TU WIEN Informatics

Hypergraph Invariants for Computational Complexity

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Matthias Paul Lanzinger, BSc
Registration Number 0919137

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:

Pablo Barceló	Andreas Pieris

Vienna, 2nd November, 2020

Matthias Paul Lanzinger

Dipl.-Ing. Matthias Paul Lanzinger, BSc

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, 2. November 2020

Matthias Paul Lanzinger

ACKNOWLEDGEMENTS

I am very grateful to my advisor Reinhard Pichler and the DBAI research group for the trust they have shown in me and the freedom that they have allowed me in my work. I also wish to explicitly thank Georg Gottlob for his mentorship.

A dissertation stands at the end of a long journey, from school to undergraduate and graduate studies, and finally the first forays into research. I feel grateful towards a great many people that have influenced me throughout this time. In that sense I also wish to thank everybody who I have argued with, dated, or had a drink with for their important role on this path.

Finally, I am thankful to my parents Charlotte and Andreas who always supported me and to Lisa for her support, understanding, and invaluable knowledge of where to place commas in german.

ABSTRACT

In the study of computational complexity, we search for lower and upper bounds for the effort – be it time, space, or something else – necessary to perform specific algorithmic tasks with a machine. Early efforts in the field focused on the separation of tasks into tractable and intractable problems. However, many natural problems in practice are intractable, motivating a more fine-grained study of computational complexity. In such a fine-grained study, one commonly studies the effect of various parameters of the problem on the problem's computation complexity. One prominent branch of such research is concerned with parameters that express the intricacy of the structure of an instance (we refer to such parameters as *widths*). In this thesis, we continue this thread of study with a particular focus on problems whose underlying structure is naturally expressed by hypergraphs.

First, we study the structure of conjunctive queries (CQs) and Constraint Satisfaction Problems (CSPs) modulo equivalence. That is, we are not only interested in the hypergraph structure of the query, but the simplest (w.r.t. some width measure) hypergraph structure of any equivalent formulation of the query, thus capturing the complexity of the question itself rather than the complexity of the formulation. Such minimal width modulo equivalence is referred to as the *semantic width* of the query. As part of this thesis, we prove characterizations of the semantic variants of hypertree width, fractional hypertree width, and submodular width. Building on these characterizations, we show that the parameterized tractability of CQs and unions of CQs (UCQs) is fully captured by

the problem structure. Specifically, we demonstrate for CQs and UCQs that their evaluation is fixed-parameter tractable exactly for classes of instances that exhibit bounded semantic submodular width. From our result for semantic fractional hypertree width, we are also able to derive new insights and results with respect to the non-parameterized tractability of CQs.

Second, we propose a unifying theoretical framework for tractable hypergraph width checking. Following that, we utilize this framework and give some novel results in fractional (hyper)graph theory to resolve important open problems on tractable width checking from the literature. Most important of which, we prove the tractability of deciding low fractional hypertree width for hypergraph classes with bounded intersection.

Finally, we propose a novel width parameter – nest-set width – that generalizes hypergraph β-acyclicity. In contrast to existing parameters that generalize β-acyclicity, our proposed width is recognizable in polynomial time and yields important new islands of tractability. In particular, we show that propositional satisfiability is fixed-parameter tractable when parameterized by nest-set width and that the evaluation of CQs with negation is tractable under bounded nest-set width.

KURZFASSUNG

In der Komplexitätstheorie suchen wir nach unteren und oberen Schranken für den Aufwand – sei es Zeit, Speicher oder anderes – den es unbedingt benötigt, um konkrete algorithmische Aufgabenstellungen maschinell zu lösen. In den Anfängen des Feldes war man vor allem um die Separierung solcher Aufgabenstellungen in effizient und nicht effizient berechenbare Probleme bemüht. Viele natürliche algorithmische Aufgaben aus praktischen Anwendungen stellten sich jedoch als nicht effizient berechenbar heraus. Aus dieser Situation heraus enstand eine detailliertere aufgelöste Studie von Komplexitätstheorie, in der der genaue Effekt unterschiedlicher Parameter auf die Komplexität eines Problems untersucht werden. Ein prominenter Zweig dieser Forschung beschäftigt sich mit Parametern, welche die strukturellen Eigenschaften einer Instanz ausdrücken. Wir bezeichnen solche Parameter als *Ausdehnungen*. In dieser Dissertation führen wir diesen Forschungszweig fort. Dabei liegt unser Schwerpunkt auf algorithmischen Aufgaben, deren Struktur sich natürlich mittels Hypergraphen beschreiben lässt.

Dazu behandeln wir als erstes die Struktur konjunktiver Abfragen (fachsprachlich conjunctive queries, im Folgenden als CQs abgekürzt) und Constraint-Satisfaction Problemen, modulo Äquivalenz. Folglich sind wir nicht nur an der Hypergraph-Struktur einer Abfrage interessiert, sondern an der einfachsten Struktur (in Bezug auf eine konkrete Ausdehnung) einer äquivalenten Formulierung der Abfrage. Damit wird nicht nur die Komplexität der Abfrage selbst, sondern die Komplexität des dahinterliegenden abstrakten Problems erfasst.

Solch eine minimale Ausdehnung modulo Äquivalenz wird als semantische Ausdehnung einer Abfrage bezeichnet. Als Teil dieser Dissertation zeigen wir Charakterisierungen von semantischen Varianten von Hyperbaum-Ausdehnung, fraktionale Hyperbaum-Ausdehnung sowie submodularer Ausdehnung. Aufbauend auf diesen Charakterisierungen zeigen wir, dass die parametrisierte effiziente Berechenbarkeit von CQs und Vereinigungen von CQs (UCQs) komplett über die Problemstruktur beschreibbar ist. Konkret demonstrieren wir für CQs und UCQs, das deren Beantwortung in fixed-parameter polynomieller Zeit genau für Abfrageklassen mit (konstant) begrenzter semantischer submodularer Ausdehnung möglich ist. Des Weiteren leiten wir aus unsere Ergebnissen für semantische fraktionale Hyperbaum-Ausdehnung auch neue Einblicke in die unparametrisierte Komplexität von CQs ab.

Als Zweites stellen wir ein vereinheitlichendes Framework für das effiziente Überprüfen von Ausdehnungsgraden vor. Darauf aufbauend lösen wir wichtige offene Probleme in der Überprüfung von Ausdehnungsgraden aus der Literatur. Dazu nutzen wir unser vorgestelltes Framework und präsentieren neue Resultate in fraktionaler Hypergraph-Theorie. Konkret zeigen wir die effiziente Überprüfbarkeit von begrenzter fraktionaler Hyperbaum-Ausdehnung für Hypergraphen mit begrenzter Kantenüberschneidung.

Abschließend stellen wir einen neuen Ausdehnungsparameter – die Nest-Mengen-Ausdehnung – vor, welcher Hypergraph β-Azyklizität verallgemeinert. Im Unterschied zu den wenigen existierend Parametern die β-Azyklizität verallgemeinern, ist der hier vorgestellte Ausdehnungsparameter in polynomieller Zeit erkennbar und ermöglicht die Definition neuer effizient lösbare Fragmente für bedeutende Probleme. Dazu zeigen wir, dass propositionelle Erfüllbarkeit (SAT) in fixed-parameter polynomieller Zeit lösbar ist, wenn das Problem durch die Nest-Mengen-Ausdehnung parametrisiert wird. Des Weiteren ist die Beantwortung von CQs mit Negation in Klassen mit konstant begrenzter Nest-Mengen-Ausdehnung in polynomieller Zeit möglich.

Contents

CHAPTER 1

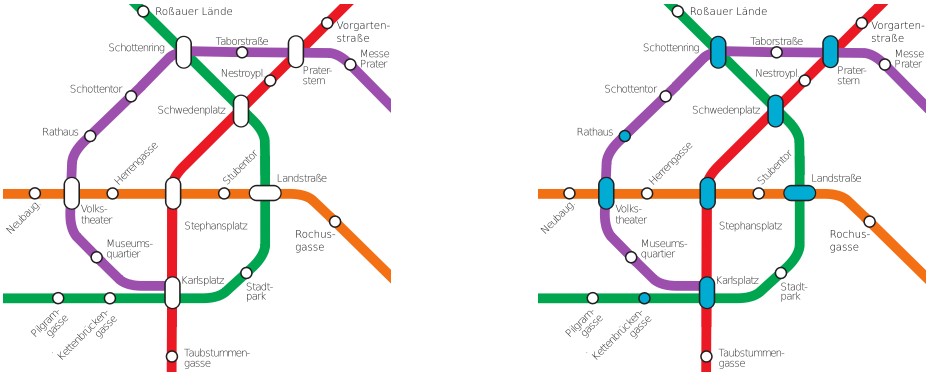
Introduction

The first three sections of this introduction intend to give an intuitive introduction to the role of structural restrictions in computational complexity theory that is approachable for a wider audience. Starting from Section 1.4, we give a more technical overview of the state of the field and the research questions that are addressed in this thesis. A reader familiar with the standard themes of parameterized complexity can skip the initial exposition and start from Section 1.4.

Much of the research that has led to the contents in this thesis has been performed in collaboration with many valued colleagues. Detailed acknowledgments as well as a mapping of the author's key publications to parts of the thesis are provided in Section 1.5.

1.1 The Tyranny of Combinatorial Explosion

In computational complexity theory we often separate problems into two categories, *tractable* and *intractable* problems. This separation is based on how



(a) Part of the metro network of Vienna.

(b) Example solution placing voting booths at 9 stations (marked in blue).

Figure 1.1: Source: both figures are modified reproductions of "Schnellverbindungen" network plan of the "Wiener Linien".

the effort of solving the problem grows in the size of the problem instance. We say that a problem is tractable, if there exists an algorithm that solves any instance of size s in $O(s^c)$ time, where c is some constant. In other words, for any tractable problem there is some polynomial (with respect to the instance size) upper bound on the effort in which it can be solved. If such a bound does not exist, i.e., if every algorithm that solves the problem requires super-polynomial time with respect to the instance size, we say the problem is intractable [1].

To begin, we will explore a central phenomenon related to the difficulty of intractable problems. Consider part of the metro network of Vienna as shown in Figure 1.1a. It consists of stations and track segments which link two stations.

Suppose there is an upcoming election and the city wants to offer voting booths on every track segment, i.e., for every track segment between station x and station y there has to be a voting booth at either station x or y (or both). This is widely considered good for democracy but, more importantly, this initiative also costs money. The government has therefore decided to persue the plan but minimize the number of voting booths to be installed in metro stations. Thus, we are now presented with a puzzle: how to place as few voting booths

[1] Other thresholds for (in)tractability are used depending on the context. The usage introduced here is the most common and can be considered standard.

as possible while satisfying the requirement that one is present on every track segment. An example solution using 9 stations is given in Figure 1.1b. To simplify the following discussion we will consider a slight variation of the question: can we satisfy this requirement by placing voting booths at c stations where c is some appropriate integer?

The naive approach would be to enumerate any combination of c stations and check for each combination if it satisfies the regulation. It is easy to see that this approach requires up to

$$\langle \#\text{of } c \text{ station combinations} \rangle \times \langle \text{steps for checking if regulation is satisfied} \rangle$$

steps to find a solution or decide that none exists. In the following, we use S to refer to the number of stations in the network and T for the number of track segments. Checking whether the regulation is satisfied is simple, iterate over all edges and check if one of the vertices is included in the set of c stations. This is possible in $O(c \cdot T)$ steps and thus in polynomial time. However, there are $\binom{S}{c}$ combinations of c stations, i.e., the number of combinations grows faster than any polynomial function in c and the size of G. For reference, the full metro network of Vienna includes 109 stations. Checking if there is a solution using 40 stations thus involves testing roughly 10^{30} combinations, far too many to check in practice[2].

From the naive approach, an interesting feature of the problem already becomes apparent. Little computational effort is required to decide whether a given choice of stations is a solution. The difficulty of finding a polynomial algorithm comes from the fact that the number of possible solutions that need to be tested grows super-polynomially in the size of the input, i.e., the solution space exhibits a form of *combinatorial explosion*. Intuitively, this situation is characteristic for an important class of computational problems that we call the NP-complete problems. The big question now is if there is a way to escape this combinatorial explosion or if it is inherent to the problem.

Our analysis of the naive approach is not sufficient to argue that the problem is intractable. As stated before, showing intractability would require us to show

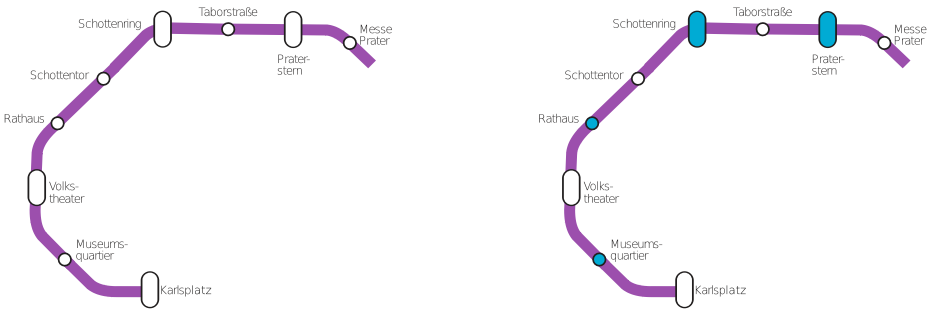
[2]The Shanghai metro network has 413 stations, checking if there is a solution with 100 stations involves checking almost 10^{100} combinations, far beyond the estimated number of atoms in the universe.

that no algorithm exists which solves the problem in a polynomial number of steps. Indeed, a definitive answer to this problem is still open and the question of whether such an algorithm exists – commonly referred to as $\mathsf{PTime} \stackrel{?}{=} \mathsf{NP}$ – is considered one of main open problems in computer science. It is widely believed that no such algorithm exists and the respective standard assumption that $\mathsf{PTime} \neq \mathsf{NP}$ is made throughout this thesis. Under this assumption, our voting booth placement problem is indeed intractable, i.e., there is no escape from the combinatorial explosion of the solution space. This is usually proven by technique called problem reduction. The tools to make such arguments are introduced in Chapter 2.

This problem of assigning voting booths to metro stations is a concrete case of a standard problem in computer science called Vertex-Cover. In general, one considers a *graph* $G = (V, E)$ consisting of a set of vertices V and a set of edges E where each edge $\{v, u\} \in E$ connects two vertices v and u with each other. A *vertex cover* of graph $G = (V, E)$ is any set $\alpha \subseteq V$ such that for every edge $\{v, u\} \in E$ we have either $v \in \alpha$ or $u \in \alpha$. There exists an assortment of related intractable problems on graphs. Just like Vertex-Cover, they frequently appear in all kinds of real-world problems and their intractability introduces tangible limitations for industry and commerce. For example, there are many natural scheduling problems in factories where finding optimal solutions is still out of reach with state of the art methods (see e.g. [111]).

1.2 Fighting Back: Structural Restrictions

While solving intractable problems can be hopeless in general, this is not the end of the story. If we look back at our voting booth example on the metro network we can identify some helpful rules for our solutions. Take any path of three stations where the middle station is not an intersection of multiple metro lines, e.g., *(Schottentor, Rathaus, Volkstheater)*. It is easy to see that if there is a solution that places booths at all three stations, then removing the booth at *Rathaus* still yields a (smaller) solution. We could therefore exclude any such configuration of stations from the total combinations that we consider. Intuitively, we see that in networks with simpler structure – i.e., less intersections – the problem becomes simpler to solve.



(a) Only the U2 line segment from Figure 1.1a.

(b) Optimal placing of voting booths for the U2 line segment.

Figure 1.2

Let us build on this observation. What if we have no intersections at all? This occurs naturally if we consider only a single line of the metro network, e.g., we will consider only the U2 line from Figure 1.1a (explicitly shown by itself in Figure 1.2a) in the following discussion.

We have already made the observation above, that it is never necessary to consider three consequent stations to find a minimal voting booth placement. Similarly, we can always assume that the ends of the line, i.e., *Karlsplatz* and *Messe-Prater* in our part of the network, have no voting booths since placing them at the neighboring stations (*Museumsquartier* and *Praterstern*, respectively) is always at least as efficient. From these observations we can construct the following simple algorithm for minimally placing voting booths on a line.

1. Mark the respective neighbors of the ends of the line as having a voting booth.
2. From some marked station, move two stations inward and mark that station.
3. Repeat the last step until there is a marked station on every track segment.

Applying the algorithm to our example produces the assignment in Figure 1.2b. In the first step, *Praterstern* and *Museumsquartier* are marked. Moving two stations inward from *Museumsquartier* leads us to mark *Rathaus*. Moving two

stations inward from *Praterstern* marks *Schottenring*. Now every track segment has at least one adjacent voting booth and the algorithm therefore terminates, having found a minimal solution.

As long as we avoid repeating the second step redundantly for the same station, this procedure will always terminate in polynomial time (in fact, the time is even linear in the number of stations). Thus, we have found a restriction (to so-called line graphs) to the structure of the problem for which the problem becomes tractable!

In a sense, we have taken our observations of how solutions behave under certain structural restrictions to circumvent the menace of combinatorial explosion. Recall, that the naive approach of simply trying all combinations of c stations is also not polynomial if we have a single line, we needed to change our algorithm to utilize the additional information we have about the structure of the network.

However, lines are maybe not the most exciting structure. Once one has found a restriction that makes the problem tractable, it is then natural to investigate if the problem remains tractable for generalizations of the restriction. Some investigation of our algorithm for line graphs reveals that it is particularly convenient that there are no cycles in a line graph. That way, a new choice of a marked station can never make an old one redundant since we always move to *new* parts of the network. Without cycles, these new parts cannot affect the previously solved parts of the problem. Indeed, if we relax our restriction from lines to networks that have no cycles (called *trees* in graph theory), we still get a fragment of the problem that is tractable. The straightforward algorithm for trees is slightly different from the one proposed above, but still makes use of the lack of cycles in a similar fashion using dynamic programming.

We start by fixing an arbitrary node of the tree as the root to orient the tree so that we can proceed in a *bottom-up* fashion. We will compute the values α_u^{in} and α_u^{out} for every station u where α_u^{in} is the size of the smallest solution where a voting booth is placed at u and α_u^{in} the size of the smallest solution if no voting booth is placed at u. Importantly, solution here refers to the local problem of the subtree rooted at u. The values at the root then correspond to the solutions of the full problem. The algorithm proceeds as follows:

1. For every leaf node ℓ we set $\alpha_\ell^{in} = 1$ and $\alpha_\ell^{out} = 0$ and mark them as *done*.

2. Choose some node u of the tree that is not marked *done* but all of its children $children(u)$ are *done*.

3. Compute the two values α_u^{in} and α_u^{out}, as follows:

$$\begin{aligned} \alpha_u^{in} &:= 1 + \textstyle\sum_{c \in children(u)} \min\{\alpha_c^{in}, \alpha_c^{out}\} \\ \alpha_u^{out} &:= \textstyle\sum_{c \in children(u)} \alpha_c^{in} \end{aligned}$$

   and then mark u as *done*

4. If u is the root node, then return $\min\{\alpha_u^{in}, \alpha_u^{out}\}$ as the lowest number of voting booth booths that satisfy the requirements. If u is not the root, continue with step 2.

That is we move bottom-up in the tree, successively solving larger and larger subproblems. When u is in the solution, we can use the smallest solutions for the individual subproblems below it (the subtrees rooted at the children of u). When u is not in the solution, we need to always consider the value of α_c^{in} for every child since there is an edge $\{u, c\}$ and a voting booth needs to be placed on at least one of them. The absence of cycles allows us to consider the solutions for the children as fixed when computing the α values for a node u since no future decision can influence the subtrees below u anymore.

The restriction to trees is already much closer to real networks than the restriction to simply lines. In fact, some smaller metro networks are in fact trees, e.g., the Kyoto metro network. Yet, looking back at our Vienna example, an algorithm for trees helps us very little when the network is not a tree. In general, we see that this kind of binary property (a network is either a tree, or not, there is no inbetween) are not optimal as more and more complex networks may require analysis of further ad hoc properties. However, there is an alternative.

Instead of further generalizing the restriction to trees to yet another binary property we can use a measure of how *tree-like* our structure is. This is commonly achieved via the notion of the *treewidth* of a graph. Intuitively, if a graph has treewidth k, we can find groupings of up to $k + 1$ nodes, such that these groupings induce a tree structure on the graph. As an example, a tree on the

groupings – formally called a *tree decomposition* – with treewidth 3 on our example network is given in Figure 1.3. A formal definition of treewidth and tree decompositions is given in Chapter 2.

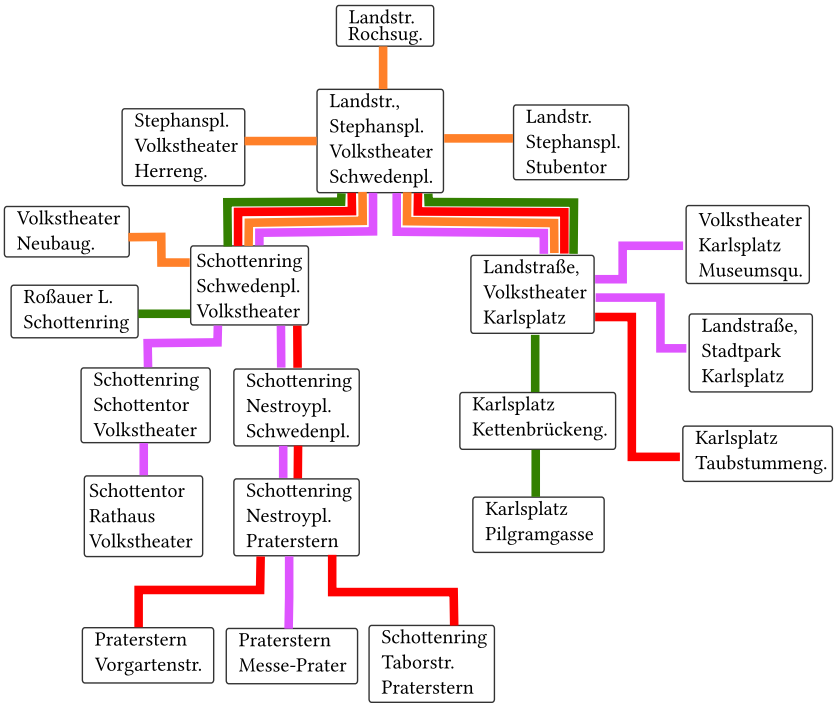


Figure 1.3: Tree Decomposition of the Metro Network Example

The simple way to algorithmically exploit a tree decomposition is again by dynamic programming on the tree. For many problems, the nodes of the tree can be viewed as their own independent subproblems, which are then integrated with the global solution via dynamic programming along the tree structure. If the subproblem for each node is small, i.e., the treewidth is at most some constant k, then we can oftentimes find local solutions for the subproblems in a polynomially boundable way.

To make this more concrete, let us again consider our voting booth example and the decomposition given in Figure 1.3. We now want to consider the subproblems for each node. By the subproblem we mean that we look only at the part of the network that contains the stations in the node as if this were its own network. Since there are at most 4 stations in any node, any subproblem

will have at most 4 stations and thus no more than 2^4 different choices of stations to mark as having a voting booth. Roughly speaking, tree decompositions are constructed in such a way that we can solve the global problem by solving the (bounded size) problem in each node of the tree and then following the algorithm for solving the problem on tree instances (with minor adaptions). Thus, intuitively a tree decomposition allows us to approach the problem as if it were cyclic only locally but acyclic from a high-level perspective. Note that this is not particular to our voting booth example but rather a common technique for problems with underlying graph structure.

A more detailed discussion of these ideas is out of the scope of this high-level section of the introduction. The interested reader is referred to a recent book on parameterized complexity by Cygan et al. [40] for a comprehensive exposition of the algorithmic uses of treewidth.

While tree decompositions will only play a tangential role in the rest of this thesis, the basic idea of these types of decompositions remains foundational. For one, the notion of graduating cyclicity in form of *width* to escape the narrowness of binary properties will follow us throughout the rest of the thesis. And secondly, decompositions that allows us to treat an algorithmic problem as an acyclic configuration of multiple small – possibly cyclic – problems, will remain the central idea for many of the parameters discussed in the following, even when we move beyond graphs.

1.3 More Dimensions More Problems

Up to this point, our discussion was focused on graphs. This is by no means an oversimplification since graphs are a sufficient abstraction for many common problems in theory and practice. Nonetheless, graphs have their limitations. Since the edges of the graph are usually used to model some kind of relationship between two vertices we are inherently restricted to binary relations and their transitive closures (via reachability) in graph models.

Consider a system that stores information on movies. Say a *movie* in this database has a *title*, a release *year*, and a *rating*. Furthermore, information on *staff* that worked on a movie is stored. In particular, it is stored who (*name*)

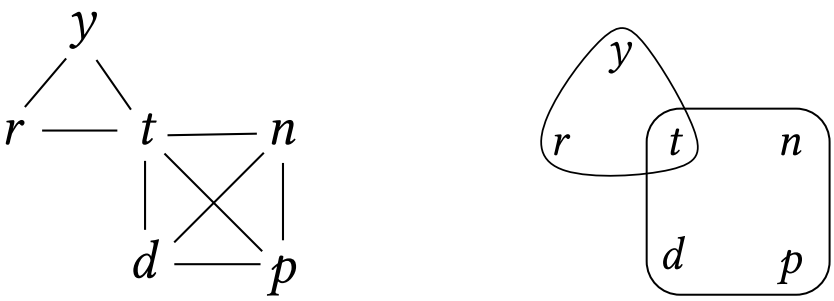
worked on which movie (*mtitle*) in which *position* and a binary flag *debut*, indicating whether this is the first recorded work by this person. In its simplest form we can treat such data as logical relations, i.e., movie is a 3-ary relation where the first component of the tuple is the *title*, the second is the *year*, and the last is the *rating*. Analogously, *staff* is a 4-ary relation with components in the order presented above.

Now suppose we are interested in finding the name and position of all staff who worked on a movie rated above 5 stars. This corresponds to finding all models – i.e., satisfying assignments to the free variables n and p – of the following logical formula ϕ

$$\phi := \exists t, y, r, d.\, \text{movie}(t, y, r) \wedge \text{staff}(t, n, p, d) \wedge r > 5 \tag{1.1}$$

First-order formulas that only allow existential quantification and conjunction are called *conjunctive queries* (CQs) and are one of the central objects of study in database theory (cf. [3]). The above problem of finding satisfying assignments for CQs is commonly referred to as *evaluating* the CQ. Evaluation of CQs is known to be intractable, even when we simplify the task to asking whether there exists a single satisfying assignment [3].

Note that the study of CQs is not only important in a classical database setting. Conjunctively linking multiple conditions together is a fundamental necessity when retrieving structured data in most contexts. Once such linked conditions are possible the system essentially implements some – usually more complex – form of conjunctive queries and thus has to struggle with their complexity. Beyond standard databases this includes important topics of modern computer science such as querying graph databases (see e.g., [99]) and data science, fields that deal with immense amounts of data. Finding tractable fragments of CQs is therefore of great interest and has far reaching consequences. Note that in the data science community the reliance to conjunctive queries is rarely acknowledged, many highly cited works (e.g., [43, 109, 101]) on the topic make no mention of the involvement of conjunctive queries and the resulting complexity theoretic challenges at all. However, connecting data from various sources inherently necessitates relational joins (in othe words, CQs) in all but the most trivial cases.



(a) The Primal Graph of ϕ.

(b) The Hypergraph of ϕ.

Figure 1.4

If we want to apply the ideas from the previous sections on graphs to this case, we first need a way to capture the structure of a CQ as a graph. Looking at Query 1.1 above, we see that there are no clear two variable groups that can form the edge relation. Instead, the most common approach is to let an edge represent that two variables occur together in a term. This is referred to as the *primal graph* of a query. The primal graph of Query 1.1 is shown in Figure 1.4a.

It is indeed possible to use tree decompositions on the primal graph to efficiently solve CQs with low treewidth [41]. However, as is apparent in Figure 1.4a, the variables that occur together in an atom – corresponding to the columns of the database table – always form a clique in the primal graph. This is problematic, a graph that contains a clique with n vertices has treewidth at least $n-1$. Thus, if we return to the database setting, we see that tables with many columns roughly speaking lead to queries with high treewidth. In consequence we can have structurally simple queries that have high treewidth (of their primal graph), demonstrating that treewidth does not describe the structural complexity of CQs well. While there are other possibilities of forming a graph for a CQ, such as the incidence graph, similar issues can be observed for those graph representations.

The fundamental issue with using a measure of graph complexity for CQs is that the relationships are not binary. By considering a graph we, in a sense, consider all partial relations that make up a table. For example, if we look at the part of the primal graph in Figure 1.4a corresponding to the term $movie(t, y, r)$ we effectively express that the 3-ary relation movie can be split in three binary

relations such that the membership of the tuple (t, y, r) in *movies* is then encoded as

$$\text{movie}_1(t, y) \wedge \text{movie}_2(y, r) \wedge \text{movie}_3(t, r)$$

In a way the treewidth of the primal graph corresponds to the purely structural complexity of rebuilding the query from only binary relations.

We see that graphs are not a sufficient formal model when it comes to such problems where the connections have naturally higher arity than 2. However, even though binary edges are insufficient the intuition of such an abstract structural view of the problem remains intact. Indeed, if one generalizes graphs to allow for edges of arbitrary size, i.e., an edge becomes an arbitrary set of vertices instead of a set of exactly two vertices, we can encode the structure of CQs much more naturally. Such generalized graphs are called *hypergraphs*. The natural hypergraph representation of query 1.1 is shown in Figure 1.4b. The structure there, such as the obvious absence of cycles, much better matches the expected structural makeup of our query.

As will become apparent throughout the next sections, consideration of the hypergraph structure of CQs has led to a highly successful area of research on tractable fragments of CQs and related problems. Just like treewidth is closely related to the complexity of many problems with natural graph representation, there are width measures for hypergraphs that allow us to define more general tractable fragments of CQs and similar problems where higher arity occurs naturally. How the *computational complexity* of such problems relates to *hypergraph invariants* that express the structural complexity of the underlying hypergraph representation is the main theme of this thesis.

At this point it should be noted that the generality of hypergraphs comes with great technical challenges beyond those that we encounter in graph theory. The challenge with hypergraphs is maybe best summarized by the abstract of a seminal paper by Füredi that reads "Almost all combinatorial questions can be reformulated as either a matching or a covering problem of a hypergraph." [55], an ominous warning in the eyes of this author. In consequence many of the theoretical tools from graph theory no longer work in hypergraphs, requiring the development of new techniques instead.

On a final note, while this section focused on CQs as an example for higher dimensional data that can not be captured well by graphs, there are many other prominent cases. In constraint satisfaction one often requires a set of variables all assigned to distinct values, i.e., $x_i \neq x_j$ for some $1 \leq i, j \leq n$. While $\neq$ itself is binary, it is not transitive, which means one would have to model all pairwise "$\neq$"-relations in the graph, again leading to a clique of size n. The situation becomes even more problematic in integer programming where we have sets of inequalities where one side is an arithmetic (linear) formula and one wants to find assignments that satisfy these inequalities. Say we have an inequality $3x - 2y + 5z \geq 0$: the variables $\{x, y, z\}$ are all clearly related and this relation should be captured in a structural representation of the problem. Yet, it is difficult to come up with sensible semantics for what a pairwise relation between two of the variables would even mean.

1.4 Hypergraph Invariants

Until now we have introduced the study of the underlying structures of hard algorithmic problems with the intention of discovering tractable fragments of the problem. We demonstrated intuitively why in particular the study of width measures such as treewidth is valuable. We then argued that the graph formalism can be too restrictive for a variety of important problems and that graph invariants such as treewidth lose some of their applicability. This motivated the step to hypergraphs and the development of hypergraph invariants for computational complexity. From here on the presentation is more technical and oriented towards a reader who is knowledgable in the field.

In the previous section, some inadequacies of treewidth – and specifically primal treewidth – for problems with higher dimensional relations were discussed. However, this should not be misunderstood as treewidth not being useful in these settings. For the sake of brevity we will refer to these problems with inherent higher dimensional relations as *(structurally) higher dimensional problems*. The focus in the literature as well as the focus of our work here lies on hypergraph invariants in the context of CQs and Constraint Satisfaction Problems (CSPs). Still, many of the results presented here are on the hypergraph level and not specific to those problems. They are therefore also of interest

to the study of other higher dimensional problems. Note that since a CQ has only one underlying hypergraph, it is common to extend the properties of the hypergraph to the query itself. For example, we say that a CQ is acyclic when its underlying hypergraph is acyclic. We apply this convention for all (hyper)graph properties of CQs from here on.

The role of (primal) treewidth in the complexity of answering CQs has been extensively studied and is well understood. First, Dalmau, Kolaitis, and Vardy [41] showed how CQs of bounded treewidth can be answered in polynomial time by reduction to Datalog. They then extend this observation to show that in fact any CQ that is equivalent to a CQ of bounded treewidth can also be answered in polynomial time. Building on this result, Grohe [72] showed that when the arity of relations is bounded by some constant, bounded treewidth of some equivalent query indeed characterizes the class of tractable instances. Note however, that the restriction to bounded arity is key here as it avoids the high arity relations which would lead to large cliques in the primal graph. Still, despite the restriction to bounded arity, these results demonstrate that the (hyper)graph structure of the query is tightly connected to the complexity of CQ answering.

Beyond the world of CQs and CSPs we can still apply fundamental results, such as Courcelle's Theorem [37], which states that any property expressible in monadic second order (MSO) logic can be detected in fixed-parameter linear time when parameterized by the structures treewidth (and the size of the formula). Hence, if some structurally higher dimensional problem is expressible in MSO on its primal graph, incidence graph, or any other graph representation, then bounded treewidth of the respective graph again gives rise to an island of tractability.

While the treewidth results for CQs clearly show how query structure and complexity are intrinsically linked, they still leave open the important case of unbounded arity. For dealing with unbounded arity it has proven fruitful to consider the *hyper*graph structure of the query directly without an intermediate transformation to some graph form. While some ideas from tree decompositions carry over to the hypergraph world, there are fundamental differences between graphs and hypergraphs that change the setting significantly. One of the key

differences is the matter of acyclicity. While defining acyclicity in graphs is straightforward, in hypergraphs the situation becomes less clear and four different kinds of acyclicity appear in the literature. In ascending order of generality they are

$$\gamma\text{-acyclic} \subset \text{Berge-acyclic} \subset \beta\text{-acyclic} \subset \alpha\text{-acyclic}$$

An extensive treatise on these notions, including their motivations and a number of equivalent characterizations for each of them and be found in [49]. Recently, a more unified framework of hypergraph acyclicity as well as some interesting additional characterizations were proposed by Duris [47]. In this work only the two most general notions – α and β-acyclicity – are of interest. Full definitions are provided in Chapter 2.

For CQs, α-acyclicity has been identified as the most relevant type of acyclicity. Importantly, α-acyclicity is incomparable to bounded primal treewidth, i.e., an α-acyclic hypergraph can have unbounded treewidth and vice versa. This type of acyclicity has long been identified to have a number of natural applications in database theory, see [19, 49, 113]. Indeed, if a CQ is α-acyclic, then the query can be answered by what has become known as *Yannakakis' Algorithm* in linear time with respect to the combined size of the input and the output [113].

Analogous to how the idea of generalizing graph acyclicity has led to the development of treewidth, these results on acyclic CQs motivated a long line of research in generalizations of hypergraph α-acyclicity. Chekuri and Rajaraman initially proposed query width [32] as such a generalization. While CQs of bounded query width are indeed answerable in polynomial time when an appropriate *query decomposition* of the query is given, finding such a decomposition was later shown to be NP-hard [68].

In a next step, Gottlob, Leone, and Scarcello [68] proposed *hypertree width* (*hw*), which properly generalizes query width, induces islands of tractability for CQ answering, and is recognizable in polynomial time for a fixed width value. At the time of writing, it remains the most general notion that satisfies all three of these properties. Since then the notion has been highly successful, with theoretical applications in a variety of fields (see e.g., [60, 59, 58]), and finding its way into experimental database systems [1, 82, 7]. Importantly, a query with bounded

hypertree width can be reduced to a query with α-acyclic hypergraph structure in polynomial time which enables us to then apply Yannakakis' algorithm.

The success of hypertree width motivated further research into more general conditions for tractable CQ answering. This led to the development of *generalized hypertree width* (*ghw*) [69] which relaxes the constraints on the hypertree decomposition, leading to a more general notion that still allows the same reduction to an acyclic query as hypertree width. However, like query width, recognizing low *ghw* is intractable. In particular, Gottlob, Miklós, and Schwentick [69] showed that deciding $ghw(H) \leq 3$ is NP-hard. The result has recently been improved upon by Fischl, Gottlob, and Pichler [54] who proved that even deciding $ghw(H) \leq 2$ is NP-hard. In a sense, *hw* can be seen as a tractable 3-approximation of *ghw* as was shown by Adler, Grohe, and Gottlob [6].

Note that even though *ghw* cannot be recognized in polynomial time, bounded *ghw* is a sufficient restriction for classes of CQs to be tractable since bounded *ghw* also implies bounded *hw*. Chen and Dalmau [34] propose an alternative approach. They give an algorithm that answers a CQ q in polynomial time under the assumption that q has *ghw* at most some constant k. However, if this promise on the width of q is broken, the algorithm is no longer sound. Since we cannot check in polynomial time if the promise was broken this approach by Chen and Dalmau leads to a kind of *non-uniform* tractability of CQ answering for bounded *ghw* that can be considered as slightly different than the tractability induced by bounded hypertree width.

The tractable evaluation of CQs of bounded *hw* or *ghw* relies on the observation that joining at most k relations is feasible in time $O(|R_{max}|^k)$ where $|R_{max}|$ is the size of the largest relation. Grohe and Marx [73] showed that this is in fact a special case of a much deeper property, namely that the size of the result of a join is bounded by the *fractional cover number* (ρ^*) of the underlying hypergraph structure. They show that it is possible to always compute a join in $|R_{max}|^{\rho^*+O(1)}$ time, a bound that was later tightened to $O(|R_{max}|^{\rho^*})$ by Ngo et al. [96]. These bounds were proven to be tight in a sense by Atserias, Grohe, and Marx [12]. Building on these results, Grohe and Marx also proposed *fractional hypertree width* (*fhw*) [73] which further generalizes *ghw*. Fischl, Gottlob, and

Pichler [54] recently showed that deciding whether $fhw(H) \leq 2$ is also NP-hard. However, there is a cubic approximation algorithm for *fhw* due to Marx [89] and therefore bounded *fhw* is also a sufficient condition for tractable CQ answering. In fact it is the *most general* such condition known at the time of writing.

Hypergraph invariants also play an important role beyond the world of plain tractability of CQs. When we move to the parameterized setting we are usually interested in CQ answering parameterized by the query. This has a clear practical motivation from the database domain where databases now regularly contain terabytes of data, while even the most complex queries are many magnitudes smaller. Thus, the trade-off of high complexity in size of the query for non-exponential complexity in the size of the database is an attractive proposition. To this end Marx [92] introduced a further generalization of *fhw*, *submodular width* (*subw*). In a highly impressive result, Marx showed that bounded *subw* in fact characterizes those hypergraph structures for which *all* CQ answering for queries of such structure is *fixed-parameter* tractable when parameterized by the query. This characterization result has recently received much interest which resulted in a variety of valuable extensions, see [106, 81, 20].

In even more recent results, Grohe's characterization has been extended to *ontology-mediate queries* (OMQs). OMQs extend the task of answering a CQ (or a UCQ) to additionally respect domain knowledge encoded in the form of logical rules (generally in description logic ontologies). In a number of closely related results, Barceló et al.[15, 14] characterize (among other results) fixed-parameter tractable evaluation of OMQs under bounded arity for ontologies expressed in various important logical formalisms. Interestingly, just as normal CQ answering, equivalence to a query of bounded treewidth characterizes fixed-parameter tractability in their setting.

Up to now we have focused on the canon of hypergraph invariants for CQs and CSPs. In the context of hypergraph structures, most research has focused on these problems due to their generality and their apparent natural connection to α-acyclicity and its generalizations (see also [13]). As was briefly mentioned, α-acyclicity itself (and some generalizations) have also been shown to be connected to the complexity of other problems such as finding Nash Equilibria [60]. However, α-acyclicity is by no means the only hypergraph acyclicity of inter-

est in computational complexity. Notably, hypergraph β-acyclicity leads to tractable fragments of various natural problems such as SAT [97], CQs with negation ($CQ^{\neg}$) [23], and polynomial optimization [100].

While acyclicity and width measures have received the majority of the attention in the literature and this section, other kinds of interesting hypergraph invariants exist. In response to the hardness proofs for checking *ghw* and *fhw* by Fischl, Gottlob, and Pichler, the same authors also studied these check problems under certain structural restrictions [54]. A particularly interesting restriction that they propose is a bound on the cardinality of intersections between some constant number of edges, which they termed *multi-intersection width*. This property emerges only in the context of hypergraphs and indeed actually only for hypergraphs of unbounded rank[3] (because the intersection can never be larger than the size of any edge itself). Furthermore, multi-intersection width is completely independent of acyclicity and related width measures. Yet, the problem of checking whether $ghw \leq k$ for constant k becomes tractable for hypergraph classes with bounded multi-intersection [54]. In addition to width measures themselves, hypergraph invariants of this type are also studied in this thesis.

1.5 Main Challenges & Contributions

As was outlined in the previous section, the study of hypergraph invariants for computational complexity has seen significant progress in recent years. Nonetheless, the field is still young and many important questions have remained unanswered. In this thesis we ask and answer some of them.

Many of the contributions listed in this section were achieved in joint work with various collaborators. The contributions of this thesis fall into three topics, the study of the parameterized complexity of CQs and UCQs, the study of the complexity of checking *ghw* and *fhw*, and the generalization of β-acyclicity. The key publications and collaborators for the respective topics are acknowledged in the relevant subsections below. We also highlight the main results of the thesis while further incidental results are discussed in the respective chapters.

[3] Recall that rank is the hypergraph equivalent of the arity in a CQ.

1.5.1 Semantic Widths and The Parameterized Complexity of (Unions of) Conjunctive Queries

The work presented in this section was performed in collaboration with Hubie Chen, Georg Gottlob, and Reinhard Pichler. The main results were published at the International Joint Conference for Artificial Intelligence (IJCAI) 2020 [35]. Earlier related work was published at the Alberto Mendelzon International Workshop on Foundations of Data Management in 2019 [63].

While the study of hypergraph invariants has produced vast islands of tractability for CQ answering, lower bounds remain open for the unbounded arity case. Recall that for bounded arity, Grohe's characterization of (fixed-parameter) tractability [72] fully answers the question. Despite the restriction to bounded arity, this result shows that query structure plays a fundamental role in the complexity of CQ answering (see also [90]). It is thus natural to strive for similar characterizations for the general case.

> **Research Challenge:** Is there a natural characterization of the tractable classes of CQs?

Recall that Dalmau, Kolaitis, and Vardi showed that CQ answering is tractable for queries that are *equivalent to* queries of bounded treewidth. Since CQs happen to be semantically equivalent if and only if they are homomorphically equivalent [31], a CQ being equivalent to a CQ of treewidth at most k is thus also commonly referred to as having treewidth at most k *modulo homomorphism*.

In related work, Barceló et al. [18, 16] investigated queries that were equivalent to α-acyclic queries, so-called *semantically acyclic* queries. In follow-up work they also studied generalized hypertree width modulo homomorphism, or as they call it *semantic generalized hypertree width* [17]. In particular they managed to show that semantic generalized hypertree width is exactly the *ghw* of the *smallest* equivalent query, the *core*.

The result by Dalmau, Kolaitis, and Vardi is an important precursor to Grohe's characterization. The study of semantic hypergraph width is therefore a natural starting point in our quest towards a general characterization theorem.

However, since *ghw* is already known to not be the most general width notion for tractability, we are interested in the properties of semantic widths beyond the important foundations laid by Barceló, Pieris, and Romero in [17].

> **Research Challenge:** Can we also characterize more general semantic widths than semantic *ghw* by the width of the core?

In a first important step towards our characterization result we are indeed able to characterize semantic fractional hypertree width and submodular width. These two widths are of particular interest for characterization theorems. Fractional hypertree width is the most general known property that allows us to define tractable classes for CQ answering and the same applies to submodular width and fixed-parameter tractability.

> **Main Result 1:** Semantic fractional hypertree width and semantic submodular width are characterized by the respective width of the core.

Recall that Marx [92] was able to characterize those hypergraphs, i.e., problem structures, that always allow for fixed-parameter tractable evaluation by the *submodular width* of the hypergraphs under assumption of the *Exponential Time Hypothesis*. However, while this result is closely related to our goal, the fact that the characterization is on the hypergraph level significantly limits its applicability in our setting: When we consider a CQ, then the complexity of answering our query does not necessarily depend on the complexity of other, unrelated, CQs that happen to have the same underlying hypergraphs. Hence, characterizing on the hypergraph level restricts us to a worst-case that may not be connected to the problem we want to study (this point is discussed in detail in Section 3.3).

Still, by a combination of Main Result 1 and a novel reduction to Marx' setting we are able to characterize the fixed-parameter tractability of CQs. We then extend this characterization even to unions of conjunctive queries, thus solving another important problem of database theory.

Main Result 2: Assuming the Exponential Time Hypothesis, a class of CSPs is fixed-parameter tractable if and only if it has bounded semantic submodular width.

Main Result 3: Assuming the Exponential Time Hypothesis, a class of UCQs is fixed-parameter tractable if and only if it has bounded semantic submodular width.

With the question of fixed-parameter tractability resolved, we shift our attention to the rest of our original research challenge, the search for a characterization of the PTime solvable classes of CQs. Here, a characterization of tractable restrictions remains an open question. While we are not able to solve the problem in this thesis, we make some progress towards a characterization of tractability when we exclude a certain kind of *degeneracy* in the query structure.

In particular, we propose the study of what we call *non-exotic* classes of CQs, which avoid this highly unnatural degeneracy. Utilizing some recent results on the connection of hypergraph width parameters and Vapnik-Chervonenkis dimension, we show that the two most important sufficient conditions for tractable CQ answering – bounded fractional hypertree width and bounded hypertree width – actually collapse for classes of non-exotic CQs. This ultimately leads to the final main result on this topic.

Main Result 4: Answering conjunctive queries is tractable for non-exotic classes of queries with bounded *semantic* fractional hypertree width.

1.5.2 Checking Generalized and Fractional Hypertree Width

A second thread of research in this thesis is concerned with the computational complexity of checking generalized and fractional hypertree width. Formally we consider the following Check problem for some width function w and integer k.

Check(w, k)	
Instance:	A hypergraph H
Question:	$w(H) \leq k$?

