Understanding Tractable Decompositions for Constraint Satisfaction

Zoltán Miklós
Abstract

Constraint satisfaction problems (CSPs) are NP-complete in general, therefore it is important to identify tractable subclasses. A possible way to find such subclasses is to associate a hypergraph to the problem and impose restrictions on its structure.

In this thesis we follow this direction. Among such structural properties, particularly important is acyclicity: it is well known that CSPs whose associated hypergraph is acyclic can be solved efficiently. In the last decade, many structural decompositions have been proposed. These concepts can be seen as generalizations of hypergraph acyclicity. The interesting decomposition concepts are those which both enable the problems in the defined subclass to be solved in polynomial time and the associated hypergraphs to be recognized efficiently. Hypertree decompositions, introduced by Gottlob et al. in [43], fall in this category and additionally, for a long time, this class was the most general concept known to have both of these desirable properties.

We study further generalizations of this concept. It was shown recently ([45]) that the recognition problem for the most straightforward generalization, for the so-called generalized hypertree decompositions, is NP-hard. Understanding the deep reasons for this intractability result enabled us to define new decompositions with tractable recognition algorithms. We not only introduce a new decomposition concept, but also a methodology to define such decompositions using subedges of the hypergraph. In this way we get a very clear picture of tractable decompositions. As an application of our method, we construct a new decomposition concept, called component hypertree decomposition, which is tractable and strictly more general than all other known tractable methods, including the recently introduced spread cut decomposition. We also define an even more general concept, which also generalizes the spread cut decompositions, according to their new definitions.

We analyze various properties of generalized hypertree decompositions and study the parallel complexity of the recognition algorithms for the known tractable
decomposition methods. Understanding their similarities and their relation to gen-
eralized hypertree decomposition, we gave upper bounds for the parallel complex-
ity of their recognition.
# Contents

Acknowledgements  

1 Introduction  
   1.1 Motivation ......................................................... 6  
   1.2 Overview of results ............................................. 10  
   1.3 Structure of the thesis ......................................... 14  

2 Preliminaries  
   2.1 Queries and hypergraphs ......................................... 15  
   2.2 Conjunctive query evaluation .................................... 17  
   2.3 Hypergraph decompositions ....................................... 19  
   2.4 Hypertree decompositions ......................................... 23  
      2.4.1 Game-theoretic characterization of hypertreewidth .......... 24  
   2.5 The Tree Projection Problem .................................... 25  
      2.5.1 Relating TPP to decomposition problems ................. 27  
   2.6 Yannakakis algorithm ............................................. 29  
   2.7 Sufficient vs. necessary conditions for tractable queries .... 30  
   2.8 Relevant results from complexity theory ....................... 31  
      2.8.1 On logspace computations ................................. 31  
      2.8.2 Models of computation ...................................... 32  
      2.8.3 The complexity class LogCFL .............................. 34
Acknowledgements

I would like to express my gratitude to my supervisor Prof Georg Gottlob for his continuous support, encouragement, advice and patience. At this point it is difficult to say and list here what else I can thank him for, I wish I could. In any case, I can say for sure that it was always very motivating to see his intense passion for science.

I am also very grateful to Prof. Thomas Schwentick for a number of very interesting discussions and also for his advice and encouragement. I would like to thank Prof. Christoph Koch for his interesting course on database theory at TU Wien and introducing me to Georg. I would like to thank my colleagues at Database and Artificial Intelligence group at the Vienna University of Technology and at the Oxford University Computing Laboratory for their support.

My doctorate studies have a fairly unusual history. Before meeting with Prof. Georg Gottlob in Vienna and starting my thesis research with him, I was involved in various research projects and I published papers in other areas, such as the Semantic Web, eLearning and distributed systems [79, 65, 78, 27, 5, 80, 64, 81, 62, 63, 54]. This research experience helped me a lot later in my work, sometimes directly, sometimes indirectly. I would like to thank for Prof. Mehdi Jazayeri and my former colleagues at the Distributed Systems Group, at Vienna University of Technology and also for Prof. Gustaf Neumann and my former group at Vienna University of Economics for all their help and support. I am also grateful for the funding I received for my work from EU IST projects Opelix and Elena, the FWF project “Complementary Approaches to Constraint Satisfaction” and also for the
scholarship from the Wolfgang Pauli Institute, Vienna. I would like to thank for St. Anne’s College, Oxford, for the supportive environment. I am grateful to András Salamon for proofreading parts of this thesis.

I would like to thank my examiners Prof. Peter Jeavons and Dr. Francesco Scarcello for their comments. In particular, I would like to thank Dr. Scarcello for his suggestions on Section 6.2.

Many thanks for all my friends and family. Last but not least, I would like to thank my parents, Miklós Istvánné Tatai Julianna and Miklós István for their support and love.

Oxford, March 2008
Chapter 1

Introduction

1.1 Motivation

 Conjunctive queries are one of the most basic types of database queries. They are equivalent in expressive power to Select-Project-Join queries. Evaluating a Boolean conjunctive query over a relational database (BCQ) is one of the most important and best-studied problems in database theory. The complexity of this problem was shown to be NP-complete in a seminal paper by Chandra and Merlin [17] in 1977.

 Another central problem in computer science and in artificial intelligence in particular, is the Constraint Satisfaction Problem (CSP). Since the pioneering work of Montanari [66] (from 1974), CSP has become one of the central research topics in artificial intelligence. The complexity of CSPs was shown to be NP-complete by Mackworth [60], in 1977.

 Since both of the problems BCQ and CSP are of central importance, both the database and constraint communities developed techniques to cope with the NP-completeness and find subclasses for which these problems can be solved in polynomial time. A possible way to find such tractable subclasses is to associate a hypergraph to the query (or to the CSP) and impose structural restrictions on this hypergraph.
Example 1.1. Consider the Boolean conjunctive query\(^1\) over a database with a binary relation \(r\) and a ternary relation \(s\):

\[
Q_0 : \quad r(X_1, X_2) \land s(X_2, X_3, X_9) \land s(X_3, X_4, X_{10}) \land r(X_4, X_5) \land s(X_5, X_6, X_9) \land s(X_6, X_7, X_{10}) \land s(X_7, X_8, X_9) \land s(X_1, X_8, X_{10}).
\]

The hypergraph \(H_0 = (V_0, E_0)\), depicted in Figure 1.1 associated with the query has a vertex set \(H_0 = \{v_1, v_2, \ldots, v_{10}\}\), where for each query variable \(X_i\) there is a vertex \(v_i\) and an edge set \(E_0\) which consists of the following edges:

\[
\begin{align*}
e_1 &= \{v_1, v_2\}, & e_2 &= \{v_2, v_3, v_9\}, & e_3 &= \{v_3, v_4, v_{10}\}, \\
e_4 &= \{v_4, v_5\}, & e_5 &= \{v_5, v_6, v_9\}, & e_6 &= \{v_6, v_7, v_{10}\}, \\
e_7 &= \{v_7, v_8, v_9\}, & e_8 &= \{v_1, v_8, v_{10}\}.
\end{align*}
\]

It is a classical result of database theory that Boolean conjunctive query evaluation, while NP-hard in general, is polynomial in case of acyclic queries, i.e., queries whose associated hypergraphs are acyclic [87]. More recently, the evaluation of acyclic Boolean conjunctive queries was also shown to be highly paral-

\(^1\)The example is adapted from [2, 4].
lelizable [41].

After many years of parallel research, in recent years the two communities became less separated. Kolaitis and Vardi [56] pointed out, that despite their different formulation and focus, BCQ and CSP are essentially the same problems, they can be both reformulated as the following question: given two finite relational structures $A$ and $B$, is there a homomorphism $h : A \rightarrow B$?

**Nearly Acyclic Hypergraphs and Hypergraph Decompositions.** Intensive efforts have been made in the last decade to generalize the class of acyclic hypergraphs to significantly larger classes and to extend the positive complexity results for hypergraph-based problems to cover instances whose associated hypergraphs belong to these larger classes. This was motivated by two facts. Firstly, it was often observed that many relevant queries are not precisely acyclic but in some sense nearly acyclic – experimental support for this was recently given in [77]. Secondly, there exists a very successful generalization of graph acyclicity, namely, bounded treewidth [72]. A large number of graph-based problems are tractable on instances of bounded treewidth [25, 8, 9, 47, 31]. There has been a quest for a suitable hypergraph decomposition method $M$ and associated $M$-width that would be a good measure of the degree of cyclicity of a hypergraph. To be usable in the context of conjunctive query processing, such a decomposition method must fulfill two important criteria:

- **Polynomial Query Evaluation.** Boolean conjunctive query evaluation must be tractable for queries whose $M$-width is bounded by a constant.

- **Polynomial Recognizability.** For each constant $k$, hypergraphs (and thus queries) of $M$-width ($MW$) bounded by $k$ must be recognizable in polynomial time, and for such queries an $M$-decomposition of width at most $k$ must be computable in polynomial time.

In the database and constraint satisfaction communities, various methods of hypergraph decompositions were defined. These methods all amount to clustering
the query hypergraph in a tree-like form and to using such a clustering for transforming the original cyclic query into an acyclic query over a modified database whose relations are obtained by taking for each cluster the natural join of the relations corresponding to the edges of that cluster. The width of the decomposition is the maximum cluster size, that is, the maximum number of edges per cluster. The different decomposition methods differ in the way the edge clusters are determined.

An overview and comparison of most of these methods can be found in [40]. In recent years, more general decomposition methods were studied, that yield better decompositions (of smaller width) for larger classes of hypergraphs. The most general of these decompositions is the generalized hypertree decomposition (GHD) [44, 4], also called acyclic guarded cover in [22].

**Generalized Hypertree Decompositions.** GHDs of the hypergraph of Example 1.1 of width 2 and 3 of $H_0$ (and of query $Q_0$) are depicted in Figure 1.2.a and 1.2.b, respectively. A GHD of a hypergraph $H$ (in our example, $H_0$) consists of a tree $T$ such that each node $p$ of $T$ is labeled with a set $\lambda(p)$ of edges of $H$ and a set $\chi(p)$ of vertices of $H$. Each edge of $H$ must be covered by at least one $\chi(p)$. For each node $p$ of the tree $T$, the set $\chi(p)$ is covered by the union of the edges in $\lambda(p)$. For each vertex $i$ of $H$, the set of all nodes of $T$ where $i$ occurs in the $\chi$-part, induces a connected subtree of $T$.

The width $\text{ghw}(D)$, also denoted by $|D|$, of a GHD $D$ is the maximum cardinality of $\lambda(p)$ over all nodes $p$ of the decomposition tree of $D$. The generalized hypertree width $\text{ghw}(H)$ of $H$ is the minimum width over all possible GHDs of $H$. Note that a hypergraph $H$ is acyclic iff $\text{ghw}(H) = 1$. In [43] it was shown that GHDs satisfy the Polynomial Query Evaluation property. In particular, given a Boolean query $Q$ of width $q$, a GHD $D$ of width $k$ and size $g$, of (the hypergraph of) $Q$, and a database $DB$ whose largest relation has size $r$, then $Q$ can be answered on $DB$ in time $O(r + g)^k \times \log(r + g)$. Therefore, computing hy-

---

2This complexity result can be obtained by using the algorithms of [19, 43] for query evalua-
pertree decompositions of smaller width improves the running time of the query evaluation algorithm.

By a recent result [45], recognizing hypergraphs with generalized hypertreewidth at most 3 is NP-complete.

1.2 Overview of results

Generalized hypertree decompositions We study the properties of hypergraphs with bounded generalized hypertreewidth. Using normal form transformations, which were originally introduced for studying hypertree decompositions, we show that bounded-width GHDS posses some useful properties. These properties are of independent interest, but we also use them later. We also give a new, alternative definition for hypertree decompositions, in terms of normal form GHDS. However, by a recent result of Marx [61], we cannot expect a better worst-case running time, even if we use a different algorithm.
Comparing decompositions  We need more precise definitions to describe our results. A decomposition method \( M \) associates for each hypergraph \( H \) a set \( M(H) \) of GHDs. The \( M \)-width of a hypergraph \( H \), \( MW(H) \) is defined as the minimal width of decompositions in \( M(H) \). Given two decomposition methods \( A \) and \( B \), we write that \( A \leq B \) iff for each hypergraph \( H \), \( AW(H) \leq BW(H) \). If \( A \leq B \) and there exists a hypergraph \( H \) such that \( AW(H) < BW(H) \), then we write \( A < B \).

Subedge-Based Decomposition Methods. A generalized hypertree decomposition of a hypergraph \( H \) satisfying a special condition \(^3\) is called hypertree decomposition \( (HD) \) and the corresponding width is referred to as the hypertreewidth \( hw(H) \) of a hypergraph \( H \). While \( k \)-bounded hypertreewidth is polynomially recognizable for each fixed \( k \) [43], recognizing hypergraphs with generalized hypertreewidth at most 3 is NP-hard [45]. We were motivated to improve hypertree decompositions and find even larger classes of hypergraphs, which which also have tractable recognition. We define the concept of subedge-based decomposition methods. A subedge of a hypergraph \( H \) is a subset of some edge of \( H \). A subedge-based decomposition method \( M \) relies on a subedge function. This is a function \( f \) which associates to each integer \( k > 0 \) and each hypergraph \( H \) a set \( f(H, k) \) of subedges of \( H \). Moreover, the set of \( k \)-width \( M \)-decompositions can be obtained as follows: (1) obtain a hypertree decomposition \( D \) of \( H' = (V, E \cup f(H, k)) \), and (2) convert \( D \) into a GHD of \( H \) by replacing each subedge \( e \in \lambda(p) \), for each decomposition node \( p \), by some edge \( e' \) of \( H \) such that \( e \subseteq e' \). We call such a decomposition method \( M \) subedge-based. We derive the following result:

|For each polynomially recognizable decomposition method \( GHD \leq M \), there exists a polynomially recognizable subedge-based decomposition method \( M' \) such that \( M' \leq M \).|

The above result is useful from a methodological point of view. In fact, it tells us that when searching for some new decomposition method \( M \) such that

\(^3\)See Chapter 2 for precise definitions.
$GHD \leq M \leq HD$, then we may concentrate on subedge-based decomposition, and thus study appropriate subedge functions. This is what we did.

