u(e')$ of each edge e' in H_{λ_u} to one of its "originator edges" e in H (i.e., an edge e in H with $e' \subseteq e$). By replacing λ_u with γ_u, we obtain an FHD $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ of width $\leq k$ of H. □

We are now ready to prove the main result of this section:

Proof of Theorem 4.4. By Theorem 4.5, it is sufficient to look for a strict hypertree decomposition (thus a query decomposition) $\mathcal{H} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ of H' of width $\leq d \cdot k$ such that for each decomposition node u of $\mathcal{H}$, $\rho^*(H'_{\lambda_u}) \leq k$ holds. This is achieved by modifying the alternating algorithm k-decomp from [51] by inserting the following two checks at each node u:

- if u has a parent r, then $\bigcup S_u \subseteq B(\lambda_r) \cup \mathit{treecomp}(u)$ where $S_u = \mathit{supp}(\lambda_u)$ and $\mathit{treecomp}(u)$ is defined as the set of vertices which have to appear in T_u and which do not appear outside T_u. This makes sure that we may set $B_u = \bigcup S_u$ without violating the connectedness condition. Hence, the resulting decomposition is strict.
- $\rho^*(H'_{\lambda_u}) \leq k$.

The so modified algorithm clearly runs in $\mathsf{ALogSpace} = \mathsf{PTime}$ □

4.4 Efficient Approximation of FHDs

We now turn our attention to approximations of the *fhw*. It is known from [74] that a tractable cubic approximation of the *fhw* always exists, i.e.: for $k \geq 1$, there exists a polynomial-time algorithm that, given a hypergraph H with $\mathit{fhw}(H) \leq k$, finds an FHD

of H of width $\mathcal{O}(k^3)$. In this section, we search for conditions which guarantee a better approximation of the *fhw*.

Natural first candidates for restricting hypergraphs are the BIP and, more generally, the BMIP. For the Check(GHD, k) problem, these restrictions guarantee tractability. We have to leave it as an open question for future research if the BIP or even the BMIP also guarantees tractability of the Check(FHD, k) problem for fixed $k \geq 1$. However, in this section, we will show that a significantly better polynomial-time approximation of the *fhw* than in the general case is possible for hypergraphs enjoying the BIP or BMIP.

4.4.1 Approximation of FHDs in case of the BIP

We first inspect the case of the bounded intersection property. We will show that the BIP allows for an arbitrarily close approximation of the *fhw* in polynomial time. Formally, the main result of this section is as follows:

Theorem 4.6. *Let $\mathscr{C}$ be a hypergraph class that enjoys the BIP and let k, ϵ be arbitrary constants with $k \geq 1$ and $\epsilon > 0$. Then there exists a polynomial-time algorithm that, given a hypergraph $H \in \mathscr{C}$ with $fhw(H) \leq k$, finds an FHD of H of width $\leq k + \epsilon$.*

In the remainder of this section, we develop the necessary machinery to finally prove Theorem 4.6. For this, we first introduce the crucial concept of a *c-bounded fractional part*. Intuitively, FHDs with c-bounded fractional part are FHDs, where the fractional edge cover γ_u in every node u is "close to an (integral) edge cover" – with the possible exception of up to c vertices in $B(\gamma_u)$.

It is convenient to first introduce the following notation: let $\gamma : E(H) \to [0, 1]$ and let $S \subseteq supp(\gamma)$. We write $\gamma|_S$ to denote the *restriction of γ to S*, i.e., $\gamma|_S(e) = \gamma(e)$ if $e \in S$ and $\gamma|_S(e) = 0$ otherwise.

Definition 4.15. Let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be an FHD of some hypergraph H and let $c \geq 0$. We say that $\mathcal{F}$ has *c-bounded fractional part* if in every node $u \in T$, the following property holds: Let $R = \{e \in supp(\gamma_u) \mid \gamma_u(e) < 1\}$. Then $|B(\gamma_u|_R)| \leq c$. ◁

Clearly, for the special case $c = 0$, an FHD with c-bounded fractional part is essentially a GHD; in this case, we can simply define $\lambda_u(e) = 1$ if $\gamma_u(e) = 1$ and $\lambda_u(e) = 0$ otherwise, for every decomposition node u. We next generalize the special condition (i.e., Condition 4 of the definition of HDs) to FHDs. To this end, we define the *weak special condition*. Intuitively, it requires that the special condition must be satisfied by the integral part of each fractional edge cover. For the special case $c = 0$, an FHD with c-bounded fractional part satisfying the weak special condition is thus essentially a GHD satisfying the special condition, i.e., an HD.

Definition 4.16. Let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be an FHD of some hypergraph H. We say that $\mathcal{F}$ satisfies the *weak special condition* if in every node $u \in T$, the following property holds: for $S = \{e \in E(H) \mid \gamma_u(e) = 1\}$, we have $B(\gamma_u|_S) \cap V(T_u) \subseteq B_u$. ◁

The proof of Theorem 4.6 will be based on two key lemmas. Consider constants $k \geq 1$, $i \geq 0$, and $\epsilon > 0$, and suppose that we are given a hypergraph H with $iwidth(H) \leq i$. We will show the following properties: if H has an FHD of width $\leq k$, then (1) H also has an FHD of width $\leq k + \epsilon$ with c-bounded fractional part, where c only depends on k, ϵ, and i, but not on the size of H, and (2) we can extend H to a hypergraph H' by adding polynomially many edges, such that H' has an FHD of width $\leq k + \epsilon$ with c-bounded fractional part satisfying the weak special condition. Theorem 4.6 will then be proved by appropriately adapting the Check(HD, k) algorithm from [51].

c-bounded fractional part and weak special condition. We start by proving two lemmas which, taken together, ensure that in case of the BIP, we can transform an arbitrary FHD into an FHD with c-bounded fractional part satisfying the weak special condition at the expense of increasing the width by at most ϵ.

Lemma 4.14. *Consider constants $k \geq 1$, $i \geq 0$, and $\epsilon > 0$, and suppose that we are given a hypergraph H with $iwidth(H) \leq i$. If H has an FHD of width $\leq k$, then it also has an FHD of width $\leq k + \epsilon$ with c-bounded fractional part, where c only depends on k, ϵ, and i, but not on the size of H. More precisely, we have $c = 2ik^2 + \frac{4k^3 i}{\epsilon}$.*

Proof. Consider an arbitrary node u in an FHD $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ of H and let γ_u be an optimal fractional edge cover of B_u with $weight(\gamma_u) \leq k$. It suffices to show that there exists a fractional edge cover γ_u^* of $B(\gamma_u)$ (and, hence, of B_u) with $weight(\gamma_u^*) \leq k + \epsilon$ and $|B(\gamma_u^*|_{R^*})| \leq c$ for $R^* = \{e \in supp(\gamma_u^*) \mid \gamma_u^*(e) < 1\}$.

We first partition the edges in $supp(\gamma_u)$ into "heavy" ones (referred to as E_h) and "light-weight" ones (referred to as E_ℓ). Moreover, we further partition the heavy edges into "big" and "small" ones (referred to as E_h^b and E_h^s, respectively). For the border between "heavy" and "light-weight" edges, we could, in principle, choose any value $w \in (0, 1)$. To keep the notation simple, we choose $w = 0.5$. For the border between "big" and "small" edges, we have to choose a specific constant d, which will be introduced below. We thus define the following sets of edges:

$E_\ell = \{e \in supp(\gamma_u) \mid \gamma_u(e) < 0.5\}$,

$E_h = \{e \in supp(\gamma_u) \mid \gamma_u(e) \geq 0.5\}$,

$E_h^s = \{e \in E_h \mid |e \cap B(\gamma_u)| < d\}$, and

$E_h^b = \{e \in E_h \mid |e \cap B(\gamma_u)| \geq d\}$ with $d = \frac{2k^2 i}{\epsilon}$

This allows us to define the subsets V_ℓ, V_s, and V_b of the vertices in $B(\gamma_u)$, s.t. V_ℓ consists of the vertices only covered by light-weight edges, V_s consists of the vertices contained in at least one heavy edge but not in a big one, and V_b consists of the vertices contained in at least one big heavy edge, i.e.:

$V_\ell = \{x \in B(\gamma_u) \mid \forall e \in E_h \colon x \notin e\}$,

$V_s = \{x \in B(\gamma_u) \mid \exists e \in E_h^s \text{ with } x \in e \text{ and } \forall e \in E_h^b \colon x \notin e\}$,

$V_b = \{x \in B(\gamma_u) \mid \exists e \in E_h^b \text{ with } x \in e\}$.

Our proof proceeds in three steps. We will first show that V_ℓ and V_s are bounded by constants. V_b can in principle become arbitrarily large (cf. edge $\{v_1, \ldots, v_n\}$ in Example 4.7). However, we will show that γ_u can be transformed into γ_u^* with $\mathit{weight}(\gamma_u^*) \leq \mathit{weight}(\gamma_u)+\epsilon$, s.t. all vertices of V_b are covered by edges of weight 1 in γ_u^*.

Claim A. $|V_\ell| < c_1$ with $c_1 = 2ik^2$.

Proof of Claim A. Let $e \in E_\ell$ and let $m = |e \cap B(\gamma_u)|$. We first show that $m < 2ik$ holds, i.e., the size of light-weight edges is bounded by a constant. (Recall from Example 4.7 that this is, in general, not the case for heavy edges.) Indeed, by $e \in E_\ell$, we know that e puts weight < 0.5 on each of its vertices. Hence, weight > 0.5 must be put on each vertex of $e \cap B(\gamma_u)$ by other edges $e' \in E_\ell$. In total, the other edges must put weight $> 0.5m$ on the vertices of e. Now we make use of the assumption that $\mathit{iwidth}(H) \leq i$ holds, i.e., the intersection of any two edges of H contains at most i vertices. Hence, whenever γ_u puts weight w on some edge $e' \neq e$, then e' puts at most weight iw in total on the vertices in e. By $\mathit{weight}(\gamma_u) \leq k$, the edges $e' \in E_\ell$ with $e' \neq e$ can put at most ik total weight on the vertices in e. We thus get the inequality $ik > 0.5m$. In other words, we have $m < 2ik$.

Now suppose that, for an arbitrary edge in E_ℓ, we have $\gamma_u(e) = w$. Then e can put at most weight mw in total on the vertices in V_ℓ. By $\mathit{weight}(\gamma_u) \leq k$, all edges in E_ℓ together can put at most weight mk on the vertices in V_ℓ. By our definition of V_ℓ, we have $V_\ell \subseteq B(\gamma_u)$, i.e., each vertex in V_ℓ receives at least weight 1. Hence, there can be at most mk vertices in V_ℓ. In total, we thus have $|V_\ell| < 2ik^2$. ◇

Claim B. $|V_s| < c_2$ with $c_2 = \frac{4k^3i}{\epsilon}$.

Proof of Claim B. First, we show that $|E_h| \leq 2k$, i.e., the number of heavy edges is bounded by a constant. (Recall from Example 4.7 that this is, in general, not the case for the light-weight edges.) By the definition of E_h, each edge in E_h has weight ≥ 0.5. By $\mathit{weight}(\gamma_u) \leq k$, the total weight of the edges in E_h is $\leq k$. Hence, there can be at most $2k$ edges in E_h.

By $E_h^s \subseteq E_h$ this implies that also $|E_h^s| \leq 2k$ holds. Since $|e \cap B(\gamma_u)| < d$ with $d = \frac{2k^2i}{\epsilon}$ holds for all edges in E_h^s, we thus have $|V_s| < \frac{4k^3i}{\epsilon}$. ◇

Claim C. For every $e \in E_h^b$, we have $\gamma_u(e) \geq 1 - \frac{\epsilon}{2k}$.

Proof of Claim C. Let $e \in E_h^b$ and let $e \cap B(\gamma_u) = m$ with $m \geq \frac{2k^2i}{\epsilon}$. Let $e' \in \mathit{supp}(\gamma_u)$ with $e' \neq e$. Moreover, since $\mathit{iwidth}(H) \leq i$, we have $|e' \cap e| \leq i$. Hence, if $\gamma_u(e') = w$, then e' can put at most total weight wi on the vertices in e. By $\mathit{weight}(\gamma_u) \leq k$, all edges $e' \in \mathit{supp}(\gamma_u)$ with $e' \neq e$ taken together can put at most total weight ki on the vertices in e. The average weight thus put on each vertex in $e \cap B(\gamma_u)$ is at most $\frac{ki}{m}$. Together with the condition $m \geq \frac{2k^2i}{\epsilon}$, we thus get the upper bound $\frac{ki\epsilon}{2k^2i} = \frac{\epsilon}{2k}$ on the average weight put on each vertex in $e \cap B(\gamma_u)$ by all of the edges $e' \neq e$. Hence, there is at least one vertex $x \in e \cap B(\gamma_u)$ for which the total weight of the edges $e' \neq e$ with $x \in e'$ is at most $\frac{\epsilon}{2k}$. Therefore, for x to be in $B(\gamma_u)$, $\gamma_u(e) \geq 1 - \frac{\epsilon}{2k}$ must hold. ◇

Conclusion of the proof of Lemma 4.14. We are now ready to construct the desired fractional cover γ_u^* of $B(\gamma_u)$ (and, hence, of B_u):

$$\gamma_u^*(e) = \begin{cases} 1 & \text{if } e \in E_h^b \\ \gamma_u(e) & \text{otherwise} \end{cases}$$

The fractional cover γ_u^* has the following properties:

- In γ_u^*, the weight of an edge is never decreased compared with γ_u. Hence, $B(\gamma_u) \subseteq B(\gamma_u^*)$ and, therefore, $B_u \subseteq B(\gamma_u^*)$ holds.
- By Claim C, for every edge $e \in E_h^b$, we have $\gamma_u(e) \geq 1 - \frac{\epsilon}{2k}$ and, thus, $\gamma_u^*(e) = 1 \leq \gamma_u(e) + \frac{\epsilon}{2k}$. Moreover, in the proof of Claim B, we have shown that $|E_h| \leq 2k$ (and, thus, also $|E_h^b| \leq 2k$) holds. Hence, we have $\mathit{weight}(\gamma_u^*) \leq \mathit{weight}(\gamma_u) + \epsilon$.
- By the definition of V_ℓ, V_s, and γ_u^*, we have $B(\gamma_u^*|_{R^*}) \subseteq V_\ell \cup V_s$ with $R^* = \{e \in \mathit{supp}(\gamma_u^*) \mid \gamma_u^*(e) < 1\}$, since all of the big, heavy edges under γ_u have weight 1 in γ_u^*. Moreover, by Claims A and B, we have $|V_\ell \cup V_s| \leq c_1 + c_2 = 2ik^2 + \frac{4k^3 i}{\epsilon}$. Hence, we also have $|B(\gamma_u^*|_{R^*})| \leq c_1 + c_2$.

By carrying out this transformation of γ_u into γ_u^* for every node u of $\mathcal{F}$, we thus get the desired FHD of H of width $\leq k + \epsilon$ with c-bounded fractional part for $c = 2ik^2 + \frac{4k^3 i}{\epsilon}$ □

The following lemma shows that, in case of the BIP, the weak special condition can be enforced without a further increase of the width.

Lemma 4.15. *Let $c \geq 0, i \geq 0$, and $k \geq 1$. There exists a polynomial-time computable function $f_{(c,i,k)}$ which takes as input a hypergraph H with $\mathit{iwidth}(H) \leq i$ and yields as output a set of subedges of $E(H)$ with the following property: if H has an FHD of width $\leq k$ with c-bounded fractional part then H' has an FHD of width $\leq k$ with c-bounded fractional part satisfying the weak special condition, where $H' = (V(H), E(H) \cup f_{(c,i,k)}(H))$. More specifically, $f_{(c,i,k)}(H)) = \{e' \mid e' \text{ is a subedge of some } e \in E(H) \text{ with } |e'| \leq ki + c\}$.*

Proof. Let H be a hypergraph with $\mathit{iwidth}(H) \leq i$ and let H' and $f_{(c,i,k)}(H))$ be as defined above. Moreover, let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be an FHD of H (and hence also of H') of width $\leq k$ with c-bounded fractional part. We assume T to be rooted, where the root can be arbitrarily chosen among the nodes of T. We have to show that $\mathcal{F}$ can be transformed into an FHD of H' of width $\leq k$ with c-bounded fractional part satisfying the weak special condition.

We proceed similarly as in the proof of Theorem 4.2 – with some simplifications due to the assumption of the BIP (rather than the less restrictive BMIP) and with some slight complications due to the fractional part. Suppose that $\mathcal{F}$ contains a violation of the weak special condition (a weak-SCV, for short), i.e., there exists a node u in T, an edge $e \in E(H)$ with $\gamma_u(e) = 1$ and a vertex $x \in e \cap V(T_u)$, s.t. $x \notin B_u$ holds. We write

(u, e, x) to denote such a weak-SCV. W.l.o.g., we can choose a weak-SCV in such a way that there exists no weak-SCV for any node u' below u. We show that this weak-SCV can be eliminated by appropriately modifying the FHD $\mathcal{F}$ of H'.

By the connectedness condition, e must be covered by some node $u^* \in T_u$, i.e., u^* is a descendant of u and $e \subseteq B_{u^*}$ holds. Let π denote the path in T from u to u^*. We distinguish two cases:

Case 1. Suppose that for every node u' along the path π with $u' \neq u$, we have $x \in B_{u'}$. Then we simply add x to B_u. Clearly, this modification does not violate any of the conditions of FHDs, i.e., the connectedness condition and the condition $B_u \subseteq B(\gamma_u)$ are still fulfilled. Moreover, the weak-SCV (u, e, x) has been eliminated and no new weak-SCV is introduced. Finally, note that γ_u is left unchanged by this transformation. Hence, the resulting FHD still has c-bounded fractional part.

Case 2. Suppose that there exists a node u' along the path π with $u' \neq u$ and $x \notin B_{u'}$. Of course, also $u' \neq u^*$ holds, since $x \in e$ and e is covered by u^*. We may also conclude that $\gamma_{u'}(e) < 1$. Indeed, suppose to the contrary that $\gamma_{u'}(e) = 1$. Then $\mathcal{F}$ would contain the weak-SCV (u', e, x) where u' is below u, which contradicts our choice of (u, e, x).

By the connectedness condition, $e \cap B_u \subseteq B_{u'}$ holds and, hence, also $e \cap B_u \subseteq e \cap B_{u'}$. Moreover, $B_{u'} \subseteq B(\gamma_{u'})$ holds by the definition of FHDs and $B(\gamma_{u'})$ is of the form $B(\gamma_{u'}) = B_1 \cup B_2$ with $B_1 = B(\gamma_{u'}|_R)$ and $B_2 = B(\gamma_{u'}|_S)$ with $R = \{e \in \mathit{supp}(\gamma_{u'}) \mid \gamma_{u'}(e) < 1\}$ and $S = \{e \in \mathit{supp}(\gamma_{u'}) \mid \gamma_{u'}(e) = 1\}$. Now let $S = \{e_1, \ldots, e_\ell\}$ denote the set of edges with weight 1 in $\gamma_{u'}$. Clearly, $\ell \leq k$, since the width of $\mathcal{F}$ is $\leq k$. In total, we have:

$$(e \cap B_u) \subseteq e \cap B_{u'} = e \cap (e_1 \cup \cdots \cup e_\ell \cup B_1) = (e \cap e_1) \cup \cdots \cup (e \cap e_\ell) \cup (e \cap B_1).$$

The first ℓ intersections each have cardinality $\leq i$ and the last intersection has cardinality $\leq c$ by our assumption of c-fractional boundedness. Together with $\ell \leq k$, we thus have $|e \cap B_u| \leq ki + c$.