In this terminology, we summarize our study Check(ghw, k) and Check(fhw, k) in this section.

This research was performed in collaboration with Georg Gottlob, Reinhard Pichler, and Igor Razgon. The contents are based on two key publications, one paper which at the time is under review by the Journal of the ACM [64] and a follow-up paper published at the International Symposium for Mathematical Foundations of Computer Science (MFCS) in 2020 [65]. Our recent results were also summarized in a recent survey paper [62].

In a recent paper by Fischl, Gottlob, and Pichler [54] the authors settle the complexity of both of the aforementioned check problems as NP-hard even for $k = 2$. In the same paper they also present a first study of *easy cases*, or structural restrictions for which the check problems become tractable. However, all of the proofs are independent of each other and little commonality is found between them. This is particularly unfortunate in this setting since recent real-world implementations (e.g., [70, 53]) are building on these theoretical results. The current proof structure makes it difficult to extract the vital insights for pragmatic implementations.

Research Challenge: Is there a common proof strategy underlying the individual tractability proofs for the Check(ghw, k) and Check(fhw, k) problems that can be abstracted to a general framework for such tractability proofs?

Main Result 5: We introduce the framework of *candidate tree decompositions* that splits the problem of tractable width checking into a single uniform algorithm and a problem specific combinatorial problem. This allows for new, vastly simpler proofs of the existing tractability results, often even lowering their upper bound to LogCFL.

In their paper, Fischl, Gottlob, and Pichler left the tractability for some interesting cases open. They show that Check(ghw, k) is tractable when the size of the intersection of any two edges of the hypergraphs is bounded (the bounded intersection property). For fhw this is left open. The situation is the same for the even more general bounded *multi*-intersection property. Using our new framework for unified tractability proofs we thus approach these open problems.

> **Research Challenge:** Is Check(fhw, k) tractable under bounded intersection or even bounded multi-intersection?

We are ultimately able to show the tractability for both properties. Both proofs require novel combinatorial techniques and come from general results on the number of possible bounded weight fractional edge covers in a hypergraph. Only the proof for bounded intersection is presented in full in the context of this thesis. For the even more involved proof for multi-intersection we give a brief overview and refer to the respective publication [65] for further details.

> **Main Result 6:** Check(fhw, k) is tractable under bounded intersection.

Our research in this area produced further incidental combinatorial results in fractional graph theory. For example, the bound on the number of fractional edge covers can also be extended to fractional vertex covers.

1.5.3 Feasible Generalizations of β-Acyclicity

The contributions presented in this final part are an extended version of a forthcoming paper that will be presented at the ACM Symposium on Principles of Database Systems (PODS) 2021 [86].

Towards the end of the previous section we gave a brief overview of some problems that become tractable when restricted to β-acyclic instances. All of them also remain NP-hard if they are only restricted to the more general α-acyclicity. However, despite the unquestionable success of the generalization

of α-acyclicity, the generalization of β-acyclicity has received little attention so far. Of the few approaches that have been made, none of them has yet been successful in properly generalizing tractability results beyond β-acyclicity (see Chapter 5 for details). Since some of the algorithmic problems that become tractable only for β-acyclicity are of central interest to logic and theoretical computer science – such as SAT and $CQ^{\neg}$ answering – further insight into their complexity is highly sought-after.

> **Research Challenge:** Is there a natural generalization of hypergraph β-acyclicity that allows us to generalize tractability results for β-acyclicity?

In Chapter 5 we answer this question by introducing a new width measure – *nest-set width* (*nsw*) – which properly generalizes β-acyclicity. Not only does nest-set width preserve many particular properties of β-acyclicity, such as being hereditary, it also achieves our goal of extending algorithmic results from β-acyclicity to bounded *nsw*. In particular, we present the following two results.

> **Main Result 7:** Answering boolean conjunctive queries with negation is tractable for classes of queries with bounded nest-set width.

> **Main Result 8:** The propositional satisfiability problem for formulas in CNF is tractable for classes of formulas with bounded nest-set width.

To further underline the naturalness of nest-set width we prove the tractability of SAT under bounded *nsw* in two different ways, both building on existing ideas that were used for β-acyclic problems. Furthermore, we discuss general conditions that are sufficient for a tractability result for β-acyclic instances to generalize to tractability under bounded *nsw*.

With the algorithmic usefulness of nest-set width established by these two results we then also address the two standard questions for every width measure. For one we are interested in the computational complexity of recognizing low

width, a problem that is heavily studied for any kind of width measures, be it on graphs or for widths generalizing α-acyclicity. The other immediate question is how nest-set width relates to other common widths so as to set the algorithmic results into a larger context.

> **Research Challenge:** What is the computational complexity of Check(nsw, k)?

> **Research Challenge:** How is nest-set width related to other common width measures?

For the first question we are able to give a complete complexity picture. We show the NP-hardness of the case where the width parameter is in the input and the tractability when the width parameter is constant. We also give a fixed-parameter polynomial algorithm for the natural parameterization of the problem.

> **Main Result 9:** Check(nsw, k) is fixed-parameter tractable when parameterized by k and NP-hard in a non-parameterized setting where k is considered part of the input.

We also establish the relationship of nest-set width to all of the important related width measures. We show that nest-set width is a special case of β-hypertree width (β-hw) introduced by Gottlob and Pichler [71]. Furthermore it is incomparable to various important width measures on the primal and incidence graph. This highlights the novelty of nest-set width and establishes an independent hierarchy on the axis of β-acyclicity, analogous to the hierarchy of generalizations of α-acyclicity.

> **Main Result 10:** Nest-set width is a specialization of β-hw and incomparable with tree- and clique-width of the primal and the incidence graph.

1.6 Organization of this Thesis

The rest of this thesis is organized as follows. In Chapter 2 we recall necessary definitions and results from the literature.

The three chapters thereafter contain the main results of this thesis where each chapter corresponds to one of the three themes of Section 1.5. Each of these topics follows a different facet of the study of hypergraph invariants. To set the context for each chapter and to further emphasize how our results fit in the scientific canon, each of these three chapters has its own introduction and conclusion.

We conclude in Chapter 6, where we recap our main results in the context of the full technical details. Furthermore, we discuss how our work relates to other open questions in the field. We also pose some new questions that arose during our research.

CHAPTER 2

Preliminary Definitions and Propositions

In this section we provide the most important definitions and results from the literature that are necessary to follow this thesis. We also fix various notations that are used throughout this thesis. The results from the literature that are given here are of general interest throughout the thesis. Results that are used for specific proofs or are similarly limited in their relation to this thesis are presented in the respective parts of the thesis where they are important.

For positive integers n we will use $[n]$ as a shorthand for the set $\{1, 2, \dots, n\}$. When X is a set of sets we sometimes write $\bigcup X$ for $\bigcup_{x \in X} x$. The same applies analogously to intersections. Furthermore, we assume the reader to be familiar with propositional logic.

2.1 (Parameterized) Complexity Theory

For most of this thesis, knowledge of standard notions from (parameterized) computational complexity is sufficient. Both topics have been formally presented in various standard works with more care than is possible in the scope of this thesis. We therefore limit ourselves to fixing terminology and notation here. We refer to [98, 78] and [40] for comprehensive overviews of computational complexity and parameterized complexity, respectively.

As our model of computation we fix Turing Machines (TM) as defined by Hopcroft and Ullman [78]. Recall, a TM consists of a tape of discrete cells that contain symbols, an internal state, and a head that points to one cell of the tape. Depending on the state and the symbol at the cell the head points to, a *transition* is performed in which the internal state is changed, a new symbol is written at the head position, and the head is moved either one cell left after writing. Formally a TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where Q is a finite set of states, Σ is a finite set of input symbols, Γ is the set of all tape symbols, B is a designated blank symbol (in Γ but not in Σ), and $F \subseteq Q$ is the set of *accepting* states. The *transition function* $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{left}, \text{right}\}$ takes a state and a tape symbol and returns the next state, a new symbol that is written to the cell, and the direction (either left or right) in which the head moves after writing.

In the initial state, only the input to the TM is present on the tape and the machine is in state q_0. We say that an input is *accepted* by a TM, if the computation eventually reaches an accepting state. The interested reader is once again referred to Hopcroft and Ullman [78] for further reading.

Note however that the exact model of computation is of little consequence in this thesis. In Section 4.1 we briefly consider the Alternating Turing Machines (ATM) of Chandra, Kozen, and Stockmeyer [30] as a model of computation. The relevant details of ATMs are given there.

We refer to the class of decision problems that can be decided in a polynomial number of transitions by some TM as PTime. Analogously, we use NP to refer to the class of decision problems that can be decided in a polynomial number of steps by a non-deterministic TM. Following standard convention we refer

to problems in PTime as *tractable* and problems in NP as *intractable*. At the time of writing it is still unknown whether NP is a proper superclass of PTime. Throughout this thesis we make the standard assumption that PTime $\neq$ NP. Finally, we also refer to the decision problems that can be decided by a TM using only a logarithmic number of writable tape cells (i.e., memory) as LogSpace. In the context of ATMs mentioned above we also consider the less common complexity class LogCFL [110]. Further discussion of this class is given with the details of ATMs.

Parameterized complexity enables a more fine-grained study of computational complexity. For an alphabet of symbols Σ, a *parameterized problem* is given as a pair (P, κ) of a problem $P \subseteq \Sigma^*$ and its parameterization κ that maps each string in Σ^* to a parameter.

We say that a parameterized problem (P, κ) is *fixed-parameter tractable* if there exists an algorithm that decides whether a given string $x \in \Sigma^*$ is in P in time $f(\kappa(x))poly(|x|)$, where f is a computable function and $poly$ is a polynomial.

A precise notion of reduction in the parameterized setting will be important in Chapter 3. Let (P, κ) and (P', κ') be two parameterized problems. A *fpt-reduction* from (P, κ) to (P', κ') is a mapping $R : \Sigma^* \to \Sigma^*$ with the following properties:

1. $x \in P \iff R(x) \in P'$ for every $x \in \Sigma^*$,
2. R is computable in time $f(\kappa(x))poly(|x|)$ (f is computable), and
3. there is a computable function g such that $\kappa'(x) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

We say (P, κ) is fpt-reducible to (P', κ'), denoted $(P, \kappa) \leq (P', \kappa')$. The class of fixed-parameter tractable problems is closed under fpt-reductions.

Our main result in Chapter 3 assume the Exponential Time Hypothesis, which states that 3-SAT with n variables can not be decided in $2^{o(n)}$ time [79]. This is a standard assumption of parameterized complexity theory.

2.2 Hypergraphs

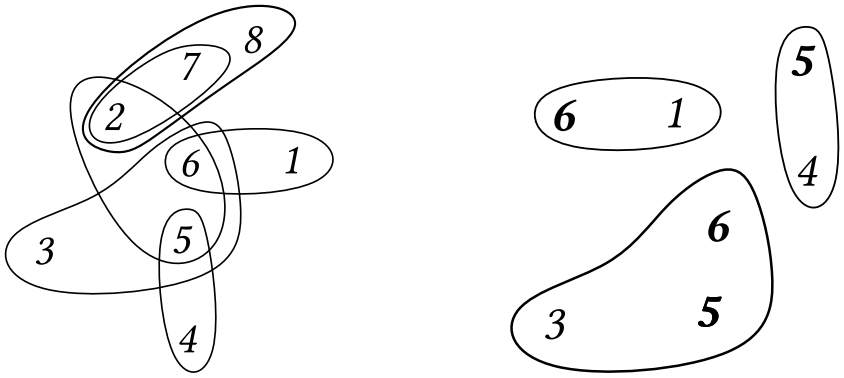
A *hypergraph* H is a pair $(V(H), E(H))$ where $V(H)$ is a set of *vertices* and $E(H) \subseteq 2^{V(H)}$ is a set of *hyperedges*. A hypergraph where every edge has exactly two elements is called a (simple) *(graph)*. For hypergraph H and vertex v, we denote the set of incident edges of v as $I(v, H) := \{e \in E(H) \mid v \in e\}$. The notation is extended to sets of vertices $s = \{v_1, \ldots, v_\ell\}$ as $I(s, H) := \bigcup_{i=1}^{\ell} I(v_i, H)$. We say an edge $e \in I(s, H)$ is *incident* to the set s. If H is clear from the context we drop H in the argument and write only $I(s)$.

A *subhypergraph* H' of H is a hypergraph with $E(H') \subseteq E(H)$ and $V(H') = \bigcup E(H')$. The *vertex induced subhypergraph* $H[U]$ of H is the hypergraph with $V(H[U]) = U$ and $E(H[U]) = \{e \cap U \mid e \in E(H)\} \setminus \{\emptyset\}$. For a set of vertices X we write $H - X$ as shorthand for the vertex induced subhypergraph $H[V(H) \setminus X]$.

Given a hypergraph H, the *dual hypergraph* H^d is defined as $V(H^d) = E(H)$ and $E(H^d) = \{\{e \in E(H) \mid v \in e\} \mid v \in V(H)\}$. The *incidence graph* of a hypergraph H is a bipartite graph (W, F) with $W = V(H) \cup E(H)$, such that, for every $v \in V(H)$ and $e \in E(H)$, there is an edge $\{v, e\}$ in F if and only if $v \in e$. Note that a hypergraph H and its dual hypergraph H^d have the same incidence graph.

To simplify some technical matters we make some standard assumptions about the hypergraphs that we consider. In particular, we assume throughout this thesis that hypergraphs have no isolated vertices and no empty edges. When considering the underlying hypergraph structure of problem instances, this is a natural assumption to make: isolated vertices have no relationship to the rest of the problem and empty edges usually do not occur by definition of the underlying hypergraph (compare with the definition of the hypergraph structure of a CQ below). Furthermore, note that our definition of $E(H)$ as a set does not allow for two distinct edges with the exact same vertices.

When we are interested in dual hypergraphs it makes sense to consider a further assumption. We call a hypergraph H *reduced* if it satisfies our standard assumptions from the previous paragraph, and if no two distinct vertices in $V(H)$ occur in precisely the same edges. If H is reduced, then we have $(H^d)^d = H$, i.e., the dual of the dual of H is H itself.



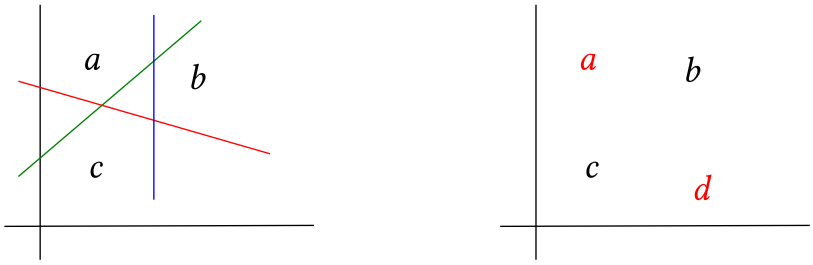
(a) Hypergraph H for Example 2.1. (b) The edges that shatter $\{6, 5\}$ in H

Figure 2.1: Illustrations for Example 2.1

In this thesis we are interested in a number of different structural properties of hypergraphs. We introduce the more common notions here and reserve the introduction of new or more niche structural properties for the respective sections where they first appear. The *rank* of H is the maximum cardinality of the edges of H. The *degree* of a vertex v is the number of edges incident to v, i.e., $\text{degree}(v) := |I(v)|$. The *degree* of a hypergraph H is the maximal degree of its vertices, i.e, $\text{degree}(H) := \max\{\text{degree}(v) \mid v \in V(H)\}$. A more complex concept that will be of interest to us is the *Vapnik-Chervonenkis dimension (VC dimension)*, a notion that originally comes from learnability theory [21]. 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 VC dimension $\text{vc}(H)$ of H is the maximum cardinality of a shattered subset of $V(H)$.

Example 2.1. These important structural notions are now illustrated on the hypergraph H in Figure 2.1a. First we can observe that the degree of H is 3 as vertices 2, 5, and 6 are all contained in 3 edges. The rank of H is similarly straightforward to observe: edges $\{2, 5, 6\}$, $\{2, 7, 8\}$, and $\{3, 5, 6\}$ all have cardinality 3 and no edge contains more than 3 vertices, hence the rank of H is 3.

When we consider the intersection width of H, it is easy to observe that it can no higher than the rank. Indeed, we even have $\text{iwidth}(H) \leq \text{rank}(H) - 1$ since no



(a) Example of three shattered points.

(b) No halfplane intersects only the points a and d.

Figure 2.2: Illustrations for Example 2.2

two distinct edges can fully overlap. It is then easy to verify that intersections of size 2 actually exist, e.g., $|\{2,7,8\} \cap \{2,7\}| = 2$, hence $\text{iwidth}(H) = 2$. For three edges we have no intersection of size 2, but many intersections of three edges contain 1 element (any vertex with degree 3 must be in the intersection of three edges), thus $3\text{-miwidth}(H) = 1$. Since the maximum degree in H is 3, no vertex is contained in four or more edges, hence $c\text{-miwidth}(H) = 0$ for $c > 3$.

Finally, H has VC-dimension 2. The set $\{5,6\}$ is the only shattered set with more than one element in H. The edges that produce the respective (non-empty) subsets of $\{5,6\}$ by intersection are explicitly shown in Figure 2.1b. △

VC dimension has been shown to be an interesting property throughout mathematics and theoretical computer science. In learnability theory (where it originated) it captures how complex a set of functions can be for them to be learnable by certain statistical classification methods. However, VC dimension has also become a widely studied property in combinatorial geometry. There, many important results rely on the assumption of bounded VC dimension. It is particularly interesting that many of the hypergraphs studied there (often under the name *set systems*) naturally have bounded VC dimension, even when infinite vertex sets are permitted [93].

Example 2.2. A standard object of interest in combinatorial geometry is the halfplane hypergraph (P, R). The vertices of the hypergraph $P \subseteq \mathbb{R}^2$ are a set of points on the real plane. The set of edges R are all intersections of P with

closed halfplanes. Intuitively, this corresponds to those subsets of P that can be separated by a straight line on the real plane (including the vertices on the line). Notably, the VC dimension of any halfplane hypergraph is at most 3, even if we admit $P = \mathbb{R}^2$ [93].

Figure 2.2a illustrates how a set of three points can always be shattered. The three colored lines represent the separation into halfplanes that contain the subsets of size 1 and 2. Of course it is easy to also find halfplanes that contain all or none of the points, hence the points in the example are shattered. On the other hand, Figure 2.2 illustrates the important case for the argument why 4 points can not be shattered. There is no straight line such that cuts off points a and d from the others, i.e., there is no halfplane that intersects only points a and d, but not c and b. Thus, these 4 points can not be shattered. It is not difficult to see that the argument holds for any combination of 4 points. △

The connection between our work, and uses of VC dimension in learnability and geometry has not yet been properly explored. Both applications still illustrate that bounded VC dimension is a very liberal restriction and classes of unbounded VC dimension are highly unnatural, especially in the context of conjunctive queries.

Finally, we recall important notions with respect to the separability of hypergraphs. Consider a hypergraph H and let $S \subseteq V(H)$. A set C of vertices with $C \subseteq (V(H) \setminus S)$ is $[S]$-*connected* if for any two distinct vertices $v, w \in C$, there exists a sequence of vertices $v = v_0, \ldots, v_h = w$ and a sequence of edges $e_0, \ldots, e_{h-1}$ $(h \geq 0)$ such that $\{v_i, v_{i+1}\} \subseteq (e_i \setminus S)$, for each $i \in \{0, \ldots, h-1\}$. A set $C \subseteq V(H)$ is an $[S]$-*component*, if C is maximal $[S]$-connected. Such a vertex set S that is used to split a hypergraph into components is referred to as a *separator*. Note that a separator S also gives rise to disjoint subsets of E with $E_C := \{e \in E(H) \mid e \cap C \neq \emptyset\}$. The *size of an* $[S]$-*component* C is defined as the number of edges in E_C. We call S a *balanced separator* if all $[S]$-components of H have size $\leq \frac{|E(H)|}{2}$.

2.3 Conjunctive Queries

Conjunctive queries can be considered the fundamental query language of database theory. Logically, they correspond to the fragment of first-order logic that only allows existential quantification and conjunction (without functions and equality). In practice, they are equivalent to Select-From-Where queries in SQL [3] and Basic Graph Patterns in SPARQL [99]. In the later parts of this thesis we also study CQs with negation (CQ$^\neg$s), a more general fragment of first-order logic. We therefore begin by defining CQ$^\neg$s and then define CQs as a special case. As a result, the definition of CQs given here takes a slightly different form than standard presentations while remaining equivalent to them. We refer to Abiteboul, Vianu, and Hull [3] for standard definitions and further discussion of CQs.

A *signature* σ is a finite set of relation symbols with associated arities. We write $ar(R)$ for the arity of relation symbol R. A *database* D (over signature σ) consists of a *finite* domain Dom and a relation R^D for each relation symbol R in the signature.

A *conjunctive query with negation* (over signature σ) is a set of literals. A literal is of the form $L(v_1, \ldots, v_m)$ where $v_1, \ldots, v_m$ are variables and L is either R or $\neg R$ for any m-ary relation symbol R in σ. If L is of the form R we call the literal *positive*, otherwise, if it is of the form $\neg R$ we say that the literal is *negative*. Note that in our setting, we consider algorithmic problems for conjunctive queries with negation to have the domain as part of the input. We can therefore always introduce explicit inequality relations in $O(|D|^2)$ time and they require no special consideration when considering polynomial time decidability.

A *conjunctive query* on the other hand is a set of *only positive* literals, which we also refer to as *atoms*. When the usage is clear from context, we commonly refer to both CQs and CQs with negation simply as *queries*. Note that we never explicitly provide the domain for CQs without negation. In those cases the domain is always the *active* domain, i.e., exactly the set of all values that occur in the database. We write $\text{vars}(q)$ for the set of all variables that occur in the literals of a CQ$^\neg$ q. We sometimes denote queries like logical formulas, i.e., $R_1(\vec{v_1}) \wedge \cdots \wedge R_n(\vec{v_n})$ with the understanding that the query is simply the set of all conjuncts.

Let q be a query and D a database over the same signature. We call a function $a\colon \mathrm{vars}(q) \to \mathit{Dom}$ an *assignment* for q. For a set of variables X we write $a[X]$ for the assignment with domain restricted to X. In a slight abuse of notation we also write $a[\vec{v}]$ for the tuple $(a(v_1), \ldots, a(v_n))$ where $\vec{v} = (v_1, \ldots, v_n)$ is a sequence of variables. An *extension* of an assignment $a\colon \mathrm{vars} \to \mathit{Dom}$ is an assignment $a'\colon \mathrm{vars}' \to \mathit{Dom}$ with $\mathrm{vars}' \supset \mathrm{vars}$ and $a(v) = a'(v)$ for every variable $v \in \mathrm{vars}$.

We say that the assignment *a satisfies a positive literal* $R(\vec{v})$ if $a[\vec{v}] \in R^D$. Similarly, *a satisfies a negative literal* $\neg R(\vec{v})$ if $a[\vec{v}] \notin R^D$. An assignment *satisfies a query* q (over database D) if it satisfies all literals of q. We write $q(D)$ for the set of all satisfying assignments for q over D. We can now define two of the main decision problem that are studied in this thesis.

BoolCQ	
Instance:	A CQ q and a database D
Question:	$q(D) \neq \emptyset$?

BoolCQ$^\neg$	
Instance:	A CQ$^\neg$ q and a database D
Question:	$q(D) \neq \emptyset$?

Note that this problem is generally referred to as *boolean* CQ evaluation. It is common to explicitly distinguish between variables that are part of the *answer* of a query and the other variables of the query. In such a setting, a *boolean* query is one with no explicit output variables, i.e., it can only return the empty set or no result. In our setting, we do not consider explicit answer variables as a matter of simplicity. In our results for CQs without negation, the extension to allow for an explicit enumeration of solutions follows immediately from [81]. Moreover, hardness in our setting implies hardness with output variables by straightforward reduction. However, our results for CQ$^\neg$ can not easily be extended to enumeration of answers as will be discussed in Chapter 5.

The notion of a *homomorphism* between CQs will play an important role in this thesis. The notion exists throughout mathematics and, abstractly, is a structure-preserving mapping from one object to another. Let q and q' be CQs with the same signature, we say that $h : vars(q) \to vars(q')$ is a homomorphism, if for every relational symbol R and every atom $R(x_1, \ldots, x_n)$ in q, there is an atom $R(h(x_1), \ldots, h(x_n))$ in q'.

Example 2.3. Consider queries $q := R(x, y) \wedge R(x, z), \wedge R(v, w)$ and $q' := R(u, v)$. The mapping h that maps $x \mapsto u$, $y \mapsto v$, $v \mapsto u$, and $z \mapsto v$ is a homomorphism from q to q'. It is easy to verify that $R(h(x), h(y))$, $R(h(x), h(z))$, $R(h(v), h(w))$ all result in $R(u, v)$, the atom in q'. Note that if q had an additional atom $S(x)$, there would be no homomorphism from q to q', since there is no atom with symbol S at all in q'. △

For two CQs (without negation) q and q', we say that q is *homomorphically equivalent* to q', or $q \simeq q'$, if there exists a homomorphism from q into q' and vice versa. The *core* of a CQ q, denoted $core(q)$, is the minimal CQ (with regards to the number of atoms) that is homomorphically equivalent to q. It is not hard to verify that every query has a unique (up to isomorphism) core [31]. For a class of CQs $\boldsymbol{Q}$, we write $core(\boldsymbol{Q})$ for the class of cores of the CQs in $\boldsymbol{Q}$.

We say that a CQ q is *contained* in another CQ q' if for every database D we have that if $q(D) \subseteq q(D')$. It is well known that q is contained in q' if and only if there exists a homomorphism from q' to q [31]. If two CQs q and q' are contained within each other, we say that they are *semantically equivalent* (we write $q \equiv q'$). Hence, if $q \equiv q'$ then for every database D we have that $q(D) \subseteq q'(D)$ and $q'(D) \subseteq q(D)$, i.e., both queries have the exact same set of solutions on every database. Furthermore, note that we have $q \equiv q'$ if and only if $q \simeq q'$, i.e., homomorphic equivalence equals semantic equivalence. In particular, q is always equivalent to $core(q)$.

A query q has an *associated hypergraph* $H(q)$. The vertices of $H(q)$ are the variables of q. Furthermore, $H(q)$ has an edge $\{v_1, \ldots, v_n\}$ if and only if there exists a literal $R(v_1, \ldots, v_n)$ or $\neg R(v_1, \ldots, v_n)$ in q.

In Chapter 5 we study the evaluation of CQ$^\neg$s in detail. To simplify arguments there we will assume that every relation symbol occurs only once in a query in

that chapter. We will therefore sometimes write the relation symbol, without the variables, to identify a literal. Note that every instance of BoolCQ$^\neg$ can be made to satisfy this property, by copying and renaming relations, in linear time.

Importantly, we do not make this assumption in Chapter 3. The difference in setting is subtle but important. In Chapter 3 we show that queries of high width (which may contain duplicate relation symbols), may in fact be equivalent to queries of lower width. The transformation from the previous paragraph takes away our chance to recognize this fact.

While we focus on the study of conjunctive queries in this thesis, our work also applies to further prominent problems via their equivalence to CQs. In particular, BoolCQ can equivalently be seen as the task of finding a homomorphism from the relational structure representing the query (also referred to as the tableau of the query) into the database. A further important equivalent problem is finding solutions of *constraint satisfaction problems* (CSPs) [108]. Finally, CQs and CQ$^\neg$s have a natural logical equivalent. When we view the database as a theory of ground facts then query answering is equivalent to the *model checking* problem for the formula that corresponds to the query, under the respective theory that corresponds to the database.

Finally, for our algorithmic considerations we assume a reasonable representation of queries and databases. In particular we assume that a relation R has a representation of size $\|R\| = O(|R| \cdot ar(R) \cdot \log |Dom|)$. Accordingly, we assume the *size of a database* D as $\|D\| = \|Dom\| + \sum_{R \in \sigma} \|R\|$ and the *size of a query* q as $\|q\| = O\left(\sum_{R \in \sigma} ar(R) \log |vars(q)|\right)$. We refer to the cardinality of the largest relation in D as $|R_{max}(D)| = \max_{R \in \sigma} |R^D|$. When the database is clear from the context we write just $|R_{max}|$.

With the size of a query now defined we can also define the main parameterized problem that we study here. In particular, we are interested in the BoolCQ problem parameterized by the size of the query. In a database context this parameterization is well motivated, the query is usually many magnitudes smaller than the database. This parameterization can also be seen as a problem inbetween the study of combined complexity (i.e., query and database are part of the input), and data complexity (i.e., the query is considered constant).

p-BoolCQ	
Instance:	A CQ q and a database D
Parameter:	$\|q\|$
Question:	$q(D) \neq \emptyset$?

We will be interested in restricting the queries of the instances to specific classes of queries. To this end we write BoolCQ($\mathcal{Q}$) to mean the BoolCQ problem with the queries in the instance restricted to queries in some class $\mathcal{Q}$. The same convention applies to $\texttt{BoolCQ}^{\neg}$ and all similar decision problems. Furthermore, for a class $\mathcal{H}$ of hypergraphs, let CQ[$\mathcal{H}$] denote all CQs whose hypergraphs are in $\mathcal{H}$. We will abbreviate the problem BoolCQ(CQ[$\mathcal{H}$]) to BoolCQ($\mathcal{H}$), i.e., BoolCQ restricted to those instances whose hypergraphs are in $\mathcal{H}$. The analogue applies to p-BoolCQ.

Unions of Conjunctive Queries

An instance of the *unions of conjunctive queries* (UCQ) problem is a set of CQs $\{q_1, \ldots, q_n\}$, we write $\bigcup_{i=1}^{n} q_i$, and a database D. We say that an instance of the UCQ problem $(\bigcup_{i=1}^{n} q_i, D)$ has a solution if for any $1 \leq i \leq n$, we have that $q_i(D) \neq \emptyset$ has a solution. The accompanying parameterized decision problem is the following

p-BoolUCQ	
Instance:	A UCQ $U = \bigcup_{i=1}^{n} q_i$ and a database D.
Parameter:	$\sum_{i=1}^{n} \|q_i\|$
Question:	Does U, D have a solution?

As for our previous problems, we write p-BoolUCQ($\mathcal{U}$) for the problem restricted to UCQs from a class $\mathcal{U}$.

As for CQs, the equivalence of UCQs will be of interest. We say that two UCQs $U = \bigcup_{i=1}^{n} q_i$ and $U' = \bigcup_{i=1}^{m} q'_i$ are *semantically equivalent* (we write $U \equiv U'$) if for every database D, (U, D) has a solution if and only if (U', D) has a solution.

A UCQ $\bigcup_{i=1}^{n} q_i$ is *non-redundant* if there are no q_i and q_j ($i \neq j$) such that q_i is contained in q_j. Note that every UCQ can be transformed into an equivalent non-redundant UCQ by repeated deletion of CQs that are contained by other CQs in the UCQ [105]). We write $nr(U)$ for the UCQ obtained by applying this procedure to make an UCQ U non-redundant. Importantly, since this procedure only deletes CQs we have $nr(U) \subseteq U$.

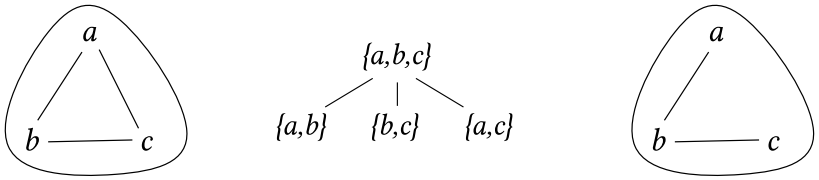
Example 2.4. Consider a UCQ $\{q_1, q_2, q_3, q_4\}$ where $q_1 \equiv q_2$ and $q_2 \subseteq q_3$ (thus, also $q_1 \subseteq q_3$). Recall, this means that $q_1(D) = q_2(D)$ and $q_2(D) \subseteq q_3(D)$ for every database D. Hence, the set of solutions $q_3(D) \cup q_4(D)$ is always the same as $q_1(D) \cup q_2(D) \cup q_3(D) \cup q_4(D)$, for any database D. This illustrates how deleting contained queries will always yield an equivalent UCQ. It is therefore easy to see that $nr(U) = \{q_3, q_4\}$ in this example.

The general procedure to obtain $nr(U)$ can be realized by two simple nested loops. The outer loop iterates over the individual CQs q of a UCQ U, and the inner loop checks whether $q \subseteq q'$ for all $q' \neq q$ in U. If the check succeeds for any q', then q is deleted from U. As illustrated in the example above, deletion of q makes no difference for the set of solutions of the UCQ. At the end, the queries that are left are clearly not contained in any other query, i.e., $nr(U)$.

As discussed above, if CQ q is contained in CQ q', then there exists a homomorphism from q' to q. Therefore, the containment check in the procedure outlined above can be realized by a homomorphism check. △

2.4 Hypergraph Acyclicity, Hypertree Width and Beyond

Earlier we introduced some basic properties of hypergraphs such as rank and degree. In this section we focus on the central structural properties of this thesis, acyclicity and its generalizations. As mentioned before, in hypergraphs there exists more than one reasonable definition of acyclicity. In this thesis we only consider the two most general such notions, α- and β-acyclicity. Note that all notions of hypertree acyclicity have numerous equivalent definitions (see e.g., [50, 24]). Here we recall only those that are important in the context of this thesis.



(a) α-acyclic, not β-acyclic (b) Join tree for Figure 2.3a. (c) α-acyclic and β-acyclic

Figure 2.3: Illustrations for Example 2.5

A *join tree* of H is a pair (T, ϵ) where T is a tree and $\epsilon : T \to E(H)$ is a bijection from the nodes in T to the edges of H such that the following holds: for every $v \in V(H)$ the set $\{u \in T \mid v \in \epsilon(u)\}$ is a subtree of T. If H has a join tree, then we say that H is *α-acyclic*.

A (weak) *β-cycle* is a sequence $(e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n, e_{n+1})$ with $n \geq 3$ where $e_1, \ldots, e_n$ are distinct hyperedges, $e_1 = e_{n+1}$, and $v_1, \ldots, v_n$ are distinct vertices. Moreover, for all $i \in [n]$, v_i is in e_i and e_{i+1} and not in any other edge of the sequence. A hypergraph is *β-acyclic* if it has no β-cycle.

An alternative (equivalent) definition of β-acyclicity is that H is β-acyclic if and only if all subhypergraphs of H are α-acyclic. In this paper, a third characterization of β-acyclicity will be important. We call a vertex v of H a *nest-point* if $I(v)$ is linearly ordered by $\subseteq$. We can then characterize β-acyclicity by a kind of elimination order for nest-points (this will be made more precise for a more general case in Definition 5.2).

Proposition 2.1 ([47]). *A hypergraph H is β-acyclic if and only if the empty hypergraph can be reached by successive removal of nest-points and empty-edges from H.*

Example 2.5. The difference between α- and β-acyclicity is of particular importance in this thesis. An important part of this difference in Figure 2.3. The hypergraph in Figure 2.3a is α-acyclic, but not β-acyclic. A join tree is for the hypergraph is given in Figure 2.3a. Clearly, $(\{c, a\}, a, \{a, b\}, b, \{b, c\}, c, \{c, a\})$ is a β-cycle. It is important to observe that – when considering α-acyclicity – cycles in the hypergraph can be ignored when they are covered by large edges. In this case the edge $\{a, b, c\}$ allows us to ignore the cycle inside.

The hypergraph in Figure 2.3c is also β-acyclic. This can be seen either by Proposition 2.1 (eliminate a or c first, then any sequence works), or directly by observing that the edge $\{a, b, c, \}$ can not close the cycle since vertices are only allowed be in two edges of the cycle and the edges of the cycle are distinct. Hence, if we have a sequence $\{a, b\}, b, \{b, c\}$, then the edge $\{a, b, c\}$ can not be used to extend the sequence to create a β-cycle since b is then contained in three edges of the sequence. △

Join trees have been successfully generalized to hypertree decompositions. A *hypertree decomposition* [68] of a hypergraph H is a tuple $\langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$, where T is a rooted tree, for every node u of the tree, $B_u \subseteq V(H)$ is called the *bag* of node u, and $\lambda_u \subseteq E(H)$ is the *cover* of u. Furthermore, a hypertree decomposition $\langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ must satisfy the following properties.

1. The subgraph $T_v = \{u \in T \mid v \in B_u\}$ for vertex $v \in V(H)$ is a tree.
2. For every $e \in E(H)$ there exists a $u \in T$ such that $e \subseteq B_u$.
3. For every node u in T it holds that $B_u \subseteq \bigcup \lambda_u$.
4. Let T_u be the subtree of T rooted at node u and let $B(T_u)$ be the union of all bags of nodes in T_u. For every node u in T it holds that $\bigcup \lambda_u \cap B(T_u) \subseteq B_u$.

The first property is commonly referred to as the *connectedness condition* and the fourth property is called the *special condition*. The *hypertree width* (hw) of a hypertree decomposition is $\max_{u \in T}(|\lambda_u|)$ and the hypertree width of H ($hw(H)$) is the minimal width of all hypertree decompositions of H.

If we exclude the special condition in the above list of properties, we obtain the definition of a *generalized hypertree decomposition* [69]. The *generalized hypertree width* of hypergraph H ($ghw(H)$) is defined analogously to before as the minimal width of all generalized hypertree decompositions of H. A more rigorous definition in follows bellow.

It is known that $hw(H) = 1$ if and only if H is α-acyclic [68]. Analogous to the definition of β-acyclicity in terms of every subhypergraph being α-acyclic, Gottlob and Pichler [71] introduced *β-hypertree width* β-hw $(H) =$

$\max\{hw(H') \mid H' \text{ is a subhypergraph of } H\}$. Note that we therefore also have $\beta\text{-}hw(H) = 1$ if and only if H is β-acyclic.

While hypertree width is conceptually and historically an important foundation for this work, we are mostly interested in width notions that generalize α-acyclicity even beyond hypertree width. We follow the framework proposed by Adler [5] and define these widths via *tree decompositions*. A tuple $\langle T, (B_u)_{u \in T} \rangle$ is a *tree decomposition* of a hypergraph H if T is a tree, every B_u is a subset of $V(H)$ and the following two conditions are satisfied:

1. For every $e \in E(H)$ there is a node $u \in T$ s.t. $e \subseteq B_u$, and
2. for every vertex $v \in V(H)$, $\{u \in T \mid v \in B_u\}$ is connected in T.

For functions $f\colon 2^{V(H)} \to \mathbb{R}^+$, the *$f$-width* of a tree decomposition is defined as $\sup\{f(B_u) \mid u \in T\}$ and the f-width of a hypergraph is the minimal f-width over all its tree decompositions. Let $\mathcal{F}$ be a class of functions from subsets of $V(H)$ to the non-negative reals, then the $\mathcal{F}$-width of H is $\sup\{f\text{-width}(H) \mid f \in \mathcal{F}\}$. All such widths are implicitly extended to CQs, UCQs, and other structures by taking the width of their respective hypergraphs.

The following properties of functions $f\colon 2^{V(H)} \to \mathbb{R}^+$ are important:

- f is *monotone* if $X \subseteq Y$ implies $f(X) \leq f(Y)$.
- f is called *edge-dominated* if $f(e) \leq 1$ for every $e \in E(H)$.
- f is called *modular* if $f(X) + f(Y) = f(X \cap Y) + f(X \cup Y)$ holds for every $X \subseteq V(H)$.
- f is called *submodular* if $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$ holds for every $X, Y \subseteq V(H)$.

We call a function $\gamma : E(H) \to \mathbb{R}^+$ an (fractional) *edge weight function*. When the co-domain is restricted to the set $\{0, 1\}$ we speak of an *integral edge weight function*. For an edge weight function γ we define the set $B(\gamma)$ of vertices *covered* by γ as

$$B(\gamma) = \{v \in V(H) \mid \sum_{e \in I_v} \gamma(e) \geq 1\}$$

We say that every subset of $B(\gamma)$ is *covered* by the weight function γ. The *weight* of an edge weight function is simply the sum of all its individual edge weights, i.e., $\text{weight}(\gamma) := \sum_{e \in E(H)} \gamma(e)$. A further important attribute of an edge weight function in this thesis is its *support*. The support consists of all edges that are assigned non-zero weight, i.e.,

$$\text{supp}(\gamma) = \{e \in E(H) \mid \gamma(e) > 0\}$$

For a set of vertices $X \subseteq V(H)$, let $\rho_H(X)$ be the minimal weight of any integral edge cover of X by edges in $E(H)$ and $\rho^*_H(X)$ the minimal weight of any fractional edge cover of X by edges in $E(H)$. In particular we write $\rho^*(H)$ for $\rho^*_H(V(H))$, which we refer to as the *fractional cover number* of H.

The f-width framework now allows us to define many of the important widths in the current literature uniformly.

(Primal) Treewidth of H [102]: $tw(H) := c$-width, where $c(X) = |X| - 1$.

Generalized hypertree width of H [68]: $ghw(H) := \rho_H$-width.

Fractional hypertree width of H [73]: $fhw(H) := \rho^*_H$-width.

Adaptive width of H [91]: $adw(H) := \mathcal{F}$-width(H), where $\mathcal{F}$ is the set of all monotone, edge-dominated, modular functions b on $2^{V(H)}$ with $b(\emptyset) = 0$. (Equivalently, $\mathcal{F}$ can be defined as the set of all functions $b\colon 2^{V(H)} \to \mathbb{R}^+$ obtained as $b(X) = \sum_{v \in X} f(x)$, where f is a fractional independent set of H.)

Submodular width of H [92]: $subw(H) := \mathcal{F}$-width(H), where $\mathcal{F}$ is the set of all monotone, edge-dominated, submodular functions b on $2^{V(H)}$ with $b(\emptyset) = 0$.

A notable omission, that is not expressible through this notion of f-width, is *hypertree width (hw)* [68], which uses the same width function as *ghw* but imposes an additional restriction on the tree decomposition. Note that these widths spawn a hierarchy in the sense that the following inequality holds for all hypergraphs H:

$$subw(H) \leq fhw(H) \leq ghw(H) \leq hw(H) \leq tw(H) + 1$$

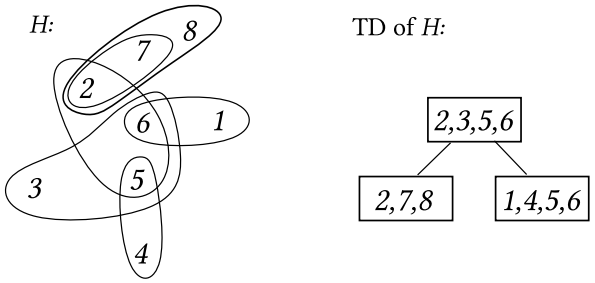


Figure 2.4: Illustration for Example 2.6

All hypergraph widths naturally extend to CQs through their hypergraphs: For a CQ q, the width of q is the width of the hypergraph of q.

Example 2.6. Consider the hypergraph H and TD of H as in Figure 2.4 (the labels of the tree represent the respective bags). To verify that this is indeed a TD of H we need to verify the two conditions stated above. First, every edge in H must be completely contained in the bag of some node. This condition is straightforward to verify, e.g., edges $\{2, 6, 5\}$ and $\{3, 5, 6\}$ are contained in the root node. Recall, the second condition (called *connectedness*) states that for every vertex $v \in V(H)$, $\{u \in T \mid v \in B_u\}$ is connected in T. For vertices, $1, 3, 4, 7, 8$ the condition is trivially satisfied as they occur only in one bag. For the others we need to check that the set of nodes where they occur, is connected (by edges of T). Consider vertex 2, which occurs in the root and the left child, those two nodes are connected and thus T_2 satisfies the condition. In fact, in the given TD, the only way to break connectedness is if a vertex occurs only in both the left and right child, but not in the root. This never occurs and therefore the connectedness condition holds. Note also that the bags $\{1, 4, 5, 6\}$ and $\{2, 3, 5, 6\}$ both have fractional (and integral) cover number 2. Hence, the *ghw* and *fhw* of this decomposition if 2 while its treewidth is 3. △

In Chapter 4 we investigate the construction of concrete decompositions that witness low *ghw* or *fhw*. This involves a detailed investigation of the structure of such decompositions and in particular of the possible ways that their bags can be covered while adhering to some width requirement. In that setting, it is useful to also associate specific covers to every bag. In those cases we will add the covers as a third element to our tree decomposition tuple in

the following way. A *generalized hypertree decomposition* (GHD) is a tuple $\langle T, (B_u)_{u\in T}, (\lambda_u)_{u\in T}\rangle$, where $\langle T, (B_u)_{u\in T}\rangle$is a tree decomposition and for every node u, λ_u is an (integral) edge cover of B_u. In a slight abuse of notation we treat the integral edge covers λ_u as sets in the sense that an edge e is in the set λ_u if and only if $\lambda_u(e) = 1$. A *fractional hypertree decomposition* (FHD) is a tuple $\langle T, (B_u)_{u\in T}, (\lambda_u)_{u\in T}\rangle$, where $\langle T, (B_u)_{u\in T}\rangle$is a tree decomposition and for every node u, γ_u is an fractional edge cover of B_u. The width of a GHD or FHD is the maximum weight over the respective covers in the decomposition.

For a class of hypergraphs $\mathcal{H}$, we say $\mathcal{H}$ has bounded width if there exists a constant k such that every hypergraph in $\mathcal{H}$ has width $\leq k$. Just like with widths on single hypergraphs, this definition applies analogously to classes of queries, which are said to have bounded width if every query in the class adheres to some constant width bound. The computational complexity of CQ answering is tightly linked to this hierarchy of parameters. This connection is summarized by the following two important propositions.

Proposition 2.2 ([73]). *Let $\mathcal{Q}$ be a class of CQs of bounded fhw. Then BoolCQ($\mathcal{Q}$) is tractable.*

Proposition 2.3 ([92]). *Let $\mathcal{H}$ be a recursively enumerable class of hypergraphs. Assuming the Exponential Time Hypothesis, p-BoolCQ($\mathcal{H}$) is fixed-parameter tractable if and only if $\mathcal{H}$ has bounded submodular width.*

We will make some comparisons to a further well-studied width notion for hypergraphs called *clique width*. Like tree-width, clique width is a width measure for graphs. Like with treewidth, we consider clique width of either primal graphs or of incidence graphs when talking about the clique width of a hypergraph. In the context of this thesis no further technical details of the width are necessary and we refer to [38] for full definitions.

CHAPTER 3

Semantic Width and the Parameterized Complexity of Conjunctive Queries

In Section 1.4 we discussed the role of Grohe's and Marx's characterization theorems for CQ answering. Despite their unquestionable importance, Grohe's and Marx's characterizations do not answer our research question. Instead, we introduce a new parameter – *semantic submodular width* (sem-*subw*) – to capture the minimal submodular width over the (infinite) equivalence class of semantically equivalent CQs.