**Component Hypertree decompositions.** We found one particularly interesting subedge function $f^C$, whose definition is based on structural properties of the input hypergraph $H$. In particular, each subedge in $f^C(H,k)$ is obtained from a full edge $e$ and some candidate decomposition block $M$ of $\leq k$ edges containing $e$, by eliminating from $e$ all vertices that are edge-connected to some induced component of $V(H) \setminus \text{vertices}(M)$, or all vertices that are not edge-connected to any component of $V(H) \setminus \text{vertices}(M)$, or all vertices from $e \cup (M \setminus e)$ that are edge-connected to some component of $V(H) \setminus \text{vertices}(M)$. The new subedge based decomposition method based on this subedge function $f^C$ is called *component hypertree decomposition (CHD)* and its associated width is referred to as *component hypertree width (chw)*. We show that:

| Component hypertree decompositions fulfill both criteria, polynomial query answering and polynomial recognizability. |

We also compared CHD to HD and spread cut decompositions, $SCD$ (defined in [22]) and found the following:

$\text{CHD < HD and CHD < SCD.}$

In particular, for the hypergraph $H_0$ of Figure 1.1, we have $hw(H_0) = 3$ but $chw(H_0) = scw(H_0) = ghw(H_0) = 2$.

**Extended Component Hypertree decompositions.** We further extended CHDs and –using subedge-functions– we defined an even more general decomposition method, called extended component hypertree decomposition ($ECHD$). We compared this decomposition with CHD also with the new definition of spread cut decomposition $SCD_{new}$, recently defined in [21].

$ECHD < CHD$ and $ECHD < SCD_{new}$ and $ECHD < SCD$. 

12
The method of extended component hypertree decompositions is thus currently the most general known polynomially recognizable hypergraph decomposition method.

**Spread cut versus hypertree decompositions.** We also study the relation of hypertree decomposition and spread cut decomposition. In [21], the authors state the following question as an open problem:

“Is there a family of hypergraphs for which the hypertree width is smaller than the width of the best spread cut decomposition by some constant factor?”

We answer this question. We first define a hypergraph which has hypertreewidth 2 and spread cut width 3. Then we give a more general example, a class of hypergraphs $G_n$ such that the hypergraphs in the class have hypertreewidth at most $2n + 1$ and spread cut width $3n$.

**Parallel complexity of structural decompositions** We studied the parallel complexity of known decompositions, surveyed in [41] and [20]. Some of the recognition problems, e.g. bounded treewidth, bounded hypertreewidth are known to be in a low parallel complexity class $\text{LogCFL}$. We show that this property also holds for each of these classes. In particular, we show that the recognition problem for bounded biconnected component width and bounded spread cut width is in $\text{LogCFL}$. For bounded cycle cutset width and cycle hypercutset width we give an even better upper bound, namely $L$. We also show that recognizing bounded pathwidth hypergraphs is feasible in $NL$.

**Bounded dimension hypergraphs** A hypergraph has bounded dimension if the maximum size of its hyperedges is bounded by a constant. We show that for a bounded dimension hypergraph $H$ we can test $ghw(H) \leq k$ in polynomial time and also in $\text{LogCFL}$. We show that for this class of hypergraphs the Tree Projection Problem (explained later) can also be solved in polynomial time. This is in contrast with the results of [45], where the authors show that these problems are NP-hard if we do not assume that $H$ has bounded dimension.
1.3 Structure of the thesis

In Chapter 2 we review some relevant concepts from graph theory and complexity theory. We recall the definitions of hypertree decomposition and the tree projection problem (TPP) and we explain how the TPP is related to the generalized hypertree decompositions. In Chapter 3 we study the properties of generalized hypertree decompositions, in particular properties of decompositions in normal form. We give a new equivalent definition for hypertree decomposition that can help to understand the restrictions posed by the “special condition”.

In Chapter 4 we introduce our framework for defining decompositions using subedges and we show that in this way for all decomposition methods $M$ that can be implemented by a polynomial time algorithm, we can always define a subedge function $f$, such that the decomposition $M_f$ defined by $f$ dominates $M$, i.e. $M_f \leq M$. We also point out that subedge-based decompositions have a game-theoretic characterization. Using the subedge-based technique, in Chapter 5 we introduce a new decomposition method called component hypertree decomposition and compare it with other decomposition methods. In Chapter 6 we define an extended version of component hypertree decompositions. $ECHDs$ also dominate the new definition of spread cut decomposition. In Chapter 6 we also compare (the new definition of) spread cut decomposition to hypertree decomposition.

In Chapter 7 we study the parallel complexity of various decomposition methods from the literature. We also study hypergraphs with bounded dimension and show that even the more general problem TPP can be solved in LogCFL, in this special case. Chapter 8 concludes the thesis and outlines some possible future research directions.
Chapter 2

Preliminaries

2.1 Queries and hypergraphs

A hypergraph is a pair \( H = (V, E) \) consisting of a set \( V \) of vertices and a set \( E \) of hyperedges. A hyperedge \( e \in E \) is a subset of \( V \). In this thesis we adopt the usual logical representation of a relational database [1], where data tuples are identified with logical ground atoms and conjunctive queries are represented as datalog rules.

There is a very natural way to associate a hypergraph \( H(Q) \) to a query \( Q \): the set of vertices consists of all variables occurring in \( Q \), and the hyperedges are all sets of variables of \( a \), such that \( a \) is an atom in the body of \( Q \). This is explained in the following example.

Example 2.1. Consider a relational database with the following relational schemas:

\[
\begin{align*}
\text{plays}(\text{Person}, \text{Film}, \text{Role}) \\
\text{directs}(\text{Person}, \text{Film}) \\
\text{relative}(\text{Person}, \text{Person})
\end{align*}
\]

The query \( Q_1 \) asks: “Are there some actors who play in a film, directed by a relative?”

\[Q_1: \text{ans} \leftarrow \text{plays}(A, F, R), \text{directs}(D, F), \text{relative}(A, D).\]
The query $Q_2$ asks: “Are there some actors whose relative directs a film?”

$Q_2: \text{ans} \leftarrow \text{plays}(A,F,R),\text{relative}(A,D),\text{directs}(D,F')$.

Figure 2.1: The hypergraph of the queries (a) $Q_1$ and (b) $Q_2$ of Example 2.1.

The hypergraphs $H(Q_1)$ and $H(Q_2)$ associated with queries $Q_1$ and $Q_2$ respectively, are depicted in Figure 2.1. A query is acyclic if its associated hypergraph is acyclic. We refer to the standard notion of hypergraph acyclicity in database theory [1]: A hypergraph $H$ is acyclic if the GYO reduct of $H$ is the empty set, i.e. $GYO(H) = \emptyset$. The GYO-reduct $GYO(H)$ of a hypergraph $H$ is a hypergraph obtained from $H$ by repeatedly applying the following two reduction rules (in arbitrary order) until this is no longer possible:

- if there is a hyperedge $e$ which is empty or contained in an another hyperedge, then remove $e$ from $H$,
- if there is a vertex $v$, which is only contained in one hyperedge, then remove $v$ from $H$.

The hypergraph $H(Q_1)$ of the query $Q_1$ is cyclic, while the hypergraph $H(Q_2)$ of $Q_2$ is acyclic. Acyclic conjunctive queries play an important role in database theory. They are intensively studied, many equivalent characterizations have been discovered, see [10]. We recall here the characterization in terms of join trees.
**Definition 2.1.** The join tree of a hypergraph $H = (V, E)$ is a pair $\langle T, \delta \rangle$, where $T = (N, F)$ is a tree and $\lambda$ is a labeling function that associates each node $p \in N$ with an edge $e \in E$, such that the following conditions hold:

1. for each edge $e \in E$, there is a node $p \in N$, such that $\lambda(p) = \{e\}$,

2. for each vertex $v \in V$, the set $\{p \in N \mid v \in \text{vertices}(\lambda(p))\}$ induces a connected subtree of $T$.

**Theorem 2.1.** ([10]) A query is acyclic iff it has a join tree.

A join tree of the query $Q_2$ from Example 2.1 is depicted in Figure 2.2. Condition 2 of Definition 2.1 is often referred as the “connectedness condition”.

![Figure 2.2: The join tree of the query $Q_2$ of Example 2.1.](image)

### 2.2 Conjunctive query evaluation

The Boolean conjunctive query evaluation (BCQ) is the following problem.\(^1\)

**Input:** $\langle DB, Q \rangle$, (relational database, conjunctive query)  
**Question:** Does $Q$ have a non-empty result over $DB$?

The problem BCQ is NP-complete in general [17], but it becomes polynomial if the query is acyclic. Yannakakis [87] provided a sequential polynomial time algorithm for acyclic BCQ. More recently, Gottlob et al. [41] have shown that in

---

\(^1\)Since both the query and the database are input to the BCQ, the complexity of this problem is often referred to in the literature as the combined complexity of the query.
the case of acyclic queries, BCQ is highly parallelizable and complete for the low complexity class LogCFL.

In the last decade there has been intensive research on finding larger classes of hypergraphs, for which the problem BCQ is tractable. There are three important computational problems related to this question.

1. Recognize the class of hypergraphs,
2. compute a decomposition tree,
3. evaluate the query using the decomposition tree.

We review here the definition of these problems.
1.) The decision problem for a hypergraph property \( P \) (e.g. acyclicity) is the following problem.

<table>
<thead>
<tr>
<th>Testing hypergraph property ( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> hypergraph ( H ),</td>
</tr>
<tr>
<td><strong>Question:</strong> Does ( H ) possess the property ( P )?</td>
</tr>
</tbody>
</table>

This problem is also referred to as the recognition problem for hypergraphs with property \( P \).

2.) Computing the decomposition tree.

<table>
<thead>
<tr>
<th>Computing a decomposition tree</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> hypergraph ( H ),</td>
</tr>
<tr>
<td><strong>Output:</strong> decomposition tree for ( D )</td>
</tr>
</tbody>
</table>

Note that in some cases the width of the decomposition is also part of the input. In other cases, the width assumed to be a fixed number. In this section we assume the latter, if we refer to the complexity of this problem, we assume that the width of the decomposition is a fixed constant number.
3.) Evaluating a Boolean conjunctive query using a decomposition tree. 

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognizing acyclic hypergraphs</td>
<td>polynomial (linear) time</td>
</tr>
<tr>
<td>Constructing a join tree</td>
<td>polynomial (linear) time</td>
</tr>
<tr>
<td>Evaluating a BCQ</td>
<td>polynomial, LogCFL-complete</td>
</tr>
</tbody>
</table>

Figure 2.3: Acyclic BCQ

2.3 Hypergraph decompositions

A number of generalizations of the concept of acyclicity have been proposed in order to apply the efficient algorithms of acyclic hypergraphs to a larger class of hypergraphs. For useful methods, it should be possible to recognize this larger class of hypergraphs in polynomial time and the construction of the join tree should also be feasible in polynomial time. A survey and systematic comparison of these structural restrictions can be found in [40, 22]. Here we recall the definitions of some important concepts.

---

1. In the case of Boolean acyclic conjunctive queries, this problem is called JTREE in [41].
2. See also Theorem 7.4.
**Tree decomposition**

*Definition 2.2.* A tree decomposition of a hypergraph \( H = (V, E) \) is a pair \( \langle T, \chi \rangle \) where \( T = (N, F) \) is a tree and \( \chi \) is a labeling function that associates to each node \( p \in N \) a set of vertices \( \chi(p) \subseteq V \), such that the following conditions hold:

1. for each vertex \( v \in V \), there is a node \( p \in N \), such that \( v \in \chi(p) \),
2. for each edge \( e \) of \( H \), there is a node \( p \in N \), such that \( \text{vertices}(e) \subseteq \chi(p) \),
3. for each vertex \( v \in V \), the set \( \{ p \in N \mid v \in \chi(p) \} \) induces a connected subtree of \( T \).

The width of a tree decomposition \( \langle T, \chi \rangle \) is \( \max_{p \in N} |\chi(p)| - 1 \). The treewidth of \( G \) is the minimal width over all of its tree decompositions. The treewidth of a hypergraph is the treewidth of its primal graph\(^4\). Condition 2 of Definition 2.2 is often referred to as the “connectedness condition”.

The complexity of the related problems is summarized in Figure 2.4.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognizing ( k )-bounded treewidth hypergraphs</td>
<td>polynomial (linear) time, in ( \log \mathrm{CFL} )</td>
</tr>
<tr>
<td>Constructing a decomposition tree</td>
<td>polynomial (linear) time</td>
</tr>
<tr>
<td>Evaluating a BCQ</td>
<td>polynomial, ( \log \mathrm{CFL} )-complete</td>
</tr>
</tbody>
</table>

Figure 2.4: \( k \)-bounded treewidth BCQ (fixed \( k \))

**Query decomposition**

Query decompositions were introduced by Chekuri and Rajaraman in \[19\].\(^5\)

Query decompositions are called edge-defined guarded covers in \[21\].

---

\(^4\)The primal graph \( G \) of a hypergraph \( H \) is a graph having the same vertex set as \( H \), which has an edge between two vertices \( x \) and \( y \) if and only if there exists a hyperedge \( e \) in \( H \), such that both \( x \) and \( y \) are contained in \( e \), i.e. both \( x \in \text{vertices}(e) \) and \( y \in \text{vertices}(e) \) holds.