Now let $e' = e \cap B_u$. We have just shown that e' is a subset of e with $|e'| \leq ki + c$. Hence, e' is an edge in H'. We can thus modify $\mathcal{F}$ by modifying γ_u to γ'_u as follows: we set $\gamma'_u(e) = 0$, $\gamma'_u(e') = 1$, and let γ'_u be identical to γ_u everywhere else. Clearly, we still have $B_u \subseteq B(\gamma'_u)$ and also $\mathit{weight}(\gamma'_u) \leq k$ still holds. Moreover, the weak-SCV (u, e, x) has been eliminated and no new weak-SCV (u, e', z) can arise since $e' = e \cap B_u$ implies $e' \subseteq B_u$. Finally, note that $B(\gamma_u|_R) = B(\gamma'_u|_R)$ and $R = \{e \in \mathit{supp}(\gamma_u) \mid \gamma_u(e) < 1\} = \{e \in \mathit{supp}(\gamma'_u) \mid \gamma'_u(e) < 1\}$. Hence, the resulting FHD still has c-bounded fractional part.

To conclude, every modification of $\mathcal{F}$ by either Case 1 or Case 2 strictly decreases the number of weak-SCVs in our FHD. Moreover, the FHD resulting from such a modification still has c-bounded fractional part. By exhaustively eliminating the weak-SCVs as described above, we thus end up with an FHD of H' of width $\leq k$ with c-bounded fractional part satisfying the weak special condition. □

Normal Form of FHDs. In order to adapt the HD algorithm from [51] to turn it into an FHD algorithm that searches for FHDs with c-bounded fractional part for some constant c and satisfying the weak special condition, we will again make use of the fractional normal form introduced in Definition 2.6. Below we argue, similar as in the proof of Lemma 4.13, that the transformations given in the proof of Theorem 2.1 applied to an FHD with c-bounded fractional part and weak special condition will result in an FHD in FNF that also satisfies these two properties.

Lemma 4.16. *Let $c \geq 0$. For each FHD $\mathcal{F}$ of a hypergraph H with c-bounded fractional part satisfying the weak special condition and with width$(\mathcal{F}) \leq k$ there exists an FHD $\mathcal{F}^+$ of H in FNF with c-bounded fractional part satisfying the weak special condition and with width$(\mathcal{F}^+) \leq k$.*

Proof Sketch. The crucial part of the transformation into normal form is to ensure Conditions 1 and 2. Here, the proof of Theorem 2.1 can be taken over literally because it only makes use of the tree structure of the decomposition, the bags, and the connectedness condition. Ensuring also Condition 3 of our FNF is easy, because we may always extend B_s by vertices from $B(\gamma_s) \cap B_r$ without violating the connectedness condition. Moreover, the transformation of HDs in [51] preserves the special condition. Analogously, when applying this transformation to FHDs (as detailed in Theorem 2.1), the weak special condition is preserved. Finally, if $\mathcal{F}$ has c-bounded fractional part then so has $\mathcal{F}^+$. This is due to the fact that, by this transformation, no set $B(\gamma_u|_R)$ with $R = \{e \in \mathit{supp}(\gamma_u) \mid \gamma_u(e) < 1\}$ is ever increased. □

Suppose that an FHD $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ is in FNF. Then, for every node $s \in T$, we define $\mathit{treecomp}(s)$ as follows:

- If s is the root of T, then we set $\mathit{treecomp}(s) = V(H)$.
- Otherwise, let r be the parent of s in T. Then we set $\mathit{treecomp}(s) = C_r$, where C_r is the unique $[B_r]$-component with $V(T_s) = C_r \cup (B_r \cap B_s)$ according to Condition 1 of the FNF.

We now carry Lemmas 5.5 – 5.7 from [51] over to fractional hypertree decompositions in fractional normal form. The proofs from [51] can be easily adapted to our setting. We therefore state the lemmas without proof.

Lemma 4.17 (Lemma 5.5 from [51])**.** *Let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be an arbitrary FHD of a hypergraph H in fractional normal form, let $u \in T$, and let $W = \mathit{treecomp}(u) \setminus B_u$. Then, for any $[B_u]$-component C such that $(C \cap W) \neq \emptyset$, we have $C \subseteq W$. Therefore, $\mathcal{C} = \{C' \subseteq V(H) \mid C' \text{ is a } [B_u]\text{-component and } C' \subseteq \mathit{treecomp}(u)\}$ is a partition of W.*

Lemma 4.18 (Lemma 5.6 from [51])**.** *Let $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be an arbitrary FHD of a hypergraph H in fractional normal form and let $r \in T$. Then, $C = \mathit{treecomp}(s)$ for some child s of r if and only if C is a $[B_r]$-component of H and $C \subseteq \mathit{treecomp}(r)$.*

Lemma 4.19 (Lemma 5.7 from [51]). *For every FHD $\mathcal{F} = \langle T, (B_u)_{u\in T}, (\gamma_u)_{u\in T}\rangle$ of a hypergraph H in fractional normal form, $|nodes(T)| \leq |V(H)|$.*

The next lemma is crucial for designing an algorithm that computes a concrete FHD. The lemma is based on Lemma 5.8 from [51]. However, the proof in the FHD-setting requires a slightly more substantial modification of the proof in the HD-setting. We therefore state the lemma together with a full proof below.

Lemma 4.20 (Lemma 5.8 from [51]). *Let $c \geq 0$ and let $\mathcal{F} = \langle T, (B_u)_{u\in T}, (\gamma_u)_{u\in T}\rangle$ be an FHD in FNF of a hypergraph H such that $\mathcal{F}$ has c-bounded fractional part and satisfies the* weak special condition*. Further, let s be a node in T and let r be the parent of s in T. Let $R_1 = \{e \in E(H) \mid \gamma_s(e) = 1\}$ and $R_2 = \{e \in E(H) \mid \gamma_s(e) < 1\}$, and let $B_s = B_1 \cup B_2$ with $B_1 = B_s \cap B(\gamma_s|_{R_1})$ and $B_2 = B_s \cap B(\gamma_s|_{R_2})$. Finally, let C be a set of vertices such that $C \subseteq treecomp(s)$. Then the following equivalence holds:*

$$C \text{ is a } [B_s]\text{-component if and only if } C \text{ is a } [B(\gamma_s|_{R_1}) \cup B_2]\text{-component.}$$

Remark. The crux of the proof of Lemma 5.8 from [51] and likewise of Lemma 4.20 stated here is the following: by the definition of FHDs, we have $B_s \subseteq B(\gamma_s|_{R_1}) \cup B_2$. Hence, every $[B(\gamma_s|_{R_1})\cup B_2]$-path is also a $[B_s]$-path, but the converse is, at first glance, not clear. However, by the weak special condition, $(B(\gamma_s|_{R_1}) \cup B_2) \setminus B_s$ only contains elements from $B_r \cap B_s$. Moreover, we are assuming that C is a subset of $treecomp(s)$, i.e., it is in the complement of B_r. Hence, $[B_s]$-paths and $[B(\gamma_s|_{R_1}) \cup B_2]$-paths actually coincide. From this it is then straightforward to conclude that, inside $treecomp(s)$, $[B_s]$-components and $[B(\gamma_s|_{R_1}) \cup B_2]$-components coincide.

Proof. Let $W = B(\gamma_s|_{R_1}) \cup B_2$. We first prove the following Property (4.1), which is the analogue of Property (1) in the proof of Lemma 5.8 from [51]:

$$W \cap treecomp(s) \subseteq B_s. \tag{4.1}$$

Proof of Property (4.1). By the definition of FHDs, we have $B_s \subseteq B(\gamma_s|_{R_1}) \cup B_2 = W$. By the weak special condition, we have $B(\gamma_s|_{R_1}) \cap V(T_s) \subseteq B_s$. By the definition of $treecomp(s)$, we have $V(T_s) = treecomp(s) \cup (B_s \cap B_r)$, i.e., also $treecomp(s) \subseteq V(T_s)$ clearly holds. In total, we thus have:

$$W \cap treecomp(s) = (B(\gamma_s|_{R_1}) \cup B_2) \cap treecomp(s) \subseteq$$

$$(B(\gamma_s|_{R_1}) \cap treecomp(s)) \cup B_2 \subseteq (B(\gamma_s|_{R_1}) \cap V(T_s)) \cup B_2 \subseteq B_s. \qquad \diamond$$

It remains to show for $C \subseteq treecomp(s)$, that C is a $[B_s]$-component if and only if C is a $[W]$-component. This proof follows the line of argumentation in the proof of Lemma 5.8 from [51] – replacing Property (1) there with our Property (4.1) proved here. For the sake of completeness, we present a detailed proof of the desired equivalence below.

Proof of the "only if"-direction. Suppose that C is a $[B_s]$-component with $C \subseteq \mathit{treecomp}(s)$. Then, in particular, $C \cap B_s = \emptyset$. Hence, by Property (1), we have $C \cap W = \emptyset$. This can be seen as follows: $C \cap W \subseteq \mathit{treecomp}(s) \cap W \subseteq B_s$ (the last inclusion uses Property (1)). Hence, also $C \cap W \subseteq C \cap B_s$ holds. Together with $C \cap B_s = \emptyset$, we thus have $C \cap W = \emptyset$.

We have to show that C is a $[W]$-component, i.e., C is $[W]$-connected and C is maximal $[W]$-connected. For the $[W]$-connectedness, consider an arbitrary pair of vertices $\{x, y\} \subseteq C$, i.e., there exists a $[B_s]$-path π between x and y. Note that this $[B_s]$-path π only goes through vertices in C. Hence, by $C \cap W = \emptyset$, π is also a $[W]$-path. Hence, C is indeed $[W]$-connected. For the maximality, we simply make us of the relationship $B_s \subseteq W$. This means that since C is maximal $[B_s]$-connected, it is also maximal $[W]$-connected.

Proof of the "if"-direction. Suppose that C is a $[W]$-component with $C \subseteq \mathit{treecomp}(s)$. By $B_s \subseteq W$, we conclude that the $[W]$-connectedness of C implies the $[B_s]$-connectedness. It remains to show that C is maximal $[B_s]$-connected. Clearly, there exists a $[B_s]$-component C' with $C \subseteq C'$. By Lemma 4.17, we have $C' \subseteq \mathit{treecomp}(s) \setminus B_s$. In particular, $C' \subseteq \mathit{treecomp}(s)$. Hence, by the "only if" part of this lemma, C' is a $[W]$-component and, therefore, C cannot be a proper subset of C'. Hence, $C = C'$. Thus, C is indeed a $[B_s]$-component.

We have thus shown for $C \subseteq \mathit{treecomp}(s)$, that C is a $[B_s]$-component if and only if C is a $[W]$-component. This concludes the proof of the lemma. □

A PTime algorithm for FHDs with c-bounded fractional part. We now adapt the HD algorithm from [51] to turn it into an FHD algorithm that searches for FHDs with c-bounded fractional part for some constant c and satisfying the weak special condition. By Lemmas 4.14 and 4.15, we know that for any constants $k \geq 1$, $i \geq 0$, and $\epsilon > 0$, there exists a constant c (which only depends on k, i, ϵ) with the following property: for every hypergraph H with $\mathit{iwidth}(H) \leq i$, if H has an FHD of width $\leq k$, then H also has an FHD of width $\leq k + \epsilon$ with c-bounded fractional part and satisfying the weak special condition. Moreover, by Lemma 4.16, the transformation into FNF can be done in such a way that it does not increase the width and preserves the c-boundedness of the fractional part and the weak special condition. Hence, in our Algorithm 4.3 "(k, ϵ, c)-`frac-decomp`", we restrict our search to FHDs of width $\leq k + \epsilon$ in FNF with c-bounded fractional part and satisfying the weak special condition. Moreover, throughout the remainder of Section 4.4, we assume that for every edge e in a given hypergraph H, all subedges e' of e of size $|e'| \leq ki + c$ (cf. Lemma 4.15) have already been added to H.

Let τ be a computation tree of the alternating algorithm (k, ϵ, c)-`frac-decomp`. We can associate with every τ an FHD $\delta(\tau) = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$, called witness tree, defined as follows: For each existential configuration in τ corresponding to the "guess" of some sets $S \subseteq E(H)$ and $W_s \subseteq V(H)$ in Step 1 during the execution of a procedure call `f-decomp` (C_r, W_r, R), the tree T contains a node s. In particular, at the initial call `f-decomp` $(V(H), \emptyset, \emptyset)$, the guesses S nad W_s give rise to the root node s_0 of T.

Algorithm 4.3: (k, ϵ, c)-`frac-decomp`

input : Hypergraph H.
output : "Accept", if H has an FHD of width $\leq k + \epsilon$ with c-bounded fractional part and weak special condition
"Reject", otherwise.

```
1  Function f-decomp (C_r, W_r: Vertex-Set, R: Edge-Set)
2      begin                                                    /* (1) Guess */
3          Guess a set S ⊆ E(H) with |S| = ℓ, s.t. ℓ ≤ k + ε ;        /* (1.a) */
4          Guess a set W_s ⊆ (V(R) ∪ W_r ∪ C_r) with |W_s| ≤ c ;      /* (1.b) */
5      end
6      begin                                                    /* (2) Check */
7          Check if ∃γ with W_s ⊆ B(γ) and weight(γ) ≤ k + ε − ℓ ;   /* (2.a) */
8          Check if ∀e ∈ edges(C_r): e ∩ (V(R) ∪ W_r) ⊆ (V(S) ∪ W_s) ; /* (2.b) */
9          Check if (V(S) ∪ W_s) ∩ C_r ≠ ∅ ;                      /* (2.c) */
10     end
11     if one of these checks fails then Halt and Reject;          /* (3) */
12     else
13         Let C := {C ⊆ V(H) | C is a [V(S) ∪ W_s]-component and C ⊆ C_r};
14     end
15     foreach C ∈ C do                                            /* (4) */
16         if f-decomp (C, W_s, S) returns Reject then
17             Halt and Reject
18         end
19     end
20     return Accept;
21 begin                                                          /* Main */
22     return f-decomp (V(H), ∅, ∅)
23 end
```

Moreover, there is an edge between nodes r and s of T, if $s \neq s_0$ and r is the node in T corresponding to the guess of sets $R \subseteq E(H)$ and $W_r \subseteq V(H)$. We will denote C_r by $comp(s)$, and r by $parent(s)$. Moreover, for the root s_0 of T, we define $comp(s_0) = V(H)$.

Each node $s \in T$ is labelled by B_s and γ_s as follows. First, we define γ_s via the mapping γ, which exists according to the check in Step 2.a:

$$\gamma_s(e) = \begin{cases} \gamma(e) & \text{for } e \in supp(\gamma) \setminus S \\ 1 & \text{for } e \in S \\ 0 & \text{otherwise} \end{cases}$$

Note that for the sets S of edges and W_s of vertices guessed in Step 1, we have $B(\gamma_s) = (V(S) \cup W_s)$. As far as the definition of the bags is concerned, we set $B_{s_0} = B(\gamma_{s_0})$ for

the root node s_0. For any other node s with $r = \mathit{parent}(s)$ and $C = \mathit{comp}(s)$, we set $B_s = B(\gamma_s) \cap (B_r \cup C)$.

The correctness proof of the (k, ϵ, c)-`frac-decomp` algorithm is along the same lines as the correctness proof of the alternating algorithm for the Check(HD, k) problem in [51]. We therefore state the analogues of the lemmas and theorems of [51] without proofs, since these can be easily "translated" from the HD setting in [51] to our FHD setting.

Lemma 4.21 (based on Lemma 5.9 from [51]). *Let H be a hypergraph and let $k \geq 1$, $\epsilon > 0$, and $c \geq 0$. If H has an FHD of width $\leq k + \epsilon$ in FNF with c-bounded fractional part and satisfying the weak special condition, then the algorithm (k, ϵ, c)-`frac-decomp` accepts input H. Moreover, every such FHD is equal to some witness tree $\delta(\tau)$ of (k, ϵ, c)-`frac-decomp` when run on input H.*

The next three lemmas will help to show the converse: whenever (k, ϵ, c)-`frac-decomp` has an accepting computation, then the corresponding witness tree is an FHD of H of width $\leq k + \epsilon$ in FNF with c-bounded fractional part and satisfying the weak special condition.

Lemma 4.22 (based on Lemma 5.10 from [51]). *Let H be a hypergraph and let $k \geq 1$, $\epsilon > 0$, and $c \geq 0$. Assume that (k, ϵ, c)-`frac-decomp` accepts input H with an accepting computation tree τ and corresponding witness tree $\delta(\tau) = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$. Then, for each node s in T:*

(a) if $s \neq \mathit{root}(T)$, then $\mathit{comp}(s)$ is a $[B_r]$-component with $r = \mathit{parent}(s)$;

(b) for any $C \subseteq \mathit{comp}(s)$, C is a $[B_s]$-component if and only if C is a $[V(S) \cup W_s]$-component.

Lemma 4.23 (based on Lemma 5.11 from [51]). *Let H be a hypergraph and let $k \geq 1$, $\epsilon > 0$, and $c \geq 0$. Assume that (k, ϵ, c)-`frac-decomp` accepts input H with an accepting computation tree τ. Let $\delta(\tau) = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ be the corresponding witness tree and $s \in T$. Then, for each node $u \in T_s$:*

$$B_u \subseteq \mathit{comp}(s) \cup B_s$$
$$\mathit{comp}(u) \subseteq \mathit{comp}(s).$$

Lemma 4.24 (based on Lemma 5.12 from [51]). *Let H be a hypergraph and let $k \geq 1$, $\epsilon > 0$, and $c \geq 0$. Assume that (k, ϵ, c)-`frac-decomp` accepts input H with an accepting computation tree τ and corresponding witness tree $\delta(\tau) = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$. Let $s \in T$ and $C_r = \mathit{comp}(s)$. Then, for every edge $e \in \mathit{edges}(C_r)$ and every edge $e' \in E(H) \setminus \mathit{edges}(C_r)$, we have $e \cap e' \subseteq B_s$.*

We are now ready to show that, whenever (k, ϵ, c)-`frac-decomp` has an accepting computation on an input hypergraph H, then the corresponding witness tree is an

FHD of H of width $\leq k + \epsilon$ in FNF with c-bounded fractional part and satisfying the weak special condition. As before, the following lemma can be shown similarly to the corresponding result in [51].

Lemma 4.25 (based on Lemma 5.13 from [51])**.** *Let H be a hypergraph and let $k \geq 1$, $\epsilon > 0$, and $c \geq 0$. If (k, ϵ, c)-`frac-decomp` accepts input H, then H has an FHD of width $\leq k + \epsilon$ in FNF with c-bounded fractional part and satisfying the weak special condition. Moreover, in case of acceptance, every witness tree $\delta(\tau)$ for H is an FHD $\mathcal{F}$ of H in FNF with c-bounded fractional part satisfying the weak special condition.*

The following result follows immediately from the above Lemmas 4.21 and 4.25.