Utilizing bounded semantic submodular width algorithmically will require finding at least one of the equivalent queries with low submodular width. In an initial step we thus follow Barceló, Pieris, and Romero who introduced semantic generalized hypertree width [17]. Akin to their characterization of semantic *ghw* in terms of the *ghw* of the core of the query (i.e., the smallest equivalent query), we show similar characterizations for more general semantic widths

here. These results are presented in Section 3.1. Furthermore, in section 3.2 we handle the special case of semantic *hypertree* width, which behaves differently from the other semantic widths discussed here. There, the width is not necessarily minimal in the core, by adding specific atoms in a way that preserves equivalence we can lower the hypertree width of any query until it equals the semantic *generalized* hypertree width.

In consequence of our results for semantic submodular width, we see that it is possible to decide the sem-*subw* of a query and to find the minimal semantically equivalent CQ. Following that, we give a reduction from Marx's setting to ours, which allows us to prove a lower bound for CQ answering with unbounded semantic submodular width. As with Marx's characterization, our result assumes the Exponential Time Hypothesis [79]; a standard assumption of parameterized complexity. A full exposition of the result is given in Section 3.4.

Through the well-known equivalence of CQ answering to the homomorphism problem as well as CQ containment [83] and solving CSPs [88], our main result also applies to those important problem families. By adapting our notion of sem-*subw* from CQs to the more general notion of *unions of conjunctive queries* (UCQs) accordingly, we can also extend our characterization result to UCQs, an important and widely studied class of query languages in database theory [105, 11]. Our results for UCQs are presented in Section 3.5.

We then discuss how our results relate to plain (non-parameterized) complexity of CQ evaluation in Section 3.6. Here, a characterization of tractable restrictions for the uniform CQ problem remains an open question. Still, we are able to present some new results on the topic. Specifically, we propose the study of *non-exotic* classes of hypergraphs, which do not exhibit a certain kind of highly unnatural exponential growth in the structure of the hypergraph. We investigate the complexity of CQ answering in the *non-exotic* case and show that it is in fact tractable even under bounded semantic *fractional* hypertree width. Note that this is not a direct consequence of tractability of CQ answering under bounded *fhw* in combination with our characterization of sem-*fhw* since the core of a CQ (i.e., the structure that we know to have bounded *fhw*) can not necessarily be found in polynomial time.

For context we would like to note that the complexity of CQ answering is

also studied from another perspective, usually in the context of finding homomorphisms. In what is called the *non-uniform* version of the problem, one considers restrictions to the database relations. That is, relations are restricted to certain properties or specific domains. Here, in a classic result, Hell and Nesetril [77] gave an elegant characterization of PTime solvability. More recently, Bulatov [27] and Zhuk [114] were able to independently establish a powerful dichotomy theorem. However, results for the nonuniform case do not translate to the uniform problem.

Finally, the problems studied here are also of particular interest in the context of CSPs. CSPs are a fundamental problem of artificial intelligence. As a unifying formal framework, they play a foundational role in many areas of AI research, see e.g., [84, 94]. However, the unifying aspect of CSPs has not yet reached its full potential. While a CSP formulation of a problem allows for reuse of common algorithmic strategies and implementations [66, 45], results in computational complexity still often require individual investigation, with little help from the framework. A complexity characterization for CSP would allow researchers to finally leverage the CSP framework also for computational complexity results, hence greatly simplifying the study of all problems that can be formulated as CSPs. The consequences and wide-reaching applications of such a characterization motivate our central research question. Note that throughout this paper, the parameterized complexity of CSPs always refers to the problem parameterized by the size of its constraint scopes.

The contents of this chapter are an extension of a paper published in collaboration with Hubie Chen, Georg Gottlob, and Reinhard Pichler [35].

3.1 Core Minimality & Determining Semantic Width

We begin our investigation in this chapter with the study of *semantic width*. That is, we are interested in hypergraph width measures applied to CQs, but instead of only considering the hypergraph of a CQ q itself we want to know the lowest possible width of any CQ that is equivalent to q. In a sense this semantic width can be seen as the actual measure of structural complexity

of the task underlying the query instead of capturing the complexity of the concrete formulation of the task given by q.

We follow Barceló, Pieris, and Romero [17] who introduced the notion of *semantic generalized hypertree width* and define the following general notion of *semantic widths* of CQs.

Definition 3.1. Let Q be the class of all CQs and $w : Q \to \mathbb{R}^+$ be invariant under isomorphism. We define *semantic* w as $\texttt{sem-}w(q) := \inf\{w(q') \mid q' \equiv q\}$.

The definition highlights the main problem with semantic widths: in general we have to deal with an infinite equivalence class of queries. For $\texttt{sem-}ghw$, Barceló, Pieris, and Romero [17] were able to prove that for every query q it holds that $\texttt{sem-}ghw(q) = ghw(core(q))$, thus showing the $\texttt{sem-}ghw$ to be decidable (they also give tighter complexity bounds). However, for our larger aim of complexity characterization $\texttt{sem-}ghw$ seems inadequate since bounded fhw is more general than bounded ghw and still yields tractable fragments for CQ answering. Our aim in this section is therefore to show similar characterizations of semantic width results for further width measures. We are able to achieve this for all hypergraph width measures that are common in the study of CQs (see Theorem 3.2). Semantic hypertree width is a special case and behaves differently and is therefore discussed separately in Section 3.2.

Theorem 3.2. *For every conjunctive query q:*

1. $\texttt{sem-}\rho^*(q) = \rho^*(core(q))$
2. $\texttt{sem-}fhw(q) = fhw(core(q))$
3. $\texttt{sem-}adw(q) = adw(core(q))$
4. $\texttt{sem-}subw(q) = subw(core(q))$

We spend the rest of this section proving the theorem. To do so we propose the following simple framework of *core minimal* functions. We then proceed to show every width measure from Theorem 3.2 to in fact be core minimal.

Definition 3.3. Let $\mathcal{Q}$ be the class of all CQs. We call a function $w\colon \mathcal{Q} \to \mathbb{R}^+$ *core minimal* if it is invariant under isomorphisms and for any $q \in \mathcal{Q}$: $w(\mathit{core}(q)) \leq w(q)$.

Lemma 3.4. *Fix $k \geq 1$, and let w be a core minimal function. For every conjunctive query q the following are equivalent:*

1. *There exists a q' homomorphically equivalent to q with $w(q') \leq k$.*
2. *$w(\mathit{core}(q)) \leq k$.*

Proof. The core of q is always homomorphically equivalent to q and therefore the upward implication follows. For the downward implication we have $w(\mathit{core}(q')) \leq w(q')$ by the virtue of w being core minimal. If q' is homomorphically equivalent to q, then their cores must be isomorphic, thus $w(\mathit{core}(q)) = w(\mathit{core}(q')) \leq w(q') \leq k$. □

Lemma 3.5. *A function w is core minimal if and only if for all conjunctive queries q we have that* $\texttt{sem-}w(q) = w(\mathit{core}(q))$.

Proof. The implication from left to right is immediate from Lemma 3.4. For the other direction we observe that for any CQ q' where $q' \simeq q$ we have $\texttt{sem-}w(q') \leq w(q)$ by definition. Thus, from $q \simeq q$ we see $w(\mathit{core}(q)) = \texttt{sem-}w(q) \leq w(q)$. □

A *homomorphism* $G \to H$ for hypergraphs is a mapping $f\colon V(G) \to V(H)$ s.t. if $e \in E(G)$, then $\{f(v) \mid v \in e\} \in E(H)$. Function application is extended to hyperedges and sets of hyperedges in the usual, element-wise, fashion: for instance, for $e \in E(G)$, we write $f(e)$ to denote $\{f(v) \mid v \in e\}$. Likewise, for $E' \subseteq E(G)$, we write $f(E)$ to denote $\{f(e) \mid e \in E'\}$. Note that if two CQs are homomorphic, then also their associated hypergraphs are homomorphic, while the converse is, in general, not true.

Example 3.1. The above definition of homomorphism of hypergraphs does not imply homomorphic CQs mainly because there is no information on relation symbols in the hypergraph. A CQ $R(x, y) \wedge R(x, z)$ is not homomorphic to a

CQ $R(x, y) \wedge S(x, z)$, while both queries have the same hypergraph, meaning the identity function a homomorphism between their hypergraphs.

Note that relation symbols also have a fixed arity, whereas hyperedges have no such limitation. Consider the two CQs $R(a, b, c)$ and $R(x, y)$ and their respective hypergraphs H and H', with $E(H) = \{\{a, b, c\}\}$ and $E(H') = \{\{x, y\}\}$. Clearly the mapping $a \mapsto x, b \mapsto y, c \mapsto y$ is a homomorphism on hypergraph level but R with arity 3 is a different relation symbol as R with arity 2, thus there is no homomorphism between the CQs.

For the other direction, consider CQs q, q' with a homomorphism f from q to q'. It is easy to see that when an atom $R(x, y, z)$ in q maps to some $R(f(x), f(y), f(z))$ in q', then the edge $\{x, y, z\}$ in $H(q)$ can always be mapped to the edge $\{f(x), f(y), f(z)\}$ in the hypergraph of q'. △

Lemma 3.6. *Let G and H be two hypergraphs and let f be a homomorphism from G to H. Given a fractional edge cover $\mathbf{x}$ of G, define $\mathbf{x}'$ s.t.*

$$x'_h = \sum_{g \in f^{-1}(h)} x_g \qquad h \in E(H).$$

Then $\mathbf{x}'$ is a fractional edge cover of $f(V(G))$ with the same total weight as $\mathbf{x}$.

Proof. We will write I_v for the set of all incident edges of a vertex v. We first show that $\mathbf{x}'$ is fractional edge cover. In an initial step we show that for every $E \subseteq E(G)$, the $\mathbf{x}'$ weight of edges in $f(E)$ will always be greater or equal to the $\mathbf{x}$ weight of E. We will (briefly) abuse notation and write $f^{-1}(f(E))$ when we in fact refer to the *union of all the preimages*, i.e., the set of all the edges that map to edges in $f(E)$. It is then easy to observe $E \subseteq f^{-1}(f(E))$ and, therefore, we also have

$$\sum_{h \in f(E)} x'_h = \sum_{h \in f(E)} \sum_{g \in f^{-1}(h)} x_g \geq \sum_{g \in f^{-1}(f(E))} x_g \geq \sum_{g \in E} x_g.$$

Now, choose an arbitrary $w \in f(V(G))$ and any $v \in f^{-1}(w)$. In combination with our previous observation we can then conclude:

$$\sum_{h \in I_w} x'_h \geq \sum_{h \in f(I_v)} x'_h \geq \sum_{g \in I_v} x_g \geq 1$$

The leftmost inequality holds, because $f(I_v) \subseteq I_w$. The rightmost inequality holds, because we are assuming that $\mathbf{x}$ is a fractional edge cover of G. We have thus shown that $\mathbf{x}'$ covers w. Since $w \in f(V(G))$ was arbitrarily chosen, we conclude that $\mathbf{x}'$ is a fractional edge cover of $f(V(G))$.

To see that the total weights of both covers are the same, observe:

$$\sum_{h \in f(E(G))} x'_h = \sum_{h \in f(E(G))} \sum_{g \in f^{-1}(h)} x_g = \sum_{g \in E(G)} x_g$$

The right equality follows from the fact that every edge of G is present in exactly one set $f^{-1}(h)$. □

Lemma 3.7. *The fractional edge cover number ρ^* of a conjunctive query is core minimal.*

Proof. Let G be the hypergraph of q and H be the hypergraph of $core(q)$. Since there is a surjective homomorphism from q to $core(q)$, there exists a surjective homomorphism from G to H. Then, by Lemma 3.6, for any fractional edge cover of G there exists a cover of H with equal weight. □

Lemma 3.8. *The functions fhw, adw, and subw are core minimal.*

Proof. Let q be a CQ and f an endomorphism from q to $core(q)$. W.l.o.g., we may assume $f(v) = v$ for all $v \in f(q)$. This can be seen as follows: suppose that $f(v) = v$ does not hold for all $v \in f(q)$. Clearly, f restricted to $core(q)$ must be a variable renaming. Hence, there exists the inverse variable renaming $f^{-1}\colon core(q) \to core(q)$. Now set $f^* = f^{-1}(f(\cdot))$. Then $f^*\colon q \to core(q)$ is the desired endomorphism from q to $core(q)$ with $f^*(v) = v$ for all $v \in f^*(q)$.

Let $H = (V(H), E(H))$ denote the hypergraph of q and $H' = (V(H'), E(H'))$ the hypergraph of $core(q) = f(q)$. Furthermore, let $(T, (B_u)_{u \in V(T)})$ be a tree decomposition of H. Then we create $(T, (B'_u)_{u \in V(T)})$ with the same structure as the original decomposition and $B'_u = B_u \cap V(H')$. This gives us a tree decomposition of H': for every edge $e \in E(H')$ with $e \subseteq B_u$, also $e \subseteq B_u \cap V(H')$ holds, because $e \subseteq V(H')$. Removing vertices completely from a decomposition cannot violate the connectedness condition. Actually, some bags B'_u might become empty but this is not problematic: either we simply allow empty bags in

the definition of the various notions of width; or we transform $(T, (B'_u)_{u \in V(T)})$ by deleting all nodes u with empty bag from T and append every node with a non-empty bag as a (further) child of the nearest ancestor node with non-empty bag.

fhw: We show that if $(T, (B_u)_{u \in V(T)})$ has ρ^*_H-width k, then $(T, (B'_u)_{u \in V(T)})$ has $\rho^*_{H'}$-width $\leq k$: By assumption, there is a fractional edge cover γ_u of every set B_u with weight $\leq k$. By Lemma 3.6, there exists a cover γ'_u of $f(B_u)$ with weight $\leq k$. What is left to show is that γ'_u also covers B'_u. Recall, that $f(v) = v$ for any $v \in V(H')$ and therefore $f(B_u \cap V(H')) = B_u \cap V(H')$. It then becomes easy to see that

$$B'_u = B_u \cap V(H') = f(B_u \cap V(H')) \subseteq f(B_u)$$

and in consequence γ'_u clearly also covers B'_u.

subw (and adw): Let $\mathcal{F}$ and $\mathcal{F}'$ be the sets of monotone, edge-dominated, submodular functions on $V(H)$ and $V(H')$ respectively. We show that for every $b' \in \mathcal{F}'$ there exists $b \in \mathcal{F}$, such that b'-width$(H') \leq b$-width(H):

Consider an arbitrary monotone, edge-dominated, submodular function $b' \colon 2^{V(H')} \to \mathbb{R}^+$ with $b'(\emptyset) = 0$. This function can be extended to a monotone, edge-dominated, submodular function $b \colon 2^{V(H)} \to \mathbb{R}^+$ on $V(H)$ by setting $b(X) = b'(X \cap V(H'))$ for every $X \subseteq V(H)$. Now, for any such b' let $(T, (B_u)_{u \in V(T)})$ be the tree decomposition for the original hypergraph with minimal b-width $= k$. Let $(T, (B'_u)_{u \in V(T)})$ refer to the tree decomposition of the core hypergraph, created by the procedure described above. Clearly $(T, (B'_u)_{u \in V(T)})$ has b'-width $= k$ because by construction $b'(B'_u) = b'(B_u \cap V(H')) = b(B_u)$ for every $u \in V(T)$.

Thus, for every monotone edge-dominated submodular function b' on the core hypergraph H', there exists a function b for H where b'-width$(H') \leq b$-width(H). As the submodular width is determined by the supremum over all permitted functions we see that $subw(H') \leq subw(H)$.

For adw observe that the definition of function b and the line of argumentation above still holds if we start off with a monotone, edge-dominated, *modular* function $b' \colon 2^{V(H)} \to \mathbb{R}^+$.

□

Theorem 3.2 now follows from a straightforward combination of Lemmas 3.8, 3.7, and 3.5.

3.2 A Special Case: Semantic Hypertree Width

Recall that the special condition of hypertree decompositions demands that if a vertex v occurs in an edge e in λ_u and in a bag in the subtree below u, then v must also appear in B_u. If this property is violated by an edge e in a GHD, we say that e causes a *special condition violation* (SCV) at node u.

The approach used for *fhw*, *adw*, *subw* in the previous section cannot be applied to hypertree width. Constructing a new tree decomposition by intersecting the bags with the vertices in H' can break the special condition.

Indeed, we will show that $\texttt{sem-}hw(q) = \texttt{sem-}ghw(q)$ for any CQ q. Our argument is based on a construction of equivalent structures that *repair* special condition violations in a generalized hypertree decomposition. This observation positions hypertree width uniquely against all other widths studied in the previous section.

Lemma 3.9. *Let q be a CQ with $ghw(q) \leq k$. Then, there exists a CQ q' with $q' \simeq q$ and $hw(q') \leq k$.*

Proof. Let $\mathcal{D} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ be a GHD of $H(q)$ with width k. We will show how to add a new tuple to q to get a new q' such that $q' \simeq q$ and $H(q')$ has a GHD with width k with *fewer* special condition violations (SCVs) than $\mathcal{D}$. Iterating this step will ultimately lead to a structure that is equivalent to q and has a GHD of width k with no SCVs, i.e., an HD.

For node u, we write T_u for the subtree rooted at u. Let u be any node in T where the special condition is violated, i.e., there is some $e^* \in \lambda_u$ such that $(e^* \cap B(T_u)) \not\subseteq (e^* \cap B_u)$. Let $v_1, \ldots, v_\ell$ be the vertices in e^* that are not in B_u and let R be the atom in q that that becomes the edge e^* in the hypergraph[1].

[1]There can be multiple tuples in different relations that correspond to the edge e^* in $H(q)$, it does not matter to which we apply the procedure.

Now, for every $i \in [\ell]$, create a fresh variable x_i. We create a new atom R' from R (using the same relation symbol, we only change the variables)by replacing every v_i (the vertices that witness the SCV) by the fresh x_i. All other variables of R are copied with no change to R'. Add R' to q to obtain the new structure q'.

We first verify that q' has a GHD $\mathcal{D}'$ with width k and one less SCV than $\mathcal{D}$. We can find such a $\mathcal{D}'$ by simply copying $\mathcal{D}$ and only updating γ_u and B_u as follows. In γ_u we replace e^* by the edge e' that corresponds to our newly created atom R'. To B_u we add all the newly created $x_1, \ldots, x_\ell$. With this change to the bag, every edge remains covered (and the new e' is now covered by B_u) and connectedness is unaffected (the new vertices $x_1, \ldots, x_\ell$ occur only in this bag). Furthermore, it is clear that we still have $B_u \subseteq \bigcup \lambda_u$ after this update. Finally, while e caused an SCV at node u, this is no longer the case with e' since e' is fully contained in B_u. Thus our new $\mathcal{D}'$ is a valid GHD of $H(q')$ with one less SCV than $\mathcal{D}$.

To finalize our argument we still need to show that $q \simeq q'$. First, since every relation in q' is a superset of a relation in q, the identity function is a homomorphism from q to q'. For the other direction, consider the function $f\colon \mathrm{vars}(q') \to \mathrm{vars}(q)$ that maps $x_i \mapsto v_i$ for $i \in [\ell]$ and every other constant in q' to itself. Clearly, the R' from our construction above maps to R under this homomorphism. For all other tuples, f is the identity function since they do not contain any of the fresh variables x_i. Thus, f is a homomorphism from q' to q.

We can therefore move along equivalent queries to (strictly) monotonically decrease the number of SCVs, ultimately yielding an HD with width k. □

Theorem 3.10. *For any CQ q it holds that* $\texttt{sem-}ghw(q) = \texttt{sem-}ghw(q)$.

Proof. Suppose $\texttt{sem-}ghw(q) = k$, then $ghw(core(q)) = k$ by Lemma 3.2. From Lemma 3.9 it now follows that there exists an q' such that $hw(q') \leq k$ and $q' \simeq core(q) \simeq q$. Thus, $\texttt{sem-}hw(q) \leq k$. On the other hand, in general for any hypergraph H we have $ghw(H) \leq hw(H)$ and therefore $\texttt{sem-}hw(q)$ can not be lower than $\texttt{sem-}ghw(q)$. Hence, $\texttt{sem-}hw(q) = k = \texttt{sem-}ghw(q)$. □

3.3 Why a Characterization on Hypergraph Level is not Enough

Many AI problems have natural CSP (and thus CQ) formulations and we ultimately wish to determine the computational complexity of all such problems through a characterization of the complexity of CSP. In this section we argue why a characterization on the hypergraph level (which ignores relation symbols), as in Proposition 2.3, is not enough for this goal. The main issue with the hypergraph characterization is that even though a CQ may have a highly complex hypergraph structure, it can still be easy to solve. Yet, the complexity of p-BoolCQ($\mathcal{H}$) expresses only the complexity of the worst-case query of the given structure. We illustrate this issue in the following example.

Consider the following problem: Given a directed graph G, can we embed (by a homomorphism) a bidirected $n \times n$-grid into G? The corresponding CQ and database $C_n = (q, D)$ have a single relation symbol E and $E^D = G$. As variables of q we have $\{x_{i,j} \mid i \in [n], j \in [n]\}$ and q contains exactly the following atoms specifying the $n \times n$-grid: $E(x_{i,j}, x_{i+1,j})$, $E(x_{i+1,j}, x_{i,j})$ for $i \in [n-1], j \in [n]$ and $E(x_{i,j}, x_{i,j+1})$, $E(x_{i,j+1}, x_{i,j})$ for $i \in [n], j \in [n-1]$.

We now consider the class C of all CQs $C_n(G)$ for $n \geq 1$ and all graphs G. The hypergraphs of C are, by definition, exactly the class of $n \times n$-grid graphs $\mathcal{G}_{n \times n}$, which is well-known to have unbounded treewidth [102]. In general, it is difficult to determine the submodular width of graphs since the definition depends on a supremum over an infinite class of functions. However, Lemma 3.11 below provides us with a convenient way to recognize that certain classes have unbounded submodular width.

Lemma 3.11. *Let H be an arbitrary hypergraph and let* $\mathrm{rank}(H)$ *be the maximum edge size in H, then*

$$tw(H) \leq \mathrm{rank}(H) \cdot subw(H)$$

Proof. Let $f : X \mapsto |X|/\mathrm{rank}(H)$ be a function on the subsets of $V(H)$. It is easy to verify that f is submodular, edge-dominated and monotone. For any node u of any tree decomposition of H we clearly have $|B_u| = \mathrm{rank}(H) \cdot f(B_u)$ and therefore also $tw(H) + 1 = \mathrm{rank}(H) \cdot f\text{-width}(H)$. Since f is submodular,

edge-dominated and monotone we also have $f\text{-width}(H) \leq subw(H)$ and the statement follows immediately. □

From Lemma 3.11 we can conclude that $\mathcal{G}_{n\times n}$ also has unbounded submodular width. From Proposition 2.3 we can thus only deduce that p-BoolCQ($\mathcal{G}_{n\times n}$) is not fixed-parameter tractable.

However, for every $C_n(G) = (q, D)$, we have that $core(q)$ is the query

$$core(q) \leftarrow E(x_1, x_2), E(x_2, x_1)$$

This is easy to verify, e.g., by observing that an undirected $n \times n$-grid is 2-colorable. Clearly, $(core(q), D)$ is solvable in polynomial time and it is equivalent to (q, D). It follows that p-BoolCQ(C) is in fact fixed-parameter tractable (and indeed tractable), despite the complexity of p-BoolCQ($\mathcal{G}_{n\times n}$). We see that a hypergraph level characterization has inherent shortcomings in establishing lower bounds for specific problem classes.

3.4 Characterizing Fixed-parameter Tractability of CQs

In this section we prove our characterization theorem for CQs. The discussion in Section 3.3 shows that unbounded submodular width can still allow for fixed-parameter tractable BoolCQ solving. Hence, we require a new, more general, property to fully capture fixed-parameter tractability. From the definition of *semantic submodular width* and the accompanying results in Section 3.1, we show that the characterization from Proposition 2.3 can indeed be strengthened to the following characterization of the fixed-parameter tractability of CQ instances.

Theorem 3.12. *Let $\mathbf{Q}$ be a recursively enumerable class of CQs. Assuming the Exponential Time Hypothesis, p-BoolCQ($\mathbf{Q}$) is fixed-parameter tractable if and only if $\mathbf{Q}$ has bounded semantic submodular width.*

Our proof of the theorem relies on two central precursors. First, we show how bounded semantic submodular width leads to fixed-parameter tractability. The

basic idea is simple, instead of solving a CQ instance with possibly arbitrarily high submodular width, we want to solve an equivalent instance with low width. However, it is not clear how to find such an equivalent instance and whether finding it is decidable. In Section 3.1, we show in Theorem 3.2, that the same connection also holds for the more complex cases of fractional hypertree width and submodular width. Note that for treewidth this property is trivial since treewidth is – in contrast to the widths considered in this chapter – hereditary, i.e., removing edges from a hypergraph can not increase its treewidth.

In the context of our main result, the most important consequence of Theorem 3.2 is that we are always able to find the equivalent queries with minimal submodular width by simply computing the core. In principle, finding the core of a CQ is intractable (formally, deciding if a query q' is the core of a query q, is DP-complete [51]). However, in our parameterized setting the computation of the core of q only depends on the parameter.

To establish a lower bound for classes with unbounded semantic submodular width we will make use of previous results from [36]. A step in our reduction will require an additional definition that helps us fix the domains of individual elements in the reduction. For a CQ q, let q^* be the expansion of q by an atom $U_v(v)$, where U_v is a fresh relation symbol, for every variable $v \in \text{vars}(q)$. For a class of CQs $\mathcal{Q}$ we write $\mathcal{Q}^*$ for $\{q^* \mid q \in \mathcal{Q}\}$. Our intention is to establish our lower bound by reduction from the hypergraph setting of Proposition 2.3. We will make use of the following two reductions.

Proposition 3.13 ([36])**.** *Let $\mathcal{Q}$ be a recursively enumerable class of conjunctive queries. Then*

$$p\text{-BoolCQ}(\text{core}(\mathcal{Q})^*) \leq p\text{-BoolCQ}(\mathcal{Q})$$

Lemma 3.14. *Let $\mathcal{Q}$ be a recursively enumerable class of conjunctive queries and let $\mathcal{H}^{\mathcal{Q}}$ be the class of hypergraphs of $\mathcal{Q}$.*

$$p\text{-BoolCQ}(\mathcal{H}^{\mathcal{Q}}) \leq p\text{-BoolCQ}(\mathcal{Q}^*)$$

Proof. Let (u, B) be an instance of $\text{BoolCQ}(\mathcal{H}^{\mathcal{Q}})$, i.e., u is the query and B the database, and let H be the hypergraph of q and Dom_u be the domain of the (u, B). Recall that a single edge can represent multiple atoms in p. For each edge

$e \in E(H)$, we consider the sets $F_{t_1}, \ldots, F_{t_k}$ of satisfying assignments $e \to \mathrm{Dom}_u$ for each of the atoms $A_1, \ldots, A_k$ of u that become edge e in the hypergraph. We then produce the set $F_e = \bigcap_{i=1}^{k} F_{t_i}$ of assignments that satisfy all the atoms for e at once. Observe that computing F_e for all $e \in E(H)$ is possible in polynomial time.

By definition there exists a CQ q^* in $\mathcal{Q}^*$ where q has hypergraph H. We can compute such a q^* by enumeration of $\mathcal{Q}$ until we find an q with a matching hypergraph and then computing q^* from q.

We will reduce (u, B) to (q^*, D) with domain Dom_q where D is constructed as follows. As the domain of D we take $\mathrm{vars}(q) \times \mathrm{Dom}_q$. Recall, that for each $v \in \mathrm{vars}(q)$ we have an atom $U_v(v)$. Let $U_v^D = \{(v, d) \mid d \in \mathrm{Dom}_q\}$. For each other relation symbol R of q and each atom $R(v_1, \ldots, v_k)$ in q, we add tuples $((v_1, f(v_1)), \ldots, (v_k, f(v_k)))$ to R^D where $f \in F_e$ and e is the hyperedge $\{v_1, \ldots, v_k\}$. Note that all atoms with the same relation symbol R contribute to the construction of R^D in this way.

We now show that (u, B) has a solution iff (q^*, D) has a solution. First, suppose $a \in u(B)$ and note that q^* and u have the same domain since q and u have the same underlying hypergraph. It is then not difficult to see that $a' : v \mapsto (v, a(v))$ is a satisfying assignment for q^* over D: For the unary atoms U_v^q, the image trivially exists in U_v^D. For the other relations, it is enough to observe that for every edge e of H, the assignment a restricted to variables in e must be in F_e.

For the other side, observe that any assignment a' from q^* to D must be of the form $v \mapsto (v, a(v))$. We argue that a is a satsifying assignment for u over B. As q and u have the same variables, a also applies to the variables of u. By definition of F_e we have that for every atom $R(\bar{x})$ in u, a maps to a tuple in R^B if $\bar{x}$ is covered by some edge of the hypergraph of q. Since the hypergraphs of q and u are the same, this holds for all atoms in u and therefore a is a satsifying assignment for u over B. □

Proof of Theorem 3.12. Let $\mathcal{H}^{core(\mathcal{Q})}$ be the class of hypergraphs of the CQs in $core(\mathcal{Q})$. We will show that the two decision problems p-BoolCQ($\mathcal{Q}$) and p-BoolCQ($\mathcal{H}^{core(\mathcal{Q})}$) are fpt-reducible to each other. In other words, we have that p-BoolCQ($\mathcal{Q}$) is fixed-parameter tractable if and only if the hypergraph

level problem p-BoolCQ($\mathcal{H}^{core(Q)}$) is fixed-parameter tractable. By Proposition 2.3 this is the case iff $\mathcal{H}^{core(Q)}$ has bounded submodular width. By Lemma 3.2, this is equivalent to Q having bounded semantic submodular width.

What is left, is to show the claim. First, we observe:

$$p\text{-BoolCQ}(Q) \leq p\text{-BoolCQ}(core(Q)) \leq p\text{-BoolCQ}(\mathcal{H}^{core(Q)})$$

The left reduction holds because (q, D) is equivalent to $(core(q), D)$ and computing the core is feasible in $f(\|q\|)$ time. The right reduction is trivial since all instances of p-BoolCQ($core(Q)$) are also instances of p-BoolCQ($\mathcal{H}^{core(Q)}$). For the other direction we get the intended reduction by straightforward combination of Lemma 3.14 and Proposition 3.13:

$$p\text{-BoolCQ}(\mathcal{H}^{core(Q)}) \leq p\text{-BoolCQ}(core(Q)^*) \leq p\text{-BoolCQ}(Q)$$

□

3.5 Characterizing Fixed-parameter Tractability of UCQs

We now extend the characterization in Theorem 3.12 from CQs to UCQs. To do so we first need to introduce a way to extend the relevant definitions to UCQs. For our width notions the natural extension to UCQs is through the maximum of its parts, i.e., for width function w and UCQ $U = \bigcup_{i=1}^{n} q_i$ let $w(U) := \max\{w(q_i) \mid i \in [n]\}$. Semantic width functions are defined the same as for CQs, i.e., $\texttt{sem-}w := \inf\{w(U') \mid U' \equiv U\}$. However, equivalence of UCQs is more complex than equivalence in CQs. In particular, the characterization by homomorphic equivalence is no longer applicable. Therefore, some additional effort is required to determine the analogue of Lemma 3.2. Using the following classic result by Sagiv and Yannakakis we can derive the fitting Lemma 3.16.

Proposition 3.15 ([105]). *Let $U = \bigcup_{i=1}^{n} q_i$ and $U' = \bigcup_{j=1}^{m} q'_j$ be non-redundant UCQs. Then $U \equiv U'$ if and only if for every q_i there is a unique q'_j such that $q_i \equiv q'_j$.*

Lemma 3.16. *Let U be an UCQ, then*

$$\text{sem-}subw(U) = \max\{subw(core(q_i)) \mid q_i \in nr(U)\}$$

Proof. It is clear that the right side of the equality is the *subw* of an UCQ that is equivalent to U. All that is to show is that this is in fact the minimal subw of an equivalent UCQ. For the sake of brevity we will write core-$subw(nr(U))$ for $\max\{subw(core(q_i)) \mid q_i \in nr(U)\}$ in the rest of the argument.

Proof is by contradiction. Suppose there exist a UCQ $V \equiv U$ where $subw(V)$ is less than core-$subw(nr(U))$. Since $V \equiv U$, clearly also $nr(V) \equiv nr(U)$. Furthermore, since $nr(V) \subseteq V$ (recall the construction of $nr(V)$) we also have $subw(nr(V)) <$ core-$subw(nr(U)$. Now, from Proposition 3.15 we have that for every $r_i \in nr(V)$, there is an equivalent $q_j \in nr(U)$. By Lemma 3.2 it follows that

$$subw(r_i) \geq \text{sem-}subw(r_i) = \text{sem-}subw(q_j) = subw(core(q_j))$$

for all such combinations of r_i and q_j. From the definition of *subw* for UCQs this then gives an direct contradiction of $subw(nr(V)) <$ core-$subw(nr(U))$. □

From Lemma 3.16 it is now easy to see, that for a class of UCQs $\mathcal{U}$ with bounded sem-*subw*, the p-BoolUCQ($\mathcal{U}$) problem is fixed-parameter tractable. For every U in $\mathcal{U}$ we can simply compute $nr(U) = \bigcup_{i=1}^{n} q_i$ and then solve the CQs $core(q_i)$ individually. In combination with Theorem 3.12 we see that this procedure is fixed-parameter tractable.

To establish the lower bound, we make use of previous work on the complexity of existential positive logic [33]. The result there is stated in a different setting but a translation is not difficult through the well-known equivalence of solving CQs and model checking of primitive positive first-order formulas.

Proposition 3.17 (Theorem 3.2 in [33])**.** *Let $\mathcal{U}$ be recursively enumerable class of non-redundant UCQs and let $\mathcal{Q}$ be the class of all individual CQs that make up the UCQs in $\mathcal{U}$. Then p-BoolCQ($\mathcal{Q}$) $\leq$ p-BoolUCQ($\mathcal{U}$).*

Theorem 3.18. *Let $\mathcal{U}$ be a recursively enumerable class of UCQs. Assuming the Exponential Time Hypothesis, p-BoolUCQ($\mathcal{U}$) is fixed-parameter tractable if and only if $\mathcal{U}$ has bounded semantic submodular width.*

Proof. For the case where $\mathcal{U}$ has bounded semantic submodular width we have already given a fixed-parameter tractable procedure for solving p-BoolUCQ($\mathcal{U}$) above. We will establish the lower bound by introducing the class $nr(\mathcal{U}) = \{nr(U) \mid U \in \mathcal{U}\}$ as an intermediate.

Suppose $\mathcal{U}$ has unbounded sem-*subw* and let $\mathcal{Q}$ be the class of all individual CQs that make up the UCQs in $nr(\mathcal{U})$. From Lemma 3.16 it follows that $nr(\mathcal{U})$ and $\mathcal{Q}$ both also have unbounded sem-*subw*. By Proposition 3.17 we have p-BoolCQ($\mathcal{Q}$) $\leq$ p-BoolUCQ($nr(\mathcal{U})$) and therefore, by Theorem 3.12, p-BoolUCQ($nr(\mathcal{U})$) can not be fixed-parameter tractable.

To finish the proof we show that p-BoolUCQ($nr(\mathcal{U})$) $\leq$ p-BoolUCQ($\mathcal{U}$). The reduction is straightforward, an instance (U, D) of p-BoolUCQ($nr(\mathcal{U})$) is reduced to the instance (U', D) of p-BoolUCQ($\mathcal{U}$) where $nr(U') \equiv U$. Such an U' can be found by enumeration of $\mathcal{U}$ in time that only depends on the parameter. Since $nr(U') \equiv U$, the reduction is trivially correct. □

3.6 Exotic Hypergraphs and Tractability

A characterization for the plain (non-parameterized) tractability of CQs remains an open question. Here we wish to highlight two consequences of our work and recent developments regarding the connection of fractional hypertree width and the Vapnik-Chervonenkis dimension of a hypergraph that was originally presented in [54].

Tractability in natural problem classes.

Bounded hypertree width (*hw*), generalized hypertree width (*ghw*) and fractional hypertree width (*fhw*) all represent sufficient conditions for tractable CQ answering, with *fhw* being the most general such property we know of. It is known that *hw* is bounded if and only if *ghw* is bounded [6]. Furthermore, there exist classes that exhibit bounded *fhw* but unbounded *hw* [73]. However, all known hypergraph classes with bounded *fhw* and unbounded *hw* involve some form of exponential growth that is unlikely to be present in natural problems. It has remained an open question if this exponential growth is essential for the separation of the two width measures.

Below, we give an answer to this question. While the results below also hold for bounded Vapnik-Chervonenkis dimension, we introduce the notion of *exotic* hypergraph classes, a (slightly more general) consequence of unbounded VC dimension, to focus on the exponential character of such classes. Instead of the technicality of shattered sets, the definition of exotic hypergraphs emphasizes the fact that, in a sense, the number of edges is exponential in the number of vertices. We are able to state that this property is indeed intrinsic to the separation of bounded *fhw* and *hw*. Alternatively, in the contrapositive, we see that for non-exotic classes, a class has bounded *fhw* if and only if it has bounded *hw*. In other words, bounded *fhw* does not allow for additional tractable cases over bounded *hw*.

Definition 3.19. Let $\mathcal{H}$ be a class of hypergraphs. We say that $\mathcal{H}$ is *exotic* if for every integer $n \geq 1$, there exists a $H \in \mathcal{H}$ with a set of n vertices $U \subseteq V(H)$ such that $H[U]$ has at least $2^n - 1$ distinct edges.

Lemma 3.20. *Let $\mathcal{H}$ be a hypergraph class. If $\mathcal{H}$ has unbounded VC dimension, then $\mathcal{H}$ is exotic.*

Proof. Assuming that $\mathcal{H}$ has unbounded VC dimension, we show for every integer $n \geq 1$ that there exists a hypergraph $H \in \mathcal{H}$ with a set of vertices $U \subseteq V(H)$ such that $H[U]$ has at least $2^n - 1$ distinct edges.

Suppose some fixed $n \geq 1$ and let $H \in \mathcal{H}$ be a hypergraph with VC dimension at least n. Since $\mathcal{H}$ has unbounded VC dimension such a H always exists. By definition H now has a shattered set of vertices X with $|X| \geq n$. From the similarity in the definition of shattered subsets and vertex induced hypergraphs we can observe $H[X] = (X, E(H)|_X \setminus \emptyset)$. Now since $E(H)|_X = 2^X$ it consists of at least 2^n distinct edges. As we remove only one (the empty set), we see that the statement holds. □

The main elements of the proof of the following Theorem 3.21 are implicitly present in a proof in [54]. The statement there puts an emphasis on the computational complexity of fhw checking and does not explicitly state the collapse. For the sake of completeness and for ease of reading we restate the theorem in

a way that fits our setting and repeat the relevant definitions and segment of the proof here.

Theorem 3.21. *For any class $\mathcal{H}$ of hypergraphs, if $\mathcal{H}$ has unbounded hypertree width and bounded fractional hypertree width then $\mathcal{H}$ is exotic.*

Definition 3.22. 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^* [74]. 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 $\rho(H)/\rho^*(H)$.

Lemma 3.23. *Let $\mathcal{H}$ be a class of hypergraphs with VC-dimension bounded by some constant d. Then for every hypergraph $H \in \mathcal{H}$ we have $hw(H) = O(fhw(H) \log fhw(H))$.*

Adapted from the full version of [54]). The proof proceeds in several steps.

Dual hypergraphs. Given a hypergraph $H = \{V, E\}$, the dual hypergraph $H^d = (W, F)$ is defined as $W = E$ and $F = \{\{e \in E \mid v \in e\} \mid v \in V\}$. For the rest of this proof we consider only reduced hypergraphs. This ensures that $(H^d)^d = H$ holds.

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., [46]):

(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 $cigap(H) = tigap(H^d)$.

VC-dimension. By a classical result ([44] Theorem (5.4), see also [26] for related results), for every hypergraph $H = (V(H), E(H))$ with at least two edges we have:

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

For hypergraphs H with a single edge only, $\mathrm{vc}(H) = 0$, and thus the above inequation does not hold. However, for such hypergraphs $\tau(H) = \tau^*(H) = 1$. By putting this together, we get:

$$tigap(H) = \tau(H)/\tau^*(H) \leq \max(1, 2\mathrm{vc}(H)\log(11\tau^*(H))).$$

Moreover, in [10], it is shown that $\mathrm{vc}(H^d) < 2^{\mathrm{vc}(H)+1}$ always holds. In total, we thus get

$$\begin{aligned} cigap(H) = tigap(H^d) &\leq \max(1, 2\mathrm{vc}(H^d)\log(11\tau^*(H^d))) \\ &\leq \max(1, 2^{\mathrm{vc}(H)+2}\log(11\rho^*(H))) \\ &\leq \max(1, 2^{d+2}\log(11\rho^*(H))), \\ &\text{which is } O(\log \rho^*(H)). \end{aligned}$$

Suppose that H has an FHD $\langle T, (B_u)_{u\in V(T)}, (\lambda)_{u\in V(T)}\rangle$ of width k. Then there exists a GHD of H of width $O(k \cdot \log k)$. 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 $O(\log k)$. Moreover, we know from [6] that $hw(H) \leq 3 \cdot ghw(H) + 1$ holds. In other words, there also exists an HD of H whose width is $O(k \cdot \log k)$. In particular, this also applies to the minimal width FHD, concluding the proof. □

Proof of Theorem 3.21. By contraposition of Lemma 3.20 we have that non-exotic classes of hypergraphs also have bounded VC dimension. From Lemma 3.23 we see that bounded fractional hypertree width implies bounded hypertree width. Lastly, a hypertree decomposition is a special case of a fractional hypertree decomposition. Hence, bounded hypertree width also implies bounded fractional hypertree width. □

We can extend the exotic property from hypergraphs to classes of CQs in the usual way. Recall, that in the context of CQs, incident edges in the hypergraph correspond to constraints that involve the variable. Hence, if vertices U have $2^{|U|} - 1$ distinct incident edges in the hypergraph, there exists at least one constraint for every possible combination of the corresponding variables in the CQ. We argue that this situation is highly unnatural and believe that this motivates further study of the complexity of non-exotic classes of CQ.

Semantic width and tractability. In the parameterized setting, it is easy to utilize low semantic width to establish upper-bounds as computing the core requires time only in the parameter. For tractability the situation is more problematic. As noted in Section 3.4, finding the core is intractable. Hence, if we have a class with bounded semantic fractional hypertree width, we know that the problem itself is not difficult, but an efficient solution depends on the hard problem of finding the core. We are caught in an unsatisfactory situation where the origin of the hardness is no longer the actual problem but the concrete formulation.

Part of the issue is that utilizing bounded *fhw* for polynomial evaluation requires a concrete decomposition with low *fhw*, which then guides the efficient solution of the CQ. Without knowing the core we cannot compute the appropriate decomposition. For bounded generalized hypertree width, Chen and Dalmau were able to show, that for classes of bounded *ghw*, there exists an algorithm for solving CQs in polynomial time without requiring the explicit computation of a decomposition [34]. Their method indeed remains polynomial if only the semantic generalized hypertree width is bounded. Thus, we are able to lift their result to bounded semantic fractional hypertree width for non-exotic classes.

Corollary 3.24. *Let $\mathcal{Q}$ be a non-exotic class of CQs with bounded semantic fractional hypertree width. Then* BoolCQ($\mathcal{Q}$) *is tractable.*

Any more general sufficient property for tractability would likely have to preserve this feature of making use of the width of the core without actually requiring the computation of the core. Furthermore, in light of Theorem 3.21 we also know that any further sufficient parameter for tractability has to be a special case of sem-*subw*. Hence, any further generalization of Corollary 3.24 for non-exotic classes would have to move along the narrow path between bounded sem-*fhw* and sem-*subw* while allowing for use of semantic width in polynomial time. We therefore conclude the section with the following conjecture.

Conjecture 1. *Let $\mathcal{Q}$ be a class of non-exotic CQs. Then* BoolCQ($\mathcal{Q}$) *is tractable if and only if $\mathcal{Q}$ has bounded semantic hypertree width.*

3.7 Summary

In this chapter we investigated semantic widths beyond semantic generalized hypertree width. We have given characterizations of semantic- hypertree width, fractional cover number, fractional hypertree width, adaptive width, and submodular width, thus covering all commonly used widths in the context of CQs (other than *ghw*).

Building on our result for semantic submodular width we presented characterizations of the fixed-parameter tractable classes of CQs and UCQs. This allows us to determine the parameterized complexity of problems that have CQ or UCQ formulations by determining if the class of these formulations has bounded sem-*subw*.

The characterization of polynomial time solvable CQs remains open. Using our result for semantic fractional hypertree width we motivated a new class of non-exotic problems that merits further research. In particular, we wish to resolve Conjecture 1, which we believe to be an important step towards the general problem.

CHAPTER 4

The Check Problem for Generalizations of α-Acylicity

In this chapter we continue a thread of research that has recently been initiated by Fischl, Gottlob, and Pichler [54]. In their paper, the authors demonstrate the NP-hardness of Check(ghw, 2) and Check(fhw, 2). However, despite this result they also present some positive news in the form of some structural restrictions for which the Check($\cdot$, k) problem becomes tractable for the respective widths.

Finding low width GHDs and FHDs is not only a problem of theoretical interest but also of immediate practical significance. Indeed, structural decomposition methods have already been integrated successfully in various commercial and academic systems [2, 8, 9, 76, 85], with speed-ups of up to a factor of 2,500 for CQ answering being reported in one systems [2]. We also refer to [53, 62, 48] for studies of the state-of-the art in computing decompositions.

When we use a decomposition of width k (be it any one of fhw, ghw, or hw) to answer a CQ (or solve a CSP, etc.), we are guaranteed an upper time bound of $O(I^k)$ where I is some measure of the size of the instance. Now, recall the

following relationship that holds for all queries q

$$fhw(q) \leq ghw(q) \leq hw(q)$$

While we can efficiently compute hypertree decompositions, they also have (possibly) higher width, thus yielding a larger exponent. For *fhw* we know that the difference can be arbitrarily large [73], while *ghw* can be at most 3 times lower than *hw* [6]. Hence, GHDs and FHDs ultimately allow for even more efficient CQ answering than HDs. Furthermore, their computation can be considered an offline activity in situations where the same query is repeated regularly.