\(^5\)The definition presented here is a simplified form of the original definition from \[19\], but has the same computational properties, as was shown in \[43\].
Definition 2.3. A query decomposition of a hypergraph $H = (V, E)$ is a triple $\langle T, \chi, \lambda \rangle$, where $\chi$ and $\lambda$ are two labeling functions that associate each node $p$ of $T$ with two sets: $\chi(p) \subseteq V$ and $\lambda(p) \subseteq E$ such that the following conditions hold:

1. for each edge $e \in E$, there is a node $p$ of $T$, such that $\text{vertices}(e) \subseteq \chi(p)$,
2. for each vertex $v \in V$, the set $\{p \in N \mid v \in \chi(p)\}$ induces a connected subtree of $T$,
3. for each $p \in N, \chi(p) = \text{vertices}(\lambda(p))$.

Intuitively, while tree decompositions try to group vertices of the hypergraph, query decompositions group entire hyperedges. The maximum number of the hyperedges in a query decomposition is the width of the decomposition and the minimum width over all possible query decompositions of a hypergraph $H$ is the query width of $H$.

If a query decomposition for a query $Q$ is given, then $Q$ can be evaluated in polynomial time, see [19]. On the other hand, the recognition problem for $k$-bounded query width is NP-complete [43]. The complexity of the related problems is summarized in Figure 2.5.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognizing $k$-bounded query-width hypergraphs</td>
<td>NP-complete ($k = 4$)</td>
</tr>
<tr>
<td>Evaluating a BCQ</td>
<td>polynomial</td>
</tr>
</tbody>
</table>

Figure 2.5: $k$-bounded query-width BCQ (fixed $k$)

**Hypertree decomposition**

A hypertree for a hypergraph $H = (V, E)$ is a triple $\langle T, \chi, \lambda \rangle$, where $T = (N, F)$ is a (rooted) tree and $\chi$ and $\lambda$ are labeling functions that associate each node $p \in N$
with two sets: $\chi(p) \subseteq V$ and $\lambda(p) \subseteq E$. We denote the subtree rooted at node $p \in N$ with $T_p$ and let $\chi(T_p) = \{v \mid v \in \chi(w), w \in T_p\}$.

**Definition 2.4.** A hypertree decomposition of a hypergraph $H = (V, E)$ is a hypertree $\langle T, \chi, \lambda \rangle$, such that the following conditions hold:

1. for each edge $e \in E$, there is a node $p \in N$, such that $\text{vertices}(e) \subseteq \chi(p)$,
2. for each vertex $v \in V$, the set $\{p \in N \mid v \in \chi(p)\}$ induces a connected subtree of $T$,
3. for each $p \in N$, $\chi(p) \subseteq \text{vertices}(\lambda(p))$,
4. for each $p \in N$, $\text{vertices}(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$.

A hypertree decomposition is complete, if for all edges $e$ of the hypergraph $H$, there is a node $p \in N$ such that $e \in \lambda(p)$ and $\text{vertices}(e) \subseteq \chi(p)$. The width of a hypertree decomposition is $\max_{p \in N} |\lambda(p)|$. The hypertreewidth of a hypergraph is the minimum width over all of its hypertree decompositions. Condition 2 of the Definition 2.4 is often referred as the “connectedness condition”.

The acyclic hypergraphs are precisely the hypergraphs with hypertreewidth one. Indeed, any join tree trivially corresponds to a (complete) hypertree decomposition, and from a given hypertree decomposition of hypertreewidth one we can easily compute a join tree [43, 44]. The complexity of the related problems is summarized in Figure 2.6.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognizing $k$-bounded hypertreewidth hypergraphs</td>
<td>polynomial time, in $\text{LogCFL}$</td>
</tr>
<tr>
<td>Constructing a decomposition tree</td>
<td>polynomial time</td>
</tr>
<tr>
<td>Evaluating a BCQ</td>
<td>polynomial, $\text{LogCFL}$-complete</td>
</tr>
</tbody>
</table>

Figure 2.6: $k$-bounded hypertreewidth BCQ (fixed $k$)
**Generalized hypertree decomposition**

The 4th condition of hypertree decomposition is called the “special condition”. It has been added to the definition to ensure that the hypergraphs with fixed bounded hypertreewidth can be recognized in polynomial time. If we drop the special condition of hypertree decomposition and require only conditions 1-3, we call the new concept *generalized hypertree decomposition*.

**Definition 2.5.** A generalized hypertree decomposition of a hypergraph \( H = (V, E) \) is a hypertree \( (T, \chi, \lambda) \), such that the following conditions hold:

1. for each edge \( e \in E \), there is a node \( p \in N \), such that \( \text{vertices}(e) \subseteq \chi(p) \),

2. for each vertex \( v \in V \), the set \( \{ p \in N \mid v \in \chi(p) \} \) induces a connected subtree of \( T \),

3. for each \( p \in N, \chi(p) \subseteq \text{vertices}(\lambda(p)) \).

The problem BCQ can be solved in polynomial time, if the associated hypergraph has fixed bounded generalized hypertreewidth. It has been stated as an open question in [44] whether hypergraphs with bounded generalized hypertreewidth can be recognized in polynomial time. This question was studied in [45], where this decision problem was shown to be NP-hard.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognizing ( k )-bounded generalized hypertreewidth hypergraphs</td>
<td>NP-complete (( k=3 ))</td>
</tr>
<tr>
<td>Evaluating a BCQ</td>
<td>polynomial, LogCFL-complete</td>
</tr>
</tbody>
</table>

Figure 2.7: \( k \)-bounded generalized hypertreewidth BCQ (for fixed \( k \))

### 2.4 Hypertree decompositions

We use the definitions and notation introduced by Gottlob et al. in [43] for hypergraph decompositions. We recall here the most important definitions.
Definition 2.6. ([43]) Let \( H = (V, E) \) be a hypergraph and let \( S \subseteq V \) be a set of vertices. Let \( X \) and \( Y \) be two vertices in \( V \). We say that \( X \) and \( Y \) are \([S]-\)adjacent, if there exists a hyperedge \( e \) in \( E \), such that \( \{X, Y\} \subseteq e \setminus S \).

Definition 2.7. ([43]) Let \( H = (V, E) \) be a hypergraph and let \( S \subseteq V \) be a set of vertices. Let \( X \) and \( Y \) be two vertices in \( V \). We say that \( X \) and \( Y \) are \([S]-\)connected, if there exists a sequence \( X = X_0, \ldots, X_r = Y \) of vertices and a sequence of hyperedges \( e_0, \ldots, e_{r-1} \) (0 ≤ \( r \)) such that \( X_i \) is \([S]-\)adjacent to \( X_{i+1} \) and \( \{X_i, X_{i+1}\} \subseteq e_i \), for each \( i = 0, \ldots, r - 1 \).

A set of vertices \( C \) is \([S]-\)connected, if for all \( X, Y \in C \), the vertices \( X \) and \( Y \) are \([S]-\)connected. An \([S]-\)component is a maximum \([S]-\)connected set of vertices in \( V \setminus S \). For an \([S]-\)connected set \( C \), we define \( \text{edges}(C) \) as

\[
\text{edges}(C) = \{ e \mid e \in E, \text{vertices}(e) \cap C \neq \emptyset \}.
\]

Let \( \langle T, \chi, \lambda \rangle \) be a hypertree decomposition and let \( p \) be a node of \( T \). We can then speak about \([\text{vertices}(\lambda(p))]\)-components and \([\chi(p)]\)-components. Following the short notation introduced in [43], if it is clear from the context, we denote \( \chi(p) \) by \( p \) and in particular, a \([p]\)-component denotes a \( \chi(p) \) component.

### 2.4.1 Game-theoretic characterization of hypertreewidth

The robber and marshals (R&M) game was introduced in [44]. The R&M game is a combinatorial game played on a hypergraph, by \( k \) marshals (1 ≤ \( k \)) who occupy hyperedges, and a robber who occupies vertices of the graph. The goal of the marshals is to capture the robber and the goal of the robber is to escape. If the marshals can capture the robber, then the marshals win the game, if the robber can always escape, then he is the winner. The robber can only move along the hyperedges, while the marshals are allowed to move from an arbitrary position to an arbitrary one. There is an additional rule about the moves of the robber: If the marshals move from position \( M_1 \) (\( M_1 \) denotes the set of edges occupied by the marshals) to a position \( M_2 \), then during this move the robber is not allowed
to move through the set \(\text{vertices}(M_1 \cap M_2)\). We call the R&M game monotone, if we also require that the marshals have to capture the robber by monotonically shrinking his escape space \(^6\). In this game, if the marshals make a move, where the escape space of the robber increases, then they lose the game.

As the following theorem shows, there is a strong connection between the number of marshals who can win the monotone variant of the game and the hypertreewidth of the hypergraph the game is played on.

**Theorem 2.2.** ([44]) A hypergraph \(H\) has hypertreewidth at most \(k\) if and only if \(k\) marshals have a winning strategy in the monotone robber and marshals game on \(H\).

### 2.5 The Tree Projection Problem

Let \(H\) and \(H'\) be two hypergraphs. We write \(H \leq H'\), if for every edge \(e\) of \(H\) there is an edge \(e'\) of \(H'\) such that \(e \subseteq e'\). The following problem is known as the Tree Projection Problem.

<table>
<thead>
<tr>
<th>Tree Projection Problem (TPP)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Two hypergraphs (H) and (H').</td>
</tr>
<tr>
<td><strong>Question:</strong> Either find an acyclic hypergraph (A), such that (H \leq A \leq H') or claim that there is no such (A).</td>
</tr>
</tbody>
</table>

Sometimes we refer to the acyclic hypergraph \(A\) –if it exists– as a tree projection of \(H'\) with respect to \(H\). We can assume that for the input hypergraphs \(H\) and \(H'\) the relation \(H \leq H'\) holds, otherwise the tree projection problem has no solution. We also use the short notation \(TPP(H, H')\) to denote the tree projection problem with input hypergraphs \(H\) and \(H'\).

\(^6\)The escape space of the robber is defined as follows: If the marshals capture the robber, then the escape space is the emptyset. Otherwise, if the marshals position is \(M\) and the game is played on the hypergraph \(H\), then the escape space is the \([\text{vertices}(M)]\)-component of \(H\) that contains the vertex, which corresponds to the position of the robber.
Example 2.2. Figure 2.8 depicts a TPP with input hypergraphs $H = ([A, B, C, D], [(A, B), (B, C), (C, D), (D, A)])$ and $H' = ([A, B, C, D], [(A, C, B), (A, C, D), (B, D)])$. An acyclic hypergraph $A$, for which the relation $H \leq A \leq H'$ holds, is depicted in Figure 2.8.

![Figure 2.8: A tree projection problem.](image)

The TPP has some similarity to the graph and hypergraph sandwich problems surveyed in [34, 33, 59]. Let us note however, that the definitions are slightly different. In the cited papers it is assumed that the three hypergraphs $H$, $H'$ and $A$ have the same number of edges which are labeled by the same set of indices $[1..m] = \{1, \ldots, m\}$ and $e_i(H) \subseteq e_i(A) \subseteq e_i(H')$ for each $i \in [1..m]$ while the TPP does not make such assumptions.

Tree projections were studied in [36, 59, 76] in the context of query optimization. In particular, in [36] the following Tree Projection Theorem was shown. A query program $P$ consisting of a sequence of selections, projections, and semi-joins solves a relational query $Q$ over a database whose schema is described by a hypergraph $H_1$ iff the output schema of the query is described by a hypergraph $H_2$ such that there exists a tree projection of $H_2$ with respect to $H_1$.

The complexity of the tree projection problem was raised by Goodman and Shmueli in 1984 and mentioned as an open problem in many papers [35, 36, 76, 59]. The TPP clearly belongs to NP, since a solution can easily be verified in
polynomial time. The Complexity of the Tree Projection Problem has been repeatedly stated as an open problem for over twenty years [36, 59, 76]. It was mentioned, for example, in [75], in a different context. Lustig and Shmueli [59] gave a polynomial time decision algorithm for the special case, when \( H \) is a so-called \( Xring \) hypergraph, but they left the general case as an open problem. For further references, see [59]. Finally, it was shown in [45], that TPP is NP-hard.

**Theorem 2.3.** ([45]) TPP is NP-complete.

This result entails hardness results for problems of query optimization discussed in [36, 59, 76].

### 2.5.1 Relating TPP to decomposition problems

Two important problems, namely recognizing \( k \)-bounded treewidth of hypergraphs and recognizing \( k \)-bounded generalized hypertreewidth graphs can be easily reduced to the TPP.

**k-bounded treewidth** Given a hypergraph \( H \) and a fixed natural number \( k \), we would like to decide, whether \( \text{tw}(H) \leq k - 1 \). Let us consider the problem \( TPP(H, H_k) \), the tree projection problem with input hypergraphs \( H \) and \( H_k \), where \( H_k \) is the hypergraph defined in the following way. \( H_k \) has the same vertices as \( H \) and the edges of \( H_k \) are the subsets of vertices of \( H \) of size at most \( k - 1 \).

\[
\begin{align*}
V(H_k) &= V(H) \\
E(H_k) &= \{S \mid S \subseteq V(H), |S| \leq k\}
\end{align*}
\]

Thus, if \( |V| = n \) then the hypergraph \( H_k \) has \( \binom{n}{k} \) edges. Let us note that \( H \) can be a graph, still the \( TPP(H, H_k) \) problem is defined in the same way.