Theorem 4.7. *Let H be a hypergraph and let $c \geq 0$ and $\epsilon > 0$. Then, (k, ϵ, c)-`frac-decomp` accepts input (H, c, ϵ) if and only if H has an FHD of width $\leq k+\epsilon$ with c-bounded fractional part and satisfying the weak special condition. Moreover, in case of acceptance, every witness tree $\delta(\tau)$ for H is an FHD $\mathcal{F}$ of H in FNF with c-bounded fractional part satisfying the weak special condition.*

It remains to establish the Ptime membership of our algorithm. Again, we can easily carry over the corresponding tractability result from Lemma 5.15 in [51]. The crux of the proof in [51] is that all data structures involved in the alternating algorithm fit into logspace. In total, our alternating algorithm (k, ϵ, c)-`frac-decomp` has to maintain the following 6 data structures: the input parameters C_r, W_r, and R of procedure `f-decomp` and the local variables S, W_s, and the component C of the next recursive procedure call. In the alternating algorithm in [51], only 4 data structures are needed, which correspond to C_r, R, S, and C in our setting. The data structures W_r and W_s are only used in our algorithm. However, these are just sets of constantly many vertices. Hence, they can of course also be stored in logspace. The rest of the proof arguments can then be easily carried over from [51]. When it comes to the complexity of the checks in step 2, we additionally have to solve a linear program in our algorithm, which can also be done in Ptime (or on an ATM using logspace). We thus get:

Lemma 4.26 (based on Lemma 5.15 from [51])**.** *The alternating algorithm (k, ϵ, c)-`frac-decomp` can be implemented on a logspace ATM.*

Theorem 4.6, now follows immediately by putting together Lemmas 4.14, 4.15, and 4.16, Theorem 4.7, and Lemma 4.26.

A polynomial time approximation scheme for finding optimal FHDs. Recall the definition of the K-Bounded-FHW-Optimization problem from Section 1.2, i.e.: given a hypergraph H, we are interested in $fhw(H)$, but only if $fhw(H) \leq K$. We will now show that, with the alternating algorithm (k, ϵ, c)-`frac-decomp` at our disposal, we are able to give a polynomial time absolute approximation scheme (PTAAS) for the bounded optimization problem. More precisely, we aim at an approximation algorithm with the following properties:

Definition 4.17 (PTAS [12, 96])**.** Let Π be an (intractable) minimization problem with positive objective function f_Π. An algorithm `Alg` is called an *approximation scheme* for Π if on input (I, ϵ), where I is an instance of Π and $\epsilon > 0$ is an error parameter, it outputs a solution s such that:

$$f_\Pi(I, s) \leq (1 + \epsilon) \cdot f_\Pi(I, s^*)$$

where s^* is an optimal solution of I, i.e. for all other solutions s' of I it holds that $f_\Pi(I, s^*) \leq f_\Pi(I, s')$.

`Alg` is called a *polynomial time approximation scheme* (PTAS), if for every fixed $\epsilon > 0$, its running time is bounded by a polynomial in the size of instance I. ◁

In our case we can even achieve something slightly better. As seen in Definition 4.17, the error of the solution given by an approximation scheme is *relative* to the optimal value of the optimization problem. Clearly, it would be preferable, if the gap between the solution returned by the approximation scheme and an optimal solution has an *absolute* bound (i.e., not depending on the optimal value). This leads to the following definition of *absolute* approximation schemes:

Definition 4.18 (PTAAS)**.** Let Π be an (intractable) minimization problem with positive objective function f_Π. We say that algorithm `Alg` is an *absolute approximation scheme* for Π if on input (I, ϵ), where I is an instance of Π and $\epsilon > 0$ is an error parameter, it outputs a solution s such that:

$$f_\Pi(I, s) \leq f_\Pi(I, s^*) + \epsilon$$

where s^* is the optimal solution of I, i.e. for all other solutions s' of I it holds that $f_\Pi(I, s^*) \leq f_\Pi(I, s')$.

`Alg` is called a *polynomial time absolute approximation scheme* (PTAAS), if for each fixed $\epsilon > 0$, its running time is bounded by a polynomial in the size of instance I. ◁

Note that every PTAAS is also a PTAS, since for any $\epsilon > 0$ it holds that $f_\Pi(I, s^*) + \epsilon \leq (1 + \epsilon) \cdot f_\Pi(I, s^*)$, provided that $f_\Pi(I, s^*) \geq 1$. Actually, we can assume w.l.o.g. that this is indeed the case [12]. We now show that, in case of the BIP, the K-Bounded-FHW-Optimization problem indeed allows for a PTAAS (and, hence, for a PTAS):

Theorem 4.8. *For every hypergraph class $\mathscr{C}$ that enjoys the BIP, there exists a PTAAS for the K-*Bounded-FHW-Optimization *problem.*

Proof. By Theorem 4.6, there exists a function `find-fhd` (H, k, ϵ, i) with the following properties:

- `find-fhd` takes as input a hypergraph H with $iwidth(H) \leq i$ and numbers $k \geq 1$, $\epsilon \geq 0$;

Algorithm 4.4: `FHW-Approximation`

input : hypergraph H with $iwidth(H) \leq i$, numbers $K \geq 1$, $\epsilon \geq 0$
output : approximation of $fhw(H)$, i.e., FHD $\mathcal{F}$ with $width(\mathcal{F}) \leq fhw(H) + \epsilon$ if $fhw(H) \leq K$

```
   /* Check upper bound                                          */
 1 if not (F = find-fhd (H, K, ε, i)) then
 2 |  return fails ;                                  /* fhw(H) > k */
 3 end
   /* Initialization                                             */
 4 L = 1;
 5 U = K + ε;
 6 ε' = ε/3;
   /* Main computation                                           */
 7 repeat
 8 |  if F' = find-fhd (H, L + (U − L)/2, ε', i) then
 9 |  |  U = L + (U − L)/2 + ε';
10 |  |  F = F';
11 |  end
12 |  else
13 |  |  L = L + (U − L)/2;
14 |  end
15 until U − L < ε;
16 return F;
```

- `find-fhd` returns an FHD $\mathcal{F}$ of H of width $\leq k + \epsilon$ if such exists and `fails` otherwise (i.e., $fhw(H) > k$ holds).

- `find-fhd` runs in time polynomial in the size of H, where k, ϵ, and i are considered as fixed.

Then we can construct Algorithm 4.4 "`FHW-Approximation`", which uses `find-fhd` as subprocedure. We claim that `FHW-Approximation` is indeed a PTAAS for the K-Bounded-FHW-Optimization problem. First we argue that the algorithm is correct; we will then also show its polynomial-time upper bound.

As for the correctness, note that the algorithm first checks if K is indeed an upper bound on $fhw(H)$. This is done via a call of function `find-fhd`. If the function call `fails`, then we know that $fhw(H) > K$ holds. Otherwise, we get an FHD $\mathcal{F}$ of width $\leq K + \epsilon$. In the latter case, we conclude that $fhw(H)$ is in the interval $[L, U]$ with $L = 1$ and $U = K + \epsilon$.

The loop invariant for the repeat loop is, that $\mathit{fhw}(H)$ is in the interval $[L, U]$ and the width of the FHD $\mathcal{F}$ is $\leq U$. To see that this invariant is preserved by every iteration of the loop, consider the function call `find-fhd` $(H, L + (U - L)/2, \epsilon', i)$: if this call succeeds then the function returns an FHD $\mathcal{F}'$ of width $\leq L + (U - L)/2 + \epsilon'$. Hence, we may indeed set $U = L + (U - L)/2 + \epsilon'$ and $\mathcal{F} = \mathcal{F}'$ without violating the loop invariant. On the other hand, suppose that the call of `find-fhd` fails. This means that $\mathit{fhw}(H) > L + (U - L)/2$ holds. Hence, we may indeed update L to $L + (U - L)/2$ and the loop invariant still holds. The repeat loop terminates when $U - L < \epsilon$ holds. Hence, together with the loop invariant, we conclude that, on termination, $\mathcal{F}$ is an FHD of H with $\mathit{width}(\mathcal{F}) - \mathit{fhw}(H) < \epsilon$.

It remains to show that algorithm `FHW-Approximation` runs in polynomial time w.r.t. the size of H. By Theorem 4.6, the function `find-fhd` works in polynomial time w.r.t. H. We only have to show that the number of iterations of the repeat-loop is bounded by a polynomial in H. Actually, we even show that it is bounded by a constant (depending on K and ϵ, but not on H): let $K' := K + \epsilon - 1$. Then the size of the interval $[L, U]$ initially is K'. In the first iteration of the repeat-loop, we either set $U = L + (U - L)/2 + \epsilon'$ or $L = L + (U - L)/2$ holds. In either case, at the end of this iteration, we have $U - L \leq K'/2 + \epsilon'$. By an easy induction argument, it can be verified that after m iterations (with $m \geq 1$), we have

$$U - L \leq \frac{K'}{2^m} + \epsilon' + \frac{\epsilon'}{2} + \frac{\epsilon'}{2^2} + \cdots + \frac{\epsilon'}{2^{m-1}}$$

Now set $m = \lceil \log(K'/\epsilon') \rceil$. Then we get

$$\frac{K'}{2^m} \leq \frac{K'}{2^{\log(K'/\epsilon')}} = \frac{K'}{(K'/\epsilon')} = \epsilon'.$$

Moreover, $\epsilon' + \frac{\epsilon'}{2} + \frac{\epsilon'}{2^2} + \cdots + \frac{\epsilon'}{2^{m-1}} < 2\epsilon'$ for every $m \geq 1$. In total, we thus have that, after $m = \lceil \log(K'/\epsilon') \rceil$ iterations of the repeat loop, $U - L < 3\epsilon' = \epsilon$ holds, i.e., the loop terminates. □

4.4.2 Approximation of FHDs in case of the BMIP

We now present a polynomial-time approximation of the *fhw* for classes of hypergraphs enjoying the BMIP. Actually, the approximation even works for slightly more general classes of hypergraphs, such that hypergraph classes with BMIP constitute a familiar subcase. For this, we will combine some classical results on the Vapnik-Chervonenkis (VC) dimension with some novel observations. This will yield an approximation of the *fhw* up to a logarithmic factor for hypergraphs enjoying the BMIP. We first recall the definition of the VC-dimension of hypergraphs.

Definition 4.19 ([85, 95])**.** Let $H = (V(H), E(H))$ be a hypergraph and $X \subseteq V(H)$ a set of vertices. Denote by $E(H)|_X$ the set $E(H)|_X = \{X \cap e \mid e \in E(H)\}$. The vertex set X is called *shattered* if $E(H)|_X = 2^X$. The *Vapnik-Chervonenkis dimension (VC dimension)* $\mathrm{vc}(H)$ of H is the maximum cardinality of a shattered subset of $V(H)$. ◁

We now provide a link between the VC-dimension and our approximation of the *fhw*.

Definition 4.20. Let $H = (V(H), E(H))$ be a hypergraph. A *transversal* (also known as *hitting set*) of H is a subset $S \subseteq V(H)$ that has a non-empty intersection with every edge of H. The *transversality* $\tau(H)$ of H is the minimum cardinality of all transversals of H.

Clearly, $\tau(H)$ corresponds to the minimum of the following integer linear program: find a mapping $w : V \to \{0, 1\}$ which minimizes $\Sigma_{v \in V(H)} w(v)$ under the condition that $\Sigma_{v \in e} w(v) \geq 1$ holds for each hyperedge $e \in E$.

The *fractional transversality* τ^* of H is defined as the minimum of the above linear program when dropping the integrality condition, thus allowing mappings $w : V \to \mathbb{R}_{\geq 0}$. Finally, the *transversal integrality gap tigap*(H) of H is the ratio $\tau(H)/\tau^*(H)$. ◁

Recall that computing the mapping λ_u for some node u in a GHD can be seen as searching for a minimal edge cover ρ of the vertex set B_u, whereas computing γ_u in an FHD corresponds to the search for a minimal fractional edge cover ρ^* [58]. Again, these problems can be cast as linear programs where the first problem has the integrality condition and the second one has not. Further, we can define the *cover integrality gap cigap*(H) of H as the ratio $\rho(H)/\rho^*(H)$. With this, we state the following approximation result for *fhw*.

Theorem 4.9. *Let $\mathscr{C}$ be a class of hypergraphs with VC-dimension bounded by some constant d and let $k \geq 1$. Then there exists a polynomial-time algorithm that, given a hypergraph $H \in \mathscr{C}$ with $\mathit{fhw}(H) \leq k$, finds an FHD of H of width $\mathcal{O}(k \cdot \log k)$.*

Proof. The proof proceeds in several steps.

Reduced hypergraphs. As in Section 4.3, Proposition 4.1, we consider, w.l.o.g., only hypergraphs H that satisfy the following 4 conditions: (1) H has no isolated vertices and (2) no empty edges. Moreover, (3) no two distinct vertices in H have the same edge-type (i.e., the two vertices occur in precisely the same edges) and (4) no two distinct edges in H have the same vertex-type (i.e., we exclude duplicate edges). Hypergraphs satisfying these conditions will be called "reduced".

Dual hypergraphs. Given a hypergraph $H = \{V, E)$, the dual hypergraph $H^d = (W, F)$ is defined as $W = E$ and $F = \{\{e \in E \mid v \in e\} \mid v \in V\}$. We are assuming that H is *reduced*. This ensures that $(H^d)^d = H$ holds. Moreover, it is well-known and easy to verify that the following relationships between H and H^d hold for any reduced hypergraph H, (see, e.g., [32]):

(1) The edge coverings of H and the transversals of H^d coincide.

(2) The fractional edge coverings of H and the fractional transversals of H^d coincide.

(3) $\rho(H) = \tau(H^d)$, $\rho^*(H) = \tau^*(H^d)$, and $\mathit{cigap}(H) = \mathit{tigap}(H^d)$.

VC-dimension. By a classical result (see [31, 22]), for every hypergraph $H = (V(H), E(H))$, we have

$$tigap(H) = \tau(H)/\tau^*(H) \leq 2\mathrm{vc}(H)\log(11\tau^*(H))/\tau^*(H).$$

Moreover, in [9], it is shown that $\mathrm{vc}(H^d) < 2^{\mathrm{vc}(H)+1}$ always holds. In total, we thus get

$$\begin{aligned} cigap(H) = tigap(H^d) &\leq 2\mathrm{vc}(H^d)\log(11\tau^*(H^d))/\tau^*(H^d) \\ &< 2^{\mathrm{vc}(H)+2}\log(11\rho^*(H))/\rho^*(H). \end{aligned}$$

Approximation of fhw by ghw. Suppose that H has an FHD $\left\langle T, (B_u)_{u\in V(T)}, (\lambda)_{u\in V(T)}\right\rangle$ of width k. Then there exists a GHD of H of width $\mathcal{O}(k \cdot \log k)$. Indeed, we can find such a GHD by leaving the tree structure T and the bags B_u for every node u in T unchanged and replacing each fractional edge cover γ_u of B_u by an optimal integral edge cover λ_u of B_u. By the above inequality, we thus increase the weight at each node u only by a factor $\mathcal{O}(\log k)$. Moreover, we know from [5] that computing an HD instead of a GHD increases the width only by the constant factor 3. □

One drawback of the VC-dimension is that deciding if a hypergraph has VC-dimension $\leq v$ is intractable [89]. However, Lemma 4.27 establishes a relationship between BMIP and VC-dimension. Together with Theorem 4.9, Corollary 4.3 is then immediate.

Lemma 4.27. *If a class $\mathscr{C}$ of hypergraphs has the BMIP then it has bounded VC-dimension. However, there exist classes $\mathscr{C}$ of hypergraphs with bounded VC-dimension that do not have the BMIP.*

Proof. [BMIP ⇒ bounded VC-dimension.] Let $c \geq 1, i \geq 0$ and let H be a hypergraph with c-*miwidth*$(H) \leq i$. We claim that then $\mathrm{vc}(H) \leq c + i$ holds.

Assume to the contrary that there exists a set $X \subseteq V$, such that X is shattered and $|X| > c + i$. We pick c arbitrary, pairwise distinct vertices $v_1, \ldots, v_c$ from X and define $X_j = X \setminus \{v_j\}$ for each j. Then $X = (X_1 \cap \cdots \cap X_c) \cup \{v_1, \ldots, v_c\}$ holds and also $|X| \leq |X^*| + c$ with $X^* \subseteq X_1 \cap \cdots \cap X_c$.

Since X is shattered, for each $1 \leq j \leq c$, there exists a distinct edge $e_j \in E(H)$ with $X_j = X \cap e_j$. Hence, $X_j = X \setminus \{v_j\} \subseteq e_j$ and also $X^* \subseteq e_1 \cap e_2 \cap \cdots \cap e_c$ holds, i.e., X^* is in the intersection of c edges of H. By c-*miwidth*$(H) \leq i$, we thus get $|X^*| \leq i$. In total, we have $|X| \leq |X^*| + c \leq i + c$, which contradicts our assumption that $|X| > c + i$ holds.

[bounded VC-dimension ⇏ BMIP.] It suffices to exhibit a family $(H_n)_{n\in\mathbb{N}}$ of hypergraphs such that $\mathrm{vc}(H_n)$ is bounded whereas c-*miwidth*(H_n) is unbounded for any constant c. We define $H_n = (V_n, E_n)$ as follows:

$V_n = \{v_1, \ldots, v_n\}$

$E_n = \{V_n \setminus \{v_i\} \mid 1 \leq i \leq n\}$

Clearly, $\mathrm{vc}(H_n) \leq 2$. Indeed, take an arbitrary set $X \subseteq V$ with $|X| \geq 2$. Then $\emptyset \subseteq X$ but $\emptyset \neq X \cap e$ for any $e \in E_n$. On the other hand, let $c \geq 1$ be an arbitrary constant and let $X = e_{i_1} \cap \cdots \cap e_{i_\ell}$ for some $\ell \leq c$ and edges $e_{i_j} \in E_n$. Obviously, $|X| \geq n - c$ holds. Hence, also $c\text{-}miwidth(H_n) \geq n - c$, i.e., it is not bounded by any constant $i \geq 0$. □

In the first part of Lemma 4.27, we have shown that $\mathrm{vc}(H) \leq c + i$ holds. For an approximation of an FHD by a GHD, we need to approximate the fractional edge cover γ_u of each bag B_u by an integral edge cover λ_u, i.e., we consider fractional vs. integral edge covers of the induced hypergraphs $H_u = (B_u, E_u)$ with $E_u = \{e \cap B_u \mid e \in E(H)\}$. Obviously, the bound $\mathrm{vc}(H) \leq c + i$ carries over to $\mathrm{vc}(H_u) \leq c + i$.

Corollary 4.3. *Let $\mathscr{C}$ be a class of hypergraphs enjoying the BMIP and let $k \geq 1$. Then there exists a polynomial-time algorithm that, given $H \in \mathscr{C}$ with $fhw(H) \leq k$, finds an FHD (actually, even a GHD) of H of width $\mathcal{O}(k \cdot \log k)$.*

CHAPTER 5

To Practice

In the previous chapter we investigated generalized and fractional hypertree decompositions from a theoretical view. For practical purposes it is important to have hypergraphs of low fractional or generalized hypertree width. In this chapter we are going to answer Research Questions 5-10.