We approach the problem by first presenting a new theoretical framework for the study of tractable width checking. Instead of considering ad hoc problems for each width, we unify the study of all f-widths in the form of *candidate tree decompositions* (CTD). This splits the problem in two parts, the mostly combinatorial problem of finding an appropriate set of *candidate bags*, and an algorithm for finding a tree decomposition from the given candidate bags. We provide a general algorithm for the second part, leaving only the task of finding the candidate bags for each individual width and structural restriction.

This framework has two important consequences. First, it greatly simplifies the existing proofs and highlights their similarities. This, in part, allows us to vastly extend the sufficient conditions for tractable *fhw* checking to the bounded multi-intersection property, solving a major open problem that was posed in [54]. Second, separating the combinatorial side from the algorithmics makes it easier to translate the key ideas to other settings. In particular, this eases the integration of the key combinatorial observations into actual implementations of decomposition algorithms. For example, the implementation presented by Gottlob, Okulmus, and Pichler in [70] uses important observations on bounded-intersection hypergraphs to speed up the computation of GHDs significantly.

The contents of this chapter are an extension of papers published in collaboration with Georg Gottlob, Reinhard Pichler, and Igor Razgon [65, 64]. In part these contents have also appeared in a recent survey article [62]. To give a self-contained presentation, parts of Section 4.2 repeat key lemmas and ideas from the original paper by Fischl, Gottlob, and Pichler [54]. Further details regarding the attribution of results is given at the beginning of that section.

4.1 The Candidate Tree Decompositions Framework

We begin by introducing our new framework for the tractability of checking f-widths. Conceptually, we can split the task of checking whether a decomposition of certain width exists into two parts: (1) deciding which sets of vertices are acceptable as bags (the *candidate bags*) and (2) deciding if there is a tree decomposition made up of only acceptable bags. In this section, we focus on the second part and show that this task is indeed tractable as long as the decompositions satisfy a certain normal form. This will allow us to show the Check problem tractable for settings where we can compute an appropriate set of candidate bags in polynomial time.

To emphasize the generality of the approach we will focus on tree decompositions in this section. Recall that a generalized hypertree decomposition of width at most k is simply a tree decomposition where every bag has an integral edge cover with weight at most k. The same is true for fractional hypertree decompositions and fractional edge covers. Hence, the Check(GHD, k) problem can be solved by computing appropriate sets S of candidate bags that can be covered by k edges and then deciding whether there exists a TD using only bags from S. If such a TD exists, it is a witness for the existence of a GHD of width at most k. Of course, the same strategy also works for Check(FHD, k).

First, we will formally define the task we are interested in as the candidate tree decomposition problem. We show that the problem is NP-complete even for acyclic graphs. Following that, we show that the problem becomes tractable if we limit our search to finding TDs that adhere to a certain normal form which is sufficient for our purposes.

Definition 4.1. Let H be a hypergraph and $\mathcal{T} = \langle T, (B_u)_{u \in T} \rangle$ be a tree decomposition of H. Let the *candidate bags* S be a family of subsets of $V(H)$. If for each $u \in T$ there exists an $S \in \mathbf{S}$ such that $B_u = S$, then we call $\mathcal{T}$ a *candidate tree decomposition* of S. We denote by $\mathbf{CTD}(\mathbf{S})$ the set of all candidate tree decompositions of H.

Example 4.1. Consider hypergraph H and TD $\mathcal{T}$ from Figure 4.1. The bags of $\mathcal{T}$ consist of the sets $\{3, 5, 6\}$, $\{2, 5, 6\}$, $\{2, 7, 8\}$, $\{4, 5\}$, and $\{1, 6\}$. Thus, $\mathcal{T}$ is a candidate tree decomposition of every S that contains those bags. In particular,

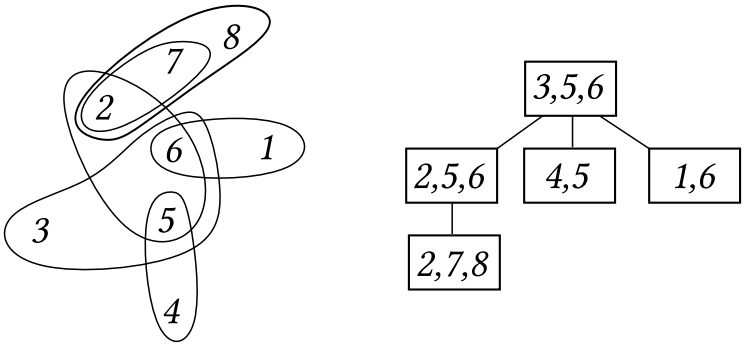


Figure 4.1: Hypergraph H with TD $\mathcal{T}$ for Example 4.1

there are a number of natural sets of candidate bags that fit this description. Let $\mathbf{S}_3$ be every subset of $V(H)$ with at most 3 elements. Then clearly every TD with treewidth at most 2 is in $\mathbf{CTD}(\mathbf{S}_3)$. Alternatively, we could consider $\mathbf{S}_E = E(H)$, i.e., every edge of H is a candidate bag. It is easy to verify that $\mathcal{T} \in \mathbf{CTD}(\mathbf{S}_E)$, and furthermore, that $\mathbf{CTD}(\mathbf{S}_E)$ corresponds exactly to the set of join trees of H. In this way, the search for candidate tree decompositions can be used as an abstraction for checking a wide array of width-related hypergraph properties. △

Theorem 4.2. *Let H be a hypergraph and $\mathbf{S} \subseteq 2^{V(H)}$. It is NP-complete to decide whether $\mathbf{CTD}(\mathbf{S}) \neq \emptyset$. The problem remains NP-complete even if we restrict the choice of H to acyclic graphs.*

Proof. The problem is clearly in NP. We show NP-hardness by reduction from the exact cover problem: Let $U = \{u_1, \ldots, u_n\}$ be the universe and let $X_1, \ldots, X_m$ be subsets of U. The exact cover problem asks for a cover of U by elements of $\{X_1, \ldots, X_m\}$ such that the sets in the cover are pairwise disjoint.

We define an acyclic graph G as follows: G is a tree with vertices $v, v_1, \ldots, v_n$, $u_1, \ldots, u_n$. The edges of G are $\{v, v_i\}$ and $\{v_i, u_i\}$ for each $1 \leq i \leq n$. Let $\mathbf{S} = \{S, S_1, \ldots, S_m\}$ where $S = \{v, v_1, \ldots v_n\}$ and each $S_i = X_i \cup \{v_j \mid u_j \in X_i\}$, i.e., by taking X_i and adding v_j for each u_j contained in X_i.

We claim that then $\mathbf{CTD}(\mathbf{S}) \neq \emptyset$ iff there is an exact cover of U by $X_1, \ldots, X_m$. One direction is easy. Let $\mathbf{X} \subseteq \{X_1, \ldots, X_m\}$ be an exact cover of U. Denote

the elements of $\mathbf{X}$ by $X'_1, \ldots, X'_q$. For $1 \leq i \leq q$, let S'_i be obtained from X'_i by adding v_i for each $u_i \in X'_i$. Clearly $S'_i \in \{S_1, \ldots, S_m\}$ and there exists a candidate tree decomposition $\langle T, (B_u)_{u \in T} \rangle$ of G where $V(T) = \{t, t_1, \ldots, t_q\}$, $E(T) = \{\{t, t_1\}, \ldots, \{t, t_q\}\}$, $B_t = S$ and for each $1 \leq i \leq q$, $B_{t_i} = S'_i$.

The other direction is more complicated. Let $\langle T, (B_u)_{u \in T} \rangle$ be a smallest (w.r.t. the number of nodes of T) candidate TD of G. In particular, this means that the bags of all the nodes are distinct. The proof proceeds in several steps:

1. There is $t \in V(T)$ such that $B_t = S$. Indeed, otherwise, v is not covered.
2. For each $1 \leq i \leq n$, there is $t' \in V(T)$ such that $u_i \in B_{t'}$ and t is adjacent to t'. Indeed, assume the opposite and let $t' \in V(T)$ be a node with $u_i \in B_{t'}$ such that t is not adjacent to t'.

   Since we assume a candidate TD, we have $B_{t'} = S_j$ for some j and hence $v_i \in B_{t'}$. Now consider the path between t and t' and let s be the node next to t on this path. By our minimality assumption we know that $B_s \neq B_t$ and thus $B_s = S_k$ with $k \neq j$. Since we assume the claim to be false we have $u_i \notin S_k$ and hence $v_i \notin S_k$. As $v_i \in B_{t'}$ and $v_i \in B_t$, the connectedness condition of the tree decomposition is violated.
3. Let $t_1, \ldots, t_q$ be the neighbors of t in T. We claim that T has no other nodes. Indeed, all the vertices of G are covered by the bags of $t, t_1, \ldots t_q$ by the previous two items. Each edge $\{v, v_i\}$ is contained in B_t. Also, each edge $\{u_i, v_i\}$ is covered by some B_{t_j} containing u_i (existing by the previous item). It follows that $T[\{t, t_1, \ldots, t_q\}]$ together with the corresponding bags form a tree decomposition of G. By the minimality assumption, T does not have other nodes.
4. We claim that for any $1 \leq i \neq j \leq q$, $B_{t_i} \cap B_{t_j} \neq \emptyset$. Indeed, otherwise, there is $u_k \in B_{t_i} \cap B_{t_j}$. However, $u_k \notin B_t$ in contradiction to the connectedness condition. It follows that $B_{t_1} \cap U, \ldots, B_{t_q} \cap U$ are disjoint elements of $\{X_1, \ldots, X_m\}$ covering all of U as required.

□

To obtain a tractable version of the problem we will introduce a generalization of the normal form that was used in the tractability proof for HDs in [68].

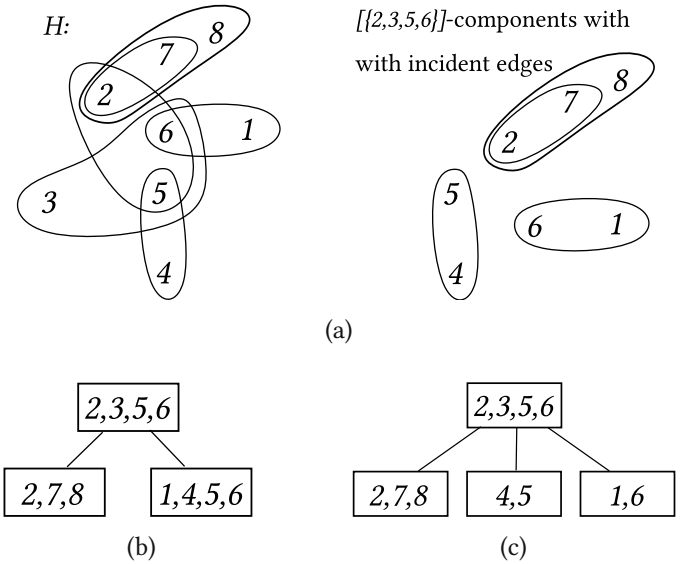


Figure 4.2: Illustrations for Example 4.2

Definition 4.3. A tree decomposition $\langle T, (B_u)_{u \in T} \rangle$ of a hypergraph H is in *component normal form (ComNF)* if for each node $r \in T$, and for each child s of r there is *exactly one* $[B_r]$-component C_s such that $V(T_s) = C_s \cup (B_r \cap B_s)$ holds. We say C_s is the component *associated with* node s.

Example 4.2. Consider the hypergraph H and the accompanying tree decompositions in Figure 4.2. We will refer to the decomposition in Figure 4.2b as $\mathcal{T}_b$ and to the decomposition in Figure 4.2c as $\mathcal{T}_c$.

Both have the same bag $B_r = \{2, 3, 5, 6\}$ in the respective root nodes r. The $[B_r]$-components of H are shown in Figure 4.2a. To help with the illustration, the components are shown including the full incident edges of the hypergraph. That is, the $[B_r]$-component $\{1\}$ is illustrated as the edge $\{6, 1\}$ and the same is true for the other components. Intuitively, in a ComNF TD each child s of a node r is a decomposition for exactly one $[B_r]$-component. In our example, $\mathcal{T}_c$ is an example of such a decomposition. It is straightforward to verify the ComNF condition for every child of the root and to match each child to a $[B_r]$-component.

On the other hand, there are three $[B_r]$-components, in a ComNF decomposition.

Thus, in ComNF, r needs to have three children[1]. In particular, $\mathcal{T}_b$ is not in ComNF. Let s be the node with bag $\{1, 4, 5, 6\}$. It is easy to verify that vertices 1 and 4 are in different $[B_r]$-components. Since also neither is in B_r, they cannot occur together in a set of the form $C \cup (B_r \cup B_s)$, where C is a $[B_r]$-component. △

By the above definition, we know that in a ComNF TD, every child s of a node r is associated with at most one $[B_r]$-component C_s. By the connectedness condition, also the converse is true, i.e., every $[B_r]$-component C_s is associated with *at most one* child s. Note that (by applying the ideas of the transformation of HDs into the normal form of [68]), every TD can be transformed in polynomial time into a TD in ComNF without increasing the width (more precisely, the bags in the resulting TD are subsets of the bags in the original TD).

Definition 4.4. Let H be a hypergraph and let $\mathbf{S}$ be a family of subsets of $V(H)$. Let $\mathcal{T} \in \mathbf{CTD}(\mathbf{S})$ be a tree decomposition in ComNF. We say $\mathcal{T}$ is a *ComNF candidate tree decomposition* of $\mathbf{S}$. We denote by $\mathbf{ComCTD}(\mathbf{S})$ the set of all ComNF candidate tree decompositions of H.

Theorem 4.5. *Let H be a hypergraph and $\mathbf{S} \subseteq 2^{V(H)}$. There exists a LogCFL algorithm that takes H and $\mathbf{S}$ as an input and decides whether $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$, and if so, returns a tree decomposition $\mathcal{T} \in \mathbf{ComCTD}(\mathbf{S})$.*

Recall that LogCFL $\subseteq$ PTime, thus the theorem implies polynomial time decidability of the problems. Since our LogCFL proof is by an algorithm for *Alternating Turing Machines* [30], it is not immediately translatable into a sensible implementable algorithm with current methods. We are not aware of a proof of Theorem 4.5 in the literature, even for only the PTime upper bound. In light of the hardness result for the general case and the difficulty of implementing LogCFL algorithm we therefore choose to first give a detailed proof of a polynomial time algorithm here.

The intuition behind a polynomial-time algorithm for this problem is simple. Every parent/child relationship in a tree decomposition corresponds to a separator

[1] Technically, a decomposition with duplicated subtrees is also allowed but this possibility is of little consequence.

S (the bag of the parent) and an $[S]$-component. It should therefore be enough to first enumerate all pairs (S, C) of separators S in $\mathbf{S}$ and $[S]$-components C, and then check if these pairs, which we will call *blocks*, can be combined to form a valid tree decomposition. Through the restriction to a specific set $\mathbf{S}$, that is part of the input, the number of blocks we have to consider is only polynomial in the input.

We note, that methods for subedge-based decompositions [69] as well as tree projections and their associated algorithms, see e.g., [57, 61, 87], are closely related to the ComNF CTD problem.

The restriction to component normal form will ultimately not restrict us in the following sections. For generalized and fractional hypertree width, ComNF can be enforced without increasing the width. Such a transformation can be found, e.g, in [68] and as part of the proof of Lemma 4.19. Indeed, we can make a slightly more general statement. As can be seen by investigation of the proof of Lemma 4.19, this transformation preserves the width for any monotone f-width. Still, some care will be required in the enumeration of the candidate bags to guarantee that they allow for a decomposition in component normal form.

We move on to the proof of the PTime version of Theorem 4.5. To do so, we present a bottom-up construction of ComNF CTDs, if they exist, using dynamic programming. Note that our presentation does not optimize for runtime, our goal is only to establish that the problem can be decided in polynomial time.

Definition 4.6. A pair (B, C) of *disjoint* subsets of $V(H)$ is a *block* if C is a $[B]$-component of H or $C = \emptyset$. Such a block is *headed* by B. Let (B, C) and (X, Y) be two blocks. We say that $(X, Y) \leq (B, C)$ if $X \cup Y \subseteq B \cup C$ and $Y \subseteq C$.

Definition 4.7. For a block (B, C) and vertex set $X \subseteq V(H)$ with $X \neq B$, we say that X *is a basis of* (B, C) if the following conditions hold:

1. Let $(X, Y_1), \ldots, (X, Y_\ell)$ be all the blocks headed by X that are less than or equal to (B, C). Then $C \subseteq X \cup \bigcup_{i=1}^{\ell} Y_i$.
2. For each $e \in E(H)$ such that $e \cap C \neq \emptyset$, $e \subseteq X \cup \bigcup_{i=1}^{\ell} Y_i$.
3. For each $i \in [\ell]$, there exists a ComNF TD of $H[X \cup Y_i]$ where the root has precisely X as its bag.

The existence of a basis X intuitively corresponds to the existence of a tree decomposition that covers the whole component C (by X together with the $[X]$-components $Y_1, \ldots, Y_\ell$) and connects X to its parent bag B. The following lemmas confirm that this definition of a basis for a block corresponds to such a TD in the expected way.

Lemma 4.8. *Let H be a hypergraph, and $\langle T, (B_u)_{u \in T} \rangle$ be a ComNF TD of H. Let $r \in T$ be a non-leaf node. For each child s of r, let C_s be the $[B_r]$-component associated with s. The following two statements are true:*

- *B_s is a basis of the block (B_r, C_s).*
- *$(B_s, D) \leq (B_r, C_s)$ if and only if D is either a component associated with a child of s or if D is empty.*

Proof. We first observe that $(B_s, D) \leq (B_r, C_s)$ if and only if D is either empty or a component associated with a child of s. Indeed, since we assume ComNF, we have that $V(T_s) = C_s \cup (B_r \cap B_s)$ and C_s is the only such $[B_r]$-component. Thus, $V(T_s) \subseteq B_r \cup C_s$.

Moreover, $B_s \cup D \subseteq V(T_s)$ holds for every block (B_s, D) where D is empty or a component associated with a child of s, which completes the proof of the "if" direction. For the "only if" direction, recall that $(B_s, D) \leq (B_r, C_s)$ requires $D \subseteq C_s$. Since there is only one node associated with C_s, we conclude $D \subseteq V(T_s)$. Since D is a $[B_s]$-component, it must have its own associated child of s.

Now that we know exactly which blocks headed by B_s are relevant, we can show that they satisfy the conditions of a basis. Let $child(s)$ be the set of all children of s. For every $u \in child(s)$, let D_u be the $[B_s]$-component associated with u. The vertices that occur in the subtree T_s are precisely $V(T_s) = B_s \cup \bigcup_{u \in child(s)} D_u$. Since we assume ComNF, we also have $C_s \subseteq V(T_s)$ and therefore Condition 1 of a basis is satisfied.

For Condition 2 it is enough to observe that if $e \cap C_s \neq \emptyset$, then it must be covered in the subtree T_s, i.e., $e \subseteq V(T_s)$. Otherwise, suppose e were only covered in some node u not in the subtree T_s. There is a vertex $v \in e \cap C_s$ that occurs in

$V(T_s)$ but not in B_r (recall B_r and C_s are disjoint). Any path from a node of T_s to u must pass through B_r, which would break connectedness for v.

Finally, for Condition 3 and each $u \in child(s)$, consider the subtree T_u^* induced by $\{s\} \cup T_u$. The root of T_u^* has bag B_s and from $V(T_u) = C_u \cup (B_s \cap B_u)$ also $B_s \cup V(T_u) = B_s \cup C_u$, i.e., T_u^* is indeed a TD of $H[B_s \cup C_u]$. To see that T_u^* is in fact in ComNF, observe that $H[B_s \cup C_u]$ has only a single $[B_s]$-component C_u. Then ComNF of T_u^* follows from the assumption that the original decomposition $\langle T, (B_u)_{u \in T} \rangle$ is in ComNF. □

For the following arguments, it is convenient to introduce the notion of a *union of TDs* that have the same root bag. Let $\langle T_1, (B_{1,u})_{u \in T_1} \rangle, \ldots, \langle T_n, (B_{n,u})_{u \in T_n} \rangle$ be rooted TDs and w.l.o.g. assume they have pairwise distinct nodes. For each $i \in [n]$, let us denote the root of T_i by r_i. Furthermore, assume that $B_{r_i} = B_{r_j}$ for all $i, j \in [n]$. We then define the union $\langle T, (B_u)_{u \in T} \rangle = \bigcup_{i=1}^{n} \langle T_i, (B_{i,u})_{u \in T_i} \rangle$ as the following structure: T is a tree with a new root node r and $B_r = B_{r_1}$. For each $i \in [n]$, all the nodes u of T_i except for the root r_i are in T and for each $u \neq r_i$ in T_i, we have $B_u = B_{i,u}$. Moreover, all edges of T_i except for the ones adjacent to the root r_i are also contained in T. Further, for every edge $[u, r_i]$ we introduce an edge $[u, r]$ in T. The following lemma establishes a sufficienct condition such that this new structure is indeed a TD.

Lemma 4.9. *Let $(B, C_1), \ldots, (B, C_\ell)$ be blocks of a hypergraph H. Assume for each $i \in [\ell]$ that there exists a ComNF TD $\mathcal{T}_i$ of $H[B \cup C_i]$ where B is the bag of the root. Then, $\mathcal{T} = \bigcup_{i=1}^{\ell} \mathcal{T}_i$ is a ComNF TD of $H[B \cup \bigcup_{i=1}^{\ell} C_i]$.*

Proof. We start by verifying that the connectedness condition is satisfied in $\mathcal{T}$. By assumption, connectedness holds for each subtree rooted at a child of the root. The condition can then only be violated if vertices occur in more than one such subtree but not in the bag of the root. As all C_i for $i \in [q]$ are $[B]$-components, or empty, they are also pairwise disjoint. So, the subtrees can only share variables in B which is precisely the bag of the root.

To see that every edge is covered, observe that for each edge e in $H[B \cup \bigcup_{i \in [q]} C_i]$, there must be at least one $i \in [q]$ such that e is also an edge in

$H[B \cup C_i]$. Otherwise, e would have to be part of more than one $[B]$-components, which is impossible as the components would then be $[B]$-connected.

It remains to show that $\mathcal{T}$ is in ComNF: the decompositions $\mathcal{T}_i = \langle T_i, (B_{i,u})_{u \in T_i} \rangle$ were already in ComNF and are left unchanged. The only new parent/child relationships are those from the root r to its children. Let u with bag B_u be a child of r. By the construction of the union of TDs, u is obtained from some node u_i with bag B_{i,u_i} in TD $\mathcal{T}_i$ for some i. That is, we have $B_u = B_{i,u_i}$. Hence, the subtree rooted at u covers a single $[B_r]$-component by the fact that $B_r = B_{r_i}$ and $\mathcal{T}_i$ is a ComNF TD of $H[B \cup C_i]$. □

Lemma 4.10. *Let H be a hypergraph and $B \subseteq V(H)$. Let (B, C) be a block of H. If there exists $X \subseteq V(H)$ that is a basis of (B, C) or if $C = \emptyset$, then there exists a ComNF TD of $H[B \cup C]$ where B is the bag of the root.*

Proof. First, if $C = \emptyset$, the TD with a single node u and $B_u = B$ is trivially a ComNF TD of $H[B]$. Otherwise, let $(X, Y_1), \ldots, (X, Y_\ell)$ be all the blocks headed by X that are less or equal (B, C). For each $i \in [\ell]$, let $\mathcal{T}_i$ be a ComNF TD of $H[X \cup Y_i]$ where X is the bag of the root node.

Let $\mathcal{T} = \langle T, (B_u)_{u \in T} \rangle$ be the union $\bigcup_{i=1}^{\ell} \mathcal{T}_i$. By Lemma 4.9, $\mathcal{T}$ is a ComNF TD of $H[X \cup \bigcup_{i=1}^{\ell} Y_i]$. Add a new root r with $B_r = B$ to $\mathcal{T}$ as the parent of the previous root to obtain $\mathcal{T}'$. We claim that $\mathcal{T}'$ is the desired ComNF TD of $H[B \cup C]$. Note that we have $V(\mathcal{T}) = X \cup \bigcup_{i=1}^{\ell} Y_i$.

Assume an edge $e \in H[B \cup C]$. If $e \cap C \neq \emptyset$, then $e \subseteq X \cup \bigcup_{i=1}^{\ell} Y_i$ because X is a basis of (B, C). Therefore, e occurs in $H[X \cup \bigcup_{i=1}^{\ell} Y_i]$ and must be covered in $\mathcal{T}$. Otherwise, if $e \cap C = \emptyset$, then $e \subseteq B$ and e is covered by the root node of $\mathcal{T}'$.

$\mathcal{T}$ satisfies the connectedness condition and has X as the bag of its root node. Hence, the only way the connectedness condition can fail in $\mathcal{T}'$ is if there is a vertex in B and $V(\mathcal{T})$ but not in X. For each $i \in [\ell]$ we have $Y_i \subseteq C$ and because B and C are disjoint we have $B \cap Y_i = \emptyset$. Therefore, $B \cap \left(X \cup \bigcup_{i=1}^{\ell} Y_i\right) = B \cap X$, i.e., any vertex in $V(\mathcal{T})$ and in B is also in the bag X at the root of $\mathcal{T}$.

It remains to show that $\mathcal{T}'$ is indeed in ComNF. We know that $\mathcal{T}$ is in ComNF. Furthermore, $\mathcal{T}$ is the single subtree of the root r and there is only one $[B]$-component in $H[B \cup C]$, namely C. So, we need to show that $V(\mathcal{T}) = C \cup (B \cap X)$.

Since $B \cup C$ is the set of all vertices in $H[B \cup C]$, we have $V(\mathcal{T}) = (B \cup C) \cap V(\mathcal{T}) = (B \cap V(\mathcal{T})) \cup (C \cap V(\mathcal{T}))$. By the connectedness shown above we have $B \cap V(\mathcal{T}) = B \cap X$. Moreover, by Condition 1 of a basis, we have $C \cap V(\mathcal{T}) = C$. In total, we thus get the desired equality $V(\mathcal{T}) = (B \cap X) \cup C$. □

Lemma 4.11. *Let H be a hypergraph and $B \subseteq V(H)$. If all blocks headed by B have a basis, then H has a ComNF tree decomposition where B is the bag of the root.*

Proof. Let $(B, C_1), \ldots, (B, C_\ell)$ be all the blocks headed by B. In consequence of Lemma 4.10 we have that for each $i \in [\ell]$, since the block (B, C_i) has a basis, there is a ComNF TD $\mathcal{T}_i$ of $H[B \cup C_i]$ with B as the bag of the root. We can then take the union $\mathcal{T} = \bigcup_{i \in [q]} \mathcal{T}_i$, which by Lemma 4.9 is precisely the required decomposition as $B \cup \bigcup_{j=1}^{l} C_j = V(H)$. □

Proof of the PTime version of Theorem 4.5. We only present the decision procedure. It is clear from the soundness argument that constructing an appropriate TD from an accepting state is trivial. We claim that Algorithm 4.1 decides, in polynomial time, whether $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$. First, the algorithm runs in polynomial time: Observe that a straightforward representation of $\mathbf{S}$ as a list of lists of vertices has size $O(|\mathbf{S}| \cdot |V(H)| \cdot \log |V(H)|)$. For asserting polynomial runtime it is therefore not necessary to distinguish between the size of the representation of $\mathbf{S}$ and $|\mathbf{S}|$. The set *blocks* has at most $|\mathbf{S}| \cdot |V(H)|$ initial elements and computing componenets is polynomial in the size of the representation of H. The checks in the innermost loop are clearly polynomial and thus, the whole algorithm requires only polynomial time.

For the soundness of the algorithm, we first observe that every marked block (B, C) in the algorithm has a ComNF TD of $H[B \cup C]$ using only bags from $\mathbf{S}$, i.e., a marked block satisfies Condition 3 of a basis. This is easily verified by structural induction in combination with Lemma 4.10. The construction of such a TD by the lemma uses only the union of TDs, which does not introduce any new bags, i.e., all bags are still elements of $\mathbf{S}$. Soundness then follows immediately from Lemma 4.11. The lemma also explicitly shows how to construct a TD from the accepting state.

Algorithm 4.1: ComNF Candidate Tree Decomposition

input: Hypergraph H and a set $\mathbf{S} \subseteq 2^{V(H)}$.
output: "Accept", if $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$
"Reject", otherwise.

```
1  begin                                                    /* Main */
2      blocks = all blocks headed by any S ∈ S ;
3      Mark all blocks (B, C) ∈ blocks where C = ∅ ;
4      repeat
5          foreach (B, C) ∈ blocks where (B, C) is not marked do
               /* Check if there exists a basis X of (B, C)    */
6              foreach X ∈ S \ {B} do
7                  blocks_X = all blocks (X, Y) with (X, Y) ≤ (B, C) ;
8                  if Not all blocks in blocks_X are marked then
9                      Continue;
10                 end
11                 V_X = X ∪ ⋃_{(X,Y)∈blocks_X} Y;
12                 if C ⊆ V_X and for each edge e where e ∩ C ≠ ∅ also
                    e ⊆ V_x then
13                     Mark (B, C);
14                 end
15             end
16         end
17         if For some S ∈ S, all blocks headed by S are marked then
18             return Accept;
19         end
20     until no new blocks marked;
21     return Reject;
22 end
```

Completeness will follow from Lemma 4.8: if there exists a $\mathcal{T} \in \mathbf{ComCTD}(\mathbf{S})$, then all blocks headed by bags of $\mathcal{T}$ are clearly contained in the set *blocks* in the algorithm. We proceed by induction on the height $h(u)$ of a node u in $\mathcal{T} = \langle T, (B_u)_{u \in T} \rangle$, where $h(u)$ denotes the maximum distance of u from a descendant leaf node. Let $u \in T$ and let C be a component associated to a child of u as in Definition 4.3, or $C = \emptyset$ if u is a leaf. We claim that after $h(u)$ iterations of the repeat-until loop, the block (B_u, C) will be marked by the algorithm.

For $h(u) = 0$, i.e., leaf nodes, the situation is clear. The block $(B_u, \emptyset)$ is marked before the loop. Suppose the claim holds for all nodes u' where $h(u') < j$. We have to show that then it also holds for $h(u) = j$: let s be a child of u with associated component C_s. By $h(s) = j - 1$ and the induction hypothesis, all blocks of B_s with an associated component of a child of s have already been marked before the j-th iteration of the loop. Therefore, in combination with Lemma 4.8, it follows that (B_r, C_s) will be marked in the j-th iteration of the loop. □

The LogCFL upper-bound.

In this subsection we show that the ComNF CTD problem also lies in the class LogCFL and is therefore highly parallelizable: Consider the LogCFL algorithm for computing hypertree decompositions presented in [68]. To guess the next separator we now, roughly speaking, guess some element of $\mathbf{S}$ instead of guessing a set of up to k edges. Since $\mathbf{S}$ is an input, it is sufficient to guess an index into $\mathbf{S}$. We will make this intuition more concrete in the following.

Recall that an *Alternating Turing Machine* (ATM) [30] is a an extension of nondeterministic turing machines where states (i.e., configurations) are partitioned into *existential* and *universal* states. Acceptance of ATMs is defined on their *computation trees*. A computation tree for an ATM and some input is constructed by nodes, labeled by the states of the ATM (edges are implied by transitions in the ATM). The root of the tree is the initial state of the ATM. For any (non-leaf) existential state the computation tree includes *one* of its successors, for a universal state the tree includes *all* of its successors. A computation tree is *accepting* if all leaves are accepting states. An accepting computation tree can therefore be considered as a witness, or *certificate*, of the acceptance of the input. Ruzzo proposed the study of the size of these certificates, introducing the notion of *tree-size* for ATMs. The *tree-size* of an ATM is the minimal size of an accepting computation tree. We refer to the original paper by Ruzzo [103] for detailed definitions of bounded tree-size in ATMs. In the following we will make use of the following characterization of LogCFL by LogSpace ATMs with bounded tree-size due to Ruzzo.

Proposition 4.12 ([103]). *The complexity class LogCFL is exactly the class of all*

decision problems that are recognized by an ATM with polynomially bounded tree size and logarithmically bounded space.

Our alternating algorithm is presented in Algorithm 4.2. Lines 3 and 11 represent existential states, all others correspond to universal states.

Algorithm 4.2: An Alternating LogCFL Algorithm for ComNF Candidate Tree Decompositions

input: Hypergraph H and a set $\mathbf{S} \subseteq 2^{V(H)}$.
output: "Accept", if $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$
"Reject", otherwise.

```
1  Function HasBasis (S : set of vertices from S, C_S : component)
2      if C_S = ∅ then return Accept;
3      Guess B ∈ S \ {S};
4      Let blocks := the set of all blocks (B, C) ≤ (S, C_S);
5      Check conditions (1) and (2) of whether B is a basis of (S, C_S)
         according to Definition 4.7;
6      if One of the checks fails then return Reject;
7      if for each (B, C) ∈ blocks, HasBasis (B, C) then
8          return Accept;
9      else return Reject;
10 begin                                              /* Main */
11     Guess a root bag S ∈ S;
12     if for each [S]-component C, HasBasis (S, C) then
13         return Accept;
14     end
15     return Reject;
16 end
```

Proof of Theorem 4.5. It is easy to see that Algorithm 4.2 is essentially a nondeterministic version of Algorithm 4.1. Instead of marking blocks with a basis in a bottom-up fashion we proceed top-down and guess a separator that works as a basis for the parent. The arguments for correctness of the algorithm are therefore the same as in the proof of the polynomial time case.

We now argue that Algorithm 4.2 is implementable on an ATM with polynomially bounded tree size and logarithmically bounded space. Establishing

that Algorithm 4.2 is implementable on an ATM with logarithmically bounded configuration sizes requires an argument about encoding and computing blocks (i.e., pairs of separators and components) in logarithmic space. For this argument we refer to the proof of Lemma 5.15 in [68] where the respective tasks are explained in great detail. Note that our set S is always an element of the input $\mathbf{S}$ and can therefore be encoded by a simple index into $\mathbf{S}$, thus a set S can be encoded in logarithmic space. What is left to show is that the checks in line 5 are feasible in LogSpace. Both only require checking a subset relationship. Instead of computing the union over all blocks (B, C_i) we can simply, check vertex-wise if the vertex is in one of the (linearly many) blocks headed by B. Indeed analogous checks (using different terminology) are already present in the alternating algorithm for hypertree decompositions in [68]. Hence, Algorithm 4.2 can be implemented on a LogSpace ATM. We now show that Algorithm 4.2 has polynomially bounded tree-size. Observe that every accepting computation tree corresponds to a TD in $\mathbf{ComCTD}(\mathbf{S})$. The guessed sets B from $\mathbf{S}$ are the bags of the decomposition while the blocks headed by B correspond to the respective children of the decomposition node with bag B. Since in that case repetition of bags is never necessary in a TD we know that if $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$ then there exists a $\mathcal{T} \in \mathbf{ComCTD}(\mathbf{S})$ with a linear number of nodes. Therefore, there is an accepting computation tree with a linear number of guesses (each of which requires a logarithmic amount of configurations, each guessing a single bit of an index). Since the checks can be done in LogSpace $\subseteq$ PTime, and are only done in relation to a guess, they add only a polynomial number of configurations to the computation tree. Thus, if the algorithm accepts, there exists an accepting computation tree with polynomially bounded tree-size. □

4.2 Check(ghw, k) under Bounded Multi-Intersection

As noted at the beginning of the chapter, parts of this section are originally due to Fischl, Gottlob, and Pichler [54]. To be precise, the contents as presented here are taken from a follow up paper by the Gottlob, Pichler, Razgon, and the author of this thesis [64], but some parts there have their origin in the preceding paper by Fischl, Gottlob and Pichler. In particular, we consider the material of

this section up to (and excluding) Definition 4.22 to be previous work and the author claims no credit for the contained contributions. A notable exception to this is Theorem 4.15 and the related Corollary 4.27, the main algorithmic results of this section. While Fischl, Gottlob, and Pichler [54] showed analogue results to these for polynomial time solvability, we give new – and arguably significantly simpler – proofs here that utilize the candidate tree decompositions framework from the previous section. In this way we are able to improve their upper bounds from PTime to LogCFL. This is particularly notable as LogCFL is known to be highly parallelizable because of its relationship to the boolean circuit classes AC^1 and NC^2 [103, 67]. As of now, it is still difficult to translate these theoretical guarantees into practical algorithms. However, the promise of highly parallel algorithms for large fragments difficult problems is enticing and provides further motivation for research on how to derive highly parallel deterministic algorithms from alternating algorithms.

Recall from the discussion in the introduction (see Section 1.5) that we are interested in finding realistic and non-trivial structural restrictions for hypergraphs that make the Check(GHD, k) problem tractable for fixed k. We thus propose here such a simple property, namely the bounded intersection of two or more edges.

Definition 4.13. 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 C be a class of hypergraphs. We say that C has the *bounded intersection property (BIP)* if there exists some integer constant i such that every hypergraph H in C has the i-BIP.

The BIP criterion properly generalizes bounded arity and is indeed non-trivial in the sense that there exist hypergraph classes of unbounded *ghw* that enjoy the BIP. Among others this includes the classes of graphs, regular hypergraphs, and linear hypergraphs.

Example 4.3. A a-regular hypergraph is a hypergraph where every edge is a set of exactly a elements. It is then easy to see, that any a-regular hypergraph H has

iwidth$(H) \leq a$. Thus, for any constant a, any class of a-regular hypergraphs enjoys the BIP.

Many such classes are well known to have unbounded ghw, the most prominent being classes of graphs (i.e., 2-regular hypergraphs) with unbounded treewidth. Consider a GHD of any graph, and observe that for every node u of the GHD, it holds that $|B_u| \leq 2 \cdot |\lambda_u|$ since every edge in λ_u can cover at most 2 vertices. It then follows that $tw(H) + 1 \leq 2ghw(H)$ and thus every class of graphs with unbounded treewidth also has unbounded ghw, while still enjoying the BIP. Indeed, it is easy to generalize this argument to see that $tw(H)+1 \leq a \cdot ghw(H)$ holds for any a-regular hypergraph H. △

Moreover, a recent empirical study [53] 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 4.14. 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 C be a class of hypergraphs. We say that the class C has the *bounded multi-intersection property (BMIP)* if there exist constants c and i such that every hypergraph H in C has the ic-BMIP.

Example 4.4. Figure 4.3 shows the hypergraph $H_0 = (V_0, E_0)$ with $ghw(H_0) = 2$ but $hw(H_0)$=3. (This example is from [69], which, in turn, is an adaption of work by Adler [4]). Figure 4.4 shows an HD of width 3 and Figure 4.5 shows GHDs of width 2 for the hypergraph H_0. The *iwidth* and the 3-*miwidth* of H_0 is 1. Starting from $c = 4$, the *c-miwidth* is 0. △

The BMIP is the most liberal restriction on classes of hypergraphs introduced in Definitions 4.13 and 4.14. 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.

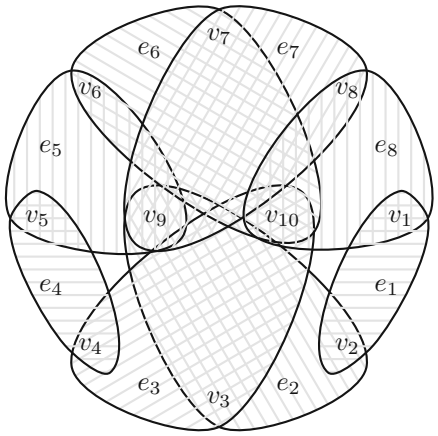


Figure 4.3: Hypergraph H_0 from Example 4.4

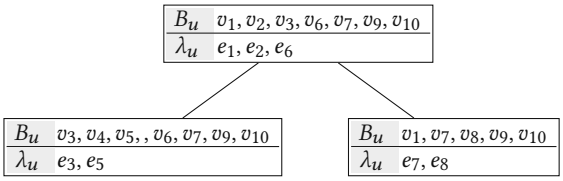


Figure 4.4: HD of hypergraph H_0 in Figure 4.3

Theorem 4.15. *For every hypergraph class C that enjoys the BMIP, and for every constant $k \geq 1$, the Check(GHD, k) problem is in LogCFL.*

Our plan is to make use of Theorem 4.5 by computing appropriate sets $\mathbf{S}$ of candidate bags such that $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$ if and only if there exists a GHD with width at most k. Example 4.5 illustrates the main challenge that needs to be tackled to compute such sets of candidate bags. A bag B_u in a GHD can be any subset of $B(\lambda_u)$ and choosing smaller subsets can decrease the width. At the same time, enumerating all subsets of $B(\lambda_u)$ is not an option if we are interested in classes of hypergraphs with unbounded rank. The main reason why the Check problem is tractable for HDs is that the additional special condition severely restricts the possible choices of B_u for given $B(\lambda_u)$.

Example 4.5 (Example 4.4 continued). In Figure 4.5, we have two GHDs of width 2 of the hypergraph H_0 from Figure 4.3. In the root u_0 of both GHDs, we have $v_2 \in B(\lambda_{u_0})$ since $v_2 \in e_2$ but $v_2 \notin B_{u_0}$. Hence, both GHDs violate the special

condition in node u_0. However, if v_2 were added to B_{u_0}, then it can be seen that covering the edges e_1, e_2, e_7, e_8 below u_0 is no longer possible in a width 2 GHD. That is why the HD in Figure 4.4 has width 3. △

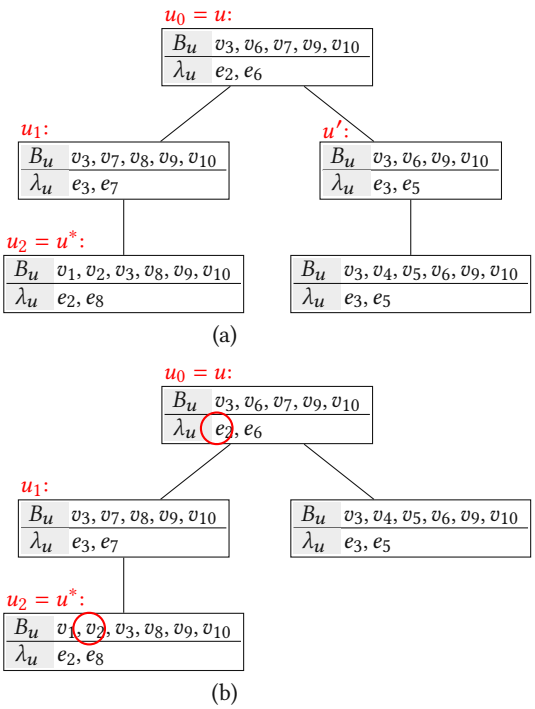


Figure 4.5: (a) non bag-maximal vs. (b) bag-maximal GHD of hypergraph H_0 in Figure 4.3

We start by introducing a useful property of GHDs, which we will call *bag-maximality*. Let $\mathcal{D} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ 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 4.16. Let $\mathcal{D} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ be a GHD of some hypergraph $H = (V(H), E(H))$. We call $\mathcal{D}$ *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 $\leq k$. However, since we want to build on the algorithm from Section 4.1, we need to show that this also holds for bag-maximal ComNF GHDs. The problem here is that adding vertices to a bag B_u, to make it maximal, can change the set of $[B_u]$-components. Fortunately, we can reuse previous arguments on the existence of hypertree decompositions to show that if H has a GHD of width $\leq k$, then it indeed also has a bag-maximal ComNF GHD of width $\leq k$.

We now carry over several properties of HDs from [68]. An inspection of the corresponding proofs in [68] reveals that these properties hold also in the generalized case. We thus state the following results below without explicitly "translating" the proofs of [68] to the generalized setting. Note that [68] deals with HDs and, therefore, in all decompositions considered there, the special condition holds. However, in Lemmas 4.17 and 4.18 below, the special condition is not needed.

We briefly recall the crucial notation for the following lemmas. For a set $V' \subseteq V(H)$, we define $\text{nodes}(V') = \{u \in T \mid B_u \cap V' \neq \emptyset\}$. If we want to make explicit the decomposition $\mathcal{G}$, we also write $\text{nodes}(V', \mathcal{G})$ synonymously with $\text{nodes}(V')$. By further overloading the nodes operator, we also write $\text{nodes}(T_u)$ or $\text{nodes}(T_u, \mathcal{G})$ to denote the nodes in a subtree T_u of T, i.e., $\text{nodes}(T_u) = \text{nodes}(T_u, \mathcal{G}) = \{v \mid v \in T_u\}$.

Lemma 4.17 (Lemma 5.2 from [68])**.** *Consider a GHD $\mathcal{D} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ 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,* $\text{nodes}(C, \mathcal{D}) \subseteq \text{nodes}(T_s)$.

Lemma 4.18 (Lemma 5.3 from [68])**.** *Consider a GHD $\mathcal{D} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ 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* $\text{nodes}(U, \mathcal{D})$ *induces a (connected) subtree of T.*

Lemma 4.19. *For every GHD $\mathcal{D} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ of width k of a hypergraph H, there exists a bag-maximal ComNF GHD $\mathcal{D}$ of H of width $\leq k$.*

Proof. Start with a GHD $\mathcal{D} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ 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.

We proceed by restating a procedure from [68] that fixes violations of the ComNF condition:. for the bag-maximal GHD obtained above, assume that there exist two nodes r and s such that s is a child of r, and the ComNF condition is violated for the pair, i.e., there does not exist a single $[B_r]$-component C_s such that $V(T_s) = C_s \cup (B_r \cap B_s)$. Let $C_1, \ldots, C_h$ be all the $[B_r]$-components containing some vertex occurring in $V(T_s)$. Hence, $V(T_s) \subseteq \left(\bigcup_{j=1}^{h} C_j \cup B_r\right)$. For each $[B_r]$-component C_j $(1 \leq j \leq h)$, consider the set nodes$(C_j, \mathcal{D})$. By Lemma 4.18, nodes$(C_j, \mathcal{D})$ induces a subtree of T, and by Lemma 4.17, nodes$(C_j, \mathcal{D}) \subseteq$ nodes(T_s). Hence nodes$(C_j, \mathcal{D})$ induces in fact a subtree of T_s.

For each node $n \in$ nodes$(C_j, \mathcal{D})$ define a new node $u_{n,j}$ and let $\lambda_{u_{n,j}} = \lambda_n$ and $B_{u_{n,j}} = B_n \cap (C_j \cup B_r)$. Note that $B_{u_{n,j}} \neq \emptyset$, because by definition of nodes$(C_j, \mathcal{D})$, B_n contains some vertex belonging to C_j. Let $N_j = \{u_{n,j} \mid n \in \text{nodes}(C_j, \mathcal{D})\}$ and, for any C_j $(1 \leq j \leq h)$, let T_j denote the (directed) graph (N_j, E_j) such that $u_{p,j}$ is a child of $u_{q,j}$ if and only if p is a child of q in T. T_j is clearly isomorphic to the subtree of T_s induced by nodes$(C_j, \mathcal{D})$, hence T_j is a tree as well.