**Lemma 2.1.** \( TPP(H, H_k) \) has a solution if and only if the treewidth of the hypergraph \( H \) is at most \( k - 1 \), i.e. \( \text{tw}(H) \leq k - 1 \).
Proof. Indeed, if there is an acyclic hypergraph $A$ such that $H \leq A \leq H_k$, then a join tree $T$ of $A$ corresponds to a tree decomposition. On the other hand, if $H$ has a tree decomposition $\langle T, \chi \rangle$ of width $k - 1$, then let us define a hypergraph $A$ as follows: $V(A) = V(H)$ and $E(A) = \{\chi(p) \mid p \in \text{nodes}(T)\}$. Clearly, $A$ is acyclic – as the tree decomposition corresponds to a join tree – and $A$ is also a solution for $TPP(H, H_k)$.

For this and many other equivalent characterizations of the treewidth, see the surveys [14, 15]. Given a graph $G$, computing treewidth is NP-hard [7]. Yet, $TW(G) \leq k - 1$ can be verified in polynomial time for a fixed $k$. The first polynomial algorithm was given in [71]; its performance time was only bounded by $n$ to a function of $k$. Then this time was successively reduced to $O(n^{k+2})$ in [7], $O(n^2)$ in [73], $O(n \log^2 n)$ in [9, 12], $O(n \log n)$ in [69], and finally to $O(n)$ in [13]. For $k \leq 5$ there are special, simpler methods; see [15] section 3.3 or [14] section 3 for a survey.

**k-bounded generalized hypertreewidth**

Given a hypergraph $H$ and a fixed natural number $k$, we would like to decide whether $ghw(H) \leq k$. Let us consider the $TPP(H, H^k)$ with input hypergraphs $H$ and $H^k$. The hyperedges of $H^k$ are constructed from at most $k$ hyperedges of $H$.

$$V(H^k) = V(H)$$

$$E(H^k) = \{(\text{vertices}(S)) \mid S \subseteq E(H), |S| \leq k\}$$

So, if $|E| = m$ then the hypergraph $H^k$ has $\binom{m}{k}$ edges.

**Lemma 2.2.** The $TPP(H, H^k)$ has a solution if and only if $ghw(H) \leq k$.

Proof. If $TPP(H, H^k)$ has a solution $A$, then a join tree of $A$ corresponds to a generalized hypertree decomposition of width $k$. On the other hand, if $H$ has a GHDD $D = \langle T, \chi \rangle$ of width $k$, then let us define a hypergraph $A$ as $V(A) = V(H)$ and $E(A) = \{\chi(p) \mid p \in \text{nodes}(T)\}$. Clearly, $A$ is acyclic – since the decomposition $D$
corresponds to a join tree– and also $H \leq A \leq H^k$ holds, thus the $TPP(H, H^k)$ has a solution.

Gottlob et al. [44] raised the question, whether bounded generalized hypertreewidth can be tested efficiently. The NP-completeness of this problem was recently shown in [45].

**Theorem 2.4. ([45])**

*Testing whether a hypergraph $H$ has generalized hypertreewidth at most 3, i.e. $ghw(H) \leq 3$ is NP-complete.*

Thus, unless $P=NP$, even for bounds as low as 3, bounded $GHW$ is not polynomially recognizable, and bounded $GHD$s, if they exist, cannot be computed in polynomial time.

### 2.6 Yannakakis algorithm

In this section we review the Yannakakis algorithm [87] for evaluating Boolean acyclic conjunctive queries over a relational database.

Let $Q$ be an acyclic Boolean conjunctive query given together with its join tree $\langle T, \lambda \rangle$ over a relational database $DB$. The Yannakakis algorithm returns “true” if the query $Q$ evaluates “true” over $DB$. Figure 2.9 contains a high level description of the Yannakakis algorithm.

In the case of generalized hypertree decompositions, the $\lambda(p)$ sets may contain more than hyperedges. In this case, when visiting a node $p$, we have first to compute the join of the relations at $p$, i.e. $P_1 \bowtie P_2 \bowtie \ldots P_k$, where the relations $P_1, \ldots, P_k$ correspond to the hyperedges $e_1, \ldots, e_k$ of $\lambda(p)$. Note that for evaluating a Boolean conjunctive query, we have to use a complete generalized hypertree decomposition.

---

7Gottlob et al. [41] have shown that a join tree for an acyclic Boolean query can be constructed very efficiently. To simplify the discussion, we assume here that the join tree for $Q$ has already been computed.
Yannakakis algorithm

**Input:** query $Q$, join tree $(T, \lambda)$, database $DB$

**Result:** “true” iff $Q$ evaluates “true” over $DB$

- Visit each node of $T$ in a bottom-up order, i.e. visit a node after having visited all of its children.
- Suppose we visit the node $r$. Then, for each child node $p$ of $r$, replace the relation $\lambda(r)$ in $DB$ by $\lambda(r) \bowtie \lambda(p)$.
- If for some child node $p$ of $r$, the relation has no tuples, i.e. $\lambda(r) \bowtie \lambda(p) = \emptyset$, then stop and return “false”.
- After visiting the root node terminate and return “true”.

Figure 2.9: Yannakakis algorithm

### 2.7 Sufficient vs. necessary conditions for tractable queries

Throughout this thesis we are interested in sufficient conditions for tractable classes of BCQ. Grohe et al. [52] studied restrictions on the primal graph of the associated hypergraph of conjunctive queries. They showed the following remarkable result.

**Theorem 2.5.** ([52])

Let $Q$ be a Boolean conjunctive query and let $G(Q)$ denote the primal graph of the hypergraph associated with the query $Q$. Let us assume that $P \neq NP$. Let $C$ be a class of graphs, such that $C$ is closed under taking minors. The following statements are equivalent.

- Each query $Q$ such that $G(Q)$ is in the class $C$ can be evaluated in polynomial time.

---

8For definitions of graph minors see e.g. [28].
– \( C \) has bounded treewidth.

The authors later also obtained a similar result for relational structures of bounded arity. Further results of this line of research can be found in [50].

Such necessary conditions are not known if we do not assume that the associated hypergraph has bounded dimension. Bounded generalized hypertreewidth is not a necessary condition for tractability: queries whose associated hypergraph has bounded fractional hypertreewidth [51] can be evaluated in polynomial time, while there are hypergraphs with bounded fractional hypertreewidth, but unbounded generalized hypertreewidth.

A Boolean conjunctive query \( Q \) can be evaluated in time \( O(r + g)^{O(k)} \), where \( k \) is the treewidth of the decomposition and \( g \) is the size of (the hypergraph of) \( Q \), and the largest relation in the database \( DB \) has size \( r \). Very recently Marx [61] also showed that the bounded treewidth restriction of the primal graph also leads to an optimal algorithm under some complexity theoretic assumptions.

### 2.8 Relevant results from complexity theory

#### 2.8.1 On logspace computations

We will use the following deep results.

**Theorem 2.6.** (Nisan, Ta-Shma [67]) \( S_L = \text{co-}S_L \)

**Theorem 2.7.** (Nisan, Ta-Shma [67]) \( S_L = L = S_L \)

**Theorem 2.8.** (Reingold [70]) \( L = S_L \)

In some of our proofs we refer to the problem \texttt{ConnectedComponentsEqual} and the following result on its complexity.

\texttt{ConnectedComponentsEqual} is the following decision problem:

**Input:** Two undirected graphs \( G_1 = (V_1, E_1) \), \( G_2 = (V_2, E_2) \)

**Question:** Is the number of connected components of \( G_1 \) equal to the number of connected components of \( G_2 \)?
**Theorem 2.9.** ([6, 67] and Theorem 2.8)

ConnectedComponentsEqual is complete for L.

A direct consequence of the above Theorem is that we can decide in logarithmic space whether a graph has exactly 2 connected components, by using a “dummy” graph containing only two isolated vertices.

### 2.8.2 Models of computation

**Alternating Turing machines**

Alternation can be seen as a generalization of nondeterminism, defined by Chandra et al. [18]. A nondeterministic Turing machine accepts its input, if and only if its start state is accepting or there exists a successor state, which leads to an accepting state. In contrast a machine that accepts the complement language would have a start state which is an accepting state or all of its successor states would lead to an accepting state. In an alternating Turing machine we allow both types of configurations: existential states are accepting if at least one of their successors is accepting, while universal states are accepting if all of their successors are accepting.

More formally, an alternating Turing machine (ATM) is a 5-tuple $M = (Q, \Gamma, \delta, q_0, g)$, where

- $Q$ is the finite set of states,
- $\Gamma$ is the finite tape alphabet,
- $\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$ is the transition function, $L$ ($R$) shifts the head to the left (right),
- $q_0$ is the initial state,
- $g : Q \rightarrow \{\forall, \exists, accept, reject\}$ is a function that associates a type to each state.
If $M$ is in an accepting (rejecting) state, then the computation is accepting (rejecting). A state $q \in Q$ with $g(q) = \exists$ is accepting if there exists a state reachable from $q$ which is accepting. A state $q \in Q$ with $g(q) = \forall$ is accepting if all states reachable from $q$ are accepting. A witness tree is a tree corresponding to an accepting computation, such that its nodes are labelled with the states of the computation. A computation tree and a witness tree of an accepting computation of an ATM is depicted in Figure 2.10.

![Figure 2.10: a) A computation tree and b) a witness tree of an accepting computation of an ATM.](image)

Alternating Turing machines can be used to define complexity classes in the usual way. For example, the class $\text{ASPACE}(f(n))$ [18] is defined as $\text{ASPACE}(f(n)) = \{L \mid L \text{ is a language decidable in } O(f(n)) \text{ space by an ATM}\}$. $AL = \text{ASPACE}(\log(n))$.

**Theorem 2.10.** (Chandra et al. [18]) $AL = P$.

**Auxiliary pushdown automaton**

A nondeterministic auxiliary pushdown automaton (NAuxPDA) [23] consist of

- a nondeterministic Turing machine with a read-only two-way input tape,
- a read/write worktape,
a pushdown store (a stack).

If the Turing machine is deterministic, the computational device is called Aux-PDA. AuxPDAs and NAuxPDAs can also be used to define complexity classes, by restricting the basic computational resources: time and space. Note that the space restriction of an NAuxPDA computation refers only to bounds used on the read/write worktape, but there is no restriction on the space used for the pushdown store. The class \( NAuxPDA-SPACE(f(n)) \) is defined as \( NAuxPDA-SPACE(f(n)) = \{ L \mid L \text{ is a language decidable in } O(f(n)) \text{ space by an NAuxPDA} \} \).

**Theorem 2.11.** ([23], [55])

\( NAuxPDA-SPACE(\log(n)) = P, \ AuxPDA-SPACE(\log(n)) = P. \)

### 2.8.3 The complexity class \( \text{LogCFL} \)

LogCFL is the class of decision problems logspace reducible to a context-free language. There are several problems, including some very natural problems, which are complete for LogCFL. These include Greibach’s hardest context-free language [49], the problem of evaluating a Boolean acyclic conjunctive query over a relational database [41], computing pure Nash equilibria of certain games [37], evaluating a positive core XPath query over an XML document [39] and the uniform membership problem for nondeterministic tree automata [58].

The relationship between LogCFL and other well-known complexity classes is summarized as follows:

\[ AC^0 \subseteq NC^1 \subseteq L = SL \subseteq NL \subseteq \text{LogCFL} \subseteq AC^1 \subseteq NC^2 \subseteq P \subseteq NP \]

Here \( L \) is logspace, \( SL \) is symmetric logspace, \( NL \) is nondeterministic logspace, \( P \) is polynomial time, \( NP \) is nondeterministic polynomial time. \( AC^i \) is the class of languages recognized by a logspace-uniform circuit family of Boolean circuits of depth \( O(\log^i n) \), while \( NC^i \) denotes the class of languages recognized by a logspace-uniform circuit family of Boolean circuits of depth \( O(\log^i n) \) having bounded fan-in. For an overview of these concepts, see [41].
Since $\text{LogCFL} \subseteq \text{AC}^1 \subseteq \text{NC}^2$, the problems in LogCFL are highly parallelizable. Figure 2.11 summarizes the computational models characterizing the class LogCFL, compared with polynomial time. In these models, LogCFL is a basic computational resource restricted class, which also underlines its central role.

<table>
<thead>
<tr>
<th>PTIME</th>
<th>LogCFL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternating Turing machine using logarithmic space [68]</td>
<td>Alternating Turing machine using logarithmic space with a polynomial size witness tree [74]</td>
</tr>
<tr>
<td>Nondeterministic Turing machine using logarithmic space with an auxiliary pushdown [55]</td>
<td>Nondeterministic Turing machine using logarithmic space with an auxiliary pushdown halting in polynomial time [82]</td>
</tr>
<tr>
<td>Logspace-uniform family of unbounded Boolean circuits [68]</td>
<td>Logspace-uniform family of semi-unbounded Boolean circuits of logarithmic depth [85]</td>
</tr>
</tbody>
</table>

Figure 2.11: Computational models: PTIME vs. LogCFL

**Theorem 2.12.** ([16])

LogCFL is closed under complementation.

LogDCFL is the class of languages logspace reducible to a deterministic context-free language. Sudborough [82] characterized the classes LogCFL and LogDCFL with simultaneous resource restrictions of auxiliary pushdown machines.

**Theorem 2.13.** ([82])

$\text{NAuxPDA} - \text{SPACE}(\log(n)) - \text{TIME}(n^{O(1)}) = \text{LogCFL},$

$\text{AuxPDA} - \text{SPACE}(\log(n)) - \text{TIME}(n^{O(1)}) = \text{LogDCFL}.$

The relation between LogCFL, LogDCFL and other known classes is depicted in Figure 2.12.
Figure 2.12: Relation between LogCFL and LogDCFL.
Chapter 3

Generalized hypertree decompositions

In this chapter we study generalized hypertree decompositions. We point out that the normal form which is defined for hypertree decompositions in [43] is a useful tool to study generalized hypertree decompositions as well. As we study various subclasses of GHDS later in the thesis, we need precise means to compare decomposition methods, which are explained in the next section.