In order to answer these questions we will first present a collection of hypergraphs converted from real-world and synthetic CQ and CSP instances in Section 5.1. In Section 5.2 we will determine the hypertree width of all our instances. We will then investigate in Section 5.3 whether the hypergraph properties (BIP, BMIP, VC-dim, BDP) that allow for efficient computation of generalized and fractional hypertree decompositions are indeed small on our benchmark instances. In Section 5.4 we will present novel algorithms for determining the generalized hypertree width. These algorithms are based on the tractability results given in Section 4.2. For fractional hypertree decompositions we will give in Section 5.5 an algorithm that determines the exact fractional hypertree width in case of bounded rank. We close this chapter in Section 5.6 with a short conclusion and comparison of the different widths.

5.1 HyperBench: A Benchmark of Hypergraphs

Our benchmark contains 3,070 hypergraphs, which have been converted from CQs and CSPs collected from various sources. Out of these 3,070 hypergraphs, 2,918 hypergraphs have never been used in a hypertree width analysis before. The hypertree width of 70 CQs and of 82 CSPs has been analysed in [54], [17], and/or [21]. An overview of all instances of CQs and CSPs is given in Table 5.1. They have been collected from various publicly available benchmarks and repositories of CQs and CSPs. In the first column, the names of each collection of CQs and CSPs are given together with references where they were first published. In the second column we display the number of hypergraphs extracted from each collection. The hw of the CQs and CSPs in our benchmark will be

	Benchmark	*No. instances*	$hw \geq 2$
CQs	SPARQL[21]	70 (out of 26,157,880)	70
	LUBM[16, 59]	14	2
	iBench[16, 8]	40	0
	Doctors[16, 42]	14	0
	Deep[16]	41	0
	JOB (IMDB) [70]	33	7
	TPC-H [15, 92]	33	1
	SQLShare [62]	290 (out of 15,170)	1
	Random [81]	500	464
CSPs	Application [11]	1,090	1,090
	Random [11]	863	863
	Other [54, 17]	82	82
	Total:	*3,070*	*2,580*

Table 5.1: Overview of benchmark instances

discussed in detail in Section 5.2. To get a first feeling of the *hw* of the various sources, we mention the number of cyclic hypergraphs (i.e., those with $hw \geq 2$) in the last column. When gathering the CQs, we proceeded as follows: of the huge benchmark reported in [21], we have only included CQs, which were detected as having $hw \geq 2$ in [21]. Of the big repository reported in [62], we have included those CQs, which are not trivially acyclic (i.e., they have at least 3 atoms). Of all the small collections of queries, we have included all.

Below, we describe the different benchmarks in detail:

- *CQs*: Our benchmark contains 535 CQs from four main sources [15, 16, 21, 62] and a set of 500 randomly generated queries using the query generator of [81]. In the sequel, we shall refer to the former queries as *CQ Application*, and to the latter as *CQ Random*. The CQs analysed in [21] constitute by far the biggest repository of CQs – namely 26,157,880 CQs stemming from SPARQL queries. The queries come from real-users of SPARQL endpoints and their hypertree width was already determined in [21]. Almost all of these CQs were shown to be acyclic. Our analysis comprises 70 CQs from [21], which (apart from few exceptions) are essentially the ones in [21] with $hw \geq 2$. In particular, we have analysed all 8 CQs with highest *hw* among the CQs analysed in [21] (namely, $hw = 3$).

  The LUBM [59], iBench [8], Doctors [42], and Deep scenarios have been recently used to evaluate the performance of chase-based systems [16]. Their queries were especially tailored towards the evaluation of query answering tasks of such systems. Note that the LUBM benchmark [59] is a widely used standard benchmark for the evaluation of Semantic Web repositories. Its queries are designed to measure the performance of those repositories over large datasets. Strictly speaking, the iBench

is a tool for generating schemas, constraints, and mappings for data integration tasks. However, in [16], 40 queries were created for tests with the iBench. We therefore refer to these queries as iBench-CQs here. In summary, we have incorporated all queries that were either contained in the original benchmarks or created/adapted for the tests in [16].

The goal of the Join Order Benchmark (JOB) [70] was to evaluate the impact of a good join order on the performance of query evaluation in standard RDBMS. Those queries were formulated over the real-world dataset Internet Movie Database (IMDB). All of the queries have between 3 and 16 joins. Clearly, as the goal was to measure the impact of a good join order, those 33 queries are of higher complexity, hence 7 out of the 33 queries have $hw \geq 2$.

The 33 TPC-H queries in our benchmark are taken from the GitHub repository originally provided by Michael Benedikt and Efthymia Tsamoura [15] for the work on [12]. Out of the 33 CQs based on the TPC-H benchmark [92], 13 queries were handcrafted and 20 randomly generated. The TPC-H benchmark has been widely used to assess multiple aspects of the capabilities of RDBMS to process queries. They reflect common workloads in decision support systems and were chosen to have broad industry-wide relevance.

From SQLShare [62], a multi-year SQL-as-a-service experiment with a large set of real-world queries, we extracted 15,170 queries by considering all CQs (in particular, no nested SELECTs). After eliminating trivial queries (i.e., queries with ≤ 2 atoms, whose acyclicity is immediate) and duplicates, we ended up with 290 queries.

The random queries were generated with a tool that stems from the work on query answering using views in [81]. The query generator allows 3 options: chain/star/random queries. Since the former two types are trivially acyclic, we only used the third option. Here it is possible to supply several parameters for the size of the generated queries. In terms of the resulting hypergraphs, one can thus fix the number of vertices, number of edges and arity. We have generated 500 CQs with 5 – 100 vertices, 3 – 50 edges and arities from 3 to 20. These values correspond to the values observed for the *CQ Application* hypergraphs. However, even though these size values have been chosen similarly, the structural properties of the hypergraphs in two groups *CQ Application* and *CQ Random* differ significantly, as will become clear from our analysis in Section 5.3.

- *CSPs*: In total, our benchmark currently contains 2,035 hypergraphs from CSP instances, out of which 1,953 instances were obtained from `xcsp.org` (see also [11]). We have selected all CSP instances from `xcsp.org` with less than 100 constraints such that all constraints are extensional. These instances are divided into CSPs from concrete applications, called *CSP Application* in the sequel (1,090 instances), and randomly generated CSPs, called *CSP Random* below (863 instances). In addition, we have included 82 CSP instances from previous hypertree width analyses provided at `https://www.dbai.tuwien.ac.at/proj/hypertree/`; all of these stem

from industrial applications and/or further CSP benchmarks. We refer to these instances as *other CSPs*. Out of the 82 instances 61 were already used as a benchmark for the HD-computation in [54]. These 61 instances are split into 15 CSP instances from DaimlerChrysler, 12 CSP instances from Grid2D, and 24 CSP instances from the ISCAS'89 benchmark on circuits. The 82 instances also include 35 instances used in the MaxSAT Evaluation 2017 [17]. There a MaxSAT encoding was created to determine the *ghw* of each of the hypergraphs. During the competition 7 out of the 35 instances were solved. We refer to the resulting 82 hypergraphs as "other CSPs". In our analysis, we keep them separated from the instances in the "CSP Application" and "CSP Random" category mentioned above because of the different provenance and for the sake of comparability with previous evaluations of hypergraph decomposition algorithms.

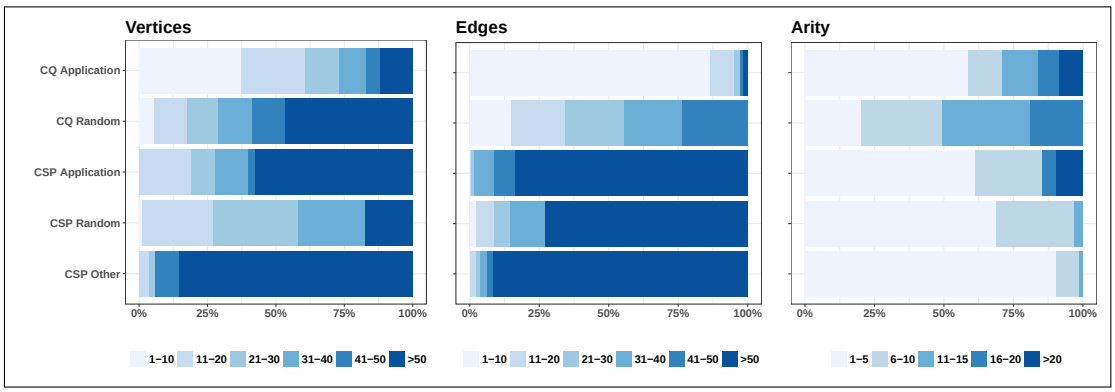

Figure 5.1: Hypergraph Sizes

Our HyperBench benchmark consists of these instances converted to hypergraphs. In Figure 5.1, we show the number of vertices, the number of edges and the arity (i.e., the maximum size of the edges) as three important metrics of the size of each hypergraph. The smallest are those coming from *CQ Application* (at most 10 edges), while the hypergraphs coming from CSPs can be significantly larger (up to 2993 edges). Although some hypergraphs are very big, more than 50% of all hypergraphs have maximum arity less than 5. In Figure 5.1 we can easily compare the different types of hypergraphs, e.g. hypergraphs of arity greater than 20 only exist in the *CSP Application* class; the *other CSPs* class contains the highest portion of hypergraphs with a big number of vertices and edges, etc.

5.1.1 Web tool

The hypergraphs in the benchmark and the results of the analyses of these hypergraphs can be accessed via a web tool, which is available at `http://hyperbench.dbai.tuwien.ac.at`. There we have uploaded 3,070 hypergraphs together with over 5,518 HDs and the output of over 16,585 further algorithm runs, where no HD of desired

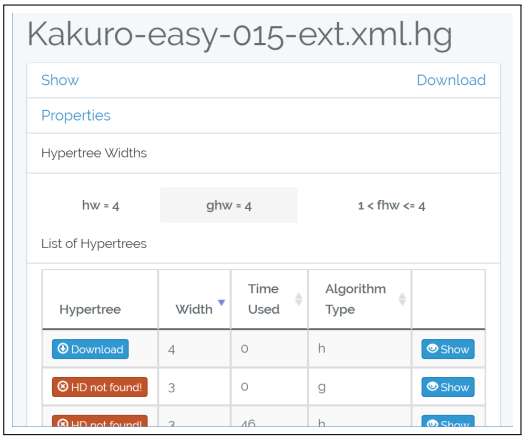

Figure 5.2: HyperBench web tool: available at

width was found (either because a lower bound on the width was established or the algorithm timed out). For example, in the screenshot in Figure 5.2 the results for the CSP instance "Kakuro-easy-015-ext.xml.hg" are displayed. The hypertree width of the instance is calculated according to the list of HDs at the bottom of the screenshot. For this instance we have several algorithm runs, some of which led to a HD, some did not ("HD not found"). With these we were able to pinpoint that $hw = ghw = 4$ holds. All instances can be explored in such a way.

Additionally, we allow the user to browse, download and inspect hypergraph categories presented in this work. In the near future, we will also provide a search interface to download instances having specific properties (e.g. $hw < 5$ or $BIP < 3$, etc.) and to contribute to the benchmark by uploading hypergraphs, which are then analysed and incorporated into our HyperBench benchmark.

5.1.2 System and Test Environment.

In [54], an implementation (called `DetKDecomp`) of the hypertree decomposition algorithm from [51] was presented. We have extended this implementation and built our new library (called `NewDetKDecomp`) of tools and algorithms upon it. This library includes the original *hw*-algorithm from [54], the tool `hg-stats` to determine properties described in Section 5.3 and the algorithms to be presented in Sections 5.4 and 5.5. The library is written in C++ and comprises around 8,500 lines of code. The code is publicly available in GitHub at `http://github.com/TUfischl/newdetkdecomp`.

To solve the linear programs (LPs) for computing fractional covers in our algorithm in Section 5.5, we use the *COIN-OR Linear Programming Solver* (CLP) version 1.16 from `https://projects.coin-or.org/Clp`. We have chosen CLP over other open-source LP solvers based on very promising empirical results reported in [41].

All the experiments reported in this paper were performed on a cluster of 10 workstations

each running Ubuntu 16.04. Every workstation has the same specification and is equipped with two Intel Xeon E5-2650 (v4) processors each having 12 cores and 256-GB main memory. Since all algorithms are single-threaded, we were allowed to compute several instances in parallel. For all upcoming runs of our algorithms we set a timeout of 3600s.

5.2 Hypertree Width

We have systematically applied the *hw*-computation from [54] to all hypergraphs in the benchmark. The results are given in Table 5.2 with their graphical representation given in Figure 5.3. In our experiments, we proceeded as follows. We distinguish between the five groups of hypergraphs as in Figure 5.4. For every hypergraph H, we first tried to solve the Check(HD, k) problem for $k = 1$. In case of *CQ Application*, we thus got 454 yes-answers and 81 no-answers. The number in each bar indicates the average runtime to find these yes- and no-instances, respectively. Here, the average runtime was "0" (i.e., less than 1 second) in both cases. For *CQ Random* we got 36 yes- and 464 no-instances with an average runtime below 1 second. For all three groups of CSP-instances, we only got no-answers with an average runtime below 1 second.

In the second round, we tried to solve the Check(HD, k) problem for $k = 2$ for all hypergraphs that yielded a no-answer for $k = 1$. Now the picture is a bit more diverse: 73 of the remaining 81 CQs from *CQ Application* yielded a yes-answer in less than 1 second. For the hypergraphs stemming from *CQ Random* (resp. CSPs), only 68 (resp. 95) instances yielded a yes-answer (in less than 1 second on average), while 396 (resp. 1932) instances yielded a no-answer in less than 7 seconds on average and 8 CSP instances led to a timeout (i.e., the program did not terminate within 3,600 seconds).

This procedure is iterated by incrementing k and running the *hw*-computation for all instances, that either yielded a no-answer or a timeout in the previous round. For instance, for queries from *CQ Application*, one further round is needed after the second round. In other words, we confirm the observation of low *hw*, which was already made for CQs of arity ≤ 3 in [21, 80]. For the hypergraphs stemming from *CQ Random* (resp. CSPs), 396 (resp. 1940)instances are left in the third round, of which 70 (resp. 232) yield a yes-answer in less than 1 second on average, 326 (resp. 1415) instances yield a no-answer in 32 (resp. 988) seconds on average and no (resp. 293) instances yield a timeout. Note that, as we increase k, the average runtime and the percentage of timeouts first increase up to a certain point and then they decrease. This is due to the fact that, as we increase k, the number of combinations of edges to be considered in each λ-label (i.e., the function λ_u at each node u of the decomposition) increases. In principle, we have to test $\mathcal{O}(n^k)$ combinations, where n is the number of edges. However, if k increases beyond a certain point, then it gets easier to "guess" a λ-label since an increasing portion of the $\mathcal{O}(n^k)$ possible combinations leads to a solution (i.e., an HD of desired width).

To answer *Research Question 6* of Section 1.1, it is indeed the case that for a big number of instances, the hypertree width is small enough to allow for efficient evaluation of CQs or CSPs: all instances of non-random CQs have $hw \leq 3$ no matter whether their arity is

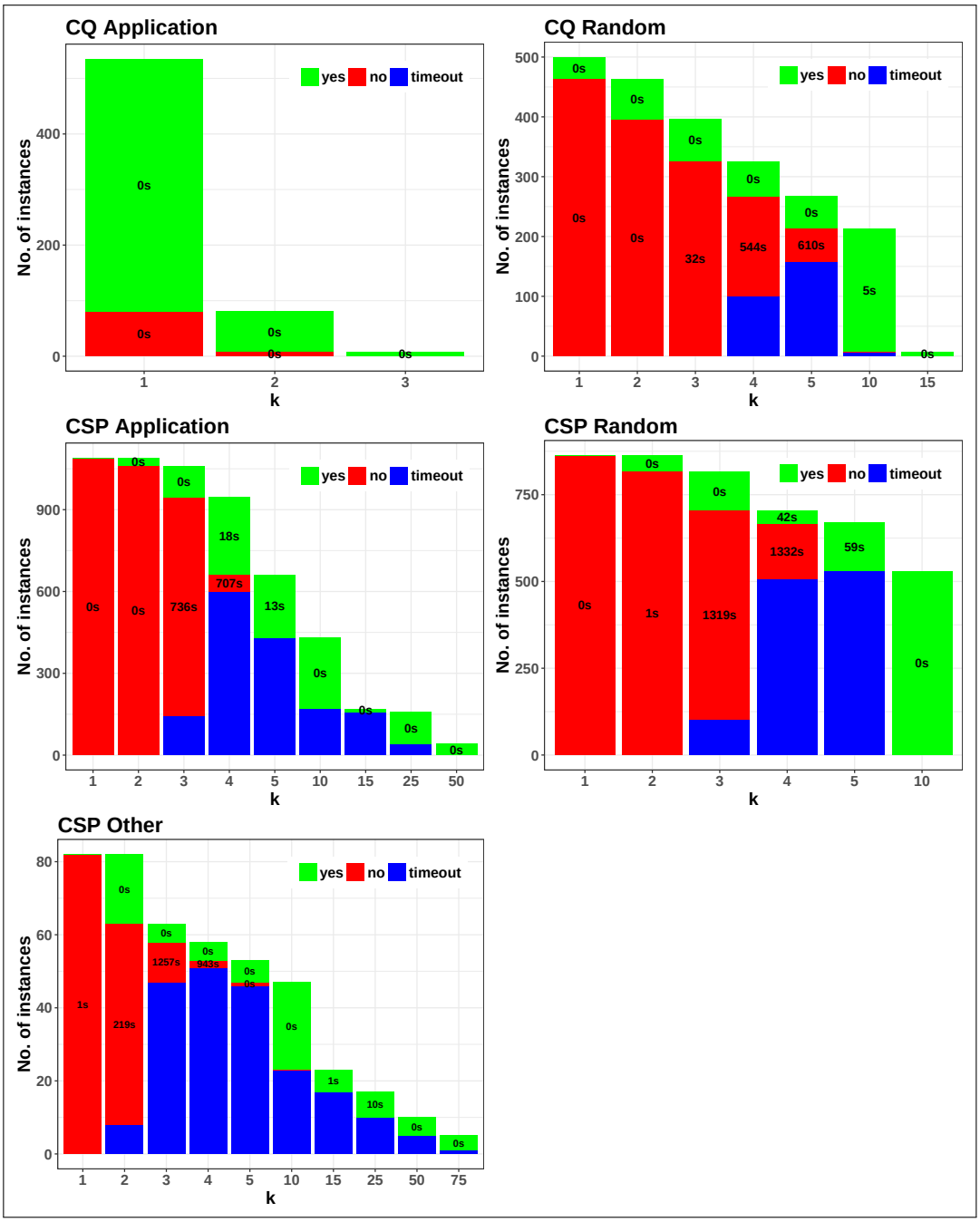

Figure 5.3: HW analysis (labels are average runtimes in s)