Now transform the GHD $\mathcal{D}$ as follows: delete the subtree T_s from T and attach to r every tree T_j for $1 \leq j \leq h$. In other words, we replace the subtree T_s by a set of trees $\{T_1, \ldots, T_h\}$. By construction, T_j contains a node $u_{n,j}$ for each node n belonging to nodes$(C_j, \mathcal{D})$ $(1 \leq j \leq h)$. Then, if we let children(r) denote the set of children of r in the new tree T obtained after the transformation above, it holds that for any $s' \in$ children(r), there exists a $[B_r]$-component C of H such that nodes$(T_{s'})$ = nodes$(C, \mathcal{D})$, and $V(T_{s'}) \subseteq (C \cup B_r)$. We want to show $V(T_{s'}) = C \cup (B_{s'} \cap B_r)$. For the "$\supseteq$"-direction, we observe that $C \subseteq T_{s'}$ clearly holds, since we have $C \subseteq T_s$ and the bags $B_{u_{n,j}}$ in $T_{s'}$ were obtained from B_n in T_s as $B_{u_{n,j}} = B_n \cap (C_j \cup B_r)$ and we are considering the component $C = C_j$ here. Moreover, $B_{s'} \cap B_r \subseteq B_{s'} \subseteq V(T_{s'})$ clearly holds. Hence, we have $C \cup (B_{s'} \cap B_r) \subseteq V(T_{s'})$.

For the "$\subseteq$"-direction, we conclude from $V(T_{s'}) \subseteq C \cup B_r$ that also $V(T_{s'}) \subseteq$

$C \cup (V(T_{s'}) \cap B_r)$ holds. Hence, it suffices to show that $(V(T_{s'}) \cap B_r) \subseteq B_{s'}$ holds. By connectedness, we have $V(T_s) \cap B_r \subseteq B_s \cap B_r$ and, therefore, also $V(T_{s'}) \cap B_r \subseteq B_s \cap B_r$. Moreover, by construction, we have $B_{s'} = B_s \cap (C \cup B_r)$ and, therefore $B_s \cap B_r \subseteq B_{s'}$. We thus also arrive at $V(T_{s'}) \subseteq C \cup (B_{s'} \cap B_r)$.

It remains to show that $T_{s'}$ is bag-maximal. Assume to the contrary that there exists a node $u_{n,j} \in T_{s'}$ and a vertex $v \in B(\lambda_{u_{n,j}}) \setminus B_{u_{n,j}}$ such that v can be added to $B_{u_{n,j}}$ without destroying the GHD properties. Recall that $B_{u_{n,j}} = B_n \cap (C_j \cup B_r)$ and $\lambda_{u_{n,j}} = \lambda_n$. Since T was bag-maximal initially and from the construction (which only makes bags smaller), the only candidates for such a v are those vertices $B_n \setminus (C_j \cup B_r)$ that got removed from the bag. However, all neighboring nodes are either r or in $T_{s'}$, which means their bags are subsets of $C_j \cup B_r$. Hence, no $v \in B_n \setminus (C_j \cup B_r)$ is contained in a neighbor of $u_{n,j}$ and adding it would break connectedness. Our newly constructed $T_{s'}$ is therefore also bag-maximal.

Iterating this procedure for all ComNF violations will eventually produce a new GHD that is still bag-maximal and in ComNF. □

Example 4.6 (Example 4.5 continued). Clearly, the GHD in Figure 4.5(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 4.5(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}. △

For the following arguments, the reader is advised to be careful in distinguishing between the bag B_u of a node u and the set of vertices $B(\lambda_u)$ that are covered by the integral edge cover λ_u. Before we prove a crucial lemma, we introduce some useful notation:

Definition 4.20. Let $\mathcal{D} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ be a GHD of a hypergraph H. Moreover, let u be a node in $\mathcal{D}$ 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 $\pi = (u_0, u_1, \ldots, u_\ell)$ with $u_0 = u$ and $u_\ell = u^*$ the *critical path* of (u, e) denoted as $\mathrm{critp}(u, e)$.

Lemma 4.21. *Let $\mathcal{D} = \langle T, (B_u)_{u\in T}, (\lambda_u)_{u\in T}\rangle$ 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.*

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

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_j} for every $j \in \{1, \ldots, \ell\}$. Therefore, $e \cap B_u \subseteq e \cap \bigcap_{j=1}^{\ell} B(\lambda_{u_j})$ holds.

"$\supseteq$": Assume to the contrary that there exists some vertex $v \in e$ with $v \notin B_u$ but $v \in \bigcap_{j=1}^{\ell} B(\lambda_{u_j})$. 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, $v \in \bigcap_{j=1}^{\ell} B(\lambda_{u_j})$ 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 $\mathcal{D}$. □

Example 4.7 (Example 4.5 continued). Consider root node u of the GHD in Figure 4.5(b). We have $e_2 \in \lambda_u$ and $e_2 \setminus B_u = \{v_2\} \neq \emptyset$. As e_2 is covered by u_2, 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. △

Lemma 4.21 characterizes the overlap of an edge with a bag as an intersection of unions. However, to utilize the proposed intersection constraints, we would prefer unions of intersections instead. A straightforward transformation from an intersection of unions to a union of intersections may introduce certain redundant terms that we would like to avoid for technical reasons. We therefore employ a particular transformation, via the $\bigcup\bigcap$-trees defined below, that avoids such redundant terms in the union.

Definition 4.22. Let H be a hypergraph, e an edge of H and let $Q_1, \ldots, Q_\ell$ be sets of edges. The $\bigcup\bigcap$-tree T of $e, Q_1 \ldots, Q_\ell$ is the output of Algorithm 4.3 with inputs $e, Q_1, \ldots Q_\ell$. We refer to the set of all leaves of T as $leaves(T)$.

Algorithm 4.3: `Union-of-Intersections-Tree`

input: An edge $e \in E(H)$, sets $Q_1, \ldots, Q_\ell$ of edges of $E(H)$
output: $\bigcup\bigcap$-tree T of $e, Q_1, \ldots Q_\ell$

`/* Initialization: compute (N,E) for T_0 */`
1 $N \leftarrow \{r\}$;
2 $E \leftarrow \emptyset$;
3 $label(r) \leftarrow \{e\}$;
4 $T \leftarrow (N, E)$;

`/* Compute T_j from T_{j-1} in a loop over j */`
5 **for** $j \leftarrow 1$ **to** ℓ **do**
6 **foreach** *leaf node* p *of* T **do**
7 **if** $label(p) \cap Q_j = \emptyset$ **then**
8 Let $Q_j = \{e_{j1}, \ldots, e_{jh_j}\}$;
9 Create new nodes $\{p_1, \ldots, p_{h_j}\}$;
10 **for** $\alpha \leftarrow 1$ **to** h_j **do** $label(p_\alpha) \leftarrow label(p_\alpha) \cup \{e_{j\alpha}\}$;
11 $N \leftarrow N \cup \{p_1, \ldots, p_{h_j}\}$;
12 $E \leftarrow E \cup \{(p, p_1), \ldots, (p, p_{h_j})\}$;
13 **end**
14 **end**
15 $T \leftarrow (N, E)$;
16 **end**

Lemma 4.23. *Let H be a hypergraph, e an edge of H and let $Q_1, \ldots, Q_\ell$ be sets of edges. Let T be the $\bigcup\bigcap$-tree of $e, Q_1, \ldots, Q_\ell$, then*

$$e \cap \bigcap_{j=1}^{\ell} \bigcup Q_j = \bigcup_{p \in leaves(T)} \bigcap label(p)$$

Proof. Proof is by induction over ℓ. For $\ell = 0$, we have $leaves(T) = \{r\}$ and $label(r) = e$ and the statement trivially holds. For $0 \leq j \leq \ell$, let T_j denote the $\bigcup\bigcap$-tree of $e, Q_1, \ldots, Q_j$. Suppose the statement is true for $\ell - 1$, then we observe the following equality

$$e \cap \bigcap_{j=1}^{\ell} \bigcup Q_j = \left(e \cap \bigcap_{j=1}^{\ell-1} \bigcup Q_j\right) \cap \bigcup Q_\ell = \bigcup_{p \in leaves(T_{\ell-1})} \left(\bigcap label(p) \cap \bigcup Q_\ell\right)$$

where the right equality follows from the induction hypothesis and distribution of $\bigcup Q_\ell$ over the union over the leaves of $T_{\ell-1}$. Now, consider a leaf $p \in leaves(T_{\ell-1})$. The construction of T_ℓ either adds new leaves new_p as children of p, or p remains a leaf in T_ℓ. We claim that in the first case $\bigcap label(p) \cap \bigcup Q_\ell = \bigcup_{p' \in new_p} \bigcap label(p')$ and in the second case, $\bigcap label(p) \cap \bigcup Q_\ell = \bigcap label(p)$. If the claim holds, we have the following equality and the statement follows immediately.

$$\bigcup_{p \in leaves(T_{\ell-1})} \left(\bigcap label(p) \cap \bigcup Q_\ell \right) = \bigcup_{p \in leaves(T_\ell)} \bigcap label(p)$$

What is left is to verify the claim. The case where new children are added to p is straightforward by distributivity as the label of each new child corresponds to a term of the union $(\bigcap label(p) \cap e_{\ell 1}) \cup \cdots \cup \left(\bigcap label(p) \cap e_{\ell h_\ell}\right) = \bigcap label(p) \cap \bigcup Q_\ell$. If p remains a leaf, then we have $label(p) \cap Q_\ell \neq \emptyset$. Thus, $\bigcap label(p) \subseteq \bigcup Q_\ell$ and therefore $\bigcap label(p) \cap \bigcup Q_\ell = \bigcap label(p)$.

□

Throughout the rest of this paper, we will be interested in how bags can be represented as combinations of edges. In particular, we will see that, under the various restrictions introduced at the beginning of this section, we are able to bound the representation of bags as *unions of intersections* of edges. After introducing some notation for such unions of intersections we can show how the BMIP allows for a bounded representation of bags for GHDs. The main result then follows by using this representation to compute an appropriate set of candidate bags, to which Algorithm 4.1 from Section 4.1 can then be applied.

Definition 4.24. Let H be a hypergraph. A (q, p)*-set* $X \subseteq V(H)$ is a set of the form $X = X_1 \cup \cdots \cup X_{q'}$ with $q' \leq q$ and where every X_i is the intersection of at most p edges. We will use q-set as shorthand for $(q, 1)$-set.

We will repeatedly make use of the fact that, by the idempotence of union and intersection, we can w.l.o.g. assume a (q, p)-set to be the union of exactly q terms, each consisting of the intersection of exactly p edges.

Lemma 4.25. *Let H be a hypergraph with c-miwidth$(H) \leq i$ and let $B_1, \ldots, B_\ell$ be $(q, 1)$-sets. For each $e \in E(H)$, there exists a (q^{c-1}, c)-set I and a subedge $e' \subseteq e$ with $|e'| \leq i\, q^c$, such that*

$$e \cap \bigcap_{j=1}^{\ell} B_j = I \cup e'$$

Furthermore, e' is the union of at most q^c subsets of intersections of exactly c edges.

Proof. For $j \in [\ell]$, fix a set of edges $Q_j = \{e_{j1}, \ldots, e_{jq}\}$ such that $B_j = e_{j1} \cup \cdots \cup e_{jq}$. Let T be the $\bigcup\bigcap$-tree of $e, Q_1, \ldots, Q_\ell$. For a node p of T, we refer to the number of edges in the path from the root to p as the depth of p, or $depth(p)$. Note that by construction, $|label(p)| = depth(p) + 1$ for each node p in T. We consider the following partition of $leaves(T)$: let $SMALL$ contain all the leaves of T at depth at most $c - 1$ and, conversely, let $FULL$ be the set of leaves at depth at least c.

By Lemma 4.23, $e \cap \bigcap_{j=1}^{\ell} B_j = \left(\bigcup_{p \in SMALL} \bigcap label(p)\right) \cup \left(\bigcup_{p \in FULL} \bigcap label(p)\right)$. Hence, to prove the lemma, it suffices to show that $\bigcup_{p \in SMALL} \bigcap label(p)$ is a (q^{c-1}, c)-set and that $|\bigcup_{p \in FULL} \bigcap label(p)| \leq iq^c$ holds.

Claim A. *$\bigcup_{p \in SMALL} \bigcap label(p)$ is a (q^{c-1}, c)-set.*

Proof of Claim A. Since each of the sets Q_j has at most q members, every node in T has at most q children. Hence, there are at most q^{c-1} leaves at depth $\leq c - 1$ and therefore $|SMALL| \leq q^{c-1}$. Furthermore, we have $|label(p)| = depth(p) + 1 \leq c$, i.e., each intersection has at most c terms. ◇

Claim B. *$|\bigcup_{p \in FULL} \bigcap label(p)| \leq iq^c$.*

Proof of Claim B. First, observe that for each $p \in FULL$, there exists a node p' in T at depth c such that $label(p) \supseteq label(p')$ and therefore also $\bigcap label(p) \subseteq \bigcap label(p')$. Note that there are at most q^c nodes p' at depth c. Furthermore, because $|label(p')| = c$ and we assume c-miwidth$(H) \leq i$, it holds that $|\bigcap label(p')| \leq i$. In total, we thus have that $\bigcup_{p \in FULL} \bigcap label(p)$ is a union of sets X_p such that each X_p is the subset of one out of at most q^c vertex sets, and each of these vertex sets has cardinality at most i.

□

For a given edge cover λ_u and arbitrary edge $e \in E(H)$ with $\lambda_u(e) = 1$, Lemma 4.21 gives us a representation of $e \cap B_u$ of the form $e \cap \bigcap_{j=1}^{\ell} B(\lambda_{u_j})$. Clearly, the sets $B(\lambda_{u_j})$ are $(k, 1)$-sets, i.e., unions of (up to) k edges. We can therefore apply Lemma 4.25 by taking $B_j = B(\lambda_{u_j})$ and $q = k$ to get a representation of the form $I \cup e'$ for each of the possible subedges $e \cap B_u$ that may ever be used in a bag-maximal ComNF GHD. This idea is formalized in the following lemma, where we identify a polynomially big family of vertex sets $\mathbf{S} \subseteq 2^{V(H)}$, such that the bags of any bag-maximal ComNF GHD of H must be a member of this family.

Lemma 4.26. *Let H be a hypergraph with c-miwidth$(H) \leq d$ and fix an integer $k > 0$. There exists a set $\mathbf{S} \subseteq 2^{V(H)}$, which can be computed in polynomial time and logarithmic space(for fixed c, d, and k), such that $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$ if and only if $ghw(H) \leq k$.*

Furthermore, for any bag-maximal ComNF GHD $\langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ of H of width $\leq k$, we have $\langle T, (B_u)_{u \in T} \rangle \in \mathbf{ComCTD}(\mathbf{S})$.

Proof. Let $n = ||H||$ refer to the size of H and $m = |E(H)|$. We define the following sets:

$\mathcal{I} = \{I \mid I \text{ is a } (k^{c-1}, c)\text{-set}\}$

$C = \{e' \mid \text{there exist distinct } e_1, \ldots, e_c \in E(H), \text{ such that } e' \subseteq e_1 \cap \cdots \cap e_c\}$

$Sub = \{I \cup \bigcup_{j=1}^{k^c} C_j \mid I \in \mathcal{I}, C_1, \ldots, C_{k^c} \in C, \text{ and } I \cup \bigcup_{j=1}^{k^c} C_j \subseteq e \text{ for some } e \in E(H)\}$.

By construction, Sub contains only $E(H)$ and subedges of H. There are no more than $m^{ck^{c-1}}$ possible (k^{c-1}, c) sets. Also, by the condition c-miwidth$(H) \leq d$, we have $|C| \leq 2^d m^c$ and, therefore, Sub has at most $m^{ck^{c-1}} m^{ck^c} 2^{dk^c}$ elements. We can then construct our desired set $\mathbf{S}$ as the set of all unions of up to k elements of Sub.

From these polynomial size bounds of the sets it is now not difficult to see that they can be computed in logarithmic space. The considerations of the previous paragraph allow us to simplify our argument for logartihmic space by observing that the cardinality of $\mathbf{S}$ is still polynomially bounded if we take no

care of removing duplicates elements, i.e., if we consider the definitions at the beginning of the proof to be over on multisets. Since k, c, and d are constants and no deduplication is necessary it is straightforward to enumerate the set $\mathcal{I}$ by $k^{c-1}c$ nested loops. Similarly, for C it is simple to enumerate up to c distinct edges, then for each such combination, enumerate all sets of vertices with up to d element and check if every vertex is present in all of the (constantly many) edges. The final union is then again simple since we can implement a union over multisets by simply copying the representations of the sets to the output.

It remains to show that this $\mathbf{S}$ indeed has the property that (1) $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$ if and only if $ghw(H) \leq k$ and (2) that for any bag-maximal ComNF GHD $\langle T, (B_u)_{u\in T}, (\lambda_u)_{u\in T}\rangle$ of H of width $\leq k$, we have $\langle T, (B_u)_{u\in T}\rangle \in \mathbf{ComCTD}(\mathbf{S})$.

First, assume $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$. Then there exists a TD of H where each bag is in $\mathbf{S}$ and therefore a union of k subedges of H. Hence, every bag of the TD can also be covered by k edges of H and thus can clearly be turned into a GHD of width at most k.

Now, assume $ghw(H) \leq k$. Let $\mathcal{D} = \langle T, (B_u)_{u\in T}, (\lambda_u)_{u\in T}\rangle$ be a bag-maximal ComNF GHD of width at most k and let u be a node of T. By Lemma 4.19, such a GHD always exists if $ghw(H) \leq k$. W.l.o.g. we assume that $B(\lambda_u) = e_1 \cup \cdots \cup e_k$ and, therefore, also

$$B_u = B_u \cap B(\lambda_u) = (B_u \cap e_1) \cup \cdots \cup (B_u \cap e_k)$$

We will show that $B_u \cap e_j \in Sub$ for each $j \in [k]$ and therefore also $B_u \in \mathbf{S}$. The case where $e_j \cap B_u = e_j$ is trivial as $e_j \in E(H) \subseteq Sub$. So, let e_j be any edge from this representation of $B(\lambda_u)$ where $e_j \cap B_u \neq e_j$. By Lemma 4.21 the following equality holds for the critical path $(u, u_1, \ldots, u_\ell)$ of (u, e_j)

$$e_j \cap B_u = \quad e_j \cap \bigcap_{j=1}^{\ell} B(\lambda_{u_j})$$

By assumption, every such $B(\lambda_{u_j})$ is a k-set and hence, by Lemma 4.25, we know that $e_j \cap \bigcap_{j=1}^{\ell} B(\lambda_{u_j})$ is precisely the union of an element of $\mathcal{I}$ and at most k^c sets from C, i.e., $e_j \cap B_u \in Sub$. Since the choice of u was arbitrary, every bag of $\mathcal{D}$ is contained in $\mathbf{S}$ and we have $\mathcal{D} \in \mathbf{ComCTD}(\mathbf{S})$. □

Proof of Theorem 4.15: We assume that C enjoys the BMIP, i.e., for every $H \in C$, we have c-miwidth$(H) \leq a \log n$. To solve the Check(GHD, k) problem, we can then simply compute, in logarithmic space, the set S from Lemma 4.26 and decide whether **ComCTD**(S) $\neq \emptyset$. By Theorem 4.5 this problem is in LogCFL and therefore, so is the whole procedure. □

Recall the definition of the degree of a hypergraph. This is a standard concept in the study of graphs and hypergraphs and we therefore also consider it here. Let C be a class of hypergraphs. We say that C has the *bounded degree property (BDP)* if there exists a constant d such that every hypergraph H in 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 4.27. *For every class C of hypergraphs of bounded degree, for each constant k, the problem Check(GHD, k) is in LogCFL.*

In case of the BMIP, the upper bound on S in the proof of Lemma 4.26, is $2^{f(i,k,c)} m^{g(k,c)}$ for some function g. Recall from Theorem 4.5 that **ComCTD**(**S**)$\neq \emptyset$ can be decided in time complexity that is polynomial in $|\mathbf{S}| \cdot |V(H)|$. We thus elegantly arrive at the related parameterized complexity result. This parameterized result was already stated by Fischl, Gottlob, and Pichler [54] with a different argument. We recall the result here since it follows naturally from our approach.

Corollary 4.28 ([54]). *For constants k and c, the Check(GHD, k) problem parameterized by the intersection-width i is fixed-parameter tractable for hypergraphs enjoying the ic-BMIP.*

4.3 From GHDs to q-Limited Monotone Width Checking

In Section 4.2, we have shown that under certain conditions (with the BIP and BDP as most specific and the BMIP as most general conditions) the problem of computing a GHD of width $\leq k$ can be reduced to finding a ComNF candidate tree decomposition for an appropriate set of candidate bags. The key to this problem reduction was to enumerate a set of subedges, which allowed us to enumerate all bags of possible bag-maximal GHDs of width $\leq k$. When trying to carry over these ideas from GHDs to FHDs, we encounter *two major challenges*: Can we adapt the candidate tree decomposition approach that we used for GHDs to work with FHDs? And is it possible to find bounded representations of all the sets of vertices that can be *fractionally* covered with weight $\leq k$?

For the second challenge, recall from the GHD-case that the possible bags $B(\lambda_u)$ could be easily computed from the given set of subedges, since each λ_u can choose at most k subedges. In contrast, for a fractional cover γ_u, we do not have such a bound on the size of the support, i.e., number of edges with non-zero weight. It is easy to exhibit a family $(H_n)_{n\in\mathbb{N}}$ of hypergraphs where it is advantageous to have unbounded $\mathrm{supp}(\gamma_n)$ even if $(H_n)_{n\in\mathbb{N}}$ enjoys the BIP, as the following example illustrates:

Example 4.8. 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 $\mathrm{iwidth}(H_n) = 1$, but an optimal fractional edge cover of H_n is obtained by the following mapping γ with $\mathrm{supp}(\gamma) = E_n$:

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

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

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

△

Below we show that, for cases where the second challenge can be resolved (i.e., we can establish an upper bound on $\text{supp}(\gamma_u)$ for all nodes u in an FHD), the check-problem of FHDs can be essentially reduced to the GHD case. The following Theorem 4.31 is thus the crucial tool for the remainder of this chapter. To introduce it in the most general way we first introduce the new notion of q-limited width. We first give the specific definition for q-limited fhw, which is more intuitive.

Definition 4.29 (q-support fractional hypertree width)**.** Let $\rho^*_q(U)$ be the minimal weight of an edge weight function γ such that $U \subseteq B(\gamma)$ and $|\text{supp}(\gamma)| \leq q$. We define the *$q$-support* fractional hypertree width of a hypergraph H as its ρ^*_q-width.

The definition is motivated by the observations from Example 4.8 above. In the q-support variant, we are only interested in the fhw that can be reached when the support of the covers must be bounded by q. Ultimately our aim is to show that there exists a constant q such that the q-support fhw always equals plain fhw. Tractability then follows by the main theorem of this section, which shows that Check(ρ^*_q-width, k) can be reduced to Check(ghw, k), as long as we have bounded multi-intersection width. Indeed, this is not a property that is particular to ρ^*_q-width but a general property of all *monotone* f-widths. The general definition is slightly different than Definition 4.29 since we do not have a notion of *support* for every f-width. We will see later that, assuming the BMIP, that q-support fhw can always be expressed as q-limited f-width in an equivalent hypergraph (that can be obtained in polynomial time).

Definition 4.30 (q-limited f-width)**.** Let f be a width function. We define the *q-limited* f-width of a hypergraph H as the minimal f-width over all tree decompositions where for each bag B_u we have that there exists a q-set G, such that $B_u \subseteq G$ and $f(G) \leq k$.

To make the reduction tractable we naturally need to restrict ourselves to functions $f\colon 2^{V(H)} \mapsto \mathbb{R}^+$ that can always be computed in polynomial time w.r.t. the size of H. We say that functions that are monotone (i.e., if $U \subseteq W$, then $f(U) \leq f(W)$) and polynomial time computable in this sense are *conservative width functions*. We are now ready to state the main result of this section.

Theorem 4.31. *Fix c, d, and q as constant integers and let f be a conservative width function. There is a polynomial time algorithm testing whether a given hypergraph H with c-miwidth$(H) \leq i$ has q-limited f-width at most k.*

Proof. We first define $\mathbf{S}$ by making use of results from the GHD-case: let $\mathbf{S}_{ghd}$ be the set from Lemma 4.26 such that $\mathbf{ComCTD}(\mathbf{S}_{ghd}) \neq \emptyset$ if and only if $ghd(H) \leq q$. We know that such a set exists and can be computed in polynomial time. We can then obtain the required set $\mathbf{S} = \{S \in \mathbf{S}_{ghd} \mid \exists q\text{-set} G : S \subseteq G \wedge f(G) \leq k\}$ by checking the condition for every element of $\mathbf{S}_{ghd}$. Since q is constant, the check can be realized in polynomial time by straightforward enumeration of all combinations of up to q edges. Recall that f is conservative and $\mathbf{S}_{ghd}$ has polynomially boundable size. Hence, also $\mathbf{S}$ can be computed in polynomial time.

We now argue that $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$ if and only if q-limited f-width$(H) \leq k$. Suppose $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$, then clearly there is a TD where every bag B_u has a q-set G such that $f(G) \leq k$ and $B_u \subseteq G$. Since f is monotone by assumption, it also follows that $f(B_u) \leq k$ and therefore we have q-limited f-width$(H) \leq k$.

For the other direction, suppose q-limited f-width of H is at most k, and let $\mathcal{T} = \langle T, (B_u)_{u \in T} \rangle$ be a TD with minimal q-limited f-width. That is, for each node $u \in T$, B_u is a subset of some set Q_u with $\rho(Q_u) \leq q$. Thus, Q_u is a q-set and can therefore be expressed in the form $Q_u = e_{u1} \cup \cdots \cup e_{uq}$. Now let $\lambda_u = \{e_{u1}, \ldots, e_{uq}\}$ for each $u \in T$. It is easy to see that $\mathcal{D} = \langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ is a GHD of H with $ghw(\mathcal{D}) \leq q$. Note that $\mathcal{D}$ is not necessarily bag-maximal or in ComNF. However, following the procedure described in Lemma 4.19, there exists a bag-maximal ComNF $\mathcal{D}' = \langle T', (B'_u)_{u \in T'}, (\lambda'_u)_{u \in T'} \rangle$ with $ghw(\mathcal{D}') \leq q$ and therefore $\langle T', (B'_u)_{u \in T'} \rangle \in \mathbf{ComCTD}(\mathbf{S}_{ghd})$.

Now recall that the transformation into a bag-maximal ComNF TD $\mathcal{D}'$ from Lemma 4.19 uses the same covers as the input decomposition. That is, every cover λ' in $\mathcal{D}'$ is equal to some cover λ in $\mathcal{D}$. By construction, then for every cover λ' in $\mathcal{D}'$ it holds that $f(\bigcup \lambda') \leq k$ and $|\lambda'| \leq q$. Since f is conservative, and therefore monotone, it follows for every node u in $\mathcal{D}'$ that $f(B'_u) \leq f(\bigcup \lambda'_u) \leq k$. Therefore, $\langle T', (B'_u)_{u \in T'} \rangle$ is in ComNF and each of its bags is in $\mathbf{S}$, i.e., $\langle T', (B'_u)_{u \in T'} \rangle \in \mathbf{ComCTD}(\mathbf{S})$.

We therefore have established that $\mathbf{ComCTD}(\mathbf{S}) \neq \emptyset$ if and only if q-limited f-width$(H) \leq k$. Since $\mathbf{S}$ is computable in polynomial time, combination with Theorem 4.5 yields a polynomial time algorithm for deciding whether q-limited f-width$(H) \leq k$. □

Despite the general statement for any conservative f-width, we only apply the theorem to *fhw* in the following. Note that ρ^* is a conservative width function since it can be computed polynomially by a straightforward linear program. However, recall that adaptive width and submodular width are defined over monotone sets of functions. Very little is known about the complexity of checking these widths and the possible application of Theorem 4.31 in those settings is a possible subject for future work. Our slight detour to the general case of arbitrary (conservative) f-widths ends here and for the rest of this chapter we will focus on fractional hypertree width.

Similarly as in the GHD-case, we will have to deal with unions of intersections of edges also in the FHD-case. The following definition and the accompanying two lemmas will be convenient for this purpose.

Definition 4.32. For a hypergraph H we write $H^\cap$ for the closure of H under intersection of edges.

It will be important to observe that adding subedges does not change the fractional hypertree width of a hypergraph. Every FHD of H is still an FHD of $H^\cap$ and every FHD of $H^\cap$ can be easily transformed into an FHD of H. It follows that we always have $fhw(H) = fhw(H^\cap)$.

Lemma 4.33. *Let H be a hypergraph and γ a fractional edge cover of a set S of vertices. Then $B(\gamma)$ is a $2^{|\operatorname{supp}(\gamma)|}$-set w.r.t. $H^\cap$.*

Proof. Let us call a subset $E \subseteq \operatorname{supp}(\gamma)$ full if $\sum_{e \in E} \gamma(e) \geq 1$ and let $\mathbf{FS}$ contain all the full subsets. For every $E \in \mathbf{FS}$, we have $\bigcap E \subseteq B(\gamma)$ and it is straightforward to verify the following equality:

$$B(\gamma) = \bigcup_{E \in \mathbf{FS}} \bigcap E$$

It is then enough to observe that $\bigcap E \in E(H^\cap)$ and that there are at most $2^{|\operatorname{supp}(\gamma)|}$ full subsets. □

Lemma 4.34. *Let H be a hypergraph with c-miwidth$(H) \leq i$. Then $H^\cap$ can be computed in polynomial time for fixed c and i. Moreover, 2^c-miwidth$(H^\cap) \leq i$ holds.*

Proof. First of all note that if $|E(H^\cap)| \leq 2^c$, then also $|E(H)| \leq 2^c$ and we can trivially compute $H^\cap$ in polynomial time by simply computing all possible intersections of edges in $E(H)$. Moreover, in this case, the condition 2^c-*miwidth*$(H^\cap) \leq i$ is void, since there are no distinct 2^c edges in $H^\cap$.

Now let $\alpha = 2^c$ and consider an intersection I of α distinct edges of $H^\cap$ of the form $I = e'_1 \cap \cdots \cap e'_\alpha$, where $e'_j \in E(H^\cap)$ for $j \in [\alpha]$. Each $e'_j \in E(H^\cap)$ is an intersection of edges from $E(H)$, i.e., there exists $\mathcal{E}_j \subseteq E(H)$ such that $e'_j = \bigcap_{e \in \mathcal{E}_j} e$. Clearly, $I = \bigcap \left(\bigcup_{j=1}^{\alpha} \mathcal{E}_j \right)$.

We claim that $|\bigcup_{j=1}^{\alpha} \mathcal{E}_j| \geq c$, i.e., I is the intersection of at least c edges from $E(H)$. Indeed, less than c distinct edges, there could only be less than $2^c - 1$ non-empty sets $\mathcal{E}_j$. It thus follows that I is a subset of an intersection of at least c edges of $E(H)$. Thus, by c-miwidth$(H) \leq i$, we conclude that $|I| \leq i$ holds. Hence, also in the case $|E(H^\cap)| > 2^c$, the condition 2^c-*miwidth*$(H^\cap) \leq i$ holds.

It remains to show that $E(H^\cap)$ can be computed from $E(H)$ in polynomial time: let $m = |E(H)|$. Then there are less than m^c intersections of less than c distinct edges from $E(H)$ and these intersections can clearly be computed in polynomial time. In order to compute also the set of intersections of at least c edges from $E(H)$, we proceed as follows: we first compute the set $\mathcal{I}_0$ of intersections of c edges. Then, for every $j \in [m - c]$, we compute the set $\mathcal{I}_j$ of intersections of α edges from $E(H)$ with $\alpha \in \{c, \ldots, c + j\}$. By c-miwidth$(H) \leq i$, we know that $|\mathcal{I}_j| \leq 2^i m^c$ holds for every $j \in [m - c]$. Hence, all these intersections can clearly be computed in polynomial time (for fixed c and i). □

With these helpful statements about $H^\cap$ in hand, it is now simple to extend Theorem 4.31 to tractable q-support *fhw* checking.

Lemma 4.35. *Let H be a hypergraph. The 2^q-limited rho*-width of $H^\cap$ is less or equal the q-support fhw of H.*

Proof. Suppose a FHD $\mathcal{F}$ with q-support fhw k. According to Lemma 4.33, for every node u in $\mathcal{F}$, every $B(\gamma)$ is a 2^q-set in $H^\cap$. Furthermore, as all edges of H are still present in $H^\cap$, clearly also $\rho^*(B(\gamma)) \leq k$ in $H^\cap$. Thus, 2^q-limited ρ^*-width of $H^\cap$ is at most the q-support fhw of H. □

Thus, we can reduce checking q-support fhw to checking 2^q-limited ρ^* by simply computing the closure under intersection of edges. By Lemma 4.34, this closure can be computed in polynomial time in the BMIP setting. Furthermore, computing the closure preserves the BMIP (one constant becomes larger). As was discussed above, ρ^* is a conservative width function and thus Theorem 4.31 applies. The correctness of the reduction follows immediately from the previous Lemma 4.35. We thus obtain the following important Corollary that will be crucial for the further results in this chapter.

Corollary 4.36. *Fix c, d, and q as constant integers and let f be a conservative width function. There is a polynomial time algorithm testing whether a given hypergraph H with c-*miwidth$(H) \leq i$ *has q-support fhw at most k.*

In order to apply Corollary 4.36, we will prove that an appropriate constant q exists such that $q - fhw(H) = fhw(H)$ for any hypergraph H under the respective restrictions that we consider. In the following section we show that such a constant indeed exists for hypergraph classes of bounded intersection width. Furthermore, in Section 4.5, we discuss how we recently extended this result even further to bounded multi-intersection width.

4.4 Checking Fractional Hypertree Width (BIP)

In this section we prove tractability of checking fhw for hypergraph classes enjoying the BIP. As mentioned before, our aim is to utilize Corollary 4.36, by showing that there exists a constant q (depending only on the intersection width i and the checked width k), such that ρ^*_q-width$(H) \leq k$ if and only if $fhw(H) \leq k$ for all hypergraphs H of such intersection width. Example 4.8 illustrates our main challenge in the fractional setting, namely the potentially unbounded size of the support of an optimal fractional edge cover. In that example, the growth of the support is linked to the increase in the degree of v_0

whereas the intersection width remains 1, no matter how large n becomes. A combinatorial result that bounds the size of the support in terms of the optimal weight of the cover and the intersection width is therefore impossible.

Instead, we will show that every vertex set $B(\gamma)$ for some fractional cover γ can be expressed by a combination of edges and vertices, where the number of both is bounded by functions of weight(γ) and the intersection width. We can then make use of Corollary 4.36 to derive our main result.

Theorem 4.37. *For every hypergraph class C that enjoys the BIP, and for every constant $k \geq 1$, the* Check*(FHD, k) problem is tractable, i.e., given a hypergraph $H \in C$, it is feasible in polynomial time to check $fhw(H) \leq k$ and, if so, to compute an FHD of width k of H.*

Throughout this subsection we consider a hypergraph H with iwidth$(H) \leq i$ for some constant i. We will investigate fractional edge covers γ of vertices $B(\gamma) \subseteq V(H)$. We write E_h and E_ℓ to denote the *heavy* and *light-weight* edges under γ, respectively. For given $k \geq 1$, we choose $1 - \frac{1}{2k}$ as the boundary between heavy and light-weight edges. More precisely, let supp(γ) denote the support of γ; then we define E_h and E_ℓ as follows:

$$E_\ell = \{e \in \text{supp}(\gamma) \mid \gamma(e) < 1 - \tfrac{1}{2k}\},$$

$$E_h = \{e \in \text{supp}(\gamma) \mid \gamma(e) \geq 1 - \tfrac{1}{2k}\},$$

We do not require γ to be optimal but we require it to be *redundancy-free* in the following sense: if γ' is a fractional cover with $\gamma'(e) < \gamma(e)$ for some $e \in \text{supp}(\gamma)$ and $\gamma'(e) = \gamma(e)$ for all other edges $e \in E$, then $B(\gamma') \subset B(\gamma)$. The set $B(\gamma)$ has the following *split and canonical representation* in terms of heavy and light-weight edges:

Definition 4.38. Let γ be an edge-weight function of some hypergraph H with weight$(\gamma) \leq k$ for some $k \geq 1$. Then $e'_1 \cup \cdots \cup e'_\ell \cup U$ with $B(\gamma) = e'_1 \cup \cdots \cup e'_\ell \cup U$ is a *split representation* of $B(\gamma)$ if the following property (1) holds:

(1) for every $\alpha \in \{1, \ldots, \ell\}$, $e'_\alpha = B(\gamma) \cap e_\alpha$ for some $e_\alpha \in E_h$;

If, additionally, the following property (2) holds, we call γ the *canonical representation* of $B(\gamma)$:

(2) $U = \{v \in B(\gamma) \mid \forall e \in E_h \colon v \notin e\}$;

Recall that we are assuming γ to be redundancy-free. Hence, also the union $e'_1 \cup \cdots \cup e'_\ell$ is non-redundant in the sense that, for every $e'_\alpha \in \{1, \ldots, \ell\}$, we have $(e'_1 \cup \cdots \cup e'_\ell) \setminus e'_\alpha \subset (e'_1 \cup \cdots \cup e'_\ell)$. Moreover, for each α, the edge $e_\alpha \in E_h$ with $e'_\alpha = B(\gamma) \cap e_\alpha$ is in fact unique. The reason for the uniqueness is that the weight of every heavy edge is greater than 0.5. Therefore, if $e'_i = e'_j$ for some indices $i \neq j$, then the weight put by γ on the vertices in e'_i (and, hence, also in e'_j) is greater than 1. We could thus safely reduce the weight of one of the edges e_i or e_j without decreasing $B(\gamma)$, which contradicts the irredundancy of γ.

In this section, we are considering hypergraphs satisfying the BIP. Hence, we can show that the number of vertices in $B(\gamma)$ which are only covered by light-weight edges, is bounded by a constant that exclusively depends on k and i.

Lemma 4.39. *Let $k \geq 1$ and $i \geq 0$ be constants, let H be a hypergraph with* $\text{iwidth}(H) \leq i$ *and let γ be an edge-weight function of H with* $\text{weight}(\gamma) \leq k$. *Moreover, let $e'_1 \cup \cdots \cup e'_\ell \cup U$ be the canonical representation of $B(\gamma)$. Then $|U| < 2ik^3$ holds.*

Proof. By definition, U is only covered by edges from E_ℓ. Let e be an arbitrary edge in E_ℓ and let $m = |B(\gamma) \cap e|$. We first show that $m < 2ik^2$. Indeed, by definition of E_ℓ, e puts weight $< 1 - \frac{1}{2k}$ on each vertex in $B(\gamma) \cap e$. Hence, weight $> \frac{1}{2k}$ has to be put on each vertex in $B(\gamma) \cap e$ by the other edges. In total, the other edges thus have to put weight $> \frac{m}{2k}$ on the m vertices $B(\gamma) \cap e$.

By the BIP, whenever γ puts weight w on some edge e' different from e, then, in total, at most weight iw is put on the vertices in e. Hence, since we are assuming $\text{weight}(\gamma) \leq k$, the total weight of all edges in E_ℓ (even the total weight of all edges in $E(H)$) is $\leq k$. Hence, in total at most weight ik can be put on the vertices in $B(\gamma) \cap e$ by the edges different from e. We therefore have $ik > \frac{m}{2k}$ or, equivalently, $m < 2ik^2$.

Now let m be the maximum size of any edge in E_ℓ and suppose that, for an arbitrary edge $e \in E_\ell$, $\gamma(e) = w$ holds. Then, in total, e puts weight $\leq mw$ on the vertices in U. Hence, the total weight put by all edges of E_ℓ on all vertices in U is $\leq mk$. Moreover, recall that every vertex in U receives weight at least 1. Together with the above bound $2ik^2$ on the size of the edges in E_ℓ, we thus get $|U| < 2ik^3$. □

Our goal now is to show that every fractional cover γ can be replaced by a fractional cover that is, in a sense, very close to an integral cover. To formalize this closeness to an integral cover, we introduce the notion of *c-bounded fractional part*. For $\gamma : E(H) \to [0, 1]$ and $S \subseteq \text{supp}(\gamma)$, 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 4.40. 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 $R = \{e \in \text{supp}(\gamma_u) \mid \gamma_u(e) < 1\}$; then $|B(\gamma_u|_R)| \leq c$.

A naive approach towards our goal of reaching an FHD with c-bounded fractional part will be to simply take the fractional edge cover γ_u at each node u and set the weight of the heavy edges to 1. Of course, this will, in general, increase the width. However, as will be illustrated below, it will not increase the width a lot. Moreover, we will establish conditions under which the increase of the width can be neglected. We first give a formal definition of the *naive cover*:

Definition 4.41. Let γ be an edge-weight function of some hypergraph H with $\text{weight}(\gamma) \leq k$ for some $k \geq 1$ and let $e'_1 \cup \cdots \cup e'_\ell \cup U$ be a split representation of $B(\gamma)$. Then we call the edge-weight function ν a *naive cover* if the following properties hold:

(1) for every $\alpha \in \{1, \ldots, \ell\}$, let $e_\alpha \in E_h$ with $e'_\alpha = B(\gamma) \cap e_\alpha$; then we set $\nu(e_\alpha) = 1$.

(2) let $U' = U \setminus (e_1 \cup \cdots \cup e_\ell)$. ν is an optimal fractional edge cover of U', i.e., let $S = \{e \mid e \in \text{supp}(\nu) \text{ and } e \cap U' \neq \emptyset\}$; then $\text{weight}(\nu|_S) = \rho^*(U')$.

If we consider the naive cover of a canonical representation, we have $U' = U$ in the above definition. Intuitively, a naive cover ν is close to an integral cover in that it assigns weight 1 to some of the edges and the fractional part of $B(\nu)$ (i.e., the vertices in $B(\nu)$ which are outside these edges with weight 1) are covered optimally. Ultimately, we will show that we can always find a naive cover where the number vertices in the fractional part is bounded by a constant. Indeed, for the naive cover of a canonical representation, we have the bound $|U| \leq 2ik^3$ by Lemma 4.39, provided that $\text{iwidth}(H) \leq i$. However, the naive cover depends not only on γ but also on the split representation it is based on. The naive cover based on the canonical representation is not necessarily the one with the least total weight.

Recall that, since we are assuming fractional covers to be redundancy-free, for each α in the definition above, the edge $e_\alpha \in E_h$ is unique. But of course, there may be several optimal fractional edge covers of U. By our definition of "heavy edges" we immediately get the inequality $\text{weight}(\nu) - \text{weight}(\gamma) \leq 0.5$, since ν increases the weight of each heavy edge by at most $\frac{1}{2k}$ and there cannot be more than k heavy edges under γ. Below we illustrate that this gap between ν and γ might be even smaller.

Example 4.9. Recall from Example 4.8 the hypergraph $H_n = (V, E)$ with $V = \{v_0, v_1, \ldots, v_n\}$ and $E = \{e_0, \ldots, e_n\}$, where $e_0 = \{v_1, \ldots, v_n\}$ and, for $\alpha \in \{1, \ldots, n\}$, $e_\alpha = \{v_0, v_\alpha\}$. That is, H contains a big edge $e_0 = \{v_1, \ldots, v_n\}$, a single vertex v_0 outside this edge and small edges connecting each of the vertices in e_0 with the outside vertex v_0.

Now let $V^* \subseteq V$ with $v_0 \in V^*$ and $|V^* \cap e_0| = d$ for some integer $d \geq 1$. For the sake of simplicity, suppose that $V^* = \{v_1, \ldots, v_d\}$. Then an optimal fractional edge cover γ of V^* would set $\gamma(e_\alpha) = \frac{1}{d}$ for each $\alpha \in \{1, \ldots, d\}$ and $\gamma(e_0) = 1 - \frac{1}{d}$. We thus get $\text{weight}(\gamma) = 2 - \frac{1}{d}$. A naive cover of the canonical representation of γ would set $\nu(e_0) = 1$ and $\nu(e_\alpha) = 1$ for a single (arbitrarily chosen) $\alpha \in \{1, \ldots, n\}$ and $\nu(e_\beta) = 0$ for all other edges e_β. △

In the above example, we observe that $\text{weight}(\nu) - \text{weight}(\gamma) \leq \frac{1}{d}$ holds. This means that, the bigger $d = |V^* \cap e_0|$ gets, the smaller the possible improvement over a naive cover will be. Of course, the above example is very simple in that E_h consists of a single edge e_0, $B(\gamma)$ contains a single vertex v_0 outside e_0, and

the light-weight edges containing v_0 cover a single vertex in e_0. The following lemma generalizes the observation that weight(ν) − weight(γ) decreases as the contribution of the heavy edges to $B(\gamma)$ increases.

Lemma 4.42. *Let $d, k \geq 1$ and $i \geq 0$ be constants and let H be a hypergraph with* iwidth$(H) \leq i$. *Moreover, let γ be an edge-weight function of H with a canonical representation $B(\gamma) = e'_1 \cup \cdots \cup e'_\ell \cup U$, s.t. $\ell \leq k$ and for every $\alpha \in \{1, \ldots, \ell\}$, the following properties hold:*

(1) $e'_\alpha = B(\gamma) \cap e_\alpha$ for some $e_\alpha \in E_h$;

(2) $|e'_\alpha| \geq d + ki$.

Then weight(ν) − weight$(\gamma) < \frac{ik^2}{d}$ *holds, where ν denotes a naive cover corresponding to the canonical representation $B(\gamma) = e'_1 \cup \cdots \cup e'_\ell \cup U$.*

Proof. We first partition supp(γ) and supp(ν) into $R \cup S$ and $R' \cup S$, respectively, with $S = \{e_\alpha \in E(H) \mid 1 \leq \alpha \leq \ell$ and $e'_\alpha = B(\gamma) \cap e_\alpha\}$ and $R =$ supp$(\gamma) \setminus S$ and $R' =$ supp$(\nu) \setminus S$.

Clearly, weight$(\gamma) =$ weight$(\gamma|_R)+$weight$(\gamma|_S)$ and weight$(\nu) =$ weight$(\nu|_{R'})+$ weight$(\nu|_S) =$ weight$(\nu|_{R'}) + \ell$ hold. Moreover, since a naive cover ν is an optimal fractional cover on U, we have weight$(\nu|_{R'}) = \rho^*(U) \leq$ weight$(\gamma|_R)$. Hence, in order to prove the lemma, it suffices to show that $\ell -$weight$(\gamma|_S) \leq \frac{ik^2}{d}$.