3.1 Decomposition methods

Definition 3.1. A decomposition method $M$ is a mapping that associates a set $M(H)$ of GHDS to a hypergraph $H$.

The width of a GHDS is the maximum number of edges in the label $\lambda(p)$, such that $p$ is a node of $T$. The decompositions in the set $M(H)$ associated to a hypergraph $H$ may have different widths. The minimal width of a decomposition in $M(H)$ associated to a hypergraph $H$ is characteristic for the decomposition method $M$.

Definition 3.2. The width $MW(H)$ of a hypergraph $H$ according to a decomposition method $M$ is the minimum width of a decomposition in the set $M(H)$.
In this way, we can compare decomposition methods. For two methods $M$ and $N$ we write $M \leq N$ iff for each hypergraph $H$, $MW(H) \leq NW(H)$. If $M \leq N$ and there exists a hypergraph $H$ such that $MW(H) < NW(H)$, then we write $M < N$.

The only method which does not fit into this framework is fractional hypertree decomposition ($FHD$). This method associates a triple $\langle T, \chi, \delta \rangle$ to a hypergraph, where $\langle T, \chi \rangle$ is a tree decomposition. The width of an $FHD$ is defined based on the $\delta$ labels.

We summarize here some of the well-known comparison results. The following approximation methods will be considered here. Query Decomposition ($QD$) [19] with the associated notion of query width ($QW$), hypertree decompositions ($HD$) [43], with the associated notion of hypertree width ($HW$), and spread cut decomposition ($SCD$) [22] with the associated notion of spread cut width ($SCW$) and fractional hypertree decomposition method [51].

By results$^1$ of [43, 4, 2, 22], $GHD < HD < QD$, and $GHD < SCD < QD$, while $SCD$ and $HD$ are incomparable. It was shown in [51] that $FHD < GHD$, but the computational properties of $FHD$ are unexplored. We conjecture $FHD$ is not polynomially recognizable unless $P=NP$.

Note also that the term “approximation method” is also appropriate in the complexity theoretic sense. In fact, the following result was shown in [4].

**Theorem 3.1.** ([4])
*For each hypergraph $H$, $ghw(H) \leq hw(H) \leq 3 \times ghw(H) + 1$.***

Our definitions for comparing decompositions are somewhat weaker than the definitions of [40]. There, the authors define stronger comparison criteria.

**Definition 3.3.** ([40])
A decomposition method $A$ strongly generalizes a decomposition method $B$, if there are hypergraphs, for which the width defined by the method $A$ is bounded by a constant number, but the width defined by the method $B$ can be unbounded.

---

$^1$The relation $SCD < QW$ follows from the definitions of [22].
If the method A strongly generalizes the method B, then clearly also the relation \( A < B \) holds. On the other hand \( A < B \) does not imply that A strongly generalizes B. For example \( GHD < HD \), but by Theorem 3.1, \( GHD \) is not a strong generalization of \( HD \).

Grohe and Marx [51] showed that \( FHD \) strongly generalizes \( GHD \). In this thesis we are interested comparing subclasses of \( GHD \)'s, in particular those which are more general than \( HD \)'s, thus, by Theorem 3.1 the weaker comparison criteria are sufficient for our investigations.

### 3.2 Normal form

The concept of normal form hypertree decompositions played a crucial role in the design of the effective recognition algorithm for bounded hypertreewidth hypergraphs. We review here the definitions from [43].

**Definition 3.4.** ([43]) A generalized hypertree decomposition \( \langle T, \chi, \lambda \rangle \) of a hypergraph \( H \) is in normal form, if for each vertex \( r \) of \( T \) and for each child \( s \) of \( r \), all the following conditions hold:

1. there is exactly one \([r]-\) component \( C_r \), such that \( \chi(T_s) = C_r \cup (\chi(r) \cap \chi(s)) \)
2. \( C_r \cap \chi(s) \neq \emptyset \), where \( C_r \) is the \([r]-\) component from condition 1,
3. \( \text{vertices}(\lambda(s)) \cap \chi(r) \subseteq \chi(s) \).

In a generalized hypertree decomposition \( \langle T, \chi, \lambda \rangle \) in normal form, there is a one-to-one correspondence between the subtrees at a node \( p \) of a decomposition tree and the \([p]-\) components (Condition 1 of normal form 3.4). This is a very useful property. The question naturally arises, whether the normal form concept can be used also for generalized hypertree decompositions. The following theorem gives a positive answer.
Theorem 3.2. Let $D = \langle T, \chi, \lambda \rangle$ be a $k$-width generalized hypertree decomposition of $H$. There exists a $k$-width generalized hypertree decomposition $D'$ in normal form.

Proof. The proof of the normal form theorem for hypertree decomposition in Gottlob et al. [43] (Theorem 5.4 in [43]) does not use the 4th special condition of hypertree decomposition, therefore the same proof also holds for generalized hypertree decompositions. The decomposition $D'$ can be obtained using the same normal form transformations as in [43].

It is important to note that the above theorem does not indicate that all subclasses of bounded generalized hypertreewidth hypergraphs are closed under normal form transformations, as hypertree decompositions. In fact, one can easily define subclasses, which are not closed.

For example, query decompositions are not closed: let us given a query decomposition of a hypergraph $H$, i.e. a generalized hypertree decomposition $D = \langle T, \chi, \lambda \rangle$ of $H$, such that at each node $p$ of $T$, $\text{vertices}(\lambda(p)) = \chi(p)$ holds, which is not in normal form. The normal form transformation $D'$ of $D$ has the same $\lambda$ labels as $D$, while the $\chi$ labels can change during the transformation steps. The decomposition $D'$ might contain a node $p_0$ such that $\chi(p_0) \subset \text{vertices}(\lambda(p_0))$. In this case, $D'$ is no query decomposition.

One could think that with the help of a normal form theorem for $GHD$ (Theorem 3.2), a recognition algorithm for bounded width $GHD$s could be designed in the same or similar way as for $HD$. Unfortunately this is not the case. The main difference between the two situations is that hypertree decompositions additionally have the following property.

Definition 3.5. (unique [p]-components property)
Let $\langle T, \lambda \rangle$ be a pair and a hypergraph $H$, such that $T$ is a tree and for all nodes $p$ of $T$, the set $\lambda(p)$ is a subset of $E(H)$. There exists a logspace transducer, which computes labels $\chi$ for the nodes of $T$, which associate sets of vertices of $H$ to the nodes of $T$, such that $\langle T, \chi, \lambda \rangle$ is a hypertree decomposition of $H$ in normal form.
As it was shown in [43, 44], the $\chi$ labels of a HD can be computed in the following way. For the root node $r_0$ of $T$, the label $\chi(r_0)$ is $\chi(r_0) = \text{vertices}(\lambda(r_0))$. Then, for a node $s$ with parent node $r$, the $\chi$ set is computed as $\chi(s) = \text{vertices}(\lambda(s)) \cap (\chi(r) \cup C_r)$, where $C_r$ is the unique $[r]$ component satisfying the condition 1 of normal form decompositions (see Definition 3.4).

Generalized hypertree decompositions in normal form are not known to have this property. In fact, the NP-hardness result in [45] suggests that the “unique $[p]$-components property” does not hold, under the usual complexity theoretic assumptions\(^2\).

In the following we study some important properties of GHDs in normal form. There is an important relation between the $[\text{vertices}(\lambda(p))]$- and $[\chi(p)]$-components (for short, $[p]$-components) of generalized hypertree decompositions.

**Lemma 3.1.** Let $\langle T, \chi, \lambda \rangle$ be a generalized hypertree decomposition in normal form and let $p$ be a node of a decomposition tree. For each $[\text{vertices}(\lambda(p))]$-component $C_\lambda$, there exists a $[p]$-component $C_\chi$, such that $C_\lambda \subseteq C_\chi$.

**Proof.** The lemma is a direct consequence of the 3rd condition of generalized hypertree decomposition: for each node $p$ of $T$, $\chi(p) \subseteq \text{vertices}(\lambda(p))$. \hfill $\Box$

Figure 3.1 depicts the relation of components in a schematic way. $C_1$ and $C_2$ are $[\text{vertices}(\lambda(p))]$-components, $C_\chi$ is a $[p]$-component. It has been shown in [43] (see Lemma 5.8 in [43]) –using the 4th special condition of hypertree decomposition– that in the case of normal form hypertree decompositions the $[\text{vertices}(\lambda(p))]$- and $[p]$-components coincide: a set of vertices $C$ is a $[p]$-component if and only if $C$ is a $[\text{vertices}(\lambda(p))]$-component. If we do not assume the 4th condition of hypertree decomposition, then there are cases, where $C_\lambda \subset C_\chi$.

We need the following technical lemma to show further useful properties of GHDs.

\(^2\)We assume that the complexity classes L and NP are different.
Lemma 3.2. Let \( D = \langle T, \chi, \lambda \rangle \) be a generalized hypertree decomposition of width \( k \) of the hypergraph \( H = (V, E) \), in normal form. We define the hypergraph \( H_D \) as follows: 
\[
V(H_D) = V, \quad \text{and} \quad E(H_D) = E(H) \cup \{e \cap \chi(p) \mid p \text{ is a node of } T, e \in \lambda(p)\}.
\]
Then,

1. the hypertreewidth of the hypergraph \( H_D \) is at most \( k \),

2. for each node \( p \) of \( T \), a set of vertices \( C \) of \( H \) is a \([p]\)-component of \( H \) if and only if \( C \) is a \([p]\)-component of \( H_D \).

Proof. 1. Let \( D' = \langle T, \chi', \lambda' \rangle \), where for each node \( p \) of \( T \), the set \( \lambda'(p) \) is obtained from \( \lambda(p) \) such that each edge \( e \in \lambda(p) \) is replaced by the edge \( e' \), for which \( \text{vertices}(e') = \text{vertices}(e) \cap \chi(p) \) holds. Then \( D' \) is a hypertree decomposition of \( H_D \). Clearly, it is a GHD for \( H_D \) and it also satisfies the special condition for each node \( p \) in \( T \) (in \( D' \)), because by the construction of \( D' \), the relation \( \text{vertices}(\lambda(p)) = \chi(p) \) holds (at each node \( p \) of \( T \)). Therefore \( D' \) is a hypertree decomposition of \( H_D \) of width \( k \).
2. Let $S$ be a set of vertices of $H$. Two vertices $X$ and $Y$ are $[S]$-connected in $H$, if and only if they are also $[S]$-connected in $H_D$. □

To state the next lemma, we recall the definition of the $treecomp(s)$ sets from [43].

**Definition 3.6.** ([43]) Let $D = \langle T, \chi, \lambda \rangle$ be a generalized hypertree decomposition of a hypergraph $H$ in normal form. The set $treecomp(s)$ is defined as follows.

- If $s$ is the root node of $T$, then $treecomp(s) = V(H)$,

- otherwise, let $r$ denote the parent node of $s$, the set $treecomp(s)$ is the unique $[r]$-component $C_r$, for which $\chi(T_s) = C_r \cup (\chi(r) \cap \chi(s))$, i.e. $treecomp(s) = C_r$.

**Lemma 3.3.** Let $D = \langle T, \chi, \lambda \rangle$ be a generalized hypertree decomposition of the hypergraph $H = (V, E)$ in normal form and let $r$ be the parent node of $s$ in $T$. Then, $treecomp(s) \subseteq treecomp(r)$.

**Proof.** For normal form hypertree decompositions, the containment $treecomp(s) \subseteq treecomp(r)$ holds, see Lemma 5.11 in [43]. Because of Lemma 3.2, the containment also holds for the $[p]$-components of $H$, defined by the decomposition $D$, as they are at the same time $[p]$-components in the hypergraph $H_D$. □

The above lemma again helps to understand the differences and similarities between $HD$s and $GHD$s of a hypergraph. The monotonicity of the $treecomp(s)$ also holds for hypertree decompositions in normal form, but by the coincidence of $[s]$- and $[\text{vertices}(\lambda(s))]$-components –Lemma 5.8 in [43]– for $HD$s, additionally the monotonicity of the $[\text{vertices}(\lambda(p))]$-components holds. In the case of $GHD$s we can only assume the monotonicity of the $treecomp(s)$-sets. The next lemma also helps to understand the structure of $GHD$s.

**Lemma 3.4.** Let $GHD = \langle T, \chi, \lambda \rangle$ be a generalized hypertree decomposition of the hypergraph $H$ in normal form and let $p$ be a node of the decomposition tree. If $v \in \text{vertices}(\lambda(p))$ is adjacent to the edges of at least two $[\text{vertices}(\lambda(p))]$-components, which are not contained in the same $[p]$-component, then $v \in \chi(p)$. 

43
Proof. Let $v \in \text{vertices}(\lambda(p))$ and let $e \in \text{edges}(C_e)$ and $f \in \text{edges}(C_f)$, such that $v \in \text{vertices}(e)$ and $v \in \text{vertices}(f)$, where $C_e$ and $C_f$ are two different $[\text{vertices}(\lambda(p))]$-components not contained in the same $[p]$-component. Since $e \in \text{edges}(C_e)$, $\text{vertices}(e) \not\subseteq \chi(p)$ and because of Lemma 3.3, $e$ is also not covered by any ancestor of $p$. Therefore, $e$ is covered by the $\chi$-set of some descendant node of $p$. Let $p_e$ be one of these nodes. Similarly, let us denote by $p_f$ a descendant node of the node $p$, for which $\text{vertices}(f) \subseteq \chi(p_f)$. The nodes $p_e$ and $p_f$ belong to two different subtrees rooted at $p$, because the decomposition is in normal form. Since $v \in \chi(p_e)$ and $v \in \chi(p_f)$, the connectedness condition implies that also $v \in \chi(p)$ must hold. □

3.3 Hypertree decomposition

In this section we give a new equivalent definition for hypertree decomposition. Although the new definition can be obtained using some technical steps from the original definition, the new form of the definition also contributes to a better understanding of the nature and consequences of the “special condition” of hypertree decomposition.