CQ Application

k	yes	no	timeout
1	454 (0)	81 (0)	0
2	73 (0)	8 (0)	0
3	8 (0)	0	0

CQ Random

k	yes	no	timeout
1	36 (0)	464 (0)	0
2	68 (0)	396 (0)	0
3	70 (0)	326 (32)	0
4	59 (0)	167 (544)	100
5	54 (0)	55 (610)	158
10	206 (5)	0	7
15	7 (0)	0	0

CSP Application

k	yes	no	timeout
1	0	1090 (0)	0
2	29 (0)	1061 (0)	0
3	116 (0)	802 (736)	143
4	283 (18)	62 (707)	600
5	231 (13)	0	431
10	261 (0)	0	170
15	12 (0)	0	158
25	118 (0)	0	40
50	40 (0)	0	0

CSP Random

k	yes	no	timeout
1	0	863 (0)	0
2	47 (0)	816 (1)	0
3	111 (0)	602 (1319)	103
4	39 (42)	160 (1332)	506
5	136 (59)	0	530
10	530 (0)	0	0

CSP Other

k	yes	no	timeout
1	0	82 (1)	0
2	19 (0)	55 (219)	8
3	5 (0)	11 (1257)	47
4	5 (0)	2 (943)	51
5	6 (0)	1 (0)	46
10	24 (0)	0	23
15	6 (1)	0	17
25	7 (10)	0	10
50	5 (0)	0	5
75	4 (0)	0	1

Table 5.2: HW of instances with average runtime in s

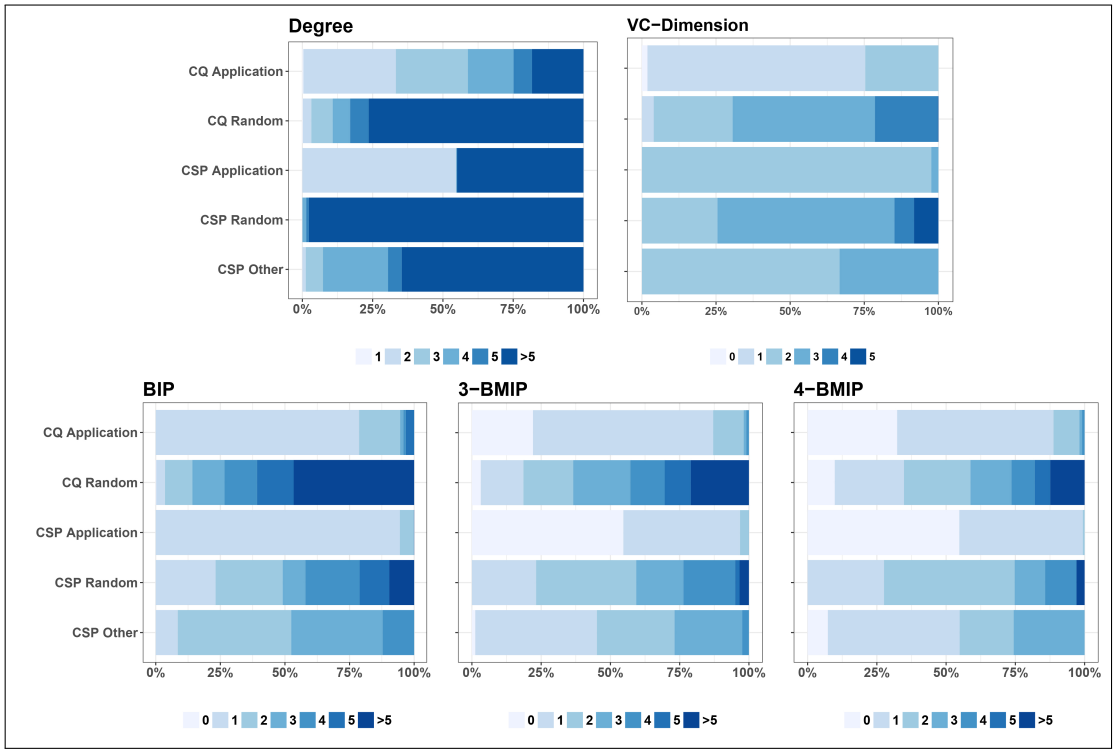

Figure 5.4: Hypergraph Properties

bounded by 3 (as in case of SPARQL queries) or not; and a large portion (at least 1027, i.e., ca. 50%) of all 2035 CSP instances have $hw \leq 5$. In total, including random CQs, 1,849 (60%) out of 3,070 instances have $hw \leq 5$, for which we could determine the exact hypertree width for 1,453 instances; the others may even have lower hw.

5.3 Hypergraph Properties

In Sections 4.2 – 4.4 several invariants of hypergraphs were used to make the Check(GHD, k) and Check(FHD, k) problems tractable or, at least, easier to approximate. We thus investigate the following properties (cf. Definitions 4.2, 4.3, 4.6, and 4.19):

- *Deg*: the degree of the underlying hypergraph
- *BIP*: the intersection width
- *c-BMIP*: the c-multi-intersection width for $c \in \{3, 4\}$
- *VC-dim*: the VC-dimension

CQ Application

i	Deg	BIP	3-BMIP	4-BMIP	VC-dim
0	0	0	118	173	10
1	2	421	348	302	393
2	176	85	59	50	132
3	137	7	5	5	0
4	87	5	5	5	0
5	35	17	0	0	0
6	98	0	0	0	0

CQ Random

i	Deg	BIP	3-BMIP	4-BMIP	VC-dim
0	0	1	16	49	0
1	1	17	77	125	20
2	15	53	90	120	133
3	38	62	103	74	240
4	31	63	62	42	106
5	33	71	47	28	1
6	382	233	105	62	0

CSP Application

i	Deg	BIP	3-BMIP	4-BMIP	VC-dim
0	0	0	596	597	0
1	0	1030	459	486	0
2	596	59	34	7	1064
3	1	0	1	0	26
4	1	0	0	0	0
5	2	0	0	0	0
>5	490	1	0	0	0

CSP Random

i	Deg	BIP	3-BMIP	4-BMIP	VC-dim
0	0	0	0	0	0
1	0	200	200	238	0
2	0	224	312	407	220
3	0	76	147	95	515
4	12	181	161	97	57
5	8	99	14	1	71
>5	843	83	29	25	0

CSP Other

i	Deg	BIP	3-BMIP	4-BMIP	VC-dim
0	0	0	1	6	0
1	0	7	36	39	0
2	1	36	23	16	51
3	5	29	20	21	26
4	19	10	2	0	0
5	4	0	0	0	0
> 5	53	0	0	0	0

Table 5.3: Hypergraph properties

The results obtained from computing *Deg*, *BIP*, 3-*BMIP*, 4-*BMIP*, and *VC-dim* for the hypergraphs in the HyperBench benchmark are shown in Table 5.3 with their graphical representation given in Figure 5.4.

Table 5.3 has to be read as follows: In the first column, we distinguish different values of the various hypergraph metrics. In the columns labelled "Deg", "BIP", etc., we indicate for how many instances each metric has a particular value. For instance, by the last row in the second column, only 98 non-random CQs have degree > 5. Actually, for most CQs, the degree is less than 10. Moreover, for the BMIP, already with intersections of 3 edges, we get $3\text{-}miwidth(H) \leq 2$ for almost all non-random CQs. Also the VC-dimension is at most 2.

For CSPs, all properties may have higher values. However, we note a significant difference between randomly generated CSPs and the rest: For hypergraphs in the groups *CSP Application* and *CSP Other*, 543 (46%) hypergraphs have a high degree (>5), but nearly all instances have BIP or BMIP of less than 3. And most instances have a VC-dimension of at most 2. In contrast, nearly all random instances have a significantly higher degree (843 out of 863 instances with a degree >5). Nevertheless, many instances have small BIP and BMIP. For nearly all hypergraphs (838 out of 863) we have $4\text{-}miwidth(H) \leq 4$. For 5 instances the computation of the VC-dimension timed out. For all others, the VC-dimension is ≤ 5 for random CSPs. Clearly, as seen in Table 5.3, the random CQs resemble the random CSPs a lot more than the CQ and CSP Application instances. For example, random CQs have similar to random CSPs high degree (382 (76%) with degree > 5), higher BIP and BMIP. Nevertheless, similar to random CSPs, the values for BIP and BMIP are still small for many random CQ instances.

To conclude, for the proposed properties, in particular BIP/BMIP and VC-dimension, most hypergraphs in our benchmark (even for random CQs and CSPs) indeed have low values.

5.3.1 Correlation Analysis.

Finally, we have analysed the pairwise correlation between all properties. Of course, the different intersection widths (BIP, 3-BMIP, 4-BMIP) are highly correlated. Other than that, we only observe quite a high correlation of the arity with the number of vertices and the hypertree width and of the number of vertices with the arity and the hypertree width. Clearly, the correlation between arity and hypertree width is mainly due to the CSP instances and the random CQs since, for non-random CQs, the *hw* never increases beyond 3, independently of the arity.

A graphical presentation of all pairwise correlations is given in Figure 5.5. Here, large, dark circles indicate a high correlation, while small, light circles stand for low correlation. Blue circles indicate a positive correlation while red circles stand for a negative correlation. In [38], we have argued that Deg, BIP, 3-BMIP, 4-BMIP and VC-dim are non-trivial restrictions to achieve tractability. It is interesting to note that, according to the correlations shown in Figure 5.5, these properties have almost no impact on the hypertree

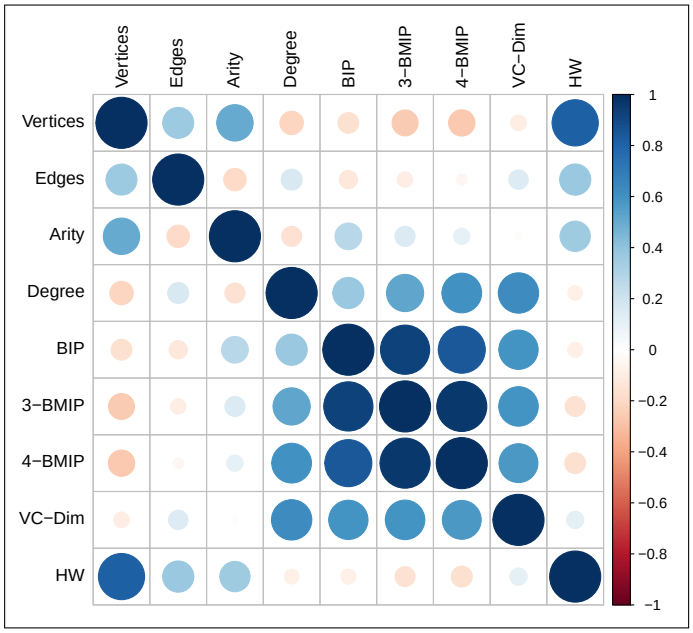

Figure 5.5: Correlation analysis.

width of our hypergraphs. This underlines the usefulness of these restrictions in the sense that (a) they make the GHD computation and FHD approximation easier [38] but (b) low values of degree, (multi-)intersection-width, or VC-dimension do not pre-determine low values of the widths.

5.4 GHW Computation

In Section 4.2, it is shown that the Check(GHD, k) problem becomes tractable for fixed $k \geq 1$, if we restrict ourselves to a class of hypergraphs enjoying the BIP. As our empirical analysis with the HyperBench has shown (see Section 5.3), it is indeed realistic to assume that the intersection width of a given hypergraph is small. We have therefore extended the *hw*-computation from [54] by an implementation of the Check(GHD, k) algorithm presented in Section 4.2, which will be referred to as the "*ghw*-algorithm" in the sequel. This algorithm is parametrized, so to speak, by two integers: k (the desired width of a GHD) and i (the intersection width of H).

The key idea of the *ghw*-algorithm is to add a polynomial-time computable set $f(H,k)$ of subedges of edges in $E(H)$ to the hypergraph H, such that $ghw(H) = k$ iff $hw(H') = k$ with $H = (V(H), E(H))$ and $H' = (V(H), E(H) \cup f(H,k))$. Tractability of Check(GHD, k) follows immediately from the tractability of the Check(HD, k) problem. Recall that the set $f(H,k)$ is defined as

$$f(H,k) = \bigcup_{e \in E(H)} \Big(\bigcup_{e_1,\ldots,e_j \in (E(H)\setminus\{e\}),\, j \leq k} 2^{(e \cap (e_1 \cup \cdots \cup e_j))} \Big),$$

Table 5.4: Comparison of GHW algorithms w. avg. runtime (s)

hw →		GlobalBIP		LocalBIP		BalSep
ghw	*total*	*yes*	*no*	*yes*	*no*	*no*
3 → 2	310	-	128 (537)	-	195 (162)	307 (12)
4 → 3	386	-	137 (2809)	-	54 (2606)	249 (54)
5 → 4	427	-	-	-	-	148 (13)
6 → 5	459	13 (162)	-	13 (60)	-	180 (288)

i.e., $f(H,k)$ contains all subsets of intersections of edges $e \in E(H)$ with unions of $\leq k$ edges of H different from e. By the BIP, the intersection $e \cap (e_1 \cup \cdots \cup e_j)$ has at most $i \cdot k$ elements. Hence, for fixed constants i and k, $|f(H,k)|$ is polynomially bounded.

5.4.1 "Global" Implementation

In a straightforward implementation of this algorithm, we compute $f(H,k)$ and from this H' and call the *hw*-computation from [54] for the Check(HD, k) problem as a "black box". A coarse-grained overview of the results is given in Table 5.4 in the column labelled as "GlobalBIP". We call this implementation of the *ghw* algorithm "global" to indicate that the set $f(H,k)$ is computed "globally", once and for all, for the *entire* hypergraph. We have run the program on each hypergraph from the HyperBench up to hypertree width 6, trying to get a smaller *ghw* than *hw*. We have thus run the *ghw*-algorithm with the following parameters: for all hypergraphs H with $hw(H) = k$ (or $hw \leq k$ and, due to timeouts, we do not know if $hw \leq k-1$ holds), where $k \in \{3,4,5,6\}$, try to solve the Check(GHD, $k-1$) problem. In other words, we just tried to improve the width by 1. Clearly, for $hw(H) \in \{1,2\}$, no improvement is possible since, in this case, $hw(H) = ghw(H)$ holds.

In Table 5.4, we report on the number of of instances where we solved the Check(GHD,$k-1$) problem for hypergraphs with $hw = k$. As before, we set a timeout of 1 hour. For instance, for the 310 hypergraphs with $hw = 3$ in the HyperBench, the "global" computation terminated in 128 cases (i.e., 41%) when trying to solve Check(GHD, 2). The average runtime was 537 seconds. All 128 cases were "no" answers, i.e. these hypergraphs have $ghw > 2$ and therefore these 128 hypergraphs of our benchmark have $ghw = hw = 3$. For the 386 hypergraphs with $hw = 4$, the "global" computation terminated in 137 cases (i.e., 35%) with average runtime 2809 when trying to solve the Check(GHD, 3) problem. The algorithm did not terminate in one hour for any of the 427 hypergraphs with $hw = 5$. Only for hypergraphs with $hw = 6$, the "global" computation returned a generalized hypertree decomposition of width 5. This was the case for 13 out of 459 (i.e., 2.8%) hypergraphs with an average runtime of 162 seconds. Overall, it turns out that the set $f(H,k)$ may be very big (even though it is polynomial if k and i are constants). Hence, H' can become considerably bigger than H. This explains the frequent timeouts in the GlobalBIP column in Table 5.4.

5.4.2 "Local" Implementation

Looking for ways to improve the *ghw*-algorithm, we closely inspect the role played by the set $f(H,k)$ in the proof of Theorem 4.2. The definition of this set is motivated by the problem that, in the top down construction of a GHD, we may want to choose at some node u the bag B_u such that $x \notin B_u$ for some variable $x \in B(\lambda_u) \cap V(T_u)$. This violates condition (4) of Definition 2.4 (the "special condition") and is therefore forbidden in an HD. In particular, there exists an edge e with $x \in e$ and $\lambda_u(e) = 1$. The crux of the *ghw*-algorithm in Section 4.2 is that for every such "missing" variable x, the set $f(H,k)$ contains a subedge $e' \subseteq e$ with $x \notin e'$. Hence, replacing e by e' in λ_u (i.e., setting $\lambda_u(e) = 0$, $\lambda_u(e') = 1$ and leaving λ_u unchanged elsewhere) eliminates the special condition violation. By the connectedness condition, it suffices to consider the intersections of e with unions of edges that may possibly occur in bags of T_u rather than with arbitrary edges in $E(H)$. In other words, for each node u in the decomposition, we may restrict $f(H,k)$ to an appropriate subset $f_u(H,k) \subseteq f(H,k)$.

The idea behind the construction of $f(H,k)$ is to assume the existence of a GHD $\mathcal{D}$ of desired width $\leq k$ and to transform $\mathcal{D}$ into an HD. Of course, if $\mathcal{D}$ is an HD, then we are done. So suppose that in some node u of $\mathcal{D}$, condition 4 (= the "special condition") of Definition 2.4 is violated. This means that λ_u contains some edge e, such that some vertex $x \in e$ occurs in $V(T_u)$ for the subtree T_u rooted at u but not in B_u. Now $f(H,k)$ is constructed in such a way that it contains an appropriate subedge $e' \subseteq e$, such that (1) $x \notin e'$ and (2) replacing e in λ_u by e' still satisfies condition 3 of Definition 2.3. We can be sure that all of e is contained in $B_{u'}$ for some descendant node u' of u. Moreover, by the connectedness condition, all vertices in $e \cap B_u$ must occur in B_v for every node v along the whole path from u to u'. It is thus shown in Section 4.2 that the only subedges $e' \subseteq e$ relevant for healing a special condition violation at node u are those obtained from intersecting e with edges in some bag λ_v on the path from u to u'. At every node u, we can therefore replace the "global" set $f(H,k)$ of subedges by a "local" set $f_u(H,k)$, where we intersect each edge e in λ_u only with edges that have to be covered yet in the subtree below u (i.e., the $[B_u]$-component C_u) or which at least have a non-empty intersection with some edge in C_u.

The results obtained with this enhanced version of the *ghw*-computation are shown in Table 5.4 in the column labelled "LocalBIP". We call this implementation of *ghw*-computation "local" because the set $f_u(H,k)$ of subedges of H to be added to the hypergraph is computed separately for each node u of the decomposition. Recall that in this table, all calls to the *ghw*-algorithm are recorded that finished within one hour. Interestingly, for the hypergraphs with $hw = 3$, the "local" computation performs significantly better (namely 63% solved with average runtime 162 seconds rather than 41% with average runtime 537 seconds). In contrast, for the hypergraphs with $hw = 4$, the "global" computation is significantly more successful. For $hw \in \{5,6\}$, the "global" and "local" computations are equally bad. A possible explanation for the reverse behaviour of "global" and "local" computation in case of $hw = 3$ as opposed to $hw = 4$ is that the restriction of the "global" set $f(H,k)$ of subedges to the "local" set $f_u(H,k)$ at each node

u seems to be quite effective for the hypergraphs with $hw = 3$. In contrast, the additional cost of having to compute $f_u(H, k)$ at each node u becomes counter-productive, when the set of subedges thus eliminated is not significant. It is interesting to note that the sets of solved instances of the global computation and the local computation are incomparable, i.e., in some cases one method is better, while in other cases the other method is better.