Consider e'_α for some $\alpha \in \{1, \ldots, \ell\}$. By iwidth$(H) \leq i$, we have $|e'_\alpha \cap e'_\beta| \leq i$ for each of the $\ell - 1$ β's with $\beta \neq \alpha$. Hence, less than ki vertices in e'_α are also contained in one of the other heavy edges e_β. Now let $e''_\alpha = \{v \in e'_\alpha \mid v \notin e'_\beta$ for every $\beta \neq \alpha\}$. Then $|e''_\alpha| > d$ holds by the assumption $|e'_\alpha| > d + ki$. Since $e''_\alpha \subseteq B(\gamma)$, the cover γ must put weight ≥ 1 on each of the vertices in e''_α. The edges e_β with $\beta \neq \alpha$ do not put any weight on the vertices in e''_α. It remains to consider the edges in R: whenever an edge in R has weight w in γ, then it can put at most weight wi in total on the vertices in e''_α. Since weight$(\gamma|_R) \leq$ weight$(\gamma) \leq k$, all of the edges in R taken together can only put $\leq ki$ weight in total on the vertices in e''_α. By $|e''_\alpha| > d$, there exists at least one vertex in e''_α that receives weight $< \frac{ki}{d}$ by $\gamma|_R$. Hence, since all vertices of e''_α are

contained in $B(\gamma)$, $\gamma(e_\alpha) > 1 - \frac{ki}{d}$ must hold. We therefore get the inequality $\text{weight}(\gamma|_S) > \ell \cdot (1 - \frac{ki}{d})$ and, thus, also $\ell - \text{weight}(\gamma|_S) < \ell \cdot \frac{ki}{d} \leq k \cdot \frac{ki}{d} = \frac{k^2 i}{d}$. □

As in Example 4.9, we again observe that $\text{weight}(\nu) - \text{weight}(\gamma)$ decreases as the contribution of the heavy edges to $B(\gamma)$ increases. Nevertheless, no matter how big the heavy edges get, this gap may remain greater than 0. However, for our purposes, a slightly weaker condition than $\text{weight}(\nu) = \text{weight}(\gamma)$ suffices, namely: for a sufficiently big lower bound on the size of the heavy edges, if $\text{weight}(\gamma) \leq k$, then also $\text{weight}(\nu) \leq k$ holds. Towards this goal, we will show next that, for relevant values of $\text{weight}(\nu)$, the difference $\text{weight}(\nu) - k$ is bounded from below by some constant which only depends on i and k. By relevant we mean that $\text{weight}(\nu)$ is in the interval $(k, k + 0.5]$. The reason for the irrelevance of the values outside this interval is that, if $\text{weight}(\nu) \leq k$, then we may simply replace γ by ν without further ado. And if $\text{weight}(\nu) > k + 0.5$, then, by $\text{weight}(\nu) - \text{weight}(\gamma) \leq 0.5$, also $\text{weight}(\gamma) > k$ would hold, which contradicts the assumption that $\text{weight}(\gamma) \leq k$ holds.

The following definitions are crucial:

Definition 4.43. For constants $c, k \geq 1$, we define:

$$\begin{aligned} \mathcal{H}(c) &= \{H = (V, E) \mid |V| \leq c\} \\ M(k, c) &= \{\rho^*(H) + j - k \mid H \in \mathcal{H}(c),\ j \in \mathbb{N},\ \text{and } \rho^*(H) + j - k \in (0, 0.5]\} \\ \mu(k, c) &= \min(M(k, c)),\ \text{if } M(k, c) \neq \emptyset \text{ and undefined otherwise.} \end{aligned}$$

Consider a fractional edge cover γ with $B(\gamma) = e'_1 \cup \cdots \cup e'_\ell \cup U$, s.t. $|U| \leq c$. Moreover, we assume that $\text{weight}(\gamma) \leq k$ holds. If $M(k, c) = \emptyset$, then $\text{weight}(\nu) \leq k$ and we are done. Note that this case arises, for instance, in Example 4.9 for $k = 2$. There we have $\ell = 1$ and $|U| = c = 1$. Hence, the fractional cover number of the induced subhypergraph $H[U]$ is 1 and the naive cover which sets $\nu(e_0) = 1$ and $\nu(e_\alpha) = 1$ for a single $\alpha \in \{1, \ldots, n\}$ clearly satisfies $\text{weight}(\nu) \leq k$.

Based on Lemmas 4.39 and 4.42, we can now prove the central combinatorial result:

Theorem 4.44. *Let H be a hypergraph with* $\text{iwidth}(H) \leq i$ *and let* $\gamma : E(H) \to [0, 1]$ *with* $\text{weight}(\gamma) \leq k$. *Then there exist a constant $c = f(k, i)$ for some*

function f and an edge-weight function $\nu : E(H) \to [0,1]$ with $\text{weight}(\nu) \leq k$ *and $B(\gamma) \subseteq B(\nu)$ such that ν has c-bounded fractional part.*

Proof. Let a canonical representation of γ be of the form $B(\gamma) = e'_1 \cup \cdots \cup e'_\ell \cup U$ with $e'_\alpha = B(\gamma) \cap e_\alpha$ such that $e_\alpha \in E_h$ for every $\alpha \in \{1, \ldots, \ell\}$. By Lemma 4.39, we have $|U| < c_0 = 2ik^3$. By our definition of heavy edges, we know that $\text{weight}(\nu) \leq \text{weight}(\gamma) + 0.5$ holds for any naive cover ν. Hence, together with the assumption $\text{weight}(\gamma) \leq k$, we conclude that $\text{weight}(\nu) \leq k + 0.5$ must hold. We now distinguish the following cases:

Case 1. Suppose that $U = \emptyset$. Let $B(\gamma) = e'_1 \cup \cdots \cup e'_\ell$. If $\gamma(e_\alpha) = 1$ for every $\alpha \in \{1, \ldots, \ell\}$, then we are done. Otherwise, let $\epsilon = \max(\{1 - \gamma(e_\alpha) \mid 1 \leq \alpha \leq \ell\}$. We consider two subcases:

Case 1.1. Suppose that $|e'_\alpha| > 2ki$ for every $\alpha \in \{1, \ldots, \ell\}$. As in the proof of Lemma 4.42, we define $e''_\alpha = \{v \in e'_\alpha \mid v \notin e'_\beta \text{ for every } \beta \neq \alpha\}$. By $\text{iwidth}(H) \leq i$ and $\text{weight}(\gamma) \leq k$, we have $|e'_\alpha \setminus e''_\alpha| \leq ki$. Hence, by $|e'_\alpha| > 2ki$, we conclude that $|e''_\alpha| > ki$ for every α. By the definition of ϵ, there exists α with $\epsilon = 1 - \gamma(e_\alpha)$. Hence, by $e''_\alpha \subseteq B(\gamma)$, the edges outside E_h must put weight at least ϵ on each of the vertices in e''_α. By $|e''_\alpha| > ki$, the edges outside E_h must put total weight $> ki\epsilon$ on all of the vertices in e''_α. By $\text{iwidth}(H) \leq i$, this requires that $\text{weight}(\gamma|_R) > k\epsilon$ for $R = \text{supp}(\gamma) \setminus E_h$ must hold. Hence, $\text{weight}(\gamma) > \text{weight}(\gamma|_{E_h}) + k\epsilon$. On the other hand, the naive cover ν increases (compared with γ) the weight of each edge $e_1, \ldots, e_\ell$ by at most ϵ. That is, $\text{weight}(\nu) \leq \text{weight}(\gamma|_{E_h}) + k\epsilon$. Hence, $\text{weight}(\nu) \leq \text{weight}(\gamma)$ and we may replace γ by ν.

Case 1.2. Suppose that there exists $\alpha \in \{1, \ldots, \ell\}$ with $|e'_\alpha| \leq 2ki$. W.l.o.g., assume that $\alpha = \ell$. Then we may represent $B(\gamma)$ as $B(\gamma) = e'_1 \cup \cdots \cup e'_{\ell-1} \cup U_1$ with $U_1 = e'_\ell$. Clearly $|U_1| \leq 2ki \leq c_0$ and we move on to Case 2 or 3 below.

Case 2. Suppose that $U \neq \emptyset$ and $M(k, c_0) = \emptyset$. The latter condition implies that $\rho^*(G) + j - k \notin (0, 0.5]$ for every hypergraph $G \in \mathcal{H}(c_0)$ and every natural number j. In particular, the induced subhypergraph $H[U]$ is in $\mathcal{H}(c_0)$. Moreover, choose $j = \ell$. Then also $\rho^*(H[U]) + \ell - k \notin (0, 0.5]$, i.e., $\text{weight}(\nu) - k \notin (0, 0.5]$. On the other hand, as argued above, $\text{weight}(\nu) \leq k + 0.5$ or,

equivalently, $\text{weight}(\nu) - k \leq 0.5$ The only possibility to satisfy both conditions is that $\text{weight}(\nu) \leq k$ holds. We may therefore again replace γ by a naive cover ν to get $B(\gamma) \subseteq B(\nu)$ and $\text{weight}(\nu) \leq k$.

Case 3. Suppose that $U \neq \emptyset$ and $M(k, c_0) \neq \emptyset$. Then we define the following value d_0 to distinguish between small and big heavy edges:

$$d_0 = \frac{ik^2}{\mu(k, c_0)}$$

We distinguish two subcases:

Case 3.1. If $|e'_\alpha| > d_0 + ki$ for every $\alpha \in \{1, \ldots, \ell\}$, then, by Lemma 4.42, we have

$$\text{weight}(\nu) - \text{weight}(\gamma) < \frac{ik^2}{d_0} = \frac{ik^2}{ik^2/\mu(k, c_0)} = \mu(k, c_0).$$

Hence, together with the assumption $\text{weight}(\gamma) \leq k$, we have $\text{weight}(\nu) - k < \mu(k, c_0)$. Moreover, as argued above, we may assume $\text{weight}(\nu) \leq k + 0.5$ or, equivalently, $\text{weight}(\nu) - k \leq 0.5$.

Note that $\text{weight}(\nu) = \ell + \rho^*(H[U])$ holds for the subhypergraph $H[U]$ of H induced by U. By $|U| \leq c_0$, we thus have $H[U] \in \mathcal{H}(c_0)$. Hence, by $\text{weight}(\nu) - k \leq 0.5$, either $\text{weight}(\nu) - k \leq 0$ or $\text{weight}(\nu) - k \in M(k, c_0)$ holds. The latter case can be ruled out because $\mu(k, c_0) = \min(M(k, c_0))$ and $\text{weight}(\nu) - k < \mu(k, c_0)$. Hence, we conclude that $\text{weight}(\nu) - k \leq 0$ or, equivalently, $\text{weight}(\nu) \leq k$ holds. We may therefore again replace γ by a naive cover ν to get $B(\gamma) \subseteq B(\nu)$ and $\text{weight}(\nu) \leq k$.

Case 3.2. Now suppose that there exists $\alpha \in \{1, \ldots, \ell\}$ with $|e'_\alpha| \leq d_0 + ki$. W.l.o.g., suppose that $\alpha = \ell$. Then we may in fact represent $B(\gamma)$ as $B(\gamma) = e'_1 \cup \cdots \cup e'_{\ell-1} \cup U_1$ with $U_1 = U \cup e''_\ell$ and $e''_\ell = \{v \in e'_\ell \mid v \notin e'_\beta \text{ for every } \beta \neq \ell\}$. Then, in particular, $|U_1| \leq c_1 = c_0 + d_0 + ki$ holds, i.e., the size of the fractional part U_1 is bounded by a constant that depends only on k and i. Note that then still Case 3 applies since $U_1 \supseteq U \neq \emptyset$ and $M(k, c_1) \supseteq M(k, c_0) \neq \emptyset$ clearly hold. Hence, after at most ℓ iterations of Case 3, eventually Case 3.1 applies and we may replace γ by a naive cover, such that the size of the fractional part is bounded by a constant that depends only on k and i. □

A cover that has c-bounded fractional part also has bounded support as the c fractionally covered vertices require at most one edge each as support. We thus immediately obtain the following corollary which may be of independent interest.

Corollary 4.45. *Let H be a hypergraph with* $\text{iwidth}(H) \leq i$ *and let* $\gamma : E(H) \to [0,1]$ *with* $\text{weight}(\gamma) \leq k$. *Then there exists an edge weight function* $\nu : E(H) \to [0,1]$ *such that* $\text{weight}(\nu) \leq k$, $B(\gamma) \subseteq B(\nu)$ *and ν has bounded support (depending on i and k).*

The theorem gives us the desired representation of the bags in an FHD of a hypergraph of intersection width bounded by some constant i. We are now ready to prove the main theorem of this subsection, namely the tractability of the Check(FHD, k) problem for hypergraph classes with bounded intersection.

Proof of Theorem 4.37. We show that Check(FHD, k) is tractable for a class C of hypergraphs with intersection width at most i. Let $H \in C$ and let H^1 be the hypergraph obtained by adding all edges of size 1 to H. Adding these edges has no effect on any fractional edge covers and, in particular, $fhw(H) = fhw(H^1)$.

Let c be the constant from Theorem 4.44 for our k and i. By Corollary 4.44, if there exists an FHD of H^1 with weight at most k, then there exists an FHD $\mathcal{F}$ where for every node u, γ_u has c-bounded fractional part. We can thus construct γ_u as a combination of c unary edges and at most k edges with weight 1. Hence, every γ_u in $\mathcal{F}$ we have $\text{supp}(\gamma_u) \leq k + c$.

Putting it all together we thus see that if $fhw(H^1) \leq k$, then also $(k + c)$-limited $fhw(H^1) \leq k$. The implication in the other direction is trivial. Then, by Corollary 4.36 we see that there is a polynomial time algorithm that decides $fhw(H^1) \leq k$. □

4.5 Multi-Intersection and Fractional Hypertree Width

In the previous section we demonstrate that Check(fhw, k) is tractable under the bounded intersection property. It was already shown by Fischl, Gottlob, and

Pichler [54], that Check(fhw, k) is tractable for classes that enjoy the bounded degree property. The natural question then is to ask if tractability also holds for classes with the bounded multi-intersection property since it both the BIP and the BDP.

However, bounded multi-intersection is a vastly more general property than both the BIP and the BDP, most evident by the fact that it generalizes these two seemingly unrelated properties. For our tractability proof we essentially follow the same proof strategy as we did in Section 4.5 for bounded intersection. This time the notion of c-bounded fractional part is however not sufficient and we focus directly on the existence of a bounded support cover that satisfies the same weight bound. The central combinatorial result for the eventual tractability proof is therefore the following.

Theorem 4.46 ([65])**.** *There is a function $h(c, d, k)$ such that the following is true. Let c, d, k be constants. Let H be a hypergraph with c-$\mathrm{miwidth}(H) \leq d$ and let $\gamma\colon E(H) \to [0, 1]$ Assume that $\mathrm{weight}(\gamma) \leq k$. Then there exists an edge weight function $\nu\colon E(H) \to [0, 1]$ such that*

- *$weight(\nu) \leq k$,*
- *$B(\gamma) \subseteq B(\nu)$,*
- *and $|\mathrm{supp}(\nu)| \leq h(c, d, k)$.*

Applied to FHDs this implies that – for hypergraph classes that enjoy the BMIP – there exists a FHD of width at most k if and only if there exists a FHD of width k where the support of every cover is bounded by some constant q. Thus, we again see that it is sufficient to check q-limited fhw for some constant q and tractability follows immediately by Theorem 4.36. Hence, we can state the following result, unifying the tractability for the BDP and BIP cases.

Theorem 4.47. *For every hypergraph class C that enjoys the BMIP, and for every constant $k \geq 1$, the Check(FHD, k) problem is tractable.*

As already stated at the beginning of this chapter, we choose to only report on our results for BMIP here. The full proof is highly technical and the main

intuitions are difficult to grasp when considering the proof directly. For this reason we instead presented the proof for the BIP case in the previous section, even though Theorem 4.47 generalizes the results there. Most of the important ideas for the proof of Theorem 4.46 appear there already, but in a more intuitive form. We consider the approachability of the proof of particular importance here since our proof strategy is (to our knowledge) novel and may have further applications in fractional graph theory.

In the following we repeat our outline (from [65]) of the full proof strategy for Theorem 4.46. For full technical details we refer to the full paper [65].

The first step of our reasoning is to consider the situation where $|B(\gamma)|$ is bounded. In this case it is easy to transform γ into the desired ν. Partition all the hyperedges of H into equivalence classes corresponding to non-empty subsets of $B(\gamma)$ such that two edges e_1 and e_2 are equivalent if and only if $e_1 \cap B(\gamma) = e_2 \cap B(\gamma)$. Then let s_X be the total weight (under γ) of all the edges from the equivalence class where $e \cap B(\gamma) = X$. Identify one representative of each (non-empty) equivalence class and let e_X be the representative of the equivalence class corresponding to X. Then define ν as follows. For each X corresponding to a non-empty equivalence class, set $\nu(e_X) = s_X$. For each edge e whose weight has not been assigned in this way, set $\nu(e) = 0$. It is clear that $B(\gamma) \subseteq B(\nu)$ and that the support of ν is at most $2^{|B(\gamma)|}$, which is bounded by assumption.

Of course, in general we cannot assume that $|B(\gamma)|$ is bounded. Therefore, as the next step of our reasoning, we consider a more general situation where we have a bounded set $\mathbf{S} = \{S_1, \ldots, S_r\}$ where each S_i is a set of at most c hyperedges such that the following conditions hold regarding $\mathbf{S}$: (i) for each $1 \leq i \leq r$, $\gamma(S_i) \geq 1$ and (ii) the set $U = B(\gamma) \setminus \bigcup_{i \in [r]} \bigcap S_i$ is of bounded size. Then the edge weight function ν as in Theorem 4.46 can be defined as follows. For each $e \in \bigcup \mathbf{S}$, set $\nu(e) = \gamma(e)$. Next, we observe that for the subhypergraph $H' = H - \bigcup \mathbf{S}$, $|B_{H'}(\gamma)|$ is bounded, where subscript H' means that we consider B for hypergraph H' and γ is restricted accordingly. Therefore, we define ν on the remaining edges as in the paragraph above. It is not hard to see that the support of the resulting ν is of size at most $c \cdot r + 2^{|U|}$. We then ultimately show that such a family of sets of edges can always be found for hypergraphs with

c- miwidth$(H) \leq d$ (after a possible modification of γ).

4.5.1 Multi-Intersection and the Dual Hypergraph

In the results of the previous sections, bounded multi-intersection has played an important role. Here we discuss some further novel observations about multi-intersection width. Notably, bounded multi-intersection width is its own dual property. This allows us to extend our results for fractional edge covers to vertex covers. The resulting theorem generalizes, in a particular sense, a well-known statement of Füredi [55].

Note that we assume reduced hypergraphs throughout this section to simplify the presentation. It is easy to check that this assumption can be made without loss of generality when studying fractional covers: if two vertices have the same edge type then they receive equivalent weight by an edge cover. Alternatively, their weight can be collapsed onto a single vertex with equivalent effect when we consider vertex covers. When considering covers one generally excludes empty edges and isolated vertices, otherwise we get into trivial situations where no covers exist. For example, the hypergraph with only a single isolated vertex has no edge cover. However, as discussed in Chapter 2 we only consider hypergraphs with no isolated vertices and no empty edges in this thesis.

We start by giving a formal definition of the fractional vertex cover problem. Let H be a hypergraph and $\beta : V(H) \to [0, 1]$ be an assignment of weights to the vertices of H. Analogous to the definition of fractional edge covers we define

- $B_v(\beta) = \{e \in E(H) \mid \sum_{v \in e} \beta(v) \geq 1\}$,
- vsupport$(\beta) = \{v \in V(H) \mid \beta(v) > 0\}$,
- and weight$(\beta) = \sum_{v \in V(H)} \beta(v)$.

A fractional vertex cover is also called a transversal in some contexts (cf. [107]). For a set of edges E' we denote the weight of the minimal fractional vertex cover β such that $E' \subseteq B_v(\beta)$ as $\tau^*(E')$. For hypergraph H, we say $\tau^*(H) = \tau^*(E(H))$.

Recall that we assume reduced hypergraphs and therefore there is a one-to-one correspondence of vertices in H and edges in H^d. We will make use of the following well-known fact about the connection of what we will call *dual weight assignments*.

Proposition 4.48. *Let H be a (reduced) hypergraph and H^d its dual. We write f_v to identify the edge in $E(H^d)$ that corresponds to the vertex v in $V(H)$. The following two statements hold:*

- *For every $\gamma : E(H) \to [0,1]$ and the function $\beta : V(H^d) \to [0,1]$ with $\beta(e) = \gamma(e)$ it holds that $B_v(\beta) = \{f_v \mid v \in B(\gamma)\}$.*
- *For every $\beta : V(H) \to [0,1]$ and the function $\gamma : E(H^d) \to [0,1]$ with $\gamma(f_v) = \beta(v)$ it holds that $B(\gamma) = \{v \mid f_v \in B_v(\beta)\}$.*

In the following we extend Theorem 4.46 to an analogous statement for fractional vertex covers thereby generalizing the previous proposition significantly. To derive the result we need a final observation about hypergraphs and their multi-intersection width. In a sense, we show that bounded multi-intersection is its own dual property.

Lemma 4.49. *Let H be a hypergraph with c-*miwidth$(H) \leq d$*. Then the dual hypergraph H^d has d+1-*miwidth$(H) \leq c$.[2]

Proof. Let $v_1, v_2, \ldots, v_{d+1}$ be $d+1$ distinct arbitrary vertices of a hypergraph H with c-miwidth$(H) \leq d$. We write $I(v) = \{e \in E \mid v \in e\}$ for the set of edges incident to a vertex v. Since H has c-multi-intersection width at most d, it must hold that $X = \bigcap_{j \in [d+1]} I(v_j)$ has no more than c elements. Otherwise, there would be at least $c+1$ edges in X that share $d+1$ vertices, i.e., a contradiction to the assumed c-multi-intersection width of H.

Now, consider the edges $f_1, f_2 \ldots, f_{d+1}$ in H^d that correspond to the vertices $v_1, v_2, \ldots, v_{d+1}$ in H. It follows from the definition of the dual hypergraph that $|\bigcap_{j \in [d+1]} f_j| = |X|$ since any two edges in H^d share exactly one vertex for each

[2]Note that the superscript of H^d only signifies that it is the dual of H. It is not connected to the integer constant d used for the multi-intersection size of H.

edge in H that they are both incident to. We know from above that $|X| \leq c$. As this applies to any choice of vertices in H, and thus also to any choice of $d+1$ edges in H^d, we see that any intersection of $d+1$ edges in H^d has cardinality less or equal c. □

Theorem 4.50. *There is a function $h(c,d,k)$ such that the following is true. Let c,d,k be constants. Let H be a hypergraph with c-miwidth$(H) \leq d$ and let β be an assignment of weights to $V(H)$. Assume that weight$(\beta) \leq k$. Then there is an assignment ν of weights to $V(H)$ such that weight$(\nu) \leq k$, $B_\nu(\beta) \subseteq B_\nu(\nu)$ and $|\text{vsupport}(\nu)| \leq h(c,d,k)$.*

Proof. Let γ be the dual weight assignment of β as in Proposition 4.48. That is, $\gamma : F \to [0,1]$ is an edge weight assignment in the dual hypergraph $H^d = (W,F)$ with $|\text{supp}(\gamma)| = |\text{vsupport}(\beta)|$ and weight(γ) = weight(β).

From Lemma 4.49 we have that H^d has d+1-miwidth$(H) \leq c$ and thus by Theorem 4.46 there is an edge weight function ν' with $B(\gamma) \subseteq B(\nu')$ and $|\text{supp}(\nu')| \leq h'(d+1,c,k)$. Let ν now be the dual weight assignment of ν'. By Proposition 4.48 we then see that also $B_\nu(\beta) \subseteq B_\nu(\nu)$ and $|\text{vsupport}(\nu)| = |\text{supp}(\nu')| \leq h'(d+1,c,k)$. □

To conclude this section we wish to highlight the connection of Theorem 4.50 to a classical result by Füredi [55] on fractional edge covers.

Proposition 4.51 ([55], page 152, Proposition 5.11.(iii))**.** *For every hypergraph H of rank r, and every fractional vertex cover w for H satisfying* weight$(w) = \tau^*(H)$*, the property $|\text{vsupport}(w)| \leq r \cdot \tau^*(H)$ holds.*

Observe that a hypergraph H with rank r is can also be considered to have 1-miwidth$(H) = r$. Hence, the above proposition means that, for a hypergraph H where it holds that 1-miwidth$(H) = r$, there is a fractional vertex cover of optimal weight whose support is bounded by a function of the weight and r. Theorem 4.50 generalizes Proposition 4.51 in two aspects. First, Theorem 4.50 considers hypergraphs with c-miwidth$(H) \leq d$ for $c \geq 1$ and second, it applies to assignments of weights to vertices in general not just to those that establish an optimal fractional vertex cover. An important aspect of Proposition 4.51 not

reflected in Theorem 4.50 is a concrete upper bound on the size of the support. Optimizing the upper bound following from Theorem 4.46 is left for future research.

4.6 Summary

In this chapter we proposed candidate tree decompositions and their use in the study of width checking. While the problem of finding candidate tree decompositions is NP-hard in general, the mild restriction to component normal form TDs is sufficient to make the problem tractable. We argue that candidate TDs greatly simplify the search for tractable fragments for width checking problems by separating algorithmic and combinatorial considerations. All further algorithmic results in this chapter are built on this novel framework.

The first of which is a slight improvement of results for ghw checking by Fischl, Gottlob, and Pichler [54]. Expanding on their ideas we show that, for hypergraph classes with bounded multi-intersection width, the Check(ghw, k) problem lies in LogCFL and is therefore highly parallelizable. We then show how, in the confines of the BMIP, we can use this result as a basis for the tractability of arbitrary (conservative) f-widths by introducing q-limited width and giving a reduction of q-limited f-width checking to Check(ghw, k).

We then solve a major open problem posed in [54] by proving that Check(fhw, k) is also tractable for classes that enjoy the BMIP. To this end, we show that there always exists a constant q such that $fhw(H) \leq k$ if and only if q-limited $fhw(H) \leq k$, which allows us to utilize our reduction to Check(ghw, k). In the process of proving the existence of such a q we make some novel combinatorial arguments that may be of independent interest, in particular in fractional graph theory.

CHAPTER 5

A Novel Generalization of β-Acylicity

Hypergraph cyclicity has been identified as a key factor for the computational complexity of multiple fundamental database and reasoning problems. While various natural notions of hypergraph acyclicity exist, the two most general ones — α- and β-acyclicity — have proven to be the most relevant in the study of the complexity of reasoning. It was already discussed in the introduction that problems that are NP-hard in general often become tractable when restricted to acyclic instances. In particular, the restriction to β-acyclic instances yields tractable classes for a variety of fundamental problems, including SAT [97], #SAT [25], and $\mathrm{CQ}^\neg$ evaluation [23, 95]. Notably, these problems remain NP-hard when restricted to α-acyclic instances or #P-hard in the case of #SAT (see [28]).

Despite the unquestionable success of the generalization of α-acyclicity, the generalization of β-acyclicity has received little attention so far. In the most prominent approach, Gottlob and Pichler [71] introduced β-hypertree width

(β-hw) as an analogue to hypertree width. In particular, they define β-hw as the maximum hypertree width over all subhypergraphs, mirroring a characterization of β-acyclicity in terms of every subhypergraph being α-acyclic. However, it is difficult to exploit low β-hw algorithmically. An inherent problem with β-hw is that a witness for low β-hw would need to include a hypertree decomposition for each of the, exponentially many, subhypergraphs. Furthermore, none of the problems listed above as tractable on β-acyclic instances are known to be tractable for bounded β-hw (beyond those that are tractable for the more general bounded hw).

In recent work, Carbonell, Romero, and Zivny introduced the notion of point-decompositions and the accompanying point-width (pw) [29], which generalizes both β-acyclicity and MIM-width[104]. They show that, given a point-decomposition of bounded point-width and polynomial size, Max-CSP can be decided in polynomial time. However, just as with β-hw, it is not known if $pw \leq k$ can be decided in polynomial time, even for constant k. We see, there are no known generalizations of β-acyclicity that are suitable for generalizing tractability results beyond β-acyclicity. In light of the importance of SAT and $\mathrm{CQ}^\neg$ this is a highly unsatisfactory situation.

In this chapter, we propose a new generalization of β-acyclicity which we call nest-set width (nsw). In contrast to β-hw and pw, it is not based on decompositions but instead generalizes a characterization of β-acyclicity by the existence of certain kinds of elimination orders. Nest-set width has several attractive properties that suggest it to be a natural extension of β-acyclicity. Importantly, $nsw \leq k$ can be decided in fixed-parameter tractable time when parameterized by k. Furthermore, we show that bounded nsw yields new islands of tractability for SAT and $\mathrm{CQ}^\neg$ evaluation.

The chapter is structured as follows. We define nest-set width and establish some basic properties in Section 5.1. We move on to establish the relationship between nsw and other width measures, most importantly β-hw, in Section 5.2. The complexity of checking nsw is discussed in Section 5.3. The tractability of $\mathrm{CQ}^\neg$ under bounded nsw is shown in Section 5.4. Finally, we discuss the effect of bounded nest-set width of CNF formulas on the well-known Davis-Putnam resolution procedure in Section 5.5. Concluding remarks for the chapter are

given in Section 5.6.

The contents of this chapter are an extension of a paper by the author of this thesis [86].

5.1 Nest-Set Width

In this section we introduce nest-set width and establish some of its basic properties. The crucial difference between β-*hw* and *nsw* is that the generalization is based on a different characterization of β-acyclicity. While β-*hw* generalizes the condition of every subgraph having a join tree, nest-set width instead builds on the characterization via nest point elimination from Proposition 2.1. We start by generalizing nest points to *nest-sets*:

Linear orders will play an important role throughout this chapter. Recall, a binary relation R is a *linear order* if it is antisymmetric, transitive and connex (either aRb or bRa holds for all a and b). We will be particularly interested in whether the subset relation $\subseteq$ is a linear order on some domain. If $\subseteq$ is a linear order for some set X, we say X is *linearly ordered* by $\subseteq$. Note that $\subseteq$ is inherently transitive and antisymmetric and we can limit our arguments to connexity.

Definition 5.1 (Nest-Set)**.** Let H be a hypergraph. A non-empty set $s \subseteq V(H)$ of vertices is called a *nest-set* in H if the set

$$I^*(s, H) := \{e \setminus s \mid e \in I(s, H)\}$$

is linearly ordered by $\subseteq$.

As the comparability by $\subseteq$ of sets minus a nest-set will appear frequently, we introduce explicit notation for it. Let H be a hypergraph and $s \subseteq V(H)$. For two sets of vertices $V, U \subseteq V(H)$, we write $V \subseteq_s U$ for $V \setminus s \subseteq U \setminus s$. We could thus alternatively define nest-sets as those sets s for which $I(s, H)$ is linearly ordered by $\subseteq_s$.

In later sections, the maximal elements with respect to $\subseteq_s$ will play an important role. For a nest-set s we will refer to a maximum edge in $I(s)$ w.r.t. $\subseteq_s$ as a *guard*

of s. Note that there may be multiple guards. However, in all of the following usage it will make no difference which guard is used and we will implicitly always use the lexicographically first one (and thus refer to *the* guard).

Like for nest points, we want to investigate how a hypergraph can be reduced to the empty hypergraph by successive removal of nest sets. We formalize this notion in the form of *nest-set elimination orderings*.

Definition 5.2 (Nest-Set Elimination Ordering)**.** Let H be a hypergraph and let $O = (s_1, \ldots, s_q)$ be a sequence of sets of vertices. Define $H_0 = H$ and $H_i := H_{i-1} - s_i$. We call O a *nest-set elimination ordering* (NEO) if, for each $i \in [q]$, s_i is a nest-set of H_{i-1} and H_q is the empty hypergraph.

Note that an elimination ordering is made up of at most $|V(H)|$ nest-sets. We are particularly interested in how large the nest-sets have to be for a NEO to exist. Hence, we introduce notation for restricted-size nest-sets and NEOs:

- If s is a nest-set of H with at most k elements then we call s a *k-nest-set*.
- A nest-set elimination ordering that consists of only k-nest-sets is a *k-nest-set elimination ordering* (k-NEO).
- Finally, the *nest-set width* $nsw(H)$ of a hypergraph H is the lowest k for which there exists a k-NEO.

It is easy to see that a hypergraph has a 1-nest-set $\{v\}$ if and only if v is a nest point. Therefore, a 1-NEO corresponds directly to a sequence of nest point deletions that eventually result in the empty hypergraph. As this is exactly the characterization of β-acyclicity from Proposition 2.1, we see that nsw generalizes β-acyclicity.

Corollary 5.3. *A hypergraph H has $nsw(H) = 1$ if and only if H is β-acyclic.*

Example 5.1. Let H_0 be the hypergraph with edges $\{a, b, c, d\}$, $\{a, d, e\}$, $\{c, d, f\}$, $\{b, e\}$, and $\{c, f\}$. Figure 5.1 illustrates the step-wise elimination of H_0 according to the 2-NEO $(\{c, f\}, \{b, e\}, \{a, d\})$.

For the first nest-set $s_1 = \{c, f\}$ we see that

$$I(s_1, H_0) = \{\{a, b, c, d\}, \{c, d, f\}, \{c, f\}\}$$

and $I^*(s_1, H_0) = \{\{a, b, d\}, \{d\}, \emptyset\}$. To verify that s_1 is a nest-set of H_0 we observe that $\{a, b, d\} \supseteq \{d\} \supseteq \emptyset$. Note that $\{f\}$ is also a nest-set of H_0 whereas $\{c\}$ is not since $\{a, b, d\}$ and $\{d, f\}$ are both in $I^*(\{c\}, H_0)$ and clearly neither $\{a, b, d\} \subseteq \{d, f\}$ nor $\{a, b, d\} \supseteq \{d, f\}$ holds.

In the second step of the elimination process we then consider $H_1 = H_0 - \{c, f\}$ and the nest-set $s_2 = \{e, b\}$. It is again straightforward to verify that $I^*(s_2, H_1) = \{\{a, d\}, \emptyset\}$ is linearly ordered by $\subseteq$. This is in fact the only nest-set of H_1. The third nest-set in the NEO, $s_3 = \{a, d\}$ only becomes a nest-set after elimination of s_2: observe that $I^*(s_3, H_1) = \{\{e\}, \{b\}, \emptyset\}$ which is not linearly ordered by $\subseteq$.

In the final step, $H_2 = H_1 - \{e, b\}$ only has two vertices left. The set of all vertices of a hypergraph is trivially a nest-set since $I^*(V(H), H)$ is always $\{\emptyset\}$. Thus, the set $V(H_2) = \{a, d\}$ is a nest-set of H_2. The hypergraph H_0 has no 1-NEO (it has a β-cycle) and therefore $nsw(H_0) = 2$. △

An important difference between α- and β-acyclicity is that only the latter is *hereditary*, i.e., if hypergraph H is β-acyclic then so is every subhypergraph of H. Nest-set width, just like β-acyclicity and β-hypertree width, is indeed also a hereditary property. In the following two simple but important lemmas, we first establish that NEOs remain valid when vertices are removed from the hypergraph (and the NEO) and then show that this also applies to removing edges.

Note that the construction in the following lemma, and Lemma 5.5 below, can technically create empty sets in the resulting NEOs. Formally speaking this is

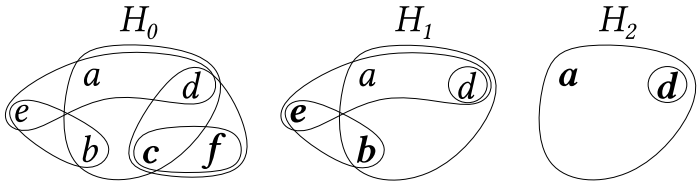


Figure 5.1: The nest-set elimination from Example 5.1

not allowed (recall that nest-sets are non-empty). Whenever this occurs the implicit meaning is that all the empty sets are removed from the NEO.

Lemma 5.4. *Let H be a hypergraph with k-NEO $O = (s_1, s_2, \ldots, s_\ell)$ and let $r \subseteq V(H)$. Then the sequence $O' = (s_1 \setminus r, s_2 \setminus r, \ldots, s_\ell \setminus r)$ is a k-NEO of $H - r$.*

Proof. We first show that for any nest-set s let $r \subseteq V(H)$ we have that $s \setminus r$ is either the empty set or a nest-set of $H - r$.

Suppose $s \setminus r$ is not empty and not a nest-set of $H - r$, then there are $e_1, e_2 \in I(s \setminus r, H - r)$ that are not comparable by $\subseteq_{s \setminus r}$. It is easy to see that there exist $e_1', e_2' \in I(s, H)$ such that $e_1 = e_1' \setminus r$ and $e_2 = e_2' \setminus r$. Since s is a nest-set in H, w.l.o.g., $e_1' \setminus s \subseteq e_2' \setminus s$ and therefore also

$$e_1 \setminus (s \setminus r) = e_1' \setminus (s \cup r) \subseteq e_2' \setminus (s \cup r) = e_2 \setminus (s \setminus r)$$

and we arrive at a contradiction.

It follows that $s_1 \setminus r$ is a k-nest-set of $H - r$. Since O is a NEO, s_2 must be a nest-set of $H - s_1$. Now, to verify O' we need to show that $s_2 \setminus r$ is a k-nest-set of $H - r - s_1$. However, this is clearly the same hypergraph as $(H - s_1) - r$ and the above obsevation applies again. We can repeat this argument for all s_i until s_ℓ and thus O' is a k-NEO. □

Lemma 5.5. *Let H be a hypergraph with k-NEO $O = (s_1, \ldots, s_\ell)$. Let H' be a connected subhypergraph of H and $\Delta = V(H) \setminus V(H')$ the set of vertices no longer present in the subhypergraph. Then the sequence $(s_1 \setminus \Delta, s_2 \setminus \Delta, \ldots s_\ell \setminus \Delta)$ is a k-NEO of H'.*

Proof. From the argument at the beginning of the proof of Lemma 5.4 we know that $s \setminus \Delta$ is empty or a nest-set of $H - \Delta$. Therefore, $I^*(s \setminus \Delta, H - \Delta)$ has a linear order under $\subseteq$. Now, since H' does not contain any vertices from Δ and is a subhypergraph of H we have $E(H') = E(H' - \Delta) \subseteq E(H - \Delta)$ and thus $I^*(s \setminus \Delta, H') \subseteq I^*(s \setminus \Delta, H - \Delta)$. Therefore $I^*(s \setminus \Delta, H')$ can be linearly ordered by $\subseteq$ and thus $s \setminus \Delta$ is a nest-set. This observation can again be iterated along the NEO in the same fashion as in the proof of Lemma 5.4 to prove the statement. □

5.2 Nest-Set Width in Comparision to Other Common Widths

A wide variety of hypergraph width measures have been studied in the literature. To provide some context for the later algorithmic results, we will first investigate how *nsw* relates to a number of prominent width notions from the literature. In particular, in this section we show that *nsw* is a specialization of β-hypertree width and incomparable to primal and incidence clique width and treewidth. The relationship to β-hypertree width is of particular interest since bounded β-*hw* also generalizes β-acyclicity. The section is structured around proving the following theorem.

Theorem 5.6. *Bounded nsw is a strictly less general property than bounded β-hw. In particular, the following two statements hold:*

1. *For every hypergraph H we have $\beta\text{-hw}(H) \leq 3\ nsw(H) + 1$.*
2. *There exists a class of hypergraphs with bounded β-hw and unbounded nsw.*

We begin by establishing a useful technical lemma that will eventually lead us to the second statement of Theorem 5.6. An important consequence of the following Lemma 5.7 is that the length (minus 1) of the longest β-cycle of H is a lower bound of $nsw(H)$ since any vertex in a cycle has to be removed at some point in any NEO.

Lemma 5.7. *Let $C = (e_1, v_1, e_2, v_2 \ldots, e_\ell, v_\ell, e_{\ell+1})$ be a β-cycle in a hypergraph H. For every nest-set s of H we have that $|s \cap \{v_1, \ldots, v_\ell\}|$ is either 0 or at least $\ell - 1$.*

Proof. Suppose the cardinality of $s \cap \{v_1, \ldots, v_\ell\}$ is not 0. That is, at least one vertex of C is in s. Since we can rotate the indices of a cycle arbitrarily we assume, w.l.o.g., that $v_1 \in s$. Then, e_2 and e_ℓ are both in $I(s)$. Recall that a β-cycles has $\ell \geq 3$ and that v_2 can occur only in e_1 and e_2 and no other edges. Similarly, v_ℓ can occur exclusively in $e_{\ell-1}$ and e_ℓ. We therefore see that $v_2 \notin e_\ell$

and $v_\ell \notin e_2$. Thus, e_2 and e_ℓ can only be comparable by $\subseteq_s$ if at least one of v_2 or v_ℓ is in s.

Suppose, w.l.o.g., $v_2 \in s$, then we have e_3 and e_ℓ in $I(s)$ and the same argument can be applied again, as long as the two edges are not adjacent in the cycle. We can then apply the argument exhaustively, until all edges of the cycle are in $I(s)$ at which point it is clear that at least $\ell - 1$ vertices are necessarily in s. □

Lemma 5.7 further emphasizes the aforementioned distinction between generalizing acyclicity in sense of tree-likeness and our approach. Any cycle graph C_n has hypertree width 2 whereas the lemma shows us that $nsw(C_n) \geq n - 1$ since any nest-set will contain at least one vertex of the cycle, so it must contain at least $n - 1$ of them. Furthermore, cycle graphs have clique width at most 4 [39] and treewidth at most 2. We therefore arrive at the following lemma.

Lemma 5.8. *There exists a class of hypergraphs that has bounded β-hw, treewidth, and clique width and unbounded nsw.*

The lemma establishes the second statement of Theorem 5.6. We can derive some further results by combining Lemma 5.8 with results from [71]. There it was shown that there exist classes of β-acyclic hypergraphs that have unbounded clique width and treewidth. In combination with the previous lemma this demonstrates that bounded clique width and bounded treewidth are incomparable to bounded *nsw*. The results in [71] also apply to incidence clique width and incidence treewidth and since the incidence graph of a cycle graph is also a cycle graph, so does Lemma 5.8. Thus, bounded *nsw* is also incomparable to bounded incidence clique width and bounded incidence treewidth. The resulting hierarchy is summarized in Figure 5.2 at the end of this section.

We move on to show that $\beta\text{-}hw\,(H) \leq 3\,nsw(H)+1$. We will give a procedure to construct a generalized hypertree decomposition of width k from a k-NEO. Since k-NEOs are hereditary, every subhypergraph of H will also have a generalized hypertree decomposition of width k. By a result of Adler, Grohe, and Gottlob in [6] we have that $hw(H) \leq 3\,ghw(H) + 1$. From there we can then derive our bound of $\beta\text{-}hw\,(H) \leq 3\,k + 1$. In particular, we make use of the observation that a nest-set is connected to the rest of the hypergraph only via its guard.

The necessary details of this observation are captured by the following two definitions and the key Lemma 5.11 below. The following construction is inspired by the hinge decompositions of Gyssens, Jeavons, and Cohen [75].

Definition 5.9 (Exhaustive Subhypergraphs)**.** Let H be a hypergraph and $E' \subseteq E(H)$. Let $E^* := \{e \in E(H) \mid e \subseteq \bigcup E'\}$ be the edges covered by E'. Then we call the subhypergraph H' with $E(H') = E(H) \setminus E^*$ the *exhaustive E'-subhypergraph* of H.

We use the term *connected exhaustive E'-subhypergraphs* of H to refer to the connected components of H' (considering each component as an individual hypergraph).

We use exhaustive subhypergraphs to express that, when we remove a set of edges E' from H, then we also want to remove the edges E^* that are covered by $\bigcup E'$. The following construction of a hypertree decomposition from a NEO will use sets of the form $\bigcup E'$ as its bags. This means that the respective bag also covers all edges in E^*. We are therefore interested in the components resulting from removing all of E^* instead of just E' from H.

In particular, we want to remove sets of edges E' in such a way that the exhaustive E'-subhypergraphs are all connected to E' via a single edge. This will allow us to bring together the decompositions of the subhypergraphs in a way that preserves all properties of hypertree decompositions.

Definition 5.10 (Exhaustive Hinges)**.** Let H be a hypergraph, $E' \subseteq E(H)$ and $C_1, \ldots, C_n$ the connected exhaustive E'-subhypergraphs of H. For an $e \in E(H)$ we say that E' is an *exhaustive e-hinge* if for every $i \in [n]$ we have that $V(C_i) \cap \bigcup E' \subseteq e$.

Lemma 5.11. *Let s be a k-nest-set of hypergraph H and let e_g be the guard of s. Then there exists an exhaustive e_g-hinge $E' \subseteq E(H)$ with the following properties:*

1. $\bigcup I(s, H) \subseteq \bigcup E'$
2. $|E'| \leq k$

Proof. Let s and e_g be as in the statement. Let λ be a minimal edge cover of $s \setminus e_g$. Observe that $|s \setminus e_g| < k$ as e_g is incident to s and therefore $|\lambda| < k$. We now claim that $E' = \lambda \cup \{e_g\}$ is the required hinge. Clearly we have $|E'| \leq k$. For the first property, recall that for every $e \in I(s, H)$ we have $e \setminus s \subseteq e_g$ and thus also $e \subseteq e_g \cup s$. It is then easy to see from the definition of E' that $e_g \cup s \subseteq \bigcup E'$ and the property follows.

What is left to show is that that E' is in fact an exhaustive e_g-hinge. Let C be one of the connected exhaustive E'-subhypergraphs of H and partition the set $V(C) \cap \bigcup E'$ in two parts: $I_1 := V(C) \cap s$ and $I_2 := V(C) \cap ((\bigcup E') \setminus s)$.

First we argue that $I_1 = \emptyset$. It was already established that $\bigcup I(s, H) \subseteq \bigcup E'$, thus every edge incident to s is removed in the exhaustive E'-subhypergraph. It is therefore impossible for a vertex of s to be in $V(C)$.

Second, observe that by construction every edge in E' is incident to s and by definition of the guard of s we thus have $((\bigcup E') \setminus s) \subseteq e_g$. It follows immediately that $I_1 \cup I_2 \subseteq e_g$ and the statement holds. □

Lemma 5.11 is the key lemma for our construction procedure. It tells us that we can always find a *small* exhaustive hinge E' in a hypergraph H if it has a k-NEO. By the first property from the lemma, the exhaustive E' subhypergraph no longer contains the vertices s. From the connected exhaustive E'-subhypergraphs we can construct subhypergraphs of H that connect to E' via a single edge and have shorter k-NEOs than H. Since the subhypergraphs are connected to E' via a single edge, it is straightforward to combine individual hypertree decompositions for every subhypergraph into a new decomposition for H. This step can then be applied inductively on the length of the k-NEO to construct a hypertree decomposition of width k for H.