Figure 3.2 depicts the relation between the sets $\text{vertices}(\lambda(p))$, $\chi(p)$ and $\text{treecomp}(p)$ in a normal form hypertree decomposition. The next lemma formulates the relation precisely.

**Lemma 3.5.** Let $D = \langle T, \chi, \lambda \rangle$ be a generalized hypertree decomposition of a hypergraph $H$ in normal form. $D$ is a hypertree decomposition if and only if for each node $p$ of $T$, $\text{vertices}(\lambda(p)) \cap \text{treecomp}(p) \subseteq \chi(p)$.

Proof. (Only if) Let $V_p$ denote $V_p = \text{vertices}(\lambda(p))$ and let $r$ be the parent node of $p$. By condition 3 and 4 of hypertree decomposition (Definition 2.4) $\chi(p) = V \cap \chi(T_p)$ and by condition 1 of normal form (Definition 3.4) $V_p \cap \chi(T_p) = V_p \cap (\text{treecomp}(p) \cup (\chi(p) \cap \chi(r))) \subseteq (V_p \cap \text{treecomp}(p)) \cup (V_p \cap (\chi(p) \cup \chi(r))) = \chi(p)$, therefore $V_p \cap \text{treecomp}(p) \subseteq \chi(p)$. 44
(If) Let $D$ be a generalized hypertree decomposition in normal form. Because of the condition of the lemma, for the root node $p$, since $\text{treecomp}(p) = V(G)$, the $\chi(p)$ set consists of all vertices in $\lambda(p)$, i.e. $\chi(p) = \text{vertices}(\lambda(p))$.

By the normal form condition, for each (non-root) node $p$ of $T$, $\chi(T_p) = \text{treecomp}(p) \cup (\chi(p) \cap \chi(r))$, where $r$ is the parent node of $p$ in $T$. Let us use the following notation: $V_p = \text{vertices}(\lambda(p))$. Now, by the normal form conditions, $V_p \cap \chi(T_p) \subseteq (V_p \cap \text{treecomp}(p) \cup (V_p \cap (\chi(p) \cap \chi(r)))) \subseteq V_p \cap (\text{treecomp}(p) \cup \chi(r))$.

By the 3rd condition of normal form $V_p \cap \chi(p) \subseteq \chi(p)$ and by the condition of this lemma $V_p \cap \text{treecomp}(p) \subseteq \chi(p)$, therefore also $V_p \cap (\text{treecomp}(p) \cup \chi(r)) \subseteq \chi(p)$. On the other hand, by the 3rd condition of generalized hypertree decomposition, $\chi(p) \subseteq V$ and clearly, $\chi(p) \subseteq \chi(T_p)$, we can conclude that also $\chi(p) \subseteq V_p \cap \chi(T_p)$ holds. Thus, for a non-root node $\chi(p) = V_p \cap (\text{treecomp}(p) \cup \chi(r))$.

We have shown that

- for the root node $p_0$ of $T$, $\chi(p_0) = \text{vertices}(\lambda(p_0))$, and
- for each non-root node $p$ of $T$, $\chi(p) = V_p \cap (\text{treecomp}(p) \cup \chi(r))$.

Using Lemma 1 in [44], we can conclude that $D$ is a hypertree decomposition. □

Figure 3.2: The relation between the sets $\text{vertices}(\lambda(p))$, $\chi(p)$ and $\text{treecomp}(p)$ in a normal form hypertree decomposition.
Chapter 4

Subedge-based decompositions

4.1 Motivation and definitions

The intractability results on generalized hypertree width motivate further research on tractable decomposition methods, which are more general than hypertree decompositions and approximate generalized hypertree decompositions (Figure 4.1). In this chapter we show that indeed there are such methods. We not only give an example of such a decomposition, but we show that each such method is basically a combination of a method to add (sub-hyper-) edges to the hypergraph with hypertree decomposition.

The following example shows that by adding subedges to a hypergraph associated to a query, we essentially do not change the query evaluation problem.

Example 4.1. Consider the relational database and query from example 2.1.

\[
\text{plays(Person,Film,Role)} \\
\text{directs(Person,Film)} \\
\text{relative(Person,Person)}
\]

\[
Q: \text{ans} \leftarrow \text{plays}(A,F,R),\text{directs}(D,F),\text{relative}(A,D).
\]

Now consider the following modified version of this example, where we add a new relation play $'$ to the database, which is defined as...
Figure 4.1: Is there a decomposition method more general than \(HD\), but still tractable?

\[
\text{plays}'(\text{Person,Film}) = \Pi_{\text{Person,Film}} \text{plays}(\text{Person,Film,Role}) \quad \text{and we also add a new atom to our query.}
\]

\[
\begin{align*}
\text{plays}'(\text{Person,Film}) \\
\text{plays}(\text{Person,Film,Role}) \\
\text{directs}(\text{Person,Film}) \\
\text{relative}(\text{Person,Person})
\end{align*}
\]

\(Q': \text{ans} \leftarrow \text{plays}'(A,F),\text{plays}(A,F,R),\text{directs}(D,F),\text{relative}(A,D).\)

Clearly, in the above example, the query \(Q\) has a nonempty result over the database \(DB\) if and only if the query \(Q'\) has a nonempty result over \(DB'\). We can formulate this observation also in a more general form.

**Proposition 4.1.** Let \(Q : \text{ans} \leftarrow r_1(X_1, \ldots, X_n), \ldots, r_m(X_{m_1}, \ldots, X_{m_n})\) be a Boolean conjunctive query over the database \(DB\). The query \(Q\) has a nonempty result over the database \(DB\) if and only if the query \(Q'\) has a nonempty result over \(DB'\), where \(Q'\) and \(DB'\) are obtained from \(Q\) and \(DB\) respectively, in the following way. Let \(r_1'(A_1, \ldots, A_{n-1})\) be the projection of \(r_1\) to the attributes \(A_1, \ldots, A_{n-1}\). \(DB'\) is obtained from \(DB\) by adding \(r_1'(A_1, \ldots, A_{n-1})\) as a new table. The query \(Q'\) is obtained from \(Q\) by adding a new query atom \(r_1'(X_1, \ldots, X_{n-1})\) with variables \(X_1, \ldots, X_{n-1}\).
Proof. If \( Q \) has a nonempty result over \( DB \) then we can easily construct also a result for \( Q' \) over \( DB' \) and vice versa. \( \square \)

As adding subedges does not change the problem essentially, we can try to add subedges in a systematic way, such that the resulting hypergraph has better and better decompositions. The following proposition, which is merely a simple observation, sets the stage for the considerations in this section.

**Proposition 4.2.** Let \( H \) be a hypergraph and let \( D = \langle T, \chi, \lambda \rangle \) be a \( GHD \) for \( H \). Then \( D' = \langle T, \chi, \lambda' \rangle \), where \( \lambda'(p) = \{ e \cap \chi(p) \mid e \in \lambda(p) \} \), for each node \( p \) of \( T \) is a \( HD \) of \( H \cup \{ e \cap \chi(p) \mid p \in T, e \in \lambda(p) \} \). Furthermore, the width of \( D' \) is at most the width of \( D \).

**Proof.** The proposition follows from the definitions of \( GHD \) and \( HD \). \( \square \)

Proposition 4.2 explains, at least to some extent, the relationship between \( HD \) and \( GHD \). More importantly, it opens a systematic way to find tractable decomposition methods as will be detailed below.

Before we dive into that, let us have a closer look at decomposition methods. Recall\(^1\) that a decomposition method \( M \) associates a set \( M(H) \) of allowed \( GHDs \) to a hypergraph \( H \). In principle, we would be interested in methods that can be implemented by tractable algorithms, but as the experience with hypertree decompositions shows, we cannot expect algorithms whose running time is polynomial independent of the parameter \( k \), the width of the decomposition. Thus, we include the width parameter in our definitions.

**Definition 4.1.** We say an algorithm \( A \) implements a decomposition method \( M \) if for each fixed value of \( k \) and for each input hypergraph \( H \), \( A \) outputs a \( GHD \) from \( M(H) \) of width at most \( k \), if there exists such a decomposition, otherwise it outputs “fail”.

Now we turn to the particular decomposition methods we are interested in. We call a subset of a hyperedge \( e \) of a hypergraph \( H \) a subedge of \( H \). Informally,

---

\(^1\)See Definition 3.1.
each function \( f \) mapping a hypergraph \( H \) to a set of subedges of \( H \) induces a decomposition method that can be computed in the following way.

1. Compute \( f(H) \),
2. compute a minimal hypertree decomposition \( D \) of \( H \cup f(H) \).

As (2) is only feasible for each fixed \( k \), it makes sense to allow \( f \) to depend on the given \( k \) as well. A subedge function associates a set of subedges to a pair \((H, k)\). To avoid technical complications, we further require that subedge functions be monotone.

**Definition 4.2.** A subedge function is a function \( f \) that associates a set of subedges of \( H \) to each pair \((H, k)\) and for each \( i < j \), \( f(H, i) \subseteq f(H, j) \).

**Definition 4.3.** Let \( D = \langle T, \chi, \lambda \rangle \) be a HD of a hypergraph \( H \cup f(H, k) \) and \( D' = \langle T, \chi', \lambda' \rangle \) be a GHD of \( H \). We say that \( D' \) covers \( D \) if, for each node \( p \) of \( T \), and for each \( e \in \lambda(p) \), there exists an edge \( e' \in \lambda'(p) \), such that \( e \) is a subedge of \( e' \), i.e. \( \text{vertices}(e) \subseteq \text{vertices}(e') \).

If we add subedges to a hypergraph \( H \), then the hypertreewidth of the resulting hypergraph \( H' \) is at most the hypertreewidth of \( H \), i.e. \( hw(H') \leq hw(H) \). Thus, for a given hypergraph \( H \), and subedge function \( f \), the hypertreewidth of \( H \cup f(H, i) \) is monotonically decreasing (in \( i \)). The hypertreewidth of \( H \cup f(H, i) \) depends on the structure of \( H \) and the definition of the subedge-function \( f \). Figure 4.2 depicts some possible patterns.

We would like to define a decomposition method using subedge-functions. In our definition we would like to relate GHDs of a hypergraph \( H \) to HDs of another hypergraph, namely \( H \cup f(H, i) \), for some \( i \). As Figure 4.2 shows, we have to be careful with the choice of \( i \). Which of the hypergraphs \( H \cup f(H, i) \) should we use in the definition of the subedge-based decomposition? To avoid further technical complications, we define \( M_f(H) \) in such a way, that for testing bounded \( M_f \)-width, i.e. testing whether \( M_f W(H) \leq k \), we have to find the smallest \( k \), for which \( hw(H \cup f(H, k)) \leq k \).

\(^2\)Our definitions satisfy this requirement, see Corollary 4.1.
Figure 4.2: The hypertreewidth of the hypergraph $H \cup f(H, i)$: some possible patterns.

**Definition 4.4.** Let $H$ be a hypergraph and let $f(H, k)$ be a subedge function. We define a decomposition method $M_f(H)$ as follows. $M_f(H)$ is the set of all GHDS $D'$ of $H$ for which there exists a $k$ such that $k \leq |D'|$ and there exists a HD $D$ of the hypergraph $H \cup f(H, k)$, such that $D'$ covers $D^3$. We call a decomposition method of the form $M_f$ subedge-based.

Thus a subedge-based decomposition of $H$ with subedge function $f$ is related to a hypertree decomposition of $H \cup f(H, k)$ (Figure 4.3). If $D'$ is a subedge-based decomposition of $H$ based on the subedge function $f$, i.e. $D' \in M_f(H)$ and $D'$ covers a hypertree decomposition $D$ of $H \cup f(H, k)$, then the width of $D'$ is at least $k$, i.e. $k \leq |D'|$.

---

$|D|$ denotes the width of the decomposition $D$. 

50
According to this definition (Definition 4.4), the hypertree decomposition method is subedge-based, as it is defined by the function: for each \( i \), \( f(H, i) = \emptyset \). On the other extreme, GHD is a subedge-based decomposition, too. In particular, GHD is equal to \( M_{f^+} \), where for each \( H \) and \( k \), \( f^+(H, i) = \text{subedges}(H) \). A related remark was made by Adler in [2].

The latter example shows that, in general, \( f(H, k) \) does not need to be of polynomial size. Nevertheless, we are interested in tractable methods. We call a subedge function \( f \) polynomially computable (logspace computable) if for each fixed \( k \), \( f(H, k) \) can be computed in polynomial time (logarithmic space).

**Lemma 4.1.** (a) If \( f \) is polynomially computable, then, for each fixed constant \( k \), it can be decided in polynomial time whether \( M_f W(H) \leq k \) and there is a tractable algorithm \( A_f \) that implements \( M_f \).

(b) If \( f \) is logspace computable, then deciding whether \( M_f W(H) \leq k \) is in the parallel complexity class \( \text{LogCFL} \).

**Proof.** For (a), given \( H \) and \( k \), \( A_f \) first computes \( f(H, k) \) and then uses the algorithm of [43] and [42] to compute a HD \( D \) of width \( i \leq k \) (\( i \) can be any number

---

\(^4\)See Lemma 6.1 in [2].