5.4.3 Balanced Separators

As we have seen so far, both "global" and "local" *ghw*-computation have not found a single instance of $hw \leq 5$ that has a smaller *ghw*. Unfortunately, also the number of instances where the *ghw*-computation did not terminate is high (e.g., 100% for instances where $hw = 5$). However, the results for $hw \in \{2, 3, 4\}$ suggest that for many of the hypergraphs with $hw \leq 6$ it is indeed the case that $ghw = hw$. Therefore, we will look for a method to especially characterize "no"-instances for the Check(GHD,k) problem.

We propose a completely new approach, based on so-called "balanced separators". The latter are a familiar concept in graph theory [34, 88] – denoting a set S of vertices of a graph G, such that the subgraph G' induced by $V(G) \setminus S$ has no connected component larger than some given size, e.g., $\alpha \cdot |V|$ for some given $\alpha \in (0, 1)$. In our setting, we may consider the label B_u at some node u in a GHD as *separator* in the sense that we can consider connected components of the subhypergraph H' of H induced by $V(H) \setminus B_u$. Clearly, in a GHD, we may consider any node as the root. So suppose that u is the root of some GHD.

In addition, we know from Theorem 2.1, that we can transform any HD, GHD, or FHD in such a way that every subtree rooted at a child node u_i of u contains exactly one connected component C_i of the subhypergraph H' induced by $V(H) \setminus B_u$. For our purposes, it is convenient to define the size of a component C_i as the number of edges that have to be covered at some node in the subtree rooted at u_i in the GHD. We thus call a separator u "balanced", if the size of each $B[u]$-component C_i is at most $|E(H)|/2$. The following observation is immediate:

Proposition 5.1. *In every GHD, there exists a node u such that B_u is a balanced separator.*

Proposition 5.1 can be easily turned into an algorithm to detect "no"-instances for the Check(GHD,k) problem. First observe that the size of B_u can be unbounded. Hence, guessing a set of vertices as a balanced separator is not an option. Since in every GHD $B_u \subseteq B(\lambda_u)$ holds, we can restrict ourselves to guessing k edges. This suffices since if there does not exist a λ_u, such that $B(\lambda_u)$ is a balanced separator then there does not exist a B_u, such that $B_u \subseteq B(\lambda_u)$ is a balanced separator. Hence, whenever we are not able to find for a hypergraph H a set $S \subseteq E(H)$ of k edges, such that $\bigcup S$ is a balanced separator, we can be sure that there does not exist a GHD of width k for H. A formal description is given in Algorithm 5.1.

Algorithm 5.1: `BalSep`

input : Hypergraph H.
output : "Accept", if H has a balanced separator $S \subseteq E(H)$ of size k
"Reject", otherwise.

```
1 begin
2 |  Guess a set S ⊆ E(H) with |S| = k ;          /* (1) Guess */
3 |  if S is a balanced separator then            /* (2) Check */
4 |  |  return Accept;
5 |  else
6 |  |  return Reject;
7 |  end
8 end
```

Clearly, Algorithm 5.1 is sound and complete for checking if there exists a balanced separator of size k. Since the check if a separator is balanced can be done in linear time, the next theorem is immediate.

Theorem 5.1. *`BalSep` is correct and runs in time $\mathcal{O}(n^k)$.*

By combining Theorem 5.1 with Proposition 5.1 we get the following corollary.

Corollary 5.1. *Let H be a hypergraph and let $k \geq 1$. If the algorithm `BalSep` outputs "**Reject**" then H does not allow for a GHD of width $\leq k$.*

In Table 5.4 we compare the output of Algorithm 5.1 with the "global" and "local" *ghw*-computation. Notice that the `BalSep` algorithm only outputs negative answers. It is indeed the case that the algorithm detects quite fast that a given hypergraph does not have a balanced separator of desired width. For hypergraphs with $hw = 3$ this is the case in 307 (99%) out of 310 instances. As we increase k, the performance deteriorates, due to k in the exponent of the running time of our algorithm. For hypergraphs with $hw = 6$, only 180 (39%) out of 459 instances are solved. Still, for negative instances the balanced separator approach performs best.

5.4.4 Empirical results.

We now look at Table 5.5 and its graphical representation in Figure 5.6, where we report for all hypergraphs with $hw \leq k$ and $k \in \{3, 4, 5, 6\}$, whether $ghw \leq k - 1$ could be verified. To this end, we run our three algorithms ("global", "local", and "balanced separators") in parallel and stop the computation, as soon as one terminates (with answer "yes" or "no"). The numbers on the bars in Figure 5.6 and in parentheses in Table 5.5 refer to the average runtime needed by the fastest of the three algorithms in each case. A timeout occurs if none of the three algorithms terminates within 3,600 seconds. It

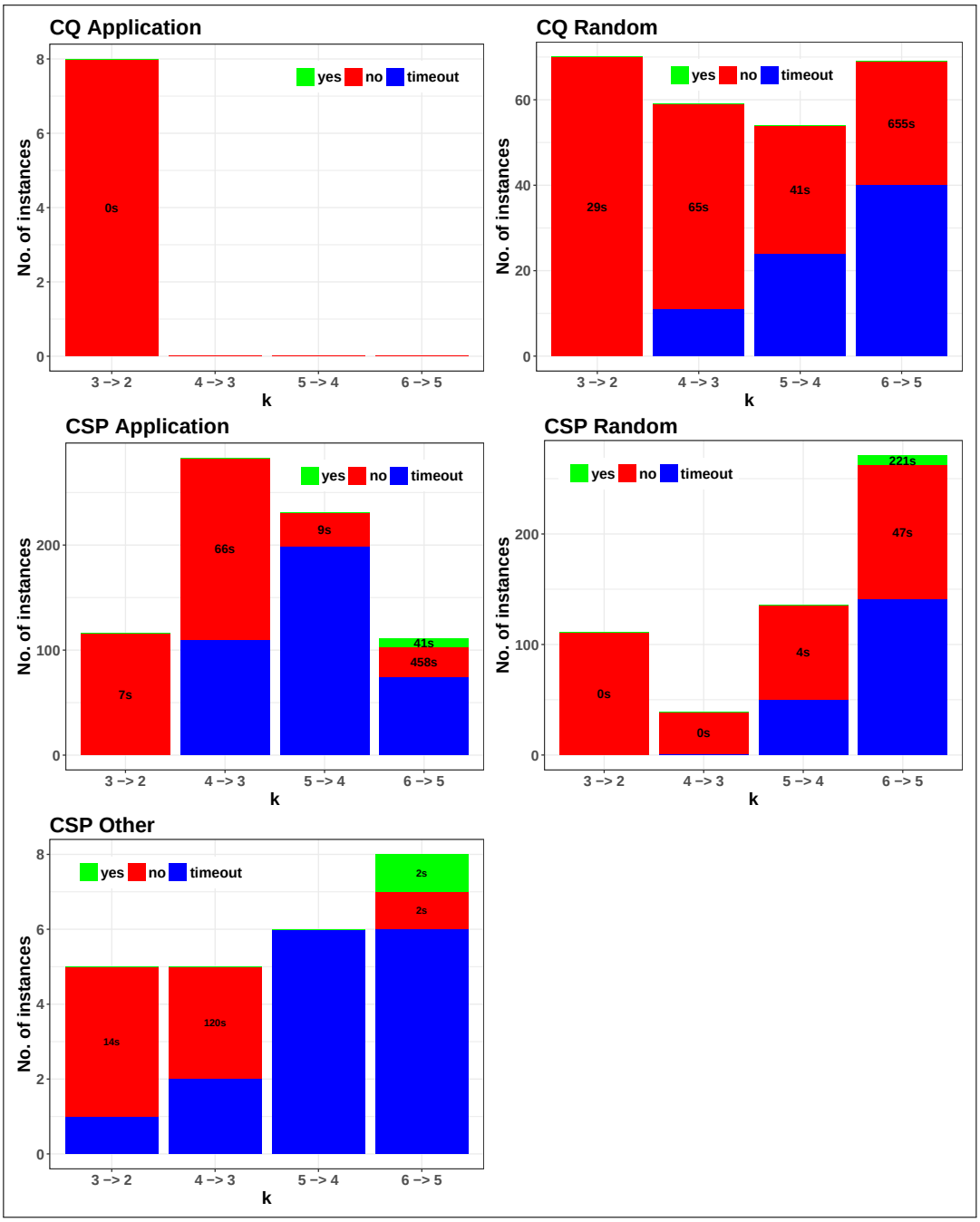

Figure 5.6: GHW analysis (labels are average runtimes in s)

is interesting to note that in the vast majority of cases, no improvement of the width is possible when we switch from *hw* to *ghw*: in 98% of the solved cases and 57% of all instances with $hw \leq 6$, *hw* and *ghw* have identical values. Actually, we think that the high percentage of the *solved cases* gives a more realistic picture than the percentage of *all cases* for the following reason: our algorithms (in particular, the "global" and "local" computations) need particularly long time for negative instances. This is due to the fact that in a negative case, "all" possible choices of λ-labels for a node u in the GHD have to be tested before we can be sure that no GHD of H (or, equivalently, no HD of H') of desired width exists. Hence, it seems plausible that the timeouts are mainly due to negative instances. This also explains why our new `BalSep` algorithm (see Algorithm 5.1), which is particularly well suited for negative instances, has the least number of timeouts.

Overall there is no big difference between the different hypergraph categories. The only hypergraphs that have a smaller generalized hypertree width are instances of the CSP categories. It also appears that the "CSP Random" instances are easier to solve than the "CSP Application" instances (66% solved vs. 48% solved, resp.).

We conclude this section with a final observation: in Figure 5.3 and Table 5.2, we had many cases, for which only some upper bound k on the *hw* could be determined, namely those cases, where the attempt to solve Check(HD, k) yields a yes-answer and the attempt to solve Check(HD, $k-1$) gives a timeout. In several such cases, we could get (with the balanced separator approach) a no-answer for the Check(GHD, $k-1$) problem, which implicitly gives a no-answer for the problem Check(HD, $k-1$). In this way, the alternative approach to the *ghw*-computation is also profitable for the *hw*-computation: for 827 instances with $hw \leq 6$, we were not able to determine the exact hypertree width. Using our new *ghw*-algorithm, we closed this gap for 297 instances; for these instances $hw = ghw$ holds.

To sum up, we now have a total of 1,778 (58%) instances for which we determined the exact hypertree width and a total of 1,406 instances (46%) for which we determined the exact generalized hypertree width. Out of these, 1,390 instances had identical values for *hw* and *ghw*. In 16 cases, we found an improvement of the width by 1 when moving from *hw* to *ghw*, namely from $hw = 6$ to $ghw = 5$. In 2 further cases, we could show $hw \leq 6$ and $ghw \leq 5$, but the attempt to check $hw = 5$ or $ghw = 4$ led to a timeout. Hence, in response to *Research Questions 9 and 10*, *hw* is equal to *ghw* in 45% of the cases if we consider all instances and in 60% of the cases (1,390 of 2,308) with small width ($hw \leq 6$). However, if we consider the fully solved cases (i.e., where we have the precise value of *hw* and *ghw*), then *hw* and *ghw* coincide in 99% of the cases (1,390 of 1,406).

CQ Application

$hw \to ghw$	yes	no	timeout
$3 \to 2$	0	8 (0)	0

CQ Random

$hw \to ghw$	yes	no	timeout
$3 \to 2$	0	70 (29)	0
$4 \to 3$	0	48 (65)	11
$5 \to 4$	0	30 (41)	24
$6 \to 5$	0	29 (655)	40

CSP Application

$hw \to ghw$	yes	no	timeout
$3 \to 2$	0	116 (7)	0
$4 \to 3$	0	173 (66)	110
$5 \to 4$	0	32 (9)	199
$6 \to 5$	8 (41)	29 (458)	74

CSP Random

$hw \to ghw$	yes	no	timeout
$3 \to 2$	0	111 (0)	0
$4 \to 3$	0	38 (0)	1
$5 \to 4$	0	86 (4)	50
$6 \to 5$	9 (221)	121 (47)	141

CSP Other

$hw \to ghw$	yes	no	timeout
$3 \to 2$	0	4 (14)	1
$4 \to 3$	0	3 (120)	2
$5 \to 4$	0	0	6
$6 \to 5$	1 (2)	1 (2)	6

Table 5.5: GHW of instances with average runtime in s

5.5 FHW Computation

Unfortunately, for the Check(FHD,k) problem the theoretical tractability results can not be easily translated into practical algorithms. There is however one class of hypergraphs for which this is possible. In this section we will focus on classes of hypergraphs having the Bounded Rank Property (BRP) (or, Bounded Arity Property).

Definition 5.1. We say that a hypergraph H has the *r-bounded rank property (r-BRP)* if $arity(H) \leq r$ holds.

Let $\mathscr{C}$ be a class of hypergraphs. We say that $\mathscr{C}$ has the *bounded rank property (BRP)* if there exists a constant r such that every hypergraph H in $\mathscr{C}$ has the r-BRP. ◁

For every hypergraph whose arity is bounded by some constant r the following lemma is immediate:

Lemma 5.1. *Let H be a hypergraph whose arity is bounded by some constant r. Then for every (fractional) edge weight function γ for H satisfying $weight(\gamma) = k$, the property $B(\gamma) \leq r \cdot k$ holds.*

Algorithm 5.2: k, r-rank-frac-decomp

input : Hypergraph H.
output : An FHD $\mathcal{F}$ of H with width $\leq k$, if it exists, "Reject", otherwise.

Function k, r-fdecomp (C_R: *Vertex-Set,* B_R: *Vertex-Set*)
  Guess a set $W \subseteq V(H)$ of $r \cdot k$ elements at most; /* **(1) Guess** */
  begin /* **(2) Check** */
    Check if $\exists\gamma$ with $W \subseteq B(\gamma)$ and $weight(\gamma) \leq k$; /* (2.a) */
    Check if $\forall e \in edges(C_R)\colon e \cap B_R \subseteq W$; /* (2.b) */
    Check if $W \cap C_R \neq \emptyset$; /* (2.c) */
  end
  if *one of these checks fails* **then Halt** and **Reject**; /* **(3)** */
  Let $\mathcal{C} := \{C \subseteq V(H) \mid C$ is a $[W]$-component and $C \subseteq C_r\}$;
  foreach $C \in \mathcal{C}$ **do** /* **(4)** */
    if k, r-fdecomp *(C, W) returns* **Reject then**
      Halt and **Reject**
    end
  end
  return Accept;
begin /* **Main** */
  return k, r-fdecomp $(V(H), \emptyset)$
end

We can use the previous lemma to provide an algorithm for solving the Check(FHD,k) for hypergraph classes having the BRP.

Theorem 5.2. *For every hypergraph class $\mathscr{C}$ that has bounded rank, and for every constant $k \geq 1$, the* Check*(FHD,k) problem is tractable, i.e., given a hypergraph $H \in \mathscr{C}$, it is feasible in polynomial time to check $fhw \leq k$ and, if so, to compute an FHD of width k of H.*

Proof Sketch. The algorithm k, r-rank-frac-decomp that checks for a given hypergraph H with arity r if $fhw \leq k$ is given in Algorithm 5.2. The algorithm is based on the alternating algorithm k-decomp solving Check(HD,k) problem presented in [51] and on the alternating algorithm (k, ϵ, c)-frac-decomp given in Algorithm 4.3. The idea of our modified algorithm is, to guess instead of a set S of k edges (and a set of W_s vertices in Algorithm 4.3), a set W of $k \cdot r$ vertices. This set of vertices corresponds to a B_u set of a possible FHD of H.

The correctness of the algorithm follows from Lemma 5.1, i.e. for every FHD $\mathcal{F} = \langle T, (B_u)_{u \in T}, (\gamma_u)_{u \in T} \rangle$ of width $\leq k$ it holds that in each node $u \in T$, it is the case that

$|B(\gamma_u)| \leq r \cdot k$ and, since $B_u \subseteq B(\gamma_u)$, also $|B_u| \leq r \cdot k$. Hence, our algorithm finds every FHD $\mathcal{F}$ of width $\leq k$ and, by the same ideas as in the proof of Lemma 4.25, every computation tree can be turned into an FHD $\mathcal{F}$.

It remains to establish PTime membership of our algorithm. Here the arguments follow closely those of the proof of Lemma 4.26. There, we argued that we can easily carry over the corresponding tractability result from Lemma 5.15 in [51]. For this we first show that all our data structures fit into logspace. The sets B_R and W are just sets of constantly many vertices, and hence storable in logspace by using indices of vertices. The variables C_R and C of Algorithm 5.2 correspond to the exact same variables of the alternating algorithm in [51]. Hence, by using the same arguments, they fit into logspace as well. The rest of the tractability proof can then be easily carried over from the proof of Lemma 4.26 and Lemma 5.15 in [51]. Especially remember, that for the checks in Step 2 of k, r-`rank-frac-decomp`, we also have to solve a linear program, which can also be done in PTime. Hence, the alternating algorithm k, r-`rank-frac-decomp` is in PTime. □

Empirical Analysis. For our empirical analysis we have implemented the k, r-`rank-frac-decomp` algorithm. From the HyperBench benchmark we have selected all hypergraphs with arity ≤ 5 and $hw \leq 6$. For all instances we tried to improve the width by 0.5, i.e. for hypergraphs with, for example, $hw = 4$ we called the k, r-`rank-frac-decomp` algorithm with $k = 3.5$. The results are given in Table 5.7 with their graphical representation given in Figure 5.7.

In total we selected 1,111 hypergraphs for our analysis. Out of these instances 265 (24%) hypergraphs have an *fhw* smaller than the *hw*. For 115 (10%) instances the k, r-`rank-frac-decomp` algorithm was not able to find a FHD with width $hw - 0.5$. Hence, we can conclude that for these "no"-instances the real $fhw \in (hw - 0.5, hw]$. For the remaining instances (731, i.e. 66%) the k, r-`rank-frac-decomp` algorithm did not terminate.