Lemma 5.12. *For any hypergraph H it holds that $ghw(H) \leq nsw(H)$.*

Proof. We show by induction on $\ell \geq 1$ that if a hypergraph H has a k-NEO of length ℓ then it has a generalized hypertree decomposition of width at most k. For the base case, $\ell = 1$, the NEO consists of a single nest-set $s = V(H)$ with $|s| \leq k$. The base case then follows from the straightforward observation that $ghw(H) \leq |V(H)|$.

Suppose the statement holds for $\ell' < \ell$. We show that it also holds for every k-NEO of length ℓ. Let $O = (s_1, \ldots, s_\ell)$ be a k-NEO of H. Let e_g be the guard of s_1 and let E' be the exhaustive e_g-hinge from Lemma 5.11 and let $C'_1, \ldots, C'_n$ be the connected exhaustive E'-subhypergraphs. Finally, for each $i \in [n]$, we add e_g to C'_i to obtain the hypergraph C_i.

By Lemma 5.5 we see that for each $i \in [n]$, C_i has a k-NEO $O_i = (s_{1,i}, \ldots, s_{\ell,i})$ since it is a subhypergraph of H. Furthermore, according to Lemma 5.5, we can assume an O_i such that $s_{1,i} \subseteq s_i$ and, since e_g in C_i, also $s_{1,i} \neq \emptyset$.

Therefore, $C_i - s_{1,i}$ has a k-NEO of length at most $\ell - 1$ and we can apply the induction hypothesis to get a GHD $\langle T_i, (B_{u,i})_{u \in T_i}, (\lambda_{u,i})_{u \in T_i} \rangle$ with $ghw \leq k$ of $C_i - s_{1,i}$. Observe that the hypergraph has an edge $e_g \setminus s_{1,i}$ which has to be covered completely by some node $u_{g,i}$ in T_i.

Let u be a fresh node with $B_u = \bigcup E'$ and $\lambda_u = E'$. For each $i \in [n]$ we now change the root of T_i to be $u_{g,i}$ and attach the tree as a child of u. A cover λ_u of a node u in T_i can contain an edge e' that are not in H because the vertices $s_{1,i}$ are removed. Such an e' is always a subedge of an edge in H and can therefore be replaced by an edge in H in a way that the bag is still covered. We claim that this newly built decomposition is a generalized hypertree decomposition of H with $ghw \leq k$. It is not difficult to verify that this new structure indeed satisfies all proprieties of a generalized hypertree decomposition.

Connectivity Each subtree below the root already satisfies connectivity. The tree structure and the bags in the subtree remains unchanged. Furthermore, by construction of the hypergraphs C_i, the sets $B(T_i)$ of vertices occuring in bags of the tree T_i are pairwise disjoint except for the vertices in e_g. Since e_g is fully in B_u the only issue for connectivity can arise if there is a vertex in $B_u \cap B(T_i)$ but not in $B_{u_{g,i}}$. We argue that this is impossible.

Since E' is an exhaustive e_g-hinge and e_g was added back into each component it is easy to see that

$$B_u \cap B(T_i) = \bigcup E' \cap V(C_i - s_{1,i}) \subseteq e_g \setminus s_{1,i}$$

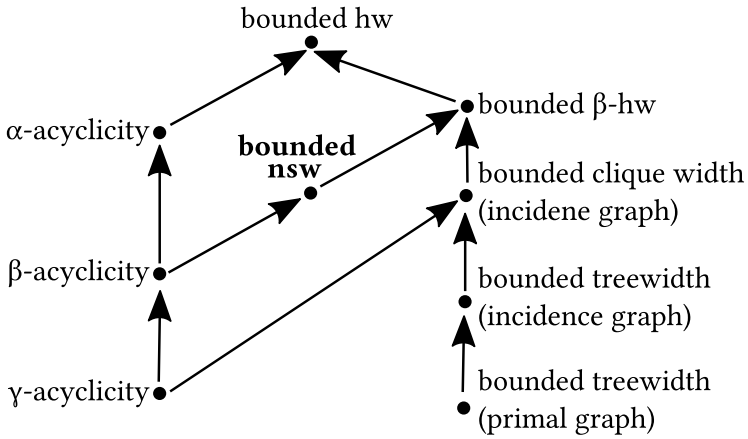


Figure 5.2: Expressive power of various hypergraph properties from [71], extended by bounded *nsw*. (Arcs are directed from less general to more general. Properties with no directed connection are incomparable.)

The rightmost term is exactly the edge that informed our choice of $u_{g,i}$, i.e., we have $B_u \cap B(T_i) = e_g \setminus s_{1,i} \subseteq B_{u_{g,i}}$ by construction.

Every edge of H is covered For every edge $e \in E(H)$ we consider two cases. Either $e \in I(s_1, H)$ or not. In the first case, by Lemma 5.11 we have $e \subseteq \bigcup E'$ and therefore it is covered in the root node u. In the second case, $e \notin I(s_1, H)$, e will occur unchanged in one of the hypergraphs $C_i - s_{1,i}$ since the removal of $s_{1,i}$ does not affect it (recall $s_{1,i} \subseteq s_i$). Since the tree decomposition of $C_i - s_{1,i}$ remain the same, except for changing which node is the root, e must be covered in the respective subtree corresponding to component $C_i - s_{i,1}$. □

Proof of Theorem 5.6 (1). By Lemma 5.12 we have that $ghw(H) \leq nsw(H)$. As mentioned above, we always have $hw(H) \leq 3\, ghw(H) + 1$ for every hypergraph and therefore also $hw(H) \leq 3\, nsw(H) + 1$. In combination with Lemma 5.5 we see that for every subhypergraph H' of H we have $hw(H') \leq 3\, nsw(H') + 1 \leq 3\, nsw(H) + 1$. □

The results of this section are summarized in Figure 5.2. The diagram extends the hierarchy given in [71] by bounded *nsw*.

5.3 The Complexity of Checking Nest-Set Width

For the existing generalizations of β-acyclicity – β-hw and pw – it is not known whether one can decide in polynomial time if a structure has width $\leq k$, even when k is a constant. This then also means that no efficient algorithm is known to compute the respective witnessing structures. In these situations, tractability results are inherently limited. One must either assume that the witnesses are given as an input or that a tractable algorithm does not use the witness at all. In comparison, deciding treewidth $\leq k$ is fixed-parameter tractable when parameterized by k [22] and checking hypertree width is tractable when k is constant [68].

When k is considered constant, it is straightforward to find a k-NEO in polynomial time, if one exists. We can simply check for all combinations of up to k vertices whether they represent a nest-set. If so, eliminate the nest-set and repeat from the beginning on the new hypergraph until it becomes empty. By Lemma 5.4, this greedy approach of always using the first found k-nest-set will result in a sound and complete procedure.

However, we can improve on this straightforward case by analyzing the following decision problem where k is part of the input.

Nest-Set-Width	
Instance:	A hypergraph H, integer k
Question:	$nsw(H) \leq k$?

We first observe that Nest-Set-Width is NP-complete in Section 5.3.1. In more positive news, we are able to show that Nest-Set-Width is fixed-parameter tractable when parameterized by k in Section 5.3.2. Importantly, the fixed-parameter algorithm explicitly constructs a k-NEO as a witness, if one exists, and can therefore serve as a basis for the algorithmic results in the following sections.

5.3.1 Checking Nest-Set Width is NP-Complete

We show NP-hardness by reduction from Vertex-Cover, a classical NP-complete problem [80]. Due to space limitations we provide only an outline of the most important ideas for the reduction here. A full proof is available in the appendix. In the Vertex-Cover problem we have as input a graph $G = (V, E)$ and an integer $k \geq 1$. The problem is to decide whether there exists a set $\alpha \subseteq V$ with $|\alpha| \leq k$ such that every edge of G is incident to at least on vertex in α. Such a set α is called a *vertex cover* of G.

Intuitively, finding a vertex cover of G can be seen as making a sequence of choices on the edges of G. For every edge $\{u, v\}$ of G either u, or v, or both must be in the vertex cover. We can intuitively encode this choice into finding a nest-set by two edges (the *choice edges*) $e_u = X \cup \{u\}$ and $e_v = X \cup \{v\}$ where X is some set of vertices that contains neither u nor v. Observe that e_u and e_v become comparable by $\subseteq$ exactly when u, or v, or both vertices are removed. Hence, by enforcing that e_u and e_v are in $I(s)$ for every nest-set s, we can encode the choice of how to cover $\{u, v\}$ into finding a nest-set.

When we encode the covering choices for each edge this way we need to be careful to not introduce any additional choices as artifacts of our construction. We therefore construct our choices in *layers* such that every layer corresponds to the vertex cover choice for an edge in G. We let both of the edges that encode the choice at layer ℓ contain *all* vertices that occur in all lower layers. Then, all edges on lower layers are already subsets of the edges at layer ℓ, even without any vertex removal. If one removes vertices such that all the choices in the individual layers are resolved, then all edges of the construction are linearly ordered.

To simplify the following proof we make two additional assumptions on the instances of Vertex-Cover. We assume that the input graph has at least 2 edges and that k is strictly less than the number of edges in G. If either assumption is violated the problem is trivial.

We first prove that it is NP-complete to decide whether a hypergraph has a k-nest-set. The hardness of Nest-Set-Width then follows from the following argument that the hypergraph H in the reduction has a $(km + k)$-nest-set if and

only if it has nest-set width at most $km + k$.

Theorem 5.13. *Nest-Set-Width is NP-complete.*

Proof. Membership is straightforward. Guess up to k vertices s and verify the orderability of $I^*(s, H)$. Hardness is by many-one reduction from Vertex-Cover. Hence, let G, k be an instance of Vertex-Cover and let n and m refer to the number of vertices and edges in G, respectively. In the following we construct a hypergraph H such that H has a $(km + k)$-nest-set if and only if G has a vertex cover of size at most k.

Let $\{e_1, \ldots, e_m\}$ be the edges of G and let $\{v_1, \ldots, v_n\}$ be the vertices of G. Our H will have as vertices $V(H) = \bigcup\{v_j^{(i)} \mid j \in [n], i \in [m+1]\}$ a copy of every vertex v_j associated to edge e_i. We will refer to the superscript (i) also as the ith *level* of H. We will write $V^{\leq i}$ for $\{v_j^\ell \in V(H) \mid \ell \leq i\}$, i.e., all the vertices at level i or lower.

For each edge $e_i = \{a, b\}$ of G, we create two edges $f_{i,1}$ and $f_{i,2}$ in H as follows.

$$f_{i,1} = V^{\leq i} \setminus \{a^{(i)}\} \qquad f_{i,2} = V^{\leq i} \setminus \{b^{(i)}\}$$

Furthermore, we also add two edges $f_{m+1,1} = V^{\leq m+1} \setminus \{a^{(m+1)}\}$ and $f_{m+1,2} = V^{\leq m+1} \setminus \{b^{(m+1)}\}$ at the final level for $e_1 = \{a, b\}$. Intuitively, these f edges at level i represents the choice between a and b for edge e_i, as one needs to be deleted for the two edges to be comparable by $\subseteq$. We will therefore refer to them as the *choice edges*. Encoding the choice for e_1 twice, at levels 1 and $m + 1$, is done for technical reasons that will become apparent later. H also contains the complete graph $K^{(j)}$ over the vertices $\{v_j^{(i)} \mid i \in [m+1]\}$ for every vertex v_j of G. Intuitively, they link the choices at every level to each other and we therefore refer to them as the *linking cliques*. Thus we have

$$E(H) = \{f_{i,1}, f_{i,2} \mid i \in [m+1]\} \cup \bigcup_{j \in [n]} E(K^{(j)})$$

We first show that if α is a vertex cover of G and $\ell \leq k$, then $s_\alpha = \bigcup_{v_j \in \alpha}\{v_j^{(i)} \mid i \in [m+1]\}$ is a $(km + k)$-nest-set of H. Note that $|s_\alpha| \leq k(m + 1)$ follows immediately from the construction.

Claim A. For each $i \in [m+1]$ we have $f_{i,1} \subseteq_{s_\alpha} f_{i,2}$ or vice versa.

Proof of claim: Let $e_i = \{a, b\}$ and, w.l.o.g., assume $a \in \alpha$ and thus also $a^{(i)} \in s_\alpha$. Clearly, $f_{i,2} \setminus \{a^{(i)}\} = V^{\leq i} \setminus \{a^{(i)}, b^{(i)}\} \subseteq f_{i,1}$. Removing further vertices from both can not change the order anymore and thus $f_{i,2} \setminus s_\alpha \subseteq f_{i,1} \setminus s_\alpha$. If $b \in \alpha$ the analogous argument yields the opposite order. The same argument also applies for $f_{m+1,1}$ and $f_{m+1,2}$ and e_1. △

By construction we always have $f_{i,1}, f_{i,2} \supseteq f_{h,1}, f_{h,2}$ for $h < i$. Thus, in combination with Claim A we have that all choice edges are linearly orderable by $\subseteq_{s_\alpha}$. The only other edges in $I(s_\alpha)$ are those of the linking cliques $K^{(j)}$ where $v_j \in \alpha$. Clearly, all edges of the clique become empty, as $V(K^{(j)}) \subseteq s_\alpha$, and thus $I^*(s_\alpha)$ is linearly orderable by $\subseteq$.

For the other direction, suppose s is a $(km+k)$-nest-set of H. We now define α as containing exactly those vertices v_j such that $v_j^{(i)} \in s$ for at least m distinct i. Note that because of the linking cliques and Lemma 5.7, a vertex $v_j^{(i)}$ occurs either for 0 or at least m distinct i in s. Using the assumptions that $2 \leq m$ and $1 \leq k < m$ from above it is straightforward to verify that $\frac{km+k}{m} < k+1$ and therefore also $|\alpha| \leq k$.

What is left is to show that every edge e_i in G is incident to a vertex in α. For any $e_i = \{a, b\}$ the choice edges $f_{i,1}, f_{i,2}$ are only comparable by $\subseteq_s$ if either $a^{(i)} \in s$ or $b^{(i)} \in s$ (or both). Then, because of the linking cliques, a or b (or both) will be in α.

It follows that if $f_{i,1}, f_{i,2} \in I(s, H)$, then e_i will be covered by α. Then, since s is not empty, there is some v_j such that $v_j^{(i)}$ occurs for m distinct i in s. In particular, then either $v_j^{(1)}$ or $v_j^{(2)}$ are in s and thus for every $2 < i \leq m+1$ we have $f_{i,1}, f_{i,2} \in I(s)$. Thus, for every edge in G there is a pair of choice edges in $I(s)$. By the argument above every edge of G is therefore incident to a vertex in α. □

We now build on this reduction to prove Theorem 5.13. Suppose the same situation as in the above proof, i.e., a vertex cover instance G, k and the hypergraph H from the reduction above. If G has a vertex cover α, then we can eliminate all the vertices s_α in H that encode the graph vertices from α. By the argument

above we have that $E(H) = I(s_\alpha)$ is linearly ordered by $\subseteq_{s_\alpha}$. It is not difficult to see that when all edges of a hypergraph are linearly ordered by $\subseteq$ then that hypergraph is β-acyclic: if a vertex v is included in every edge then $\{v\}$ is a nest-set. Thus, $H - s_\alpha$ has a 1-NEO O' and thus prepending s_α to O' gives us a $(km + k)$-NEO of H.

On the other hand, suppose H has a $(km + k)$-NEO $O = (s_1, \dots)$. As s_1 is a $(km + k)$-nest-set of H, the arguments from the proof apply and we have that G has a vertex cover of size at most k. This now also completes the proof of Theorem 5.13.

On a final note, one may notice similarities between finding nest-sets and an important work by Yannakakis [113] on vertex-deletion problems in bipartite graphs. Yannakakis gives a complexity characterization for vertex-deletion problems on bipartite graphs that extends to hypergraphs via their incidence graph. Furthermore, the specific problem of finding a vertex-deletion such that the edges of the hypergraph become linearly ordered by $\subseteq$ is stated to be polynomial. While this strongly resembles the nest-set problem, the results of Yannakakis are not applicable here since we are not interested in a global property of the hypergraph but only in the orderability of the edges that are incident to the deleted vertices.

5.3.2 Checking Nest-Set Width in a Parameterized Setting

Recall that every nest-set s has a maximal edge with respect to $\subseteq_s$; the guard of s. The main idea behind the algorithm presented in this section is to always fix an edge e_g and check if there exists a nest-set that specifically has e_g as its guard. This will allow us to incrementally build a nest-set s relative to the guard e_g. We first demonstrate this principle in the following example.

Example 5.2. We consider a hypergraph H with three edges $e_1 = \{a, b, c, d\}$, $e_2 = \{a, b, c, g\}$, and $e_3 = \{c, d, g, f\}$. We want to find a nest-set with guard e_1. The hypergraph with e_1 highlighted is shown in Figure 5.2. To start, if s is a nest-set with guard e_1, then at least one vertex of e_1 must be in s. For this example let $a \in s$.

Since $a \in s$ we also have that $e_2 \in I(s)$. For s to be a nest-set with guard e_1 it must then hold that $e_2 \setminus s \subseteq e_1 \setminus s$. Since g is in e_2 but not in e_1 we can deduce

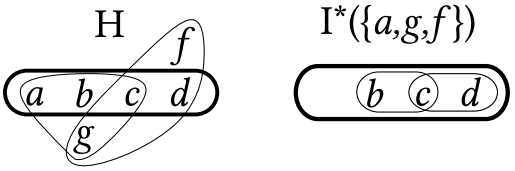


Figure 5.3: Illustration of Example 5.2

that also $g \in s$. More generally, any vertex that occurs in an edge from $I(s)$ but not in e_1 must be part of the nest-set s. Now, since $g \in s$ it follows that $e_3 \in I(s)$ and therefore, by the previous observation, also $f \in s$.

At this point we have deduced that if a is in s, then so are g and f. We now have the situation that for every edge $e \in I(s)$ we have $e \setminus \{a, g, f\} \subseteq e_1 \setminus \{a, g, f\}$. However, as illustrated in Figure 5.2, $I^*(\{a, g, f\})$ is not linearly ordered by $\subseteq$. A nest-set must therefore contain further vertices. In this case it is easy to see that either removing b or d is enough. In conclusion we have shown that if $a \in s$, then there are two 4-nest-sets $\{a, b, e, f\}$ and $\{a, d, e, f\}$ that have guard e_1. △

What makes the problem difficult is that there can be many possible ways of making edges linearly ordered by vertex deletion. In Example 5.2 both choices, removing either b or d, lead to a 4-nest-set. However, suppose there were an additional edge $e_4 = \{b, x\}$. Then, choosing b would also imply $e_4 \in I(s)$ and $x \in s$. Choosing d would lead to a smaller nest-set.

In general, this type of complication can occur repeatedly and it is therefore necessary to continue this expansion procedure for all possible (minimal) ways of ordering the known incident edges of s. We will therefore first establish an upper bound on these possible expansions.

Intuitively, when we have edges $\{a, b\}$ and $\{b, c\}$, the only way they become comparable by $\subseteq$ is if either a or c is removed. The existence of a linear order over all the edges thus requires resolving all such conflicts. By encoding these conflicts in a kind of conflict graph we can see that the problem is equivalent to finding a vertex cover in the conflict graph.

Definition 5.14 ($\subseteq$-conflict graph)**.** Let H be a hypergraph, we define the $\subseteq$-*conflict graph* of H as the graph obtained by the following construction (with

$V(G) = \bigcup E(G)$): For every two distinct edges $e_1, e_2 \in E(H)$, if $v \in e_1 \setminus e_2$ and $u \in e_2 \setminus e_1$, then add an edge $\{v, u\}$ to G. We say that u and v have a $\subseteq$-*conflict* in H.

Lemma 5.15. *Let H be a hypergraph and let $s \subseteq V(H)$. Then $E(H - s)$ is linearly ordered by $\subseteq$ if and only if s is a vertex cover of the $\subseteq$-conflict graph of H.*

Proof. Let G be the $\subseteq$-conflict graph of H. We first show the implication from right to left. Let s be a vertex cover for G and suppose that $E(H - s)$ is not linearly ordered by $\subseteq$. Hence, there are two edges $e_1, e_2 \in E(H - s)$ that are incomparable, i.e., there exist vertices $v \in e_1 \setminus e_2$ and $u \in e_2 \setminus e_1$ and neither v nor u is in the vertex cover s. A conflict can not be introduced by removing the vertices of s and therefore it was already present in H. Therefore, there must be an edge $\{v, u\}$ in G that is not covered by s, contradicting that s is a vertex cover.

For the other direction let $s \subseteq V(H)$ such that $E(H - s)$ is linearly ordered by $\subseteq$. Then for every $\subseteq$-conflict, i.e., every pair of vertices u, v where there are $e_1, e_2 \in E(H)$ with $v \in e_1 \setminus e_2$ and $u \in e_2 \setminus e_1$, at least one of u, v must be in s. All edges of G are exactly between such pairs of vertices, hence s contains at least one vertex of each edge in G. Therefore s is also a vertex cover of G. □

This correspondence allows us to make use of the following classical result by Fernau [52] on the enumeration of all *minimal vertex covers*. A vertex cover is called a minimal vertex cover if none of its subsets is a vertex cover.

Proposition 5.16 ([52])**.** *Let G be a graph with n vertices. There exist at most 2^k minimal vertex covers with size $\leq k$ and they can be fully enumerated in $O(2^k k^2 + kn)$ time.*

In combination with Lemma 5.15 we therefore also have an upper bound on computing all minimal vertex deletions that resolve all $\subseteq$-conflicts. With this we are now ready to state Algorithm 5.1 which implements the intuition described at the beginning of this section. The algorithm is given a hypergraph and an edge e_g to use as guard and tries to find a k-nest-set with guard e_g by exhaustively following the steps described in Example 5.2. We are able to show that this

Algorithm 5.1: Find nest-set with fixed guard.

input: Hypergraph H, edge e_g, and an integer $k \geq 1$.
output: "Accept", if there exists a nest-set s with guard e and $|s| \leq k$
"Reject", otherwise.

```
1  Function NestExpand (s : set of vertices)
2      if |s| > k then
3          return Reject
4      end
5      Δ ← ⋃ I(s, H) \ (s ∪ e_g);
6      if Δ ≠ ∅ then
7          return NestExpand (s ∪ Δ);
8      end
9      H_g ← The hypergraph I*(s, H);
10     if H_g has no ⊆-conflicts then
11         return Accept;
12     end
13     A ← all minimal vertex covers of the ⊆-conflict graph of H_g with
         size at most k − |s|;
14     foreach α ∈ A do
15         if NestExpand (s ∪ α) accepts then
16             return Accept;
17         end
18     end
19     return Reject;
20 begin                                              /* Main */
21     foreach v ∈ e_g do
22         if NestExpand ({v}) then
23             return Accept;
24         end
25     end
26     return Reject;
27 end
```

indeed leads a correct procedure for finding k-nest-sets with a specific guard. See the appendix for a full proof of this statement.

Lemma 5.17. *Algorithm 5.1 is sound and complete.*

Proof. The algorithm has one base-case for acceptance, when $\Delta = \emptyset$ and $I^*(s, H)$ is linearly orderable by $\subseteq$. Clearly then s is a nest-set. As $\Delta = \emptyset$, for every (sub)edge $e \in I^*(s, H)$ we have $e \subseteq e_g$, i.e., e_g is a guard of s. From the check at the beginning of the NestExpand we have $|s| \leq k$. Hence, if the algorithm accepts then the current s is a k-nest-set with guard e_g.

To establish completeness we show that if a k-nest-set s with guard e_g exists, then the algorithm will accept. In particular we claim that for every call NestExpand(s'), if there exists a k-nest-set s with guard e_g and $s' \subseteq s$, then either s' is a k-nest set or $s \cup X \subseteq s$ where $s \cup X$ is the parameter of one of the calls made directly by NestExpand(s').

We distinguish two cases. First suppose there are edges $e \in I(s')$ such that $e \setminus s' \not\subseteq e_g \setminus s'$. Since e_g is the guard of s, and $I(s') \subseteq I(s)$, every element of $e \setminus e_g$ must necessarily also be in s. This corresponds directly to the set Δ in the algorithm. Hence, $s' \cup \Delta \subseteq s$ when $\Delta \not\emptyset$, which is clearly the only parameter of a child call.

In the other case, there are no such edges. The claim then states that either s' is a k-nest-set and the algorithm accepts, or that $s' \cup \alpha \subseteq s$ for some $\alpha \in \mathbf{A}$. Now suppose the claim is false, i.e., there exists a k-nest-set $s \supseteq s'$ with guard e_g such that s' is not a k-nest-set and $\forall \alpha \in \mathbf{A}. s' \cup \alpha \not\subseteq s$. By Lemma 5.15, $\mathbf{A}$ contains all possible minimum deletions with size at most $k - |s'|$ that make $I^*(s')$ linearly orderable by $\subseteq$. It follows that some $\subseteq$-conflict from H_g must remain in $I^*(s)$ as otherwise some $\alpha \in \mathbf{A}$ would be a subset of s. This contradicts the fact that that assumption s is a nest-set and thereby proves the claim.

With the claim established, completeness then follows from the fact if e_g is a guard of nest-set s, then $e_g \cap s \neq \emptyset$. Hence, if there exists a k-nest-set s with guard e_g it must contain at least one vertex v of e_g. Inductive application of the claim then proves that a k-nest-set will be found by the algorithm (and accepted) when starting from NestExpand($\{v\}$). □

For the sake of simplicity, Algorithm 5.1 is stated as a decision procedure. Even so, it is easy to see that a k-nest-set with the appropriate guard has been constructed at any accepting state. It is then straightforward to use Algorithm 5.1 to decide in fixed-parameter polynomial time if a hypergraph

has any k-nest-set, and if so output one. In the following we use $\|H\| = |V(H)| + |E(H)|$ for the *size* of hypergraph H.

Theorem 5.18. *There exists a $2^{O(k^2)}poly(\|H\|)$ time algorithm that takes as input hypergraph H and integer $k \geq 1$ and returns a k-nest-set s of H if one exists, or rejects otherwise.*

Proof. We simply call Algorithm 5.1 once for each edge of H as the guard. Since every nest-set has a guard Lemma 5.17 implies that this will find an appropriate nest-set if one exists. If all calls reject, then there can be no nest-set with at most k elements as it is not guarded by any edge of H.

What is left to show is that Algorithm 5.1 terminates in $2^{O(k^2)}poly(k\,\|H\|)$ time. Calling the procedure $|E(H)|$ times clearly preserves this bound. First, observe that every recursive call of NestExpand increases the cardinality of s by at least one. The call tree of the recursion therefore has maximum depth k. Furthermore, by Proposition 5.16 every node in the call tree has at most 2^k children if $\Delta = \emptyset$, or exactly one when $\Delta \neq \emptyset$. Hence, at most $2^{(k^2)}|e_g|$ calls to NestExpand are made in one execution of Algorithm 5.1.

In each call, the computation of Δ and H_g as well as all the checks are feasible in $O(poly(\|H\|))$ time. Since $\|H_g\| \leq \|H\|$, the set $\mathbf{A}$ can be computed in $O(2^k k^2 + poly(k\,\|H\|))$ time according to Proposition 5.16. Recall that we assume $k \leq V(H)$ since the problem is trivial otherwise. The overall execution time of Algorithm 5.1 is therefore in $2^{O(k^2)}poly(\|H\|)$. □

Once we can find individual k-nest-sets, finding k-NEOs becomes simple. Recall from Lemma 5.4 that vertex removal preserves k-NEOs. Thus straightforward greedy removal of k-nest-sets is a sound and complete algorithm for finding k-NEOs. Since at most $|V(H)|$ nest-set removals are required to reach the empty hypergraph, using the procedure from Theorem 5.18 to find the k-nest-sets yields a $2^{O(k^2)}poly(\|H\|)$ time algorithm for Nest-Set-Width.

Corollary 5.19. *Nest-Set-Width parameterized by k is fixed-parameter tractable.*

5.4 Nest-Set Width & Conjunctive Queries with Negation

We move on to prove our main algorithmic result. Recall that a query q has an associated hypergraph $H(q)$. We define the nest-set width of the query q as $nsw(q) = nsw(H(q))$. We say that a class $\mathcal{Q}$ of $\texttt{BoolCQ}^\neg$ instances has *bounded nsw* if there exists a constant c, such that every query q in $\mathcal{Q}$ has $nsw(q) \leq c$.

Theorem 5.20. *For every class $\mathcal{Q}$ of $\texttt{BoolCQ}^\neg$ instances with bounded nsw, $\texttt{BoolCQ}^\neg$ is decidable in polynomial time.*

As usual, the result can be extended to unions of conjunctive queries with negation (UCQ$^\neg$) when the *nsw* of a UCQ$^\neg$ is defined to be the maximum *nsw* of its CQ$^\neg$ parts.

While the complexity of CQs without negation has been extensively studied and is well understood, few results extend to the case where negation is permitted. When there are only positive literals, then the satisfying assignments for each literal are explicitly present in the database. Finding a solution for the whole query thus becomes a question of finding a consistent combination of these explicitly listed partial assignments. However, with negative literals it is possible to implicitly express a large number of satisfying assignments. Recovering an explicit list of satisfying assignments for a negative literal may require exponential time and space.

This additional expressiveness of negative literals has important implications for the study of structural parameters. While evaluation of CQs is NP-complete with and without negation, permitting negation allows for expressing problems as queries with a *simpler* hypergraph structure. Such a change in expressiveness relative to structural complexity must also be reflected in structural parameters that capture tractable classes of the problem.

As an example, consider SAT for propositional formulas in conjunctive normal form (CNF). Recall, that for a formula F in CNF, the corresponding hypergraph $H(F)$ has as its vertices the variables of the formula and every edge is the set of variables of some clause in the formula. A clause $C = l_1 \vee \cdots \vee l_q$ has $2^q - 1$ satisfying assignments to the variables of the clause. Thus, a corresponding

positive literal in a CQ, that contains all the satisfying assignments, will be of exponential size (unless the size of clauses is considered bounded). On the other hand, there is a single assignment to $vars(C)$ that does not satisfy C. It is therefore possible to compactly encode SAT by having a negative literal for each clause that excludes the respective non-satisfying assignment. Since this reduction preserves the hypergraph structure of the SAT formula it follows that structural restrictions can only describe a tractable fragment of $\texttt{BoolCQ}^\neg$ if they also make SAT tractable. For example, SAT is NP-hard when restricted to α-acyclic formulas [97], and thus so is $\text{CQ}^\neg$ evaluation. In contrast, evaluation of CQs without negation is tractable for α-acyclic queries [112].

Theorem 5.21 (Implicit in [97])**.** $\texttt{BoolCQ}^\neg$ *is NP-hard even when restricted to* α*-acyclic queries.*

Simplifying Assumptions To simplify the presentation we make the following assumptions on the instances of $\texttt{BoolCQ}^\neg$. First we assume that queries in Nest-Set-Width instances are always *safe*, i.e., no variable occurs only in negative literals. An unsafe query can always be made safe: If a variable v occurs only in negative literals, we simply add a new literal $R(v)$ with $R^D = \{(d) \mid d \in Dom\}$ to the query. The resulting query is clearly equivalent to the unsafe one on the given domain. Importantly, the additional unary literals does not change the nest-set width of the query. At some points in the algorithm we operate on (sub)queries that are not safe. The assumption of safety is made for the starting point of the procedure.

Our second assumption is that the size of the domain is exactly a power of 2, i.e., $|Dom| = 2^d$ for some integer d. Since we already assume safe queries, increasing the size of the domain has no effect on the solutions since the newly introduced constants cannot be part of any solution. Furthermore, this assumption increases the size of the domain at most by a constant factor less than 2.

5.4.1 Relation to Previous Work

The algorithm presented here builds on the work of Brault-Baron [23] for the β-acyclic case. While we can reuse some of the main ideas, the overall approach

used there does not generalize to our setting. There the tractability is first shown for boolean domains, i.e., the domain is restricted to only two values. BoolCQ$^\neg$ over arbitrary domains is reduced to the problem over the boolean domain by blowing up each variable in such a way as to encode the full domain using boolean variables. This naturally requires every variable in the original query to be replaced by $\log_2 |Dom|$ many new variables. While this operation preserves β-acyclicity, it can increase nsw by a factor $\log_2 |Dom|$.

Example 5.3. Consider the following query and a domain with 8 elements.

$$q = \neg R(a, b) \wedge \neg S(b, c) \wedge \neg T(a, c)$$

The reduction to a query over the boolean domain will then replace every variable v by three variables v_1, v_2, v_3, resulting in the equivalent query q_b over the boolean domain

$$\begin{aligned} q_b = &\neg R(a_1, a_2, a_3, b_1, b_2, b_3) \wedge \neg S(b_1, b_2, b_3, c_1, c_2, c_3) \wedge \\ &\neg T(a_1, a_2, a_3, c_1, c_2, c_3) \end{aligned}$$

It is easy to see that q has $nsw(q) = 2$ because any combination of two variables is a nest-set of q. However, while $\{a, b\}$ is a nest-set of q, this does not translate to the existence of a 2-nest-set in q_b. It is easy to verify that any $\{a_i, b_j\}$ for $i, j \in [3]$ is not a nest-set. Indeed, applying the ideas from Section 5.3.2 it is easy to see that in general, for such a triangle query, $nsw(q_b) = 2 \log_2 |Dom|$. △

A subtle but key observation must be made here. While the previous example shows that the variable blowup from the binary encoding affects the nest-set width in general, this does not happen when $nsw(q) = 1$. Consider a nest-set $\{v\}$ of some hypergraph H. The edges incident to v are linearly ordered by $\subseteq$. If we add a new vertex v' in all the edges that contain v, then clearly the edges incident to v' are the same as those of v and therefore also linearly ordered by $\subseteq$.

Lemma 5.22. *Let H be a hypergraph with a nest-set $\{v\}$. Let H' be a hypergraph obtained by adding a new variable v' to H that occurs exactly in the same edges as v. Then $\{v\}$ and $\{v'\}$ are both nest-sets of H'.*

This subtle difference between 1-nest-sets and larger nest-sets will be principal to the following section.

5.4.2 Eliminating Variables

The BoolCQ$^\neg$ algorithm in the following Section 5.4.3 will be based around successive elimination of variables from the query. This elimination will be guided by a nest-set elimination ordering where we eliminate all variables of a nest-set at once. This elimination of a nest-set s is performed in three steps.

1. Eliminate all occurrences of variables from s in positive literals.
2. Extend the negative literals incident to s in such a way that they form a β-acyclic subquery.
3. Eliminate the variables of s from the β-acyclic subquery.

In this section we introduce the mechanisms used for these steps. For steps 1 and 2 we need to extend literals in such a way that their variables include all variables from some set s. We do this in a straightforward way by simply extending the relation by all possible tuples for the new variables. It is then easy to see that such extensions are equivalent with respect to their set of satisfying solutions.

Definition 5.23. Consider a literal $L(v_1, \ldots, v_n)$ where L is either R or $\neg R$ and the respective relation R^D. Let s be a set of variables and let $s' = s \setminus \{v_1, \ldots, v_n\}$ be the variables in s that are not used in the literal. We call the literal $L(v_1 \ldots, v_n, \vec{s'})$ with the new relation $R^D \times Dom^{|s'|}$ the *s-extension of R* (where Dom^m represents the m-ary Cartesian power of the set Dom and we use the relational algebra semantics of the product $\times$).

Lemma 5.24. *Let $L'(\vec{v}, \vec{s'})$ be the s-extension of $L(\vec{v})$. Then the following holds: an assignment $a\colon vars(L) \to Dom$ satisfies $L(\vec{v})$ if and only if every extension of a to $vars(L')$ satisfies $L'(\vec{v}, \vec{s'})$.*

Proof. Let $L(\vec{v})$ where L is either R or $\neg R$ and let $L'(\vec{v}, \vec{s'})$ be the s-extension. Let $a\colon vars(L) \to Dom$ be an assignment that satisfies L. If L is positive, we have $a[\vec{v}] \in R^D$ and then every extension of the tuple to $vars(L')$ exists by the semantics of the relational product. If L is negative, then $a[\vec{v}] \notin R^D$. The

relational product for creating the relation of the s-extension will therefore also not create any tuples where $a[\vec{v}]$ occurs in the projection to $\vec{v}$.

On the other hand, let $a\colon vars(L) \to Dom$ such that every $a'\colon vars(L') \to Dom$ that extends a satisfies L'. If L is positive, then any such a' also satisfies $\pi_{vars(L)}(L')$ and therefore a satisfies L as the relational product does not change the tuples of L. If L is negative, consider the tuple $t = a[\vec{v}]$. Suppose $t \in R^D$, i.e., a does not satisfy L. But then any extension of a would also be in the relation of the s-extension, contradicting our assumption that every extension satisfies L'. □

The process for positive elimination is simple. Straightforward projection is used to create a positive literal without the variables from s. A new negative literal then restricts the extensions of satisfying assignments for the new positive literal to exactly those that satisfy the old positive literal. A slightly simpler form of this method was already used in [23].

Lemma 5.25. *Let $R(\vec{x}, \vec{s})$ be a positive literal. Define new literals $P(\vec{x})$ with $P^D = \pi_{\vec{x}}(R^D)$ and $\neg C(\vec{x}, \vec{s})$ with $C^D = P_s^D \setminus R^D$ where P_s is the s-extension of P. Then an assignment a satisfies $R(\vec{x}, \vec{s})$ if and only if a satisfies $P(\vec{x}) \wedge \neg C(\vec{x}, \vec{s})$.*

Proof. Let a be a satisfying assignment for $R(\vec{x}, \vec{s})$. Clearly, a also satisfies $P(\vec{x})$ and by Lemma 5.24 it also satisfies P_s. Furthermore, by construction $a[\vec{x}, \vec{s}]$ is explicitly not in C^D and hence a also satisfies $\neg C(\vec{x}, \vec{s})$.

On the other hand. Let a be a satisfying assignment for $P(\vec{x}) \wedge \neg C(\vec{x}, \vec{s})$. By construction it is then clear that a satisfies only those extensions of $\vec{x}$ that correspond to a tuple in R^D. At the same time $a[\vec{x}]$ is in $\pi_x(R^D)$. Hence, a always corresponds to a tuple in R^D and we see that a satisfies $R(\vec{x}, \vec{s})$. □

For the elimination of variables that occur only negatively we build upon a key idea from [23]. There, a method for variable elimination is given for the case where the domain is specifically $\{0, 1\}$. We repeat parts of the argument here to highlight some important details. Consider a query $q = \{\neg R_1, \ldots, \neg R_n\}$. The main observation is that the satisfiability of the negative literals $\neg R_1, \ldots, \neg R_n$

with variables $x_1, \ldots, x_m$ is equivalent to satisfiability of the formula

$$\bigwedge_{i=1}^{n} \bigwedge_{(a_1,\ldots,a_q)\in R_i} \left(a_1 \neq x_{i_1} \vee \cdots \vee a_q \neq x_{i_q}\right)$$

Since we are in the domain $\{0, 1\}$ we have only two cases for the inequalities. Either $0 \neq x$ or $1 \neq x$, which are equivalent to $x = 1$ and $x = 0$, respectively. We can therefore equivalently rewrite the clauses in the formula above as the propositional formula

$$\sigma(a_1)x_{i_1} \vee \cdots \vee \sigma(a_q)x_{i_q}$$

where $\sigma(1) = \neg$ and $\sigma(0) = \epsilon$, i.e., the empty string.

Recall, if we have two clauses $x \vee \ell_1 \vee \cdots \vee \ell_\alpha$ and $\overline{x} \vee \ell'_1 \vee \ell'_\beta$ the *x-resolvent* of the two clauses is $\ell_1 \vee \cdots \vee \ell_\alpha \vee \ell'_1 \vee \cdots \vee \ell'_\beta$. Removing all clauses containing variable x and adding all x-resolvents as new clauses to a given formula in CNF yields an equi-satisfiable formula without the variable x. This process is often referred to as Davis-Putnam resolution [42]. If we then reverse the initial transformation from query to propositional formula, we obtain a new query q' (and corresponding database) that no longer contains the variable x. The new q' has a solution if and only if q has a solution.

It was already shown in [97] that resolution on a nest point will never increase the number of clauses. After conversion back to the CQ$^\neg$ setting this means that every relation will contain at most as many tuples as it did before the variable elimination. In combination with other standard properties of resolution one then arrives at the following statement.

Proposition 5.26 (Implicit in Lemma 16 in [23])**.** *Let q be the query*

$$\{\neg R_1(\vec{x_1}, y), \neg R_2(\vec{x_2}, y), \ldots, \neg R_n(\vec{x_n}, y)\}$$

on database D with domain $\{0, 1\}$ and let $\{y\}$ be a nest-set of q. There exists a query q' of the form $\neg R_1(\vec{x_1}), \neg R_2(\vec{x_2}), \ldots, \neg R_n(\vec{x_n})$ and a database D' with the following properties:

1. *If $a \in q(D)$, then $a[vars(q')] \in q'(D')$.*

2. *If* $a' \in q'(D')$ *then there exists* $c \in Dom$ *such that* $a' \cup \{y \mapsto c\} \in q(D)$.

3. q' *and* D' *can be computed in* $O(\|D\|)$ *time.*

4. *For every* $i \in [n]$ *we have* $|R_i^{D'}| \leq |R_i^D|$

It was discussed in the previous section that we can not, in general, reduce a query to an equivalent query with 2 element domain without increasing the nest-set width by a $\log |Dom|$ factor. However, as mentioned in the outline above, our plan is to temporarily transform certain subqueries in such a way that they become β-acyclic and that for any variable v that we want to eliminate, $\{v\}$ is a nest-set of the transformed subquery.

By the observation from Lemma 5.22, the reduction to a 2 element domain by binary encoding preserves β-acyclicity and allows us to eliminate the encoding variables of v one-by-one using Proposition 5.26. Afterwards, we can revert the binary encoding by mapping everything back into the original domain. This strategy allows us to lift Proposition 5.26 to a much more general form, allowing for variable elimination in *arbitrarily large* domains. This will ultimately allow us to circumvent the obstacles described in Section 5.4.1.

Lemma 5.27. *Let* q *be the query*

$$\{\neg R_1(\vec{x_1}, y), \neg R_2(\vec{x_2}, y), \ldots, \neg R_n(\vec{x_n}, y)\}$$

on database D *with* $|Dom| = 2^k$ *and let* $\{y\}$ *be a nest-set of* q. *There exists query* q' *of the form* $\neg R_1(\vec{x_1}), \neg R_2(\vec{x_2}), \ldots, \neg R_n(\vec{x_n})$ *and a database* D' *with the following properties:*

1. *If* $a \in q(D)$, *then* $a[vars(q')] \in q'(D')$.

2. *If* $a' \in q'(D')$ *then there exists* $c \in Dom$ *such that* $a' \cup \{y \mapsto c\} \in q(D)$.

3. q' *and* D' *can be computed in* $O(\|D\| \log^2 |Dom|)$ *time given* q *and* D *as input.*

4. *For every* $i \in [n]$ *we have* $|R_i^{D'}| \leq |R_i^D|$.

Proof. Since we have $|Dom| = 2^k$ there exists a bijection $f : Dom \to \{0,1\}^k$ that can be efficiently computed. We then consider the binary version q_b of q where every variable x is substituted by variables $x_1, x_2, \ldots, x_k$ and the respective database D_b where every tuple $(a_1, \ldots, a_{ar(R)}) \in R^D$ becomes a $(f(a_1), \ldots, f(a_{ar(R)})) \in R^{D_b}$. We thus clearly have that $a \in q(D)$ if and only if $f(a) \in q_b(D_b)$.

We now proceed to eliminating the variables $y_1, \ldots, y_k$ that encode the variable y in q_b. Observe that for every $i \in [k]$ we have that $\{y_i\}$ is a nest-set of q_b. Using Proposition 5.26 we can then successively remove all the $\{y_i\}$ successively, each elimination requiring $O(\|D_b\|)$ time. Recall from Lemma 5.4 that nest-sets are preserved when vertices are deleted from the hypergraph. Let q'_b and D'_b be the result of eliminating y_i for every $i \in [k]$ in this fashion.

Since exactly the substitution of y was deleted we can then clearly reverse the transformation from before and create a q' of the form

$$\neg R_1(\vec{x_1}), \neg R_2(\vec{x_2}), \ldots, \neg R_n(\vec{x_n})$$

from q'_b, as well as the corresponding database D' from D'_b. Again, clearly $a \in q'(D')$ if and only if $f(a) \in q'_b(D'_b)$.

Now, if $a \in q(D)$, then $f(a) \in q_b(D_b)$. By Proposition 5.26 we can then observe that $f(a)[vars(q'_b)] \in q'_b(D'_b)$ and in turn also $f^{-1}(f(a)[vars(q'_b)]) = a[vars(q')] \in q'(D')$. For the other direction, we proceed similarly, if $a' \in q'(D')$ then also $f(a') \in q'_b(D'_b)$. Again, by Proposition 5.26 this can be extended to an assignment $a_b \in q_b(D_b)$ and thus also implicitly to a $f^{-1}(a_b) \in q(D)$. Since $f(a')$ is extended by assignments for y_1 through y_k it follows that $f^{-1}(a_b)$ extends a' by some assignment for y.

Finally, the transformations to binary form and back are simple rewritings and can be done in linear time. The elimination of the y_i variables requires $O(k \|D\|)$ time and we have $k = \log_2(|Dom|)$. □

5.4.3 The Elimination Procedure

We are now ready to define our algorithm for eliminating nest-sets from CQs with negation. Our procedure for eliminating the variables of a nest-set s from a

Algorithm 5.2: Eliminate nest-set s from q, D.