\(^5\)The algorithm in [43] is a decision algorithm for \( hw(H) \leq k \). Gottlob et al. [42] have shown
such that $1 \leq i \leq k$ for $H \cup f(H, k)$, if one exists. Note that for each subedge $e$ used in $D$, there is an edge $e'$ of $H$ with $e \subseteq e'$. Thus, replacing each such $e$ by a respective $e'$ yields a $GHD D'$ of width $i$ for $H$.

Note that $D'$ might not be in $M_f(H)$, because by Definition 4.4 $M_f(H)$ contains $GHD$s of width at least $k$, and maybe $k \leq |D'|$ does not hold. Thus, let $p$ be a node of the underlying tree $T$ of $D'$ with $|\lambda(p)| = i$ and let $e_1, \ldots, e_{k-i}$ be hyperedges of $H$ which are not yet in $|\lambda(p)|$. By adding these edges to $\lambda(q)$ for each node $q$ of $T$ we get a $GHD$ of width $k$, which is the output of $A_f$. As $A_f$ works in polynomial time, the decision problem can be answered in polynomial time as well. The case of (b) is similar: one only has to carefully compose the logspace algorithm to compute $f(H, k)$ with the LogCFL check [43, 41] whether the hypertree width of $H \cup f(H, k)$ is $\leq k$ (in the standard way known from complexity theory). Since LogCFL is closed under logspace reductions, we can conclude that deciding $M_fW(H) \leq k$ is feasible in LogCFL.

From the proof of Lemma 4.1 we can conclude:

**Corollary 4.1.** For a hypergraph $H$, $M_fW(H)$ is the smallest $k$ for which $HW(H \cup f(H, k)) \leq k$.

For reference in the next section we state the following, which can be shown by a similar argument.

**Theorem 4.1.** Let $A$ and $B$ be two subedge–defined decomposition methods, defined by the functions $f_A$ and $f_B$, respectively. If for all positive integers $i$, $f_A(i, H) \subseteq f_B(i, H)$, then $BW(H) \leq AW(H)$.

**Proof.** Similar to the proof of Lemma 4.1. \qed

---

6If no such edges exist, $H$ has less than $k$ hyperedges and $M_fW(H) \leq k$ holds trivially by definition of $M_f$. 

---

that computing a certificate, which is in this case a hypertree decomposition of $H$ of width $k$, is feasible in functional LogCFL.
We have seen that decomposition methods $M_f$ with tractable $f$ lead to tractable GHD-computations. We next show that, on the other hand, each tractable decomposition method is basically of the form $M_f$.

**Theorem 4.2.** For each decomposition method $M$ which can be implemented by an algorithm $A$, which runs in polynomial time, there is a polynomially computable subedge function $f$ such that $M_f \leq M$.

**Proof.** Let $M$ and $A$ be as stated. Given a hypergraph $H$ and a number $i$ ($1 \leq i$), let $D = (T, \chi, \lambda)$ be the GHD of width $i$ for $H$ computed by $A$. Let $D' = (T, \chi, \lambda')$ be defined as in Proposition 4.2. Then we define $g(H, i) = \bigcup_p \lambda'(p)$, where $p$ ranges over all nodes of $T$. As $A(H, i)$ can be computed in polynomial time, $g(H, i)$ is polynomial.

Furthermore, by Proposition 4.2, $D'$ has width $\leq i$ and is in $M_i(H)$. As this holds, for each $k$, the function $f(H, k) = \bigcup_{i=1}^k g(H, i)$, is monotone. Thus it is a subedge-function and we can conclude that $M_f W(H) \leq M(H)$.

Of course, the function $f$ in the proof of Theorem 4.2 depends on the ability of already computing a GHD. Thus, the reader might get the impression that the detour through the subedge-function $f$ is not very useful. Nevertheless, in the next chapter (Chapter 5) we exhibit a polynomial subedge function $f$ which is defined entirely in terms of $H$ and does not involve the construction of a decomposition.

Let us make the following observation.

**Lemma 4.2.** Let $f$ be a subedge-function and let $H$ be a hypergraph. $H$ is acyclic if and only if $M_f W(H) \leq 1$.

**Proof.** If the hypergraph $H$ is acyclic, then it has a join tree $(T, \lambda)$. We can easily obtain a join tree for $H \cup f(H, 1)$, too, in the following way. For each subedge $e \in f(H, 1)$, such that $e$ is a subedge of $e'$, there is a node $p$ in $T$, where $e'$ is covered. Thus we can add a child node $p_e$ to $p$, with $\lambda(p_e) = \{e\}$, $\chi(p_e) = \text{vertices}(e)$. In this way we obtain a join tree for $H \cup f(H, 1)$, thus it is acyclic and therefore also $M_f W(H) \leq 1$ holds.
If $M_f W(H) \leq 1$, then $hw(H \cup f(H, 1)) \leq 1$. Therefore $ghw(H) \leq 1$, thus $H$ is acyclic. □

4.2 Game-theoretic characterization

Hypergraphs with $k$-bounded hypertreewidth have an intuitive characterization in terms of monotone R&M games. In [2], Adler studied non-monotone R&M games. She observed that the class of hypergraphs characterized by non-monotone games with $k$ marshals is larger than the class of hypergraphs with hypertreewidth at most $k$. She also observed, that this class is not identical with the class of hypergraphs with $k$-bounded generalized hypertreewidth.

In this Section we point out that hypergraphs with bounded width, defined by a subedge-based decomposition, also have a game theoretic characterization. Although the game-theoretic characterization for this class follows easily from the definitions, we think that it helps to clear up the picture of decompositions and to understand the relation between monotone and non-monotone games.

Theorem 4.3. Let $H$ be a hypergraph and $f$ a subedge function. A hypergraph has $M_f$ width at most $k$ if and only if $k$ marshals have a winning strategy in the monotone R&M game on the hypergraph $H \cup f(H, k)$.

Proof. If the decomposition $D'$ is in $M_f(H)$ and has width $k$, then by Definition 4.4 there exists a hypertree decomposition $D$ of $H \cup f(H, k)$, such that $D'$ covers $D$. Thus, the width of $D$ is at most $k$. Therefore, by the results in [44], there must be a game on $H \cup f(H, k)$, where $k$ marshals have a winning strategy.

On the other hand, if $k$ marshals have a winning strategy on the hypergraph $H \cup f(H, k)$, then the hypergraph $H \cup f(H, k)$ has a HD of width $k$. In this case a subedge-based decomposition of $H$ of width $k$ can be constructed as it is explained in the proof of Lemma 4.1. □

As HD and GHD are also subedge based decompositions, they are also characterized by games. In the case of HD, the game is identical with the monotone
R&M game. Note that Theorem 4.3 gives a characterization for $GHD$ in terms of a monotone game.

Theorem 4.3 refers to a game on the hypergraph $H \cup f(H, k)$, but there is a straightforward way to define an equivalent game on the hypergraph $H$. The rules are the following. A robber who occupies a vertex is allowed to move on the edges and tries to escape the $k$ marshals who can occupy either edges of $H$ or subedges from the set $f(H, k)$. Marshals can move on helicopters between positions. If the marshals move from position $M$ to position $M'$ then the robber is not allowed to move through the vertices of $M \cap M'$.

In the R&M game for hypertreewidth, the $k$ marshals occupy entire hyperedges, thus we can represent the marshals’ position in a succinct way, by $k$ edges, thus in logarithmic space. In the game for generalized hypertreewidth the marshals can occupy any possible subedges, thus such a succinct representation is unlikely to exist. Intuitively, the lack of such a succinct representation of marshals positions makes it difficult to design a tractable recognition algorithm for bounded width $GHD$s.

Figure 4.4 depicts a possible winning strategy for two marshals on the hypergraph of Example 1.1. The marshals use a subedge to win the game.
The marshals use a subedge at this position of the game.

Figure 4.4: A winning strategy for two marshals in the monotone subedge-based R&M game
In this section we give a specific subedge defined decomposition, called “component hypertree decomposition”, that strictly generalizes both hypertree decomposition [43] and spread cut decomposition [22] and is also tractable. The definition of the subedge function \( f^C \) of component hypertree decomposition is based only on some structural attributes of the hypergraph, namely on the intersection properties of specific edges and components.

### 5.1 Definitions

We need some additional technical definitions, before we define the subedge function of component hypertree decomposition.

**Definition 5.1.** Let \( M \) be a set of edges of the hypergraph \( H \). We define \( \text{prop}(e, M) \), the proper part of an edge related to \( M \) as \( \text{prop}(e, M) = e \setminus \bigcup_{e' \in M, e \neq e'} e' \).

**Example 5.1.** Let us consider the set \( M \), a set of edges of a hypergraph depicted in Figure 5.1. \( M \) contains three edges, \( a(A, B, C) \), \( b(C, D, E, F) \), \( c(F, G, H) \). Then, \( \text{prop}(b, M) = \{D, E\} \).
Definition 5.2. Let $M$ be a set of edges of the hypergraph $H$ and let $e$ be an edge in $M$. We define the set $\text{internal}(e, M) = \{v \mid v \in \text{vertices}(e) \text{ and there exists no } [\text{vertices}(M)]\text{-component } C \text{ of } H, \text{ such that } v \in \text{vertices}(\text{edges}(C)) \}.$

Example 5.2. Let us consider the set $M = \{e(A, B, C), f(C, D, E)\}$, depicted in Figure 5.2. Then, $\text{internal}(e, M) = \{B\}$.

Definition 5.3. Let $H$ be a hypergraph, let $M$ be a set of edges of $H$, let $e$ be an edge in $M$ and let $C$ be a $[\text{vertices}(M)]$-component. The function $\text{elim}(M, C, e)$ associates a set containing the following three subedges to a triple $(M, C, e)$

1. $e \cap \text{vertices}(\text{edges}(C))$,  
2. $\text{prop}(e, M) \cap \text{vertices}(\text{edges}(C))$,  
3. $\text{internal}(e, M)$.
Now, we are prepared to define the subedge function of the component hypertree decomposition.

**Definition 5.4.** Let $H$ be a hypergraph and let $k$ ($0 \leq k$) be an integer. We define the subedge function $f^C$ as:

$$f^C(H, k) = \{ e \setminus e' \mid M \text{ is a set of } \leq k \text{ hyperedges of } H,$$

$$e \in M,$$

$$D \text{ is a } \{\text{vertices}(M)\}-\text{component,}$$

$$\text{and } e' \in \text{elim}(M, D, e)\}.$$

The decomposition method $M_{f^C}$ will be referred to as component hypertree decomposition (CHD).

According to our definition, the generalized hypertree decomposition of the hypergraph of Example 1.1 in Figure 1.2 a) is a component hypertree decomposition. A hypertree decomposition of the hypergraph $H \cup f^C(H, 2)$ is depicted in Figure 5.3.

![Figure 5.3: Hypertree decomposition of width 2 of the hypergraph $H \cup f^C(H, 2)$, $e'_2(v_3, v_9) \subseteq e_2$, $e'_3(v_3, v_{10}) \subseteq e_3$.](image)

Let us note that for fixed $k$, the set $f^C(H, k)$ is computable using only logarithmic space. In this case, the sets $M$ and $N$ are of constant size $k$, the $\{\text{vertices}(M)\}$-components can be represented also in logarithmic space, and all of the required computations (computing connected components, intersections and difference of

---

*A generalized hypertree decomposition of width 2 is depicted in Figure 1.2.*
sets) is feasible in logspace, see e.g. [41]. Therefore, by Lemma 4.1, deciding whether for a fixed constant $k$, a given hypergraph $H$ has component hypertreewidth at most $k$, is feasible in LogCFL.

### 5.2 Comparison with other tractable decompositions

In this section we show that component hypertree decompositions strictly generalize both hypertree decompositions [43] and spread cut decompositions [22]. First we recall the definition of spread cut decomposition from [22].

**Definition 5.5.** ([22]) A normal form \(^2\) generalized hypertree decomposition $\langle T, \chi, \lambda \rangle$ of a hypergraph $H$ is called spread cut decomposition (SCD) if additionally the following conditions hold:\(^3\)

1. for each node $p$ of $T$, each $[p]$-component meets –i.e. has non-empty intersection with– at most one $[\text{vertices}(\lambda(p))]$-component,

2. for each node $p$ of $T$, for all pairs of edges $e_1, e_2 \in \lambda(p)$, such that $e_1 \neq e_2$, $e_1 \cap e_2 \subseteq \chi(p)$.

3. for each node $p$ of $T$, for each edge $e \in \lambda(p)$,

   - (a) either $\forall v \in \text{internal}(e, \lambda(p)), v \in \chi(p)$,
   - (b) or $\forall v \in \text{internal}(e, \lambda(p)), v \notin \chi(p)$ and for every $[\text{vertices}(\lambda(p))]$-component $C$, $\text{vertices}(\text{edges}(C) \cap e) \subseteq \chi(p)$.

We will also use the following terminology from [22]. A **guarded block** of a hypergraph $H(V,E)$ is a pair $\langle \lambda, \chi \rangle$, such that $\lambda$ is a subset of edges of $E(H)$

\(^2\)The original definition in [22] does not ensure the existence of a tractable recognition algorithm, see [21], thus we assumed that the decomposition is in normal form. The authors of [22] defined a new tractable variant of spread cut decomposition [21]. We compare component hypertree decompositions with this new variant in Chapter 6.