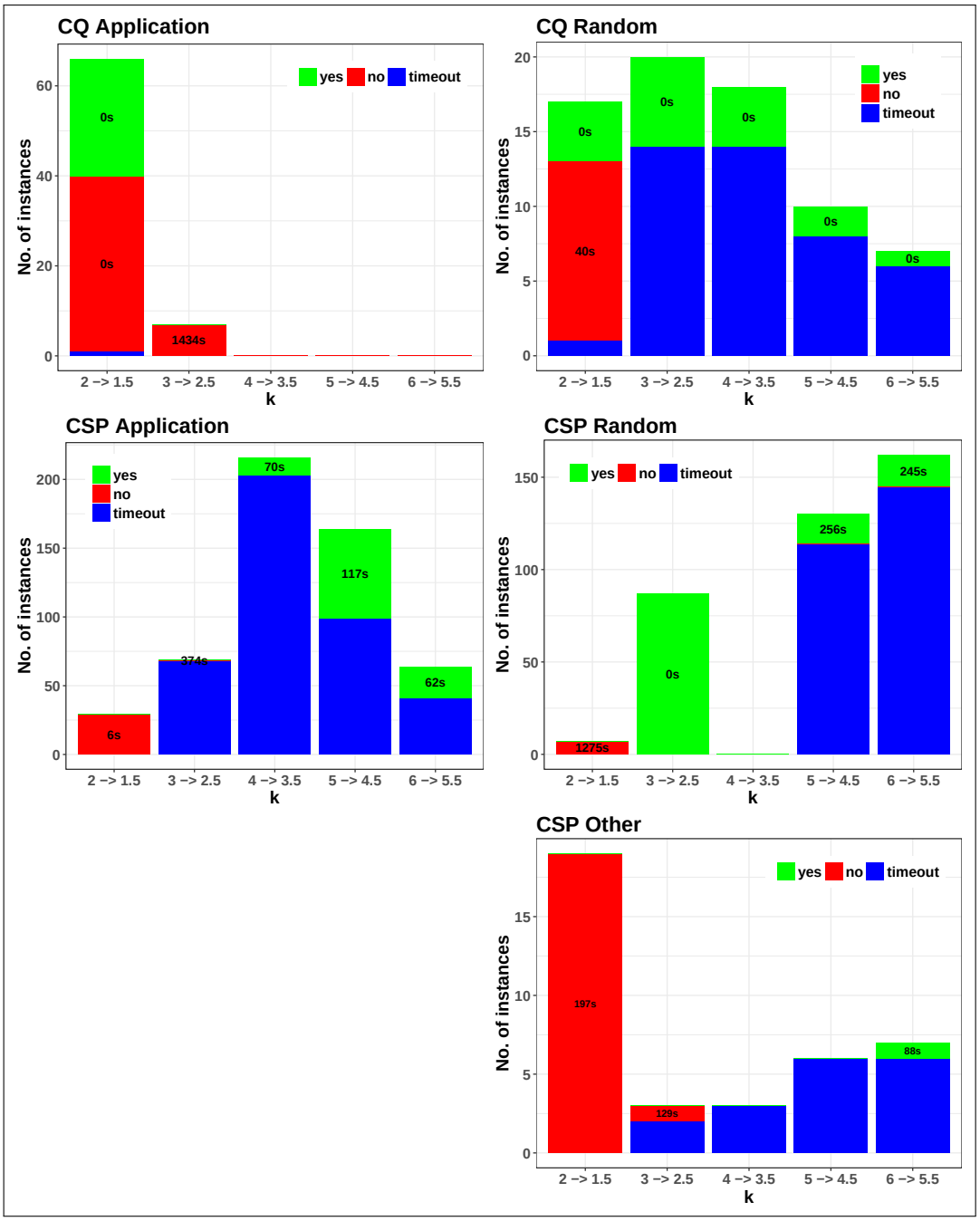

Figure 5.7: Results of FHW computation with k, r-`rank-frac-decomp` (labels are average runtimes in s)

CQ Application

hw → *fhw*	yes	no	timeout
2 → 1.5	26 (0)	39 (0)	1
3 → 2.5	0	7 (1434)	0

CSP Application

hw → *ghw*	yes	no	timeout
2 → 1.5	0	29 (6)	0
3 → 2.5	0	1 (374)	68
4 → 3.5	13 (70)	0	203
5 → 4.5	65 (117)	0	99
6 → 5.5	23 (62)	0	41

CQ Random

hw → *ghw*	yes	no	timeout
2 → 1.5	4 (0)	12 (40)	1
3 → 2.5	6 (0)	0	14
4 → 3.5	4 (0)	0	14
5 → 4.5	2 (0)	0	8
6 → 5.5	1 (0)	0	6

CSP Random

hw → *ghw*	yes	no	timeout
2 → 1.5	0	7 (1275)	0
3 → 2.5	87 (0)	0	0
4 → 3.5	0	0	0
5 → 4.5	16 (256)	0	114
6 → 5.5	17 (245)	0	145

CSP Other

hw → *ghw*	yes	no	timeout
2 → 1.5	0	19 (197)	0
3 → 2.5	0	1 (129)	2
4 → 3.5	0	0	3
5 → 4.5	0	0	6
6 → 5.5	1 (88)	0	6

Table 5.6: FHW of instances with average runtime in s

5.6 Comparison of Different Width Measures

We conclude this chapter with a summary of the results and a comparison of the different width notions, in particular the difference of *hw* and *fhw*. We will combine the results presented here, with the results of [36, 72]. There we presented two algorithms for the Check(FHD,k) problem based on fractionally improved hypertree decompositions. A *fractionally improved hypertree decomposition* is computed from a hypertree decomposition where in each node the integral edge cover is substituted by a fractional edge cover. Based on this idea two algorithms were presented [36]:

- The simplest way to obtain a fractionally improved (G)HD is to take either a GHD or HD as input and compute a fractionally improved (G)HD. To this end, the algorithm visits each node u of a given GHD or HD and computes an optimal fractional edge cover γ_u for the set B_u of vertices. This algorithm is simple and computationally inexpensive, provided that one can start off with a GHD or HD that

was computed before. Clearly, this approach is rather naive and the dependence on a concrete HD is unsatisfactory. Therefore the more sophisticated second algorithm is described next.

- The algorithm `FracImproveHD` has as input a hypergraph H and numbers $k, k' \geq 1$, where k is an upper bound on the hw and k' the desired fractionally improved hw. We search for an FHD $\mathcal{D}'$ with $\mathcal{D}' = \mathit{SimpleImproveHD}(\mathcal{D})$ for some HD $\mathcal{D}$ of H with $\mathit{width}(\mathcal{D}) \leq k$ and $\mathit{width}(\mathcal{D}') \leq k'$. In other words, this algorithm searches for the best fractionally improved HD over all HDs of width $\leq k$. Hence, the result is independent of any concrete HD.

The results from the previous sections together with the results from [36] are gathered and can be retrieved on our HyperBench website (see Section 5.1). From these results we derive for each hypergraph upper and lower bounds for the hw, ghw, and fhw. For this we also make use of the well known inequality: $fhw \leq ghw \leq hw$, i.e. an upper bound for the hw automatically gives us an upper bound for the ghw and for the fhw; a lower bound for the fhw automatically gives us a lower bound for ghw and for the hw.

Remark. The way our experimental analyses are conducted in the previous sections, we often have that solving the Check(HD,$k-1$) (or Check(GHD,$k-1$), resp.) problem did not terminate, whereas solving Check(HD,k) (or Check(GHD,k), resp.) problem terminated. For such cases we have that $hw \leq k$ ($ghw \leq k$, resp.), but not that $hw > k-1$ ($ghw > k-1$, resp.). Hence, we are often not able to match lower and upper bounds. In numbers, for all 3,070 hypergraphs the lower bounds do not match the upper bounds for 1,408 (46%) instances in case of hw, 1,544 (51%) instances in case of ghw, and for 2,581 (84%) instances in case of fhw. For fhw the remaining 489 instances are those, which are acyclic, i.e. they have $fhw = 1$.

We therefore compare in the upcoming paragraphs computed and derived upper bounds of hw, ghw, and fhw. In order to make these paragraphs more accessible for the reader, we will simply talk about hw, ghw, and fhw, neglecting the fact that the real hw, ghw, and fhw might be smaller. If this is not the case (i.e. matching lower bounds exist) we will explicitly state it in the text.

Let us first compare hw with ghw, which is visualized in Figure 5.8. The charts can be read as follows: on the x-axis we have the computed hw in increasing order and on the y-axis the computed ghw in increasing order. A dot is drawn if there is an instance that has hw equal to the value on the x-axis and ghw equal to the value on the y-axis. The colour of the dot corresponds to the hypergraph category and the size to the number of instances having the corresponding combination of hw and ghw. The top two charts show all instances, the bottom two charts only instances where $hw \leq 6$. The left two charts show instances in non-random hypergraph categories, i.e. the "CQ Application", the "CSP Application", and the "CSP Other" category. The right two charts show instances in random hypergraph categories, i.e. the "CQ Random" and the "CSP Random" category.

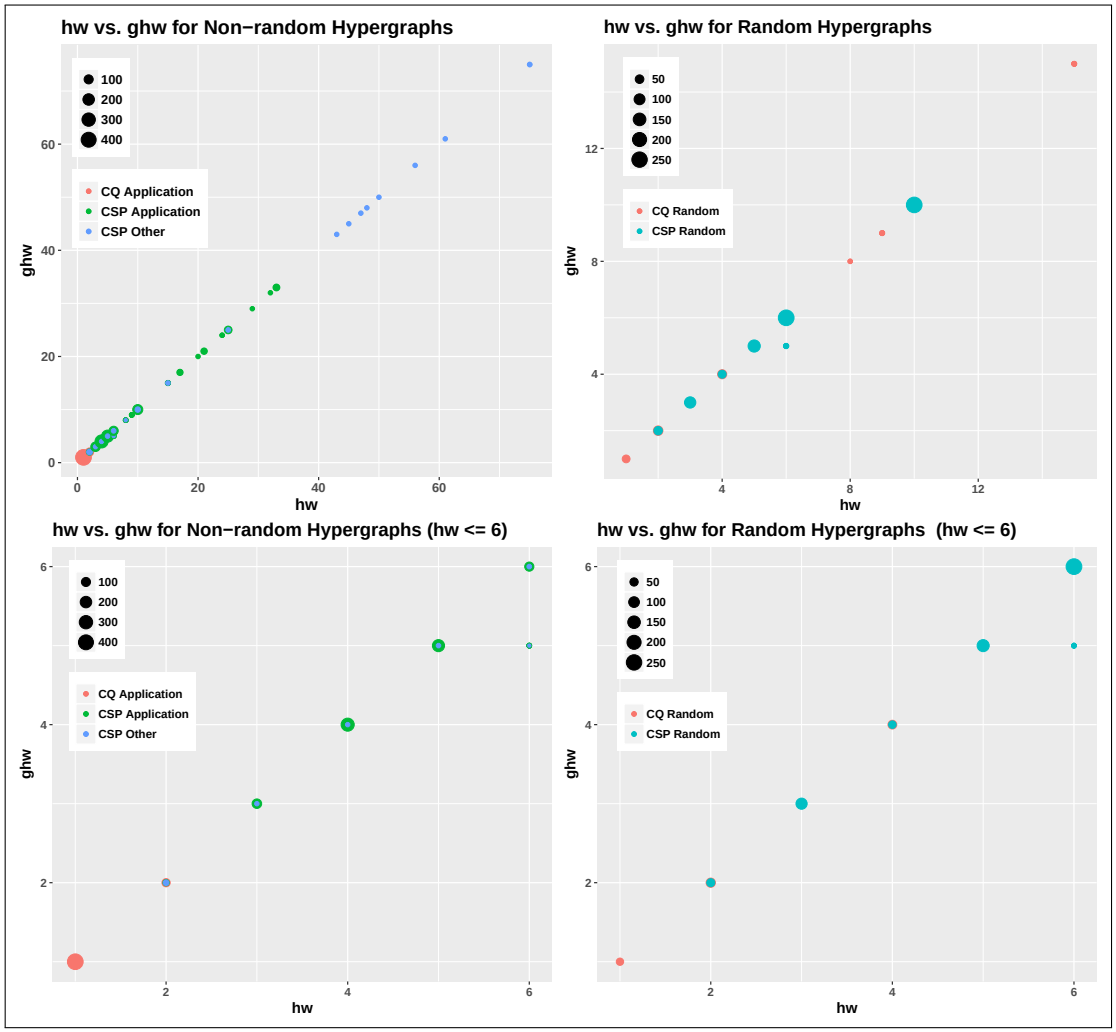

Figure 5.8: Comparison of *hw* vs. *ghw*

First notice that the CQ instances have lower *hw* and *ghw* than the CSP instances (in both non-random and random categories). But altogether, the differences between *hw* and *ghw* are marginal. There were only 16 instances were the *ghw* is smaller than the *hw*. All these instances are from the CSP categories and have $hw \leq 6$ and $ghw = 5$. Unfortunately, we were not able to solve the Check(HD,5) problem for these instances. This means, that it is still possible that these instances have $hw = ghw = 5$. Before we look at the more interesting *hw* vs. *fhw* comparison, note that these 16 instances have not been solved with any of the *fhw* algorithms.

The comparison of *hw* against *fhw* is shown in Figure 5.9. The top left chart shows that hypergraphs in the "CSP Other" category have the greatest difference in *hw* and *fhw* (by up to 20). For the other non-random categories, "CQ Application" and "CSP Application"

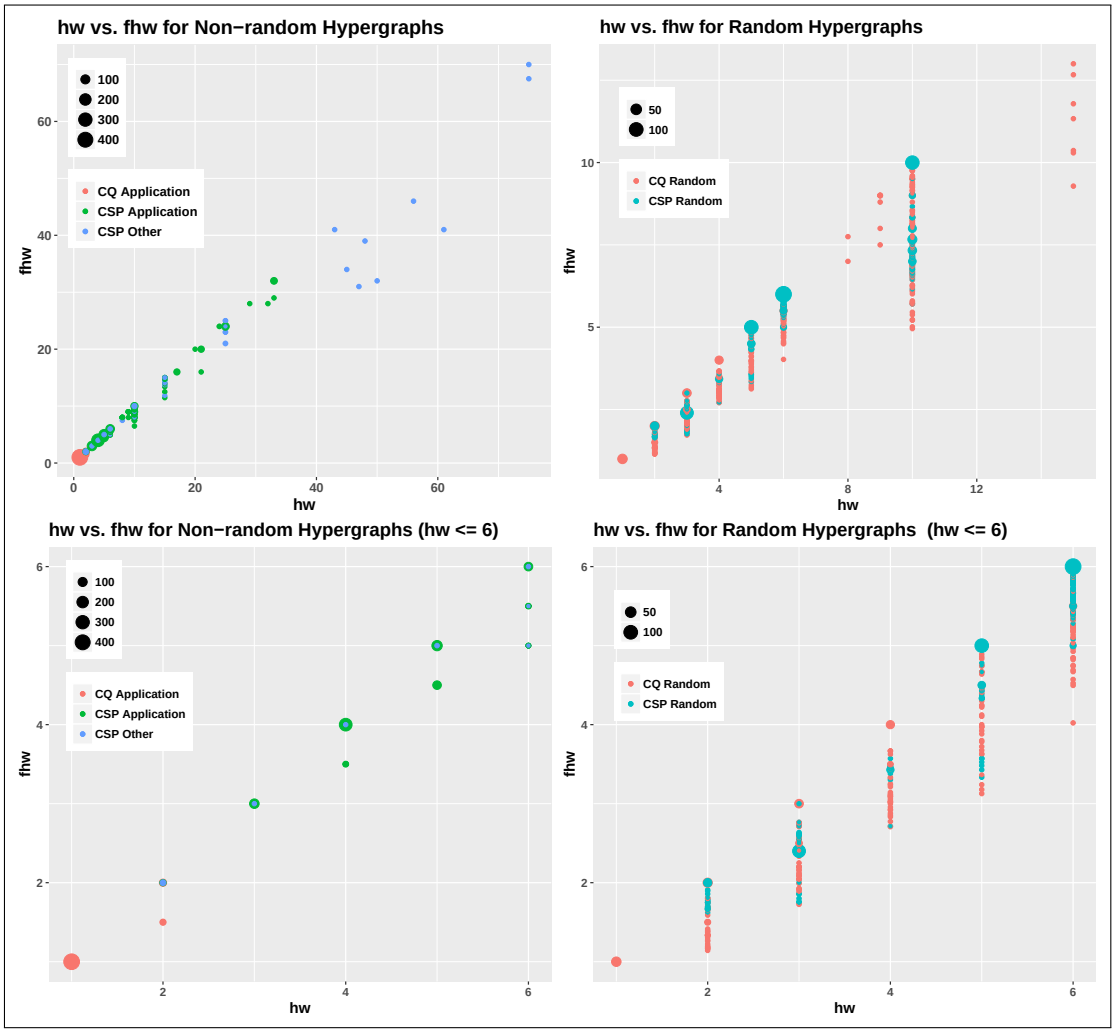

Figure 5.9: Comparison of *hw* vs. *fhw*

the improvement by our *fhw* algorithms is at most 1 for hypergraphs with $hw \leq 6$. Hence, for non-random hypergraphs of low width, computing an FHD instead of an HD might not justify the exponential blow-up in computation. Also notice the CQs where $hw = 2$ and $fhw \leq 1.5$. A closer inspection of these CQs reveals that they stem from querying a triangle, i.e. for example the query $A(X, Y) \wedge B(Y, Z) \wedge C(Z, X)$. In total there are 430 (25%) out of 1,707 non-random instances where it holds that $fhw < hw$. The charts for random queries show a completely different picture. The largest improvement by computing an FHD instead of an HD for random queries is just 5.7. However, in total there are 841 (62%) out of 1,363 instances that have lower *fhw* than *hw*. Counting both together we have 1,271 (41%) instances out of all 3,070 where the *fhw* is lower than the *hw*.

CHAPTER 6

Conclusion

6.1 Summary

In this work, we have settled the complexity of deciding $fhw(H) \leq k$ for fixed constant $k \geq 2$ and $ghw(H) \leq k$ for $k = 2$ by proving the NP-completeness of both problems. This gives negative answers to two open problems. On the positive side, we have identified rather mild restrictions such as the BDP (i.e., the bounded degree property), BIP (i.e., the bounded intersection property), LogBIP, BMIP (i.e., the bounded multi-intersection property), and LogBMIP, which give rise to a PTime algorithm for the Check(GHD, k) problem. Moreover, we have shown that the BDP ensures tractability also of the Check(FHD, k) problem. For the BIP, we have shown that an arbitrarily close approximation of the *fhw* in polynomial time exists. In case of the BMIP, we have proposed a polynomial-time algorithm for approximating the *fhw* up to a logarithmic factor. As our empirical analyses show, these restrictions are very well-suited for instances of CSPs and, even more so, of CQs, i.e. hypergraphs stemming from such instances have low degree, *iwidth*(,) etc.

For these empirical analyses we have gathered more than 3,000 hypergraphs stemming from real-world and randomly generated CQs and CSPs. We have seen that these instances indeed have low *hw*, which is important, since all algorithms answering CQs and solving CSPs using HDs, GHDs, or FHDs depend exponentially on the *hw*, *ghw*, or *fhw*, resp. The reduction of the *ghw*-computation problem to the *hw*-computation problem in case of low intersection width turned out to be more problematical than the theoretical tractability results had suggested. Even the improvement by "local" computation of the additional subedges did not help much. However, with the help of balanced separators we were able to find many instances for which it holds that $ghw > k$ for some k. With these insights at hand, we have found out, that, most surprisingly, the discrepancy between *hw* and *ghw* is much lower than expected. Theoretically, only the upper bound $hw \geq 3 \cdot ghw + 1$ is known. However, in practice, when considering

hypergraphs of $hw \leq 6$, we could show that in 53% of all cases, hw and ghw are simply identical. Moreover, in all cases when one of our implementations of ghw-computation terminated on instances with $hw \leq 5$, we got identical values for hw and ghw. For solving the Check(FHD,k) problem we have developed a novel algorithm that has a runtime polynomial for instances of bounded arity. With this algorithm we have shown that for instances with arity ≤ 5 in 265 cases it holds that $fhw < hw$.