Function `Elim-s-Positive` (q)
  Let $P_1, \ldots, P_n$ be the positive literals incident to s in q;
  Let $\neg R_1, \ldots, \neg R_m$ be the negative literals incident to s where $vars(R_j) \subseteq \bigcup_{i=1}^{n} vars(P_i)$;
  $q_J \leftarrow \{P_1, \ldots, P_n, \neg R_1, \ldots, \neg R_m\}$;
  $J \leftarrow$ all solutions of $q_J(D)$;
  $P \leftarrow \pi_{vars(J) \setminus s}(J)$;
  $P_s \leftarrow$ the s-extension of P;
  $C \leftarrow P_s \setminus J$;
  $q_1 \leftarrow (q \setminus q_J) \cup \{P, \neg C\}$;
  return q_1
Function `Elim-s-Negative` (q_1)
  Let $\neg C, \neg N_1, \ldots, \neg N_\ell$ be the literals incident to s in q_1;
  foreach $i \in \ell$ **do**
    $N_i' \leftarrow$ the s-extension of N_i;
  end
  $q^\neg \leftarrow \{\neg C, \neg N_1', \ldots, \neg N_\ell'\}$;
  Let $q^* = \{\neg C^*, \neg N_1^*, \ldots, \neg N_\ell^*\}$ be the query obtained by successively eliminating every $v \in s$ from $q^\neg$ using Lemma 5.27;
  $q_{-s} \leftarrow (q_1 \setminus q^\neg) \cup \{\neg N_1^*, \ldots, \neg N_\ell^*\}$;
  Update relation P to $P - C^*$ in D^{-s};
  return q_{-s}
begin `/* Main */`
  $q_1 \leftarrow$ `Elim-s-Positive` (q);
  return `Elim-s-Negative` (q_1);
end

`BoolCQ`$^\neg$ instance q, D is described in Algorithm 5.2. Updates to the database are implicit in the algorithm. This is to be understood as adding the corresponding relation for every literal that is added to the query and removing the relations for the deleted literals. We refer to the new instance q', D' returned by the algorithm as the *s-elimination* of q, D.

The procedure begins by eliminating all positive occurrences of s via the function `Elim-s-Positive`. To do so, it considers the subquery q_J, which contains all the positive literals incident to s as well as those negative literals that

are fully covered (w.r.t. their variable scopes) by these positive literals. It is straightforward to compute all the solutions of q_J by first joining all the positive literals and then incorporating the negative literals via anti-joins. This can be done efficiently since the variables of q_J can be covered by at most k positive literals by a similar argument as in the proof of Lemma 5.11. The set of solutions of q_J is taken as a new relation J, to which we apply the mechanism from Lemma 5.25. The resulting literals P and $\neg C$ replace the subquery q_J in q to form q_1.

The resulting query q_1 thus is equivalent to q and has no variable in s occurring in a positive literal. The only literals incident to s that are left are those negative literals that contain variables beyond those in q_J, and the new literal $\neg C$. The second subprocedure, `Elim-s-Negative`, eliminates the variables in s from these negative literals using Lemma 5.27. To eliminate a variable $v \in s$ with Lemma 5.27 we need a β-acyclic subquery where v is a nest-point. We therefore do not consider the literals $\neg N_i$ directly but instead operate on their s-extensions. Observe that the literals $\neg N_i$ are all incident to s and therefore their variables are linearly ordered under $\subseteq_s$. Furthermore, for every $i \in [\ell]$ we have $vars(N_i) \setminus s \supseteq vars(C) \setminus s$. Thus, for the s-extensions N_i' of N_i we have

$$vars(C) \subseteq vars(N_{i_1}') \subseteq vars(N_{i_2}') \subseteq \cdots \subseteq vars(N_{i_\ell}')$$

Therefore, $q^\neg$ on line 16 in the algorithm is clearly β-acyclic and all variables in s are present in every literal. Thus also every variable of s is a nest-point of $q^\neg$ and Lemma 5.27 can be used to eliminate all of them. After the elimination we get the new set of literals q^* which we replace $q^\neg$ with in q_1. The literal $\neg C^*$ always has the same set of variables as the P that was introduced in `Elim-s-Positive`. We thus can simply account for $\neg C^*$ by subtracting the relation of C^* from the relation of P instead of adding $\neg C^*$ as a literal. This way we avoid the possibility of increasing the number of new literals in the resulting final query q_{-s}.

Example 5.4. Consider a query q with nest-set $s = \{a, b, c\}$. The query has literals $P_1(a, b, c)$, $P_2(b, d)$, $\neg N_1(a, d, e, f, g)$, and $\neg N_2(c, d, e)$ incident to s. This setting is illustrated on hypergraph level in Figure 5.4 where the components C_1 and C_2 abstractly represent the rest of the query.

Algorithm 5.2 first computes the intermediate relation J containing all the solutions for $P_1(a, b, c) \wedge P_2(b, d)$. We remove the variables in s from J by

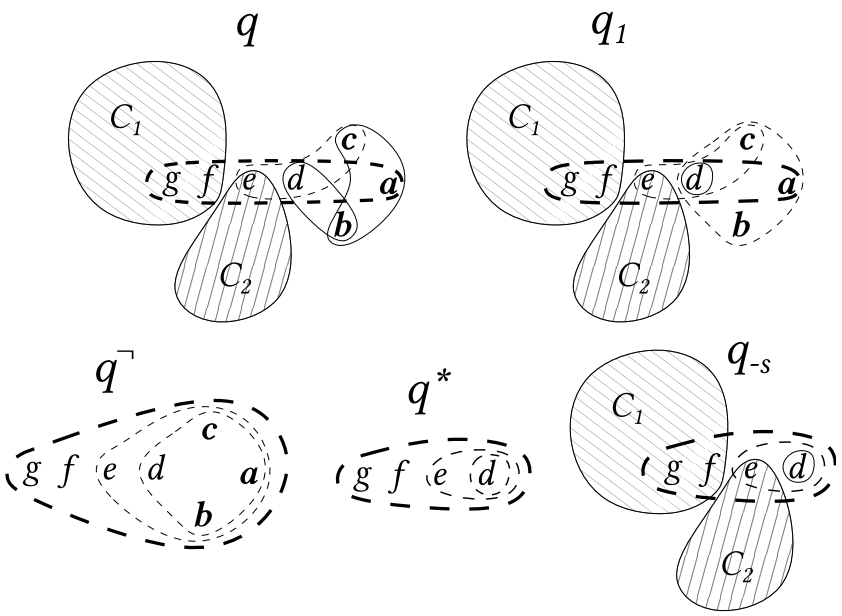


Figure 5.4: Example of an s-elimination on hypergraph level. (Dashed edges correspond to negative literals.)

projection to obtain the new literal $P(d)$. We furthermore add $\neg C(a, b, c, d)$ as in Lemma 5.25 to make q_1 equivalent to q. Note that variables from s now occur only in negative literals of q_1.

The procedure then moves on to eliminating s from the negative literals. First all the negative literals are expanded to cover all variables of s. The expanded negative literals $\neg N_1'(a, d, e, f, g, b, c)$, $\neg N_2'(c, d, e, a, b)$, $\neg C(a, b, c, d)$ make up the subquery $q^\neg$. As discussed above, this expansion modifies the hypergraph structure in such a way that all the variables of s now correspond to 1-nest-sets of $q^\neg$ (see also Figure 5.4). They can therefore be eliminated using Lemma 5.27 to obtain q^*.

Finally, replacing $q^\neg$ in q_1 by q^* (and simplifying $\neg C^*(d) \wedge P(d)$) will produce the final query without variables from s, the s-elimination of q. △

What is left is to prove that an s-elimination has a solution if and only if the original query has a solution. This follows from the combination of the observations in Section 5.4.2. Moreover, the s-elimination is always smaller

than the original query and it can be computed in polynomial time when the size of s is bounded. From these three properties it will then be straightforward to establish our main result, the tractability of $\texttt{BoolCQ}^\neg$ under bounded *nsw*. Full proofs are available in the appendix.

Lemma 5.28. *Let q', D' be the s-elimination of some $\texttt{BoolCQ}^\neg$ instance q, D. Then $q'(D') \neq \emptyset$ if and only if $q(D) \neq \emptyset$.*

Proof. From Lemma 5.25 it follows that $P \wedge \neg C$ is equivalent to q_J. Therefore we have $q_1(D_1) = q(D)$.

Suppose $a \in q_1(D_1)$. Then, for any $i \in [\ell]$ we have that a satisfies $\neg N_i$. From Lemma 5.24 it follows that a also satisfies $\neg N_i'$. Hence, $a[vars(q^\neg)]$ satisfies $q^\neg$. By Lemma 5.27 it then follows that $a[vars(q^*)] \in q^*(D^*)$. Since the literals in $q^\neg$ were the only literals incident to s we have that $a[vars(q) \setminus s]$ satisfies all literals in q_{-s}. The update of $P^{D_{-s}}$ at the end is trivial since $vars(P) = vars(C^*)$.

For the other side of the equivalence now assume that $a' \in q'(D')$. Then a in particular satisfies all literals of q^*. By Lemma 5.27 there exists an assignment a_s to s such that $a = (a' \cup a_s)$ satisfies $q^\neg$. Now, observe that for every $j \in [\ell]$, if $a[vars(N_j')] \notin N_j'$ then also $a[vars(N_j)] \notin N_j$ by Lemma 5.24. All other literals remain the same between q' and q_1 and thus $a \in q_1(D) = q(D)$. □

Lemma 5.29. *Let q, D be an instance of $\texttt{BoolCQ}^\neg$ and let s be a k-nest-set of q. Then the s-elimination of q, D can be computed in $O(|R_{max}|^k |Dom|^k poly(\|q\| + \|D\|))$ time.*

Proof. We need two observations to prove the bound. First, q_J can be covered by k positive literals. Since s is a nest-set, one of the P_i is maximal (among the positive literals) with regards to $\subseteq_s$. Just like in Lemma 5.11 we are left with at most $k - 1$ variables of s that are not covered by this maximal positive literal. Hence, $|J| \leq O(|R_{max}|^k)$ and can be computed in the stated time bound naturally using joins and anti-joins. Second, computing the s-elimination can only add at most k variables to a literal, increasing the number of tuples by at most $|Dom|^k$. Time bounds for all other steps then follow in a straightforward fashion from these two observations and the time bound of Lemma 5.27.

We first argue that $vars(q_J)$ can be covered by at most k positive literals. Since s is a nest-set, one of the P_i is maximal (among the positive literals) with regards to $\subseteq_s$. Just like in Lemma 5.11 we are left with at most $k-1$ variables of s that are not covered by this maximal positive literal. As we assume safety, every such variable requires at most one positive literal to cover it and the claim holds. Thus, J will contain a subset of the tuples formed by a join of k positive literals, hence $|J| \leq O(|R_{max}|^k)$.

A join $A \bowtie B$ is feasible in $O(|A||B| \max\{ar(A), ar(B)\} \log |Dom|)$ time by a straightforward nested loop join. As $ar(R) \leq \|Q\|$ for any relation symbol R we simplify to $O(|R_{max}|^2 \|q\| \log |Dom|)$. As usual, once k positive literals that cover $vars(q_J)$ are joined, any further join is simply a semi-join and requires only linear time. It follows that the joins over all the positive literals of q_J can be computed in $O(|R_{max}|^k \|q\| \log |Dom|)$ time. The negative literals in q_J can then all be removed by anti-joins, which also require linear time, i.e., $A \triangleright B$ takes $O(\|A\| + \|B\|)$ time. At most $|q|$ anti-joins need to be performed and we see that we have an upper bound $O(|R_{max}|^k \|q\|^2 \log |Dom|)$ for the time required to compute J.

Computing P is clearly linear in $\|J\|$ while computing C is feasible in $O(\|J\| \cdot \|Dom^k\|)$ time. We use the fact that $ar(J) \leq \|q\|$ to simplify the final bound.

For every $i \in [\ell]$, the s-expansion from N_i to N_i' requires $O(\|R_{max}\| \cdot \|Dom^k\|)$ time. Finally, for each variable $v \in s$, eliminating v requires $O(\|D^\neg\| \log^2 |Dom|)$ time by Lemma 5.27 and is performed k times. As the s-extension can increase the relation size by a factor of at most $|Dom|^k$ we have $\|D^\neg\| \leq \|D\| \, |Dom|^k$. Note that the linear factor k is simplified away in the final bound by observing $k \leq vars(q)$. □

Lemma 5.30. *Let q', D' be the s-elimination of some* `BoolCQ`$^\neg$ *instance q, D. Then $\|q'\| \leq \|q\|$ and $\|D'\| \leq \|D\|$.*

Proof. For the query, we remove n positive literals. Observe that $n \geq 1$ because we assume q is safe and $s \neq \emptyset$. We only add one new positive literal P. Hence, the number of positive literals can not increase through s-elimination. Every new negative literal in q' corresponds one-to-one to a negative literal that was

removed from q. We thus have less or equal literals and strictly less variables in q' than in q.

For the database observe that the variables of the literals $P_1, \ldots, P_n$ can be ordered as follows

$$vars(P_{i_1}) \setminus s \subseteq vars(P_{i_2}) \setminus s \subseteq \cdots \subseteq vars(P_{i_n}) \setminus s$$

By construction we have that $vars(P) = \left(\bigcup_{i=1}^{n} vars(P_i)\right) \setminus s$. As the union over a chain of subsets is simply the maximal element of the chain we have $vars(P) = vars(P_{i_n}) \setminus s$. It is then easy to see from the construction that $|P^{D'}| \leq |P^D_{i_n}|$. For the new negative literals we have $|N_i^*| \leq |N_i|$ by Lemma 5.27. Since no arities can increase in the s-elimination we arrive at $\|D'\| \leq \|D\|$. □

Finally, note that the construction of the s-elimination preserves the simplifying assumptions made in the beginning of the section. The domain is never modified by the procedure and if q was safe, then so is its s-elimination q'. Moreover, we also have $H(q') = H(q) - s$ and can therefore repeatedly apply this elimination along a k-NEO to decide whether $q(D) \neq \emptyset$.

Proof of Theorem 5.20. Let $\mathcal{Q}$ be a class of BoolCQ$^\neg$ instances and say there exists a constant k such that the nest-set width of every query in $\mathcal{Q}$ is at most k.

Let q, D be an instance of BoolCQ$^\neg$ with $nsw(q) \leq k$. First, we compute a k-NEO $\mathcal{O} = (s_1, \ldots, s_\ell)$ which is feasible in polynomial time for constant k by Corollary 5.19. Then perform the following procedure that starts with $q_0 := q, D_0 := D, i := 1$:

1. If $i > \ell$, accept the input. Otherwise continue with the next step.
2. Let q_i, D_i be the s_i-elimination of q_{i-1}, D_{i-1}. Rename the new P literal to P_i.
3. If $P_i^{D_i} = \emptyset$, reject the input. Otherwise increment i by 1 and continue from step 1.

In case of acceptance, the procedure has eliminated all variables and only the 0-ary literal P_ℓ is left in q_ℓ. Since the procedure did not reject in the step before,

we have $P_\ell^{D_\ell} \neq \emptyset$, i.e., it contains the empty tuple and thus $q_\ell(D_\ell) = \{()\} \neq \emptyset$. On the other hand, if the procedure rejects at step i, then $P_i^{D_i} = \emptyset$. The literal P_i occurs positively in q_i and it follows that $q_i(D_i) = \emptyset$.

By Lemma 5.28 we have $q_i(D_i) \neq \emptyset$ if and only if $q(D) \neq \emptyset$ for all $i \in [\ell]$. The described procedure is therefore sound and complete. The computation of q_i, D_i from q_{i-1}, D_{i-1} is performed at most $\ell \leq vars(q)$ times. By Lemmas 5.30 and 5.29 the procedure requires only polynomial time in $\|q\|$ and $\|D\|$. □

5.4.4 An Application: SAT Parameterized by Nest-Set Width

Note that the nest-set width appears only in the exponent of $|R_{max}|$ and $|Dom|$ in the time bound from Lemma 5.29. A reduction to $\texttt{BoolCQ}^\neg$ where these two cardinalities can be constantly bounded thus shows fixed-parameter tractability of the original problem when parameterized by *nsw*.

We show in the following that this implied fixed-parameter tractability of SAT when parameterized by *nsw*. In order for formulas to relate directly to hypergraphs we consider only propositional formulas in conjunctive normal form (CNF). The hypergraph $H(F)$ of a formula F has as its vertices the variables of F and edges $E(H(F)) = \{vars(C) \mid C \text{ clause in } F\}$. We then alternatively refer to $nsw(H(F))$ as the nest-set width $nsw(F)$ of the propositional CNF formula F.

Recall the reduction from SAT to $\texttt{BoolCQ}^\neg$ given in the beginning of Section 5.4. For a formula F in CNF consisting of clauses $C_1, \ldots, C_n$. For every clause C_i with variables $v_1, \ldots, v_\ell$ we create a literal $\neg R_i(v_1, \ldots, v_\ell)$ where the relation R_i^D contains the single tuple corresponding to the only assignment that does not satisfy the clause. To satisfy our assumption of safety we also create a positive literal $V_j(v_j)$ for every variable v_j with $V_j^D = \{(0), (1)\}$, i.e., the whole domain.

We see that this reduction produces a $\texttt{BoolCQ}^\neg$ instance with $|R_{max}| = 2$, and $|Dom| = 2$ while $\|q\|$ and $\|D\|$ are linear in the size of F. Plugging these values into the bound from Lemma 5.29 gives us a $2^{O(k)}poly(|F|)$ time bound for an s-elimination in this query, where $k = nsw(H(F))$ is the *nest-set width of the formula* F. Hence, repeated s-elimination along a k-NEO gives us a fixed-parameter tractable procedure for SAT.

Theorem 5.31. *SAT for propositional CNF formulas is fixed-parameter tractable when parameterized by the nest-set width of the formula.*

Interestingly, the standard Davis-Putnam resolution procedure for SAT is also fixed-parameter polynomial when parameterized by nest-set width if resolution is performed according to a NEO. The resulting algorithm is notably different from the one resulting from the above reduction and requires some further auxiliary results. A full proof of the claim is given in Appendix 5.5. We consider this further evidence that *nsw* is a natural generalization of β-acyclicity with interesting algorithmic consequences.

Building on the discussions from Section 5.2 it is also interesting to note that SAT is known to be W[1]-hard when the problem is parameterized by incidence clique width [97].

5.5 Nest-Set Width & Davis-Putnam Resolution

In the previous section we showed how our CQ$^\neg$ evaluation result also implies the fixed-parameter tractability of SAT when parameterized by *nsw*. In this section we show that the same parameterized tractability result for SAT can also be reached via Davis-Putnam (DP) resolution [42], the method proposed by Ordyniak, Paulusma, and Szeider for β-acyclic formulas [97].

Building on the approach used in [97], we show that DP resolution on a nest-set will always decrease the number of clauses in the formula. While the number of clauses can increase in the intermediate steps, before resolution on every variable from the nest-set has been performed, the intermediate blowup can be bounded in the size of the nest-set.

5.5.1 Resolution

For this section we consider a *clause* C as a set of literals x or $\overline{x}$ where x is a variable. If a literal is of the form x we say it is positive and otherwise it is negative. This is also referred to as the *phase* of the literal. If C is a clause, we write $\overline{C}$ for the clause $\{\overline{\ell} \mid \ell \in C\}$, i.e., every clause is switched (note that $\overline{\overline{x}} = x$). A formula F (in CNF) is a set of clauses.

For two clauses $C, D \in F$ with $C \cap \overline{D} = \{x\}$ we call $(C \cup D) \setminus \{x, \overline{x}\}$ the *x-resolvent* of C and D. Note that we don't need to care about cases where $C \cap \overline{D} \supset \{x\}$. Say the intersection equals $\{x, y\}$. Resolving on x would yield a new clause containing both y and $\overline{y}$. Such a new clause is therefore trivially satisfied and of no interest.

Let $DP_x(F)$ be the formula obtained by first adding all x-resolvents to F and then removing all clauses where x occurs. Such a step is commonly called a *(Davis-Putnam) resolution step* [42]. It is well-known that F and $DP_x(F)$ are equisatisfiable for all variables x.

We write $DP_{x_1,\dots,x_q}(F)$ for $DP_{x_q}(DP_{x_{q-1}}(\cdots DP_{x_1}(F)\cdots))$. We also write DP_s for a set $s \subseteq vars(F)$ if the particular order does not matter. The procedure $DP_{vars(F)}(F)$ will produce either an empty clause at some step or end in an empty formula. In the first case, F is unsatisfiable, and conversely, F is satisfiable in the second case.

The hypergraph $H(F)$ of a formula F has vertex set $vars(F)$ and a set of edges $\{vars(C) \mid C \in F\}$. For a set of variables $s = \{v_1, \dots, v_q\}$ we define $\overline{s} := \{\overline{v_1}, \dots, \overline{v_q}\}$. Let C be a clause and s a set of variables. We write $C - s$ for $C \setminus (s \cup \overline{s})$, the clause with all literals of variables from s removed. We extend this notation to formulas as $F - s := \{C - s \mid C \in F\}$ for formula F.

5.5.2 Davis-Putnam Resolution over Nest-Sets

In general, $DP_x(F)$ can contain more clauses than F and the whole procedure can therefore require exponential time (and space). However, as we will see, if we eliminate nest-sets then the increase in clauses is only temporary. We start by showing that resolution on a variable in a nest-set will only produce resolvents that remain, in a sense, local to the nest-set.

Lemma 5.32. *Let $s = \{v_1, \dots, v_q\}$ be a nest-set of $H(F)$. For every $s' \subseteq s$ and any clauses $C, D \in DP_{s'}(F)$ that contain a variable from s we have that $vars(C) \subseteq_s vars(D)$, or vice versa.*

Proof. Proof is by induction on the cardinality of s'. If $s' = \emptyset$ no resolution takes place and the statement follows from the fact that s is a nest-set.

Say $s' = \{v_1, \ldots, v_k\}$ and consider $C, D \in DP_{v_1,\ldots,v_{k-1}}(F)$ such that both clauses contain a variable from s. According to the induction hypothesis, w.l.o.g., we have $vars(C) \subseteq_s vars(D)$ (the other case is symmetric). To show that the statement also holds for s' we show that each v_k-resolvent of $DP_{v_1,\ldots,v_{k-1}}(F)$ also satisfies this property.

In particular, for such C, D and $C \cap \overline{D} = \{v_k\}$ we show that $vars(R) \setminus s = vars(D) \setminus s$, where R is the v_k-resolvent of C and D. Since $vars(D) \setminus r$ is comparable to all other clauses that contain a variable of s, so is $vars(R) \setminus s$ and the statement also holds for $DP_{s'}(F)$.

It is not hard to see that the two sets are in fact the same: Since $R = (C \cup D) \setminus \{v_k, \overline{v_k}\}$ and $v_k \in s$ we also have $var(R) \setminus s = (var(C) \cup var(D)) \setminus s$. Recall that we have $(var(C) \setminus s) \subseteq (var(D) \setminus s)$ and therefore $(var(C) \cup var(D)) \setminus s = var(D) \setminus s$. Note that the argument doesn't change if $D \cap \overline{C} = \{v_k\}$. □

Lemma 5.33. *Let $s = \{v_1, \ldots, v_q\}$ be a nest-set of $H(F)$. For every $s' \subseteq s$ and any clause $C \in DP_{s'}(F)$ we have that $C - s \in F - s$.*

Proof. Proof is by induction on the cardinality of s'. For $s' = \emptyset$ the statement is true by definition.

Suppose the statement holds for $|s'| < k$ and let $s' = \{v_1, \ldots, v_k\}$. Consider a $C \in DP_{s'}(F) \setminus DP_{s' \setminus \{v_k\}}(F)$, i.e., a new clause obtained by the resolution on v_k after resolution on all the other variables of s' was already performed. Thus, $C = (C_1 \cup C_2) \setminus \{v_k, \overline{v_k}\}$ for some $C_1, C_2 \in DP_{s' \setminus \{v_k\}}(F)$ where $C_1 \cap \overline{C_2} = \{v_k\}$. By Lemma 5.32 we have, w.l.o.g., $vars(C_1) \setminus s \subseteq vars(C_2) \setminus s$ (the other case is symmetric).

Now, for every variable $v \in (vars(C_1) \cap vars(C_2)) \setminus s$, we know that v occurs in the same phase in both clauses since $C_1 \cap \overline{C_2} = \{v_k\}$. Thus, also $C_1 - s \subseteq C_2 - s$ and therefore also $C - s = C_2 - s$. By the induction hypothesis we have $C_2 - s \in F - s$ and the proof is complete. □

While Lemma 5.33 has $DP_s(F) \leq |F|$ as a direct consequence, it also gives insight into the size of the intermediate formulas. In particular, for any non-empty $s' \subseteq s$ we have $|DP_{s'}(F)| \leq 3^{k-|s'|}|F|$. This can be observed from noting that any

clause $C \in DP_{s'}(F)$ is an extension of a clause from $F - s$ by any combination of literals for the variables $s \setminus s'$. Specifically, there are three possibilities for every such variable, it either occurs positively, negatively or not at all in C. Thus, any clause in $F - s$ has only $3^{|s \setminus s'|} = 3^{k-|s'|}$ extensions.

From Section 5.3.2 we know that we can compute a k-NEO of $H(F)$ in fixed-parameter polynomial time when parameterized by k. From the size bound above it is then easy to see that each resolution step in a nest-set can be performed in fixed-parameter polynomial time. Hence, repeating the resolution step along a k-NEO is an fixed-parameter tractable procedure for SAT parameterized by k.

Theorem 5.34. *SAT for propositional CNF formulas is fixed-parameter tractable when parameterized by the nest-set width of the formula.*

5.6 Summary

In this chapter, we introduced a new generalization of hypergraph β-acyclicity which we call *nest-set width*. Our aim was to emulate the successfully generalizations of α-acyclicity by finding a novel measure of structural complexity that relates to the complexity of important problems and which can be recognized efficiently.

We argue that nest-set width fulfills these goals. We show that two fundamental algorithmic problems in logic – propositional satisfiability and $CQ^{\neg}$ evaluation (i.e., model checking for existential first-order logic) – indeed become tractable under bounded nest-set width. Recall that both of these problems remain NP-hard even when their underlying hypergraphs are α-acyclic.

We show that bounded nest-set width is a distinct restriction from other widths in the literature, be it graph widths like treewidth or clique-width, or previous generalizations of β-acyclicity like β-hw. In particular, we show that bounded nsw is a specialization of bounded β-hw.

CHAPTER 6

Conclusion

In this chapter we briefly summarize our main contributions one final time, discuss how they progress the field, and finally discuss possibilities for further research.

We began our investigation in Chapter 3 with the study of the semantic width of conjunctive queries. We first showed that the semantic variant of many important widths is equal to the width of the core. Importantly, this also leads to an upper bound for the complexity of checking such semantic widths (which were otherwise not even known to be decidable). These new insights into semantic width then enabled us to derive (parameterized) tractability results based on bounded semantic widths. In the parameterized case we also show that this is as good it gets by demonstrating that bounded semantic submodular width characterizes the fixed-parameter tractability of CQs and UCQs.

With these results we have fully answered the question of when CQs and UCQs are fixed-parameter tractable. We believe that beyond their theoretical importance, our characterization results can be of great use in the context of CSPs. In the right setting they can arguably significantly simplify the determination of

the parameterized complexity of new CSP formulations. In particular, showing that a CSP formulation of some problem is not fixed-parameter tractable can be greatly simplified by showing unbounded sem-$subw$ instead of finding a reduction. Additionally, we also make some progress with respect to finding the exact border of plain tractability of CQs, the main remaining open problem in this area.

In the following Chapter 4, we shift our focus to the tractability of checking ghw and fhw. We show that while Check(ghw, k) is NP-hard, the problem is in fact highly parallelizable (it is in the complexity class LogCFL), improving on a result by Fischl, Gottlob, and Pichler [54]. We then show how Check(fhw, k) can be reduced to Check(ghw, k) for hypergraphs with bounded (multi-)intersection width, thus demonstrating its tractability under those restrictions.

We see the work in this chapter an important bridge between theory and practice. While the tractability proofs may seem abstract, the underlying combinatorial ideas are important for the practical computation of GHDs and FHDs. Recent work by Fischl et al. [53] as well as Gottlob, Okulmus, and Pichler [70] has already demonstrated this approach with great success. Moreover, we make some first steps towards the yet largely unstudied topic of width checking of arbitrary f-widths. Finally, our study of fhw width checking has yielded a number of novel techniques and results in fractional hypergraph theory that may be of independent interest.

For our third main theme, we moved on to the study of β-acyclicity. This kind of acyclicity has received significantly less attention than α-acyclicity up to now despite appearing to be the natural hypergraph acyclicity notion for important problems such as SAT and CQ$^\neg$ answering. Our aim is to apply the fruitful ideas from research on α-acyclicity also to β-acyclicity. To this end we introduce nest-set width which, analogous to hypertree width, is the most general know generalization of β-acyclicity that is recognizable in polynomial time. Moreover, we show that nest-set width is an effective width notion by showing that bounded nest-set width induces tractable fragments of CQ$^\neg$ answering and SAT.

We believe that our work on nest-set width opens up the study of the effect of hypergraph structure for new problems, similar to how generalizations of

α-acyclicity have led to deep insight into the complexity of CQ answering. Since this applies to a number of problems that are considered distinct at the time, e.g., SAT, CQ$^\neg$ answering, and polynomial optimization, it may also reveal some deeper yet unknown connection between these problems.

6.1 Outlook

To fully conclude this thesis we list some opportunities for further research that builds on the progress made in this thesis.

As discussed in the previous section, we believe that our characterization theorems from sections 3.4 and 3.5 have the potential to simplify hardness proofs for CSP formulations of algorithmic problems. Instead of finding a reduction from a problem that is not fixed-parameter tractable, we now have the alternative option of showing that the class of CSPs has unbounded sem-*subw*. This motivates further work on theoretical tools that help to show whether a class has bounded sem-*subw*. We believe that further study of adaptive width which is bounded if and only if *subw* is bounded [92] but can be defined over only modular functions, can be a productive avenue of research here.

Moreover, we are intrigued by the connections to Vapnik-Chervonenkis dimension. VC-dimension is an important parameter in learnability theory, particularly in binary classification, [21]. To our knowledge this connection between learnability theory and CQ answering is largely unexplored. We are particularly interested in how the VC-dimension of the instance hypergraph relates to the learnability of parameters defined on the hypergraph. Going even further we are interested in how the connection to learnability relates to our claim of the unnaturalness of queries with unbounded VC-dimension.

Recent work has proposed *join-width*, a hybrid width parameter, which also depends on the database, for the study of the computational complexity of CQs [56]. Investigation of a semantic variant of join-width is a natural further avenue of research.

In the context of our tractability result for Check(fhw, k) under bounded multi-intersection width we are again interested in bounded VC-dimension. It is known that bounded VC-dimension is a more general restriction than

bounded multi-intersection width [54], and therefore the next natural candidate to consider in the search for tractable fragments. However, a preliminary investigation suggests that, if at all possible, this extension to bounded VC-dimension will require a completely new proof strategy.

Our results for Check(fhw, k) checking also motivate a more practically motivated question. Can we further lower our complexity bounds to match the ghw case? We have shown that, under the BMIP, Check(ghw, k) is in LogCFL and thus well suited for parallelization. For practical algorithms the possibility of parallelization is enticing and we are interested in whether our tractability results for Check(fhw, k) can also be strengthened to LogCFL membership. In the fhw case the main blocker here is the complexity of checking whether a set can be fractionally covered with weight k. More sophisticated methods than linear programming will be necessary to perform this check more efficiently.

With respect to nest-set width and the results from Chapter 3 the possibilities for future work are plentiful. Our results make us hopeful that nsw can find broader application beyond the problems discussed in this thesis. We believe that most algorithmic results for β-acyclic instances of problems can be generalized to also hold under bounded nest-set width. This includes, the tractability results for #SAT and Max-CSP by Brault-Baron, Capelli and Mengel [25], the worst-case bounds for β-acyclic CQ evaluation by Ngo et al. [95], and a very recent tractability result for polynomial optimization by Del Pia and Di Gregorio[100].

An interesting question has been left open in Section 5.2: the relationship between nest-set width and point-width. Since the tractable computation of point-decompositions remains an open question, the applicability of point-width for algorithmic results in our setting is not clearly established. However, we believe that the question of whether point-width generalizes nest-set width (or β-hw) is an important open problem for the overall program of β-acyclicity generalizations. Proving that point-width generalizes β-acyclicity [29] already requires considerable effort and it is therefore likely that showing the relationship to nsw will be more challenging and requires individual study.

Finally, with our two new islands of tractability for SAT and $CQ^{\neg}$, there comes a question of whether the result can be generalized further or if this is the limit of tractability for the problem. Both kinds of answers would be of great interest for

$CQ^{\neg}$ evaluation as well as SAT. With respect to our parameterized tractability result for SAT it may also be of interest further investigate the relationship to clique-width since SAT is known to be W[1]-hard when parameterized by incidence clique-width [97].

Glossary

bottom-up Bottom-up commonly refers to an algorithmic strategy where one first solves small subproblems and then solves the larger subproblems on the basis of the previously solved smaller subprolems..

clique A clique is a graph where every vertex has an edge to every other vertex. Also called a *complete graph*..

conservative width functions A conservative width function $f\colon 2^{V(H)} \mapsto \mathbb{R}^+$ is monotone and it is possible to check in polynomial time whether $f(X) \leq k$ for any $X \subseteq 2^{V(H)}$ and a constant k..

degree The degree of a vertex is the number of incident edges. The degree of a hypergraph H is the maximal degree of its vertices..

edge weight function A function $\gamma\colon E(H) \mapsto \mathbb{R}^+$ that maps every edge of a hypergraph H to a *weight*. Also exists in the form of *integral* edge weight function when the co-domain is $\{0, 1\}$..

Exponential Time Hypothesis The Exponential Time Hypothesis, which states that 3-SAT with n variables can not be decided in $2^{o(n)}$ time. For further details, see [79]..

fractional cover number The fractional cover number $\rho^*(H)$ of a hypergraph H is the minimal weight of a fractional edge cover of H..

graph A graph is a mathematical structure cosisting of a set of vertices and a set of edges, where an edge is a connection from one vertex to another. Alternatively, a graph is a hypergraph where every hyperedge has cardinality 2..

hereditary We call a hypergraph property P hereditary if the following holds: If H has property P, then every subhypergraph of H also has property P..

hypergraph Hypergraphs are a generalization of graphs where the edges are arbitrarily sized sets of vertices (instead of being limited to edges containing exactly two vertices)..

leaf The leaf of a tree is a vertex with degree 1..

multi-intersection width The c-multi-intersection width of a hypergraph H is the maximal cardinality of an intersection of c *distinct* edges..

rank The rank of a hypergraph H is the maximal cardinality of any edge in the hypergraph, i.e., $\max\{|e| \mid e \in E(H)\}$..

reduced hypergraph A hypergraph is said to be reduced if the following conditions hold: (1) there are no isolated vertices, (2) there are no empty edges, and (3) no two edges are contained in the exact same set of edges..

semantic width The semantic width (for some width notion) of a CQ (or UCQ) q is the minimal width of all queries that are semantically equivalent to q..

support The support of an edge weight function is the set of edges that are assigned non-zero weight..

tree A tree is a connected acyclic graph..

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 *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco*, pages 431–446, 2016.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[4] Isolde Adler. Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory*, 47(4):275–296, 2004.

[5] Isolde Adler. *Width functions for hypertree decompositions*. PhD thesis, Albert-Ludwig Universität Freiburg, 2006.

[6] Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007.

[7] Foto N. Afrati, Manas R. Joglekar, Christopher Ré, Semih Salihoglu, and Jeffrey D. Ullman. GYM: A multiround distributed join algorithm. In

Proceedings of the 20th International Conference on Database Theory, ICDT 2017, Venice, pages 4:1–4:18, 2017.

[8] Kamal Amroun, Zineb Habbas, and Wassila Aggoune-Mtalaa. A compressed generalized hypertree decomposition-based solving technique for non-binary constraint satisfaction problems. *AI Comm.*, 29(2):371–392, 2016.

[9] 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 *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne*, pages 1371–1382, 2015.

[10] Patrick Assouad. Densité et dimension. *Annales de l'Institut Fourier*, 33(3):233–282, 1983.

[11] Albert Atserias, Anuj Dawar, and Phokion G. Kolaitis. On preservation under homomorphisms and unions of conjunctive queries. *J. ACM*, 53(2):208–237, 2006.

[12] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013.

[13] Albert Atserias and Phokion G. Kolaitis. Consistency, acyclicity, and positive semirings. *CoRR*, abs/2009.09488, 2020.

[14] Pablo Barceló, Victor Dalmau, Cristina Feier, Carsten Lutz, and Andreas Pieris. The limits of efficiency for open- and closed-world query evaluation under guarded tgds. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland*, pages 259–270. ACM, 2020.

[15] Pablo Barceló, Cristina Feier, Carsten Lutz, and Andreas Pieris. When is ontology-mediated querying efficient? In *Proceedings of the 34th ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver*, pages 1–13. IEEE, 2019.

[16] Pablo Barceló, Georg Gottlob, and Andreas Pieris. Semantic acyclicity under constraints. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA*, pages 343–354, 2016.

[17] Pablo Barceló, Andreas Pieris, and Miguel Romero. Semantic optimization in tractable classes of conjunctive queries. *SIGMOD Record*, 46(2):5–17, 2017.

[18] Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. Semantic acyclicity on graph databases. *SIAM J. Comput.*, 45(4):1339–1376, 2016.

[19] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[20] Christoph Berkholz and Nicole Schweikardt. Constant delay enumeration with fpt-preprocessing for conjunctive queries of bounded submodular width. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, Aachen*, pages 58:1–58:15, 2019.

[21] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[22] Hans L. Bodlaender and Babette de Fluiter. Reduction algorithms for constructing solutions in graphs with small treewidth. In *Proceedings of the Second Annual International Computing and Combinatorics Conference, COCOON 1996, Hong Kong*.

[23] Johann Brault-Baron. A negative conjunctive query is easy if and only if it is beta-acyclic. In *Proceedings of the 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, Fontainebleau*, pages 137–151, 2012.

[24] Johann Brault-Baron. Hypergraph acyclicity revisited. *ACM Comput. Surv.*, 49(3):54:1–54:26, 2016.

[25] Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic cnf-formulas. In *Proceedings of the*

32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, Garching, pages 143–156. Schloss Dagstuhl, 2015.

[26] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, Dec 1995.

[27] Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA*, pages 319–330, 2017.

[28] Florent Capelli, Arnaud Durand, and Stefan Mengel. Hypergraph acyclicity and propositional model counting. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing, SAT 2014, Vienna*, pages 399–414. Springer, 2014.

[29] Clément Carbonnel, Miguel Romero, and Stanislav Zivny. Point-width and max-csps. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Saarbrücken*, pages 1–13. ACM, 2019.

[30] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[31] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, Boulder, CO*, pages 77–90, 1977.

[32] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

[33] Hubie Chen. On the complexity of existential positive queries. *ACM Trans. Comput. Log.*, 15(1):9:1–9:20, 2014.

[34] Hubie Chen and Víctor Dalmau. Beyond hypertree width: Decomposition methods without decompositions. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP 2005*, pages 167–181, 2005.

[35] Hubie Chen, Georg Gottlob, Matthias Lanzinger, and Reinhard Pichler. Semantic width and the fixed-parameter tractability of constraint satisfaction problems. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1726–1733, 2020.

[36] Hubie Chen and Moritz Müller. The fine classification of conjunctive queries and parameterized logarithmic space. *TOCT*, 7(2):7:1–7:27, 2015.

[37] Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 313–400. 1997.

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

[39] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000.

[40] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[41] Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming, CP 2002, Ithaca, NY*, pages 310–326, 2002.

[42] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[43] Vasant Dhar. Data science and prediction. *Commun. ACM*, 56(12):64–73, 2013.

[44] Guo-Li Ding, Paul Seymour, and Peter Winkler. Bounding the vertex cover number of a hypergraph. *Combinatorica*, 14(1):23–34, 1994.

[45] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artif. Intell.*, 132(2):151–182, 2001.

[46] Pierre Duchet. Hypergraphs. In *Handbook of combinatorics (vol. 1)*, pages 381–432. MIT Press, 1996.

[47] David Duris. Some characterizations of γ and β-acyclicity of hypergraphs. *Inf. Process. Lett.*, 112(16):617–620, 2012.

[48] M. Ayaz Dzulfikar, Johannes Klaus Fichte, and Markus Hecher. The PACE 2019 parameterized algorithms and computational experiments challenge: The fourth iteration (invited paper). In *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, Munich*, pages 25:1–25:23, 2019.

[49] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.

[50] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.

[51] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.

[52] Henning Fernau. On parameterized enumeration. In *Proceedings of the 8th Annual International Computing and Combinatorics Conference, COCOON 2002, Singapore*, pages 564–573. Springer, 2002.

[53] Wolfgang Fischl, Georg Gottlob, Davide Mario Longo, and Reinhard Pichler. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam*, pages 464–480, 2019.

[54] Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX*, pages 17–32, 2018.

[55] Zoltán Füredi. Matchings and covers in hypergraphs. *Graphs Comb.*, 4(1):115–206, 1988.

[56] Robert Ganian, Sebastian Ordyniak, and Stefan Szeider. A join-based hybrid parameter for constraint satisfaction. In *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT*, pages 195–212, 2019.

[57] Nathan Goodman and Oded Shmueli. The tree projection theorem and relational query processing. *J. Comput. Syst. Sci.*, 28(1):60–79, 1984.

[58] Georg Gottlob and Gianluigi Greco. Decomposing combinatorial auctions and set packing problems. *J. ACM*, 60(4):24:1–24:39, 2013.

[59] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: Questions and answers. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA*, pages 57–74, 2016.

[60] Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Pure nash equilibria: Hard and easy games. *J. Artif. Intell. Res.*, 24:357–406, 2005.

[61] Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Tree projections and constraint optimization problems: Fixed-parameter tractability and parallel algorithms. *J. Comput. Syst. Sci.*, 94:11–40, 2018.

[62] Georg Gottlob, Matthias Lanzinger, Davide Mario Longo, Cem Okulmus, and Reinhard Pichler. The hypertrac project: Recent progress and future research directions on hypergraph decompositions. In *Proceedings of the 17th Internatioal Conferenece on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR 2020, Vienna*, pages 3–21, 2020.

[63] Georg Gottlob, Matthias Lanzinger, and Reinhard Pichler. Semantic width revisited (extended abstract). In *Proceedings of the 13th Alberto Mendelzon International Workshop on Foundations of Data Management, Asunción*, 2019.

[64] Georg Gottlob, Matthias Lanzinger, Reinhard Pichler, and Igor Razgon. Complexity analysis of general and fractional hypertree decompositions. *CoRR*, abs/2002.05239, 2020.

[65] Georg Gottlob, Matthias Lanzinger, Reinhard Pichler, and Igor Razgon. Fractional covers of hypergraphs with bounded multi-intersection. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, Prague*, pages 41:1–41:14, 2020.

[66] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

[67] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Computing LOGCFL certificates. *Theor. Comput. Sci.*, 270(1-2):761–777, 2002.

[68] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[69] Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: Np-hardness and tractable variants. *J. ACM*, 56(6):30:1–30:32, 2009.

[70] Georg Gottlob, Cem Okulmus, and Reinhard Pichler. Fast and parallel decomposition of constraint satisfaction problems. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1155–1162, 2020.

[71] Georg Gottlob and Reinhard Pichler. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming, ICALP 2001, Crete*, pages 708–719. Springer, 2001.

[72] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007.

[73] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014.

[74] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014.

[75] Marc Gyssens, Peter Jeavons, and David A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artif. Intell.*, 66(1):57–89, 1994.

[76] 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.

[77] Pavol Hell and Jaroslav Nesetril. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990.

[78] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition.* Pearson international edition. Addison-Wesley, 2007.

[79] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[80] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, Yorktown Heights, NY*, The IBM Research Symposia Series, pages 85–103. Plenum Press, 1972.

[81] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL*, pages 429–444, 2017.

[82] Christoph Koch, Yanif Ahmad, Oliver Kennedy, Milos Nikolic, Andres Nötzli, Daniel Lupei, and Amir Shaikhha. Dbtoaster: higher-order delta processing for dynamic, frequently fresh views. *VLDB J.*, 23(2):253–278, 2014.

[83] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

[84] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[85] Mohammed Lalou, Zineb Habbas, and Kamal Amroun. Solving hypertree structured CSP: sequential and parallel approaches. In Marco Gavanelli and Toni Mancini, editors, *Proceedings of the 16th RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion, RCRA@AI*IA 2009, Reggio Emilia*, volume 589 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[86] Matthias Lanzinger. Tractability beyond β-acyclicity for conjunctive queries with negation. *CoRR*, abs/2007.08876, 2020.

[87] Aviv Lustig and Oded Shmueli. Acyclic hypergraph projections. *J. Algorithms*, 30(2):400–422, 1999.

[88] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[89] Dániel Marx. Approximating fractional hypertree width. *ACM Trans. Algorithms*, 6(2):29:1–29:17, 2010.

[90] Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6(1):85–112, 2010.

[91] Dániel Marx. Tractable structures for constraint satisfaction with truth tables. *Theory Comput. Syst.*, 48(3):444–464, 2011.

[92] Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013.

[93] Jiří Matoušek. Geometric set systems. In *European Congress of Mathematics*, pages 1–27. Springer, 1998.

[94] David E. Narváez. Constraint satisfaction techniques for combinatorial problems. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial*

Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, pages 8028–8029, 2018.

[95] Hung Q. Ngo, Dung T. Nguyen, Christopher Ré, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2014, Snowbird, UT*, pages 234–245. ACM, 2014.

[96] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018.

[97] Sebastian Ordyniak, Daniël Paulusma, and Stefan Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theor. Comput. Sci.*, 481:85–99, 2013.

[98] Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.

[99] Jorge Pérez, Marcelo Arenas, and Claudio Gutiérrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009.

[100] Alberto Del Pia and Silvia Di Gregorio. On the complexity of binary polynomial optimization over acyclic hypergraphs. *CoRR*, abs/2007.05861, 2020.

[101] Foster J. Provost and Tom Fawcett. Data science and its relationship to big data and data-driven decision making. *Big Data*, 1(1):51–59, 2013.

[102] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[103] Walter L. Ruzzo. Tree-size bounded alternation. *J. Comput. Syst. Sci.*, 21(2):218–235, 1980.

[104] Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving #sat and MAXSAT by dynamic programming. *J. Artif. Intell. Res.*, 54:59–82, 2015.

[105] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[106] Francesco Scarcello. From hypertree width to submodular width and data-dependent structural decompositions. In *Proceedings of the 26th Italian Symposium on Advanced Database Systems, Castellaneta Marina (Taranto)*, 2018.

[107] Edward Scheinerman and Daniel Ullman. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs.* Dover Publications, Inc., 2011.

[108] Edward P. K. Tsang. *Foundations of constraint satisfaction.* Computation in cognitive science. Academic Press, 1993.

[109] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition.* Springer, 2016.

[110] H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. Syst. Sci.*, 43(2):380–404, 1991.

[111] Felix Winter, Nysret Musliu, Emir Demirovic, and Christoph Mrkvicka. Solution approaches for an automotive paint shop scheduling problem. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA*, pages 573–581, 2019.

[112] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Databases, VLDB 1981, Cannes*, pages 82–94. VLDB, 1981.

[113] Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981.

[114] Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA*, pages 331–342, 2017.