6.2 Future Work

We plan to explore several further issues regarding the computation and approximation of the generalized and fractional hypertree width. We find the following tasks particularly urgent and/or rewarding.

- Does the special condition defined by Grohe and Marx [58] lead to tractable recognizability also for FHDs, i.e., in case we define "$sc\text{-}fhw(H)$" as the smallest width an FHD of H satisfying the special condition, can $sc\text{-}fhw(H) \leq k$ be recognized efficiently?

- Our tractability result in Section 5 for the Check(FHD, k) problem is weaker than for Check(GHD, k). In particular, for the BIP and BMIP, we have only obtained efficient approximations of the fhw. It is open if the BIP or even the BMIP suffices to ensure tractability of Check(FHD, k). And if not, we should at least search for a better approximation of the fhw in case of the BMIP. Or can non-approximability results be obtained under reasonable complexity-theoretic assumptions?

- So far, we have only implemented the ghw-computation in case of low intersection width. In Section 4.2, tractability of the Check(GHD, k) problem was also proved for the more relaxed bounded multi-intersection width. Our empirical results in Figure 5.4 show that, apart from the random CQs and random CSPs, the 3-multi-intersection is ≤ 2 in almost all cases. It seems therefore worthwhile to implement and test also the BMIP-algorithm.

- Our new approach to ghw-computation via "balanced separators" proved quite effective in our experiments. However, further theoretical underpinning of this approach is missing. The empirical results obtained via balanced separators suggest that the number of balanced separators is often drastically smaller than the number of arbitrary separators. We want to determine a realistic upper bound on the number of balanced separators in terms of n (the number of edges) and k (an upper bound on the width).

- Finally, we want to further extend the HyperBench benchmark and tool in several directions. We will thus incorporate further implementations of decomposition algorithms from the literature such as the GHD- and FHD computation in [76] or the polynomial-time FHD computation for hypergraphs of bounded degree given in

Section 4.3. Moreover, we will continue to fill in hypergraphs from further sources of CSPs and CQs. For instance, in [1, 23, 43, 44] a collection of CQs for the experimental evaluations in those papers is mentioned. We will invite the authors to disclose these CQs and incorporate them into the HyperBench benchmark.

- Very recently, a new, huge, publicly available query log has been reported in [73]. It contains over 200 million SPARQL queries on Wikidata. In the paper, the anonymisation and publication of the query logs is mentioned as future work. However, on their web site, the authors have meanwhile made these queries available. At first glance, these queries seem to display a similar behaviour as the SPARQL queries collected by Bonifatti et al. [21]: there is a big number of single-atom queries and again, the vast majority of the queries is acyclic. A detailed analysis of the query log in the style of [21] constitutes an important goal for future research.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Christopher R. Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun, and Christopher Ré. Emptyheaded: A relational engine for graph processing. *ACM Trans. Database Syst.*, 42(4):20:1–20:44, 2017.

[2] Christopher R. Aberger, Susan Tu, Kunle Olukotun, and Christopher Ré. Emptyheaded: A relational engine for graph processing. In *Proc. of SIGMOD 2016*, pages 431–446. ACM, 2016.

[3] Christopher R. Aberger, Susan Tu, Kunle Olukotun, and Christopher Ré. Old techniques for new join algorithms: A case study in RDF processing. *CoRR*, abs/1602.03557, 2016.

[4] Isolde Adler. Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory*, 47(4):275–296, 2004.

[5] Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007.

[6] Kamal Amroun, Zineb Habbas, and Wassila Aggoune-Mtalaa. A compressed generalized hypertree decomposition-based solving technique for non-binary constraint satisfaction problems. *AI Commun.*, 29(2):371–392, 2016.

[7] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and implementation of the LogicBlox system. In *Proc. of SIGMOD 2015*, pages 1371–1382. ACM, 2015.

[8] Patricia C. Arocena, Boris Glavic, Radu Ciucanu, and Renée J. Miller. The iBench integration metadata generator. *PVLDB*, 9(3):108–119, November 2015.

[9] Patrick Assouad. Densité et dimension. *Annales de l'Institut Fourier*, 33(3):233–282, 1983.

[10] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013.

[11] Gilles Audemard, Frédéric Boussemart, Christoph Lecoutre, and Cédric Piette. XCSP3: an XML-based format designed to represent combinatorial constrained problems. `http://xcsp.org`, 2016.

[12] Giorgio Ausiello. *Complexity and approximation: combinatorial optimization problems and their approximability properties*. Springer, 1999.

[13] Nurzhan Bakibayev, Tomás Kociský, Dan Olteanu, and Jakub Závodný. Aggregation and ordering in factorised databases. *PVLDB*, 6(14):1990–2001, 2013.

[14] Catriel Beeri, Ronald Fagin, David Maier, Alberto O. Mendelzon, Jeffrey D. Ullman, and Mihalis Yannakakis. Properties of acyclic database schemes. In *Proc. STOC 1981*, pages 355–362, 1981.

[15] Michael Benedikt. CQ benchmarks, 2017. Personal Communication.

[16] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the chase. In *Proc. PODS 2017*, pages 37–52. ACM, 2017.

[17] Jeremias Berg, Neha Lodha, Matti Järvisalo, and Stefan Szeider. Maxsat benchmarks based on determining generalized hypertree-width. *MaxSAT Evaluation 2017*, page 22, 2017.

[18] Philip A. Bernstein and Nathan Goodman. Power of natural semijoins. *SIAM J. Comput.*, 10(4):751–771, 1981.

[19] Hans Bodlaender. TreewidthLIB: A benchmark for algorithms for treewidth and related graph problems. `http://www.cs.uu.nl/research/projects/treewidthlib/`.

[20] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[21] Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *PVLDB*, 11(2):149–161, 2017.

[22] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, Dec 1995.

[23] Nofar Carmeli, Batya Kenig, and Benny Kimelfeld. Efficiently enumerating minimal triangulations. In *Proc. PODS 2017*, pages 273–287. ACM, 2017.

[24] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of STOC 1977*, pages 77–90. ACM, 1977.

[25] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

[26] Fan R. K. Chung, Zoltan Fueredi, MR Garey, and Ronald L. Graham. On the fractional covering number of hypergraphs. *SIAM journal on discrete mathematics*, 1(1):45–49, 1988.

[27] David A. Cohen, Peter Jeavons, and Marc Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *J. Comput. Syst. Sci.*, 74(5):721–743, 2008.

[28] Rina Dechter. *Constraint Processing (.* The Morgan Kaufmann Series in Artificial Intelligence). 2003.

[29] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artif. Intell.*, 38(3):353–366, 1989.

[30] Artan Dermaku, Tobias Ganzow, Georg Gottlob, Benjamin J. McMahan, Nysret Musliu, and Marko Samer. Heuristic methods for hypertree decomposition. In *Proc. of MICAI 2008*, pages 1–11, 2008.

[31] Guo-Li Ding, Paul Seymour, and Peter Winkler. Bounding the vertex cover number of a hypergraph. *Combinatorica*, 14(1):23–34, 1994.

[32] Pierre Duchet. Hypergraphs. In *Handbook of combinatorics (vol. 1)*, pages 381–432. MIT Press, 1996.

[33] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.

[34] Uriel Feige and Mohammad Mahdian. Finding small balanced separators. In *Proc. STOC 2006*, pages 375–384. ACM, 2006.

[35] Johannes K. Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. An smt approach to fractional hypertree width. In *Proc. CP 2018*, volume 11008 of *LNCS*, pages 109–127. Springer, 2018.

[36] Wolfgang Fischl, Georg Gottlob, Davide M. Longo, and Reinhard Pichler. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. In *Proc. of PODS 2019 (to appear)*, 2019.

[37] Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases (full version). *CoRR*, abs/1611.01090, 2016.

[38] Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In *Proc. PODS 2018*, pages 17–32. ACM, 2018.

[39] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4):755–761, 1985.

[40] Zoltán Füredi. Matchings and covers in hypergraphs. *Graphs and Combinatorics*, 4(1):115–206, 1988.

[41] Jared L Gearhart, Kristin L Adair, Richard J Detry, Justin D Durfee, Katherine A Jones, and Nathaniel Martin. Comparison of open-source linear programming solvers. *Tech. Rep. SAND2013-8847*, 2013.

[42] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. Mapping and cleaning. In *Proc. ICDE 2014*, pages 232–243. IEEE, 2014.

[43] Lucantonio Ghionna, Luigi Granata, Gianluigi Greco, and Francesco Scarcello. Hypertree decompositions for query optimization. In *Proc. ICDE 2007*, pages 36–45. IEEE Computer Society, 2007.

[44] Lucantonio Ghionna, Gianluigi Greco, and Francesco Scarcello. H-DB: a hybrid quantitative-structural sql optimizer. In *Proc. CIKM 2011*, pages 2573–2576. ACM, 2011.

[45] Nathan Goodman and Oded Shmueli. Tree queries: A simple class of relational queries. *ACM Trans. Database Syst.*, 7(4):653–677, 1982.

[46] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: Questions and answers. In *Proc. of PODS 2016*, pages 57–74. ACM, 2016.

[47] Georg Gottlob, Nicola Leone, and Francesco Scarcello. On tractable queries and constraints. In *Proc. of DEXA 1999*, pages 1–15, 1999.

[48] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

[49] Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

[50] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: A survey. In *Proc. of MFCS 2001*, pages 37–57, 2001.

[51] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[52] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci.*, 66(4):775–808, 2003.

[53] Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM*, 56(6):30:1–30:32, 2009.

[54] Georg Gottlob and Marko Samer. A backtracking-based algorithm for hypertree decomposition. *ACM Journal of Experimental Algorithmics*, 13, 2008.

[55] Gianluigi Greco and Francesco Scarcello. The power of tree projections: local consistency, greedy algorithms, and larger islands of tractability. In *Proc. of PODS 2010*, pages 327–338, 2010.

[56] Gianluigi Greco and Francesco Scarcello. Greedy strategies and larger islands of tractability for conjunctive queries and constraint satisfaction problems. *Inf. Comput.*, 252:201–220, 2017.

[57] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *Proc. of SODA 2006*, pages 289–298. ACM Press, 2006.

[58] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014.

[59] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[60] Marc Gyssens, Peter Jeavons, and David A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artif. Intell.*, 66(1):57–89, 1994.

[61] Zineb Habbas, Kamal Amroun, and Daniel Singer. A forward-checking algorithm based on a generalised hypertree decomposition for solving non-binary constraint satisfaction problems. *J. Exp. Theor. Artif. Intell.*, 27(5):649–671, 2015.

[62] Shrainik Jain, Dominik Moritz, Daniel Halperin, Bill Howe, and Ed Lazowska. Sqlshare: Results from a multi-year sql-as-a-service experiment. In *Proc. SIGMOD 2016*, pages 281–293, New York, NY, USA, 2016. ACM.

[63] Shant Karakashian, Robert J. Woodward, and Berthe Y. Choueiry. Reformulating r(*, m)c with tree decomposition. In *SARA*. AAAI, 2011.

[64] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst-case and beyond. In *Proc. of PODS 2015*, pages 213–228. ACM, 2015.

[65] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *Proc. of PODS 2016*, pages 13–28. ACM, 2016.

[66] Phokion G. Kolaitis. Constraint satisfaction, databases, and logic. In *Proc. of IJCAI 2003*, pages 1587–1595, 2003.

[67] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

[68] Mohammed Lalou, Zineb Habbas, and Kamal Amroun. Solving hypertree structured CSP: sequential and parallel approaches. In *Proc. RCRA@AI*IA 2009*, 2009.

[69] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *PVLDB*, 9(3):204–215, November 2015.

[70] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *The VLDB Journal*, Sep 2017.

[71] Nicola Leone, Alfredo Mazzitelli, and Francesco Scarcello. Cost-based query decompositions. In *Proc. of SEBD 2002*, pages 390–403, 2002.

[72] Davide M. Longo. Improving Hypertree Decompositions using Fractional Edge Covers. Master's thesis, Universitá della Calabria, Italy, 2018.

[73] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In *Proc. ISWC 2018*, pages 376–394. Springer, 2018.

[74] Dániel Marx. Approximating fractional hypertree width. *ACM Trans. Algorithms*, 6(2):29:1–29:17, 2010.

[75] Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013.

[76] Lukas Moll, Siamak Tazari, and Marc Thurley. Computing hypergraph width measures exactly. *Inf. Process. Lett.*, 112(6):238–242, 2012.

[77] Nysret Musliu. Generation of tree decompositions by iterated local search. In *Proc. of EvoCOP 2007*, pages 130–141, 2007.

[78] Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, 2015.

[79] Adam Perelman and Christopher Ré. DunceCap: Compiling worst-case optimal query plans. In *Proc. SIGMOD 2015*, pages 2075–2076. ACM, 2015.

[80] François Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In *Proc. SWIM 2011*, page 7. ACM, 2011.

[81] Rachel Pottinger and Alon Halevy. MiniCon: A scalable algorithm for answering queries using views. *The VLDB Journal*, 10(2-3):182–198, September 2001.

[82] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[83] Yehoshua Sagiv and Oded Shmueli. Solving queries by tree projections. *ACM Trans. Database Syst.*, 18(3):487–511, 1993.

[84] Marko Samer. Hypertree-decomposition via branch-decomposition. In *Proc. of IJCAI 2005*, pages 1535–1536, 2005.

[85] Norbert Sauer. On the density of families of sets. *J. Combinatorial Theory (A)*, 13(1):145–147, 1972.

[86] Francesco Scarcello, Gianluigi Greco, and Nicola Leone. Weighted hypertree decompositions and optimal query plans. *J. Comput. Syst. Sci.*, 73(3), 2007.

[87] Werner Schafhauser. New heuristic methods for tree decompositions and generalized hypertree decompositions, 2006. Master Thesis, TU Wien.

[88] Aaron Schild and Christian Sommer. On balanced separators in road networks. In *Proc. SEA 2015*, volume 9125 of *LNCS*, pages 286–297. Springer, 2015.

[89] Ayumi Shinohara. Complexity of computing vapnik-chervonenkis dimension and some generalized dimensions. *Theor. Comput. Sci.*, 137(1):129–144, 1995.

[90] Sathiamoorthy Subbarayan and Henrik Reif Andersen. Backtracking procedures for hypertree, hyperspread and connected hypertree decomposition of csps. In *Proc. of IJCAI 2007*, pages 180–185, 2007.

[91] Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

[92] Transaction Processing Performance Council (TPC). TPC-H decision support benchmark. `http://www.tpc.org/tpch/default.asp`, 2014.

[93] Susan Tu and Christopher Ré. Duncecap: Query plans using generalized hypertree decompositions. In *Proc. of SIGMOD 2015*, pages 2077–2078. ACM, ACM, 2015.

[94] René van Bevern, Rodney G. Downey, Michael R. Fellows, Serge Gaspers, and Frances A. Rosamond. Myhill-nerode methods for hypergraphs. *Algorithmica*, 73(4):696–729, 2015.

[95] Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

[96] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

[97] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proc. of VLDB 1981*, pages 82–94. IEEE Computer Society, 1981.

CV

DI Wolfgang Fischl, BSc

Personal Data

Address	Setzgasse 32 7072 Mörbisch am See
Phone:	0676 / 602 79 86
E-Mail:	wolfgang.fischl@live.at
Born:	June 1st, 1986 in Eisenstadt

Education

since 12/2013

TU Wien
Ph.D. Computer Science

09/2010 – 11/2013

TU Wien
Master program Computational Intelligence
Graduated with honors: Diplom-Ingenieur (comp. to MSc)

09/2006 – 12/2010

University of Vienna
Bachelor program computer science (bioinformatics)
Graduated with: Bachelor of science (BSc)

09/2004 – 06/2005

HTBLuVA Wr. Neustadt: Abteilung EDVO
Graduated with honors: Matura

09/2003 – 06/2004

Northville High School, Michigan, USA

09/2000 – 06/2003

HTBLuVA Wr. Neustadt: Department EDVO

09/1996 – 06/2000

BG u. BRG Eisenstadt

09/1992 – 06/1996

Volksschule Mörbisch

Employment History

12/2013 – 11/2018 (40 hpw)

TU Wien – Institute for Logic and Computation
Database and Artificial Intelligence Group

- University assistant
- Ph.D. student

03/2013 – 11/2013 (20 hwp)

TU Wien – Institute for Logic and Computation
Database and Artificial Intelligence Group

- Project assistant

01/2008 – 03/2013 (10 hpw)

Max F. Perutz Laboratories GmbH, Vienna
Center for Integrative Bioinformatics

- Linux system admin. (desktop & server)
- Administration of a Linux cluster
- Troubleshooting of hard- and software problems

07/2007 – 09/2007 u. 05/2006 – 09/2006 (40 hpw)

cargo-partner AG, Fischamend
IT-Abteilung

- IT support
- Linux server administration & -monitoring

07/2004 u. 07/2003 (40 hpw)

Bit-Studio, Eisenstadt
Team Technik

- IT support
- Troubleshooting of hardware problems

07/2002 (40 hpw)

Authorities of the Govt. of Burgenland, Eisenstadt
Department: management and controlling

- Secretaries office

Awards

06/2014: *Distinguished Young Alumnus Award* of the Faculty of Informatics, TU Wien

Scientific Publications (abridged)

- Wolfgang Fischl, Georg Gottlob, Davide M. Longo, Reinhard Pichler:
  HyperBench: A Benchmark and Tool for Hypergraphs and Empirical Findings. To appear PODS 2019
- Wolfgang Fischl, Georg Gottlob, Reinhard Pichler:
  General and Fractional Hypertree Decompositions: Hard and Easy Cases. PODS 2018: 17-32
  Invited to submit to the Journal of the ACM (in review)
- Shqiponja Ahmetaj, Wolfgang Fischl, Markus Kröll, Reinhard Pichler, Mantas Simkus, Sebastian Skritek:
  The Challenge of Optional Matching in SPARQL. FoIKS 2016: 169-190
- Shqiponja Ahmetaj, Wolfgang Fischl, Reinhard Pichler, Mantas Simkus, Sebastian Skritek:
  Towards Reconciling SPARQL and Certain Answers. WWW 2015: 23-33
- Diego Calvanese, Wolfgang Fischl, Reinhard Pichler, Emanuel Sallinger, Mantas Simkus:
  Capturing Relational Schemas and Functional Dependencies in RDFS. AAAI 2014: 1003-1011

Teaching

09/2014 – 08/2018: TU Wien, Vienna
Faculty of Informatics
Courses: VU Database Systems (winter term) &
VU Semi Structured Data (summer term)
Responsibilities: Exercises
Support of students during the exercises
1-2 lectures per term

10/2013 – 02/2014: University of Vienna, Vienna
Faculty of Informatics
Courses: PR Optimization and Simulation
Responsibilities: Support of 2 groups of 60 students during the exercises

10/2012 – 02/2013: University of Vienna, Vienna
Courses: PR Optimization and Simulation
Responsibilities: Teaching assistant

10/2011 – 02/2012: University of Vienna, Vienna
see 10/2012 – 02/2013

10/2010 – 02/2011: University of Vienna, Vienna
see 10/2012 – 02/2013

10/2009 – 02/2010: University of Vienna, Vienna
see 10/2012 – 02/2013