\(^3\)Condition 3 follows from the “canonical form” theorem, proven in [22] (Theorem 7.6). We find it convenient to include this condition in the definition.
and $\chi$ is a subset of vertices of the edges in $\lambda$. A guarded block $\langle \lambda, \chi \rangle$ covers a hyperedge $e$, if $\text{vertices}(e) \subseteq \chi$. A set $S$ of guarded blocks of $H$ is called guarded cover for $H$ if each hyperedge of $H$ is covered by some block in $S$. A guarded block $\langle \lambda, \chi \rangle$ has unbroken components if every $[p]$-component meets –i.e. has non-empty intersection with– at most one $[\text{vertices}(\lambda(p))]$-component.

**Lemma 5.1.** Let $\langle T, \chi, \lambda \rangle$ be a spread cut decomposition of a hypergraph $H$. Let $p$ be a node of $T$. For each $e \in \lambda(p)$, exactly one of the following conditions is true:

1. $e \setminus \chi(p) = \text{internal}(e, \lambda(p))$,

2. there exists a unique $[\text{vertices}(\lambda(p))]$-component $C_e$, such that $e \setminus \chi(p) = \text{prop}(e, \lambda(p)) \cap \text{vertices}(\text{edges}(C_e))$.

**Proof.** Assume that $e \setminus \chi(p)$ contains a vertex from $\text{internal}(e, \lambda(p))$. Then, by condition 3 of Definition 5.5, $e \setminus \chi(p) = \text{internal}(e, \lambda(p))$. Now, assume, $e \setminus \chi(p)$ does not contain any internal vertex, then let us assume indirectly that $v, w \in \text{vertices}(e) \setminus \chi(p), (v \neq w)$ and there are two different $[\text{vertices}(\lambda(p))]$-components $C$ and $D$, such that $v \in \text{vertices}(\text{edges}(C))$ and $w \in \text{vertices}(\text{edges}(D))$. Then $v$ and $w$ are $[p]$-connected, since $\{v, w\} \subseteq \text{vertices}(e) \setminus \chi(p)$. So, there exist two different vertices $v_C \in C$ and $v_D \in D$, such that $v_C$ and $v_D$ are $[p]$-connected to $v$ and $w$, respectively, therefore also $v_C$ and $v_D$ are also $[p]$-connected. From this follows that the $[p]$-component containing $v$ and $w$ meets more than one $[\text{vertices}(\lambda(p))]$-components. Contradiction. \hfill $\Box$

**Definition 5.6.** For $M, H, C$ and $e$ as in Definition 5.3 let $\text{elim}^*(M, C, e)$ be defined as in Definition 5.3 except that we only associate two subedges to a triple $(M, C, e)$, namely those mentioned in points 2 and 3 in Def. 5.3.

**Definition 5.7.** We define the subedge function $f^*$ as $f^*(H, k) = \{e \setminus e' \mid M \text{ is a set of at most } k \text{ hyperedges of } H, e \in M, C \text{ is a } [\text{vertices}(M)]\text{-component, } e' \in \text{elim}^*(M, C, e)\}$.
Note that for each hypergraph $H$ and for each positive $k$, $f^*(H,k) \subseteq f^C(H,k)$.

**Lemma 5.2.** $SC \leq M_f$.

*Proof.* Let $D = \langle T, \chi, \lambda \rangle$ be a spread cut decomposition of $H$ of width $k$. It is sufficient to show that for each node $p$ of $T$ and for each edge $e$ in $\lambda(p)$, $e \cap \chi(p) \in \text{subedges}(H \cup f^*(H,k))$. But this follows from the definition of $f^*$ (Definition 5.7) and Corollary 4.1 and Lemma 5.1. □

**Theorem 5.1.** $CHD < HD$ and $CHD < SCD$.

*Proof.* Clearly, by Corollary 4.1, $CHD \leq HD$. For hypergraph $H_0$ in the introduction, see Figure 1.1, $HW(H) = 3$, $CHW(H) = 2$, therefore $CHD < HD$.

Let us first prove that $CHD \leq SCD$. Given that for each hypergraph $H$ and for each positive $k$, $f^*(H,k) \subseteq f^C(H,k)$, by Corollary 4.1 and Lemma 5.2, $CHD = M_{f^c} \leq M_f \leq SC$.

To show that $CHD < SCD$, we give examples of hypergraphs whose component hypertreewidth is strictly smaller than their spread cut width. For the hypergraph $\mathcal{H}$ of Example 5.3, $CHW(\mathcal{H}) = 5$, $SCW(\mathcal{H}) = 6$. For proofs, see Section 5.3. □

### 5.3 Separating examples

In this section we define hypergraphs having strictly smaller component hypertreewidth than spread cut width. The examples are adaptations of examples from [2], [4], [21], for our purposes.

#### 5.3.1 Hypergraph with $chw(H) = 5$ vs. $scw(H) = 6$

**Example 5.3.** The hypergraph $\mathcal{H} = (V, E)$ is defined as follows. The vertices of $\mathcal{H}$ are the “ground” vertices $\{A, B, C, D, E, F, A_1, B_1, C_1, D_1, E_1, F_1\}$ and the 32 “balloon” vertices, represented as stars in the figure (Figure 5.4). Each balloon
vertex is connected by an edge to each ground vertex. All other edges are depicted in the figure. More precisely,

\[\mathcal{B} = \{G_{ij} \mid i, j \in \{1, 2, 3, 4\}\} \cup \{F_{ij} \mid i, j \in \{1, 2, 3, 4\}\}\]

\[\mathcal{V} = \mathcal{B} \cup \{A, B, C, D, E, F, A_1, B_1, C_1, D_1, E_1, F_1\}\]

\[\mathcal{E} = \{(g, p) \mid g \in \mathcal{B}, p \in \mathcal{V} \setminus \mathcal{B}\} \cup \{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4\}\]

\[\cup\{(A, A_1), (A, B), (B, C), (A, D), (C, D), (D, E), (D, F), (A_1, B_1), (B_1, C_1),\]

\[\{A_1, D_1\}, \{C_1, D_1\}, \{D_1, E_1\}, \{D_1, F_1\}\},\] where

\[a_1 = \{G_{11}, G_{12}, G_{13}, G_{14}, F_{11}, F_{12}, F_{13}, F_{14}, F_1\},\]

\[a_2 = \{G_{21}, G_{22}, G_{23}, G_{24}, F_{21}, F_{22}, F_{23}, F_{24}, F_1\},\]

\[a_3 = \{G_{31}, G_{32}, G_{33}, G_{34}, F_{31}, F_{32}, F_{33}, F_{34}, E\},\]

\[a_4 = \{G_{41}, G_{42}, G_{43}, G_{44}, F_{41}, F_{42}, F_{43}, F_{44}, E\},\]

\[b_1 = \{G_{11}, G_{21}, G_{31}, G_{41}, F_{11}, F_{21}, F_{31}, F_{41}, F\},\]

\[b_2 = \{G_{12}, G_{22}, G_{32}, G_{42}, F_{12}, F_{22}, F_{32}, F_{42}, F\},\]

\[b_3 = \{G_{13}, G_{23}, G_{33}, G_{43}, F_{13}, F_{23}, F_{33}, F_{43}, E_1\},\]

\[b_3 = \{G_{14}, G_{24}, G_{34}, G_{44}, F_{13}, F_{23}, F_{33}, F_{44}, E_1\}.\]

The vertices in \(\mathcal{B}\) are called balloon vertices, the other vertices are called ground vertices. For simplicity, we use the following notation \(g_{ij} = \{F_{ij}, G_{ij}\}\). The hypergraph is depicted in Figure 5.4.

We need some technical lemmas to compare the spread cut width and component hypertreewidth of \(\mathcal{H}\).

**Lemma 5.3.** Let \(\langle T, \chi, \lambda \rangle\) be a generalized hypertree decomposition of the hypergraph \(\mathcal{H}\) in normal form. If, for a node \(p\) of \(T\), \(\mathcal{B} \not\subseteq \chi(p)\), then \(p\) is a leaf node.

**Proof.** Since every vertex in \(V(\mathcal{H})\) is directly connected to every vertex in \(\mathcal{B} \setminus \chi(p)\), there is only one \([p]\)-component at node \(p\). Thus, since the decomposition is in normal form, there is only one branch of \(T\) out of the node \(p\), which must contain the parent node of \(p\). \(\square\)
Figure 5.4: $hw(H) = 6$, $scw(H) = 6$, $ghw(H) = 5$, $chw(H) = 5$

**Lemma 5.4.** Let $(T, \chi, \lambda)$ be a generalized hypertree decomposition of the hypergraph $H$ in normal form. If, for a node $p$ of $T$, $\mathcal{B} \not\subseteq \chi(p)$, then the decomposition $(T \setminus \{p\}, \chi, \lambda)$, i.e. the decomposition $D$ without the node $p$ is also a GHD of $H$.

**Proof.** By lemma 5.3, the node $p$ must be a leaf node. Let us assume indirectly that there is an edge $e$, which is only covered by $\chi(p)$, thus removing the node $p$ from the decomposition, the condition 1 of generalized hypertree decomposition (Definition 2.5) is violated. Note that if we remove a leaf node from a GHD, then the resulting tree may only violate the first condition of the definition of GHD, the second and third conditions must remain valid.

Since $e$ is covered only by $p$, there is a vertex $X_e \in vertices(e)$, which is not covered by the parent node $r$ of $p$, i.e. $X \notin \chi(r)$. Since $\mathcal{B} \not\subseteq \chi(p)$, there is a vertex $Y \in \mathcal{B} \setminus \chi(p)$. By the definition of $H$, there exists a binary edge $(X, Y)$ in $H$. This binary edge $(X, Y)$ is not covered by $\chi(p)$. This is a contradiction, since $X \in \chi(p)$, $X \notin \chi(r)$, and there is a node $s$ (different from $p$ and $r$), where the edge $(X, Y)$ is covered, i.e. $X \in \chi(s)$, therefore the connectedness condition of GHD is
violated.

Thus, by the above lemmas (Lemma 5.3 and 5.4) we can assume without loss of generality that in a GHD of $\mathcal{H}$ (in normal form), for all nodes $p$ of $T$, the containment $\mathcal{B} \subseteq \chi(p)$ holds.

**Lemma 5.5.** Let $\langle T, \chi, \lambda \rangle$ be a generalized hypertree decomposition of the hypergraph $\mathcal{H}$ of width at most 5 in normal form. If, for a node $p$, $\mathcal{B} \subseteq \chi(p)$, then either $\{a_1, a_2, a_3, a_4\} \subseteq \lambda(p)$ or $\{b_1, b_2, b_3, b_4\} \subseteq \lambda(p)$.

**Proof.** If $\{a_1, a_2, a_3, a_4\} \not\subseteq \lambda(p)$ and $\{b_1, b_2, b_3, b_4\} \not\subseteq \lambda(p)$ then $\lambda(p)$ contains at most 3 edges from the set $\{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4\}$. They cover at most $3 \times 8 + 2 = 26$ vertices from $\mathcal{B}$. Each of the remaining edges covers at most one vertex from $\mathcal{B}$. Contradiction. □

**Lemma 5.6.** There exists no generalized hypertree decomposition of the hypergraph $\mathcal{H}$ of width 4.

**Proof.** Indirectly, let us assume, that there exists a generalized hypertree decomposition $D = \langle T, \chi, \lambda \rangle$ of width 4. By Theorem 3.2, we can assume that $D$ is in normal form. By the Lemmas 5.3 and 5.4 we can assume without loss of generality that for all nodes $p$ of $T$ of a GHD of $\mathcal{H}$, the containment $\mathcal{B} \subseteq \chi(p)$ holds. This is only possible for a decomposition of width 4, if for all nodes $p$ of $T$, either $\lambda(p) = \{a_1, a_2, a_3, a_4\}$ or $\lambda(p) = \{b_1, b_2, b_3, b_4\}$. Then the edge $(B, C)$ is not contained in the $\chi$ set of any node of $T$. Contradiction. □

**Lemma 5.7.** Let $\langle T, \chi, \lambda \rangle$ be a generalized hypertree decomposition of the hypergraph $\mathcal{H}$ of width 5 in normal form. Then, there exists a node $p$ for which $E \notin \chi(p)$ or $F \notin \chi(p)$ or $E_1 \notin \chi(p)$ or $F \notin \chi(p)$.

**Proof.** For all nodes $p$ of $T$, either $\{a_1, a_2, a_3, a_4\} \subseteq \lambda(p)$ or $\{b_1, b_2, b_3, b_4\} \subseteq \lambda(p)$. It is not possible, that for all nodes $p$, $\{a_1, a_2, a_3, a_4\} \subseteq \lambda(p)$ holds, because then we could construct a GHD of width 1 for the cyclic subhypergraph $(A_1, B_1), (B_1, C_1), (C_1, D_1), (D_1, E_1), (E_1, A_1)$. 65
Without loss of generality, let us assume that \( p_a \) is the parent node of \( p_b \), such that \( \{a_1, a_2, a_3, a_4\} \subseteq \lambda(p_a) \) and \( \{b_1, b_2, b_3, b_4\} \subseteq \lambda(p_b) \). Let us assume that \( E \in \chi(p_a) \) and \( F_1 \in \chi(p_a) \), otherwise \( p_a \) satisfies the criteria of the lemma. Then, either \( E \notin \chi(p_b) \) or \( F_1 \notin \chi(p_b) \), because \( \{b_1, b_2, b_3, b_4\} \subseteq \lambda(p_b) \), the width of the decomposition is at most 5 and there is no hyperedge containing both \( E \) and \( F_1 \).

**Lemma 5.8.** \( hw(H) = 6 \), \( scw(H) = 6 \), \( ghw(H) = 5 \), \( chw(H) = 5 \).

**Proof.** It is easy to see that \( chw(H) \leq 5 \): a CHD of \( H \) of width 5 is depicted in Figure 5.5. Therefore, since \( 5 = ghw(H) \leq chw(H) \leq 5 \), also \( chw(H) = 5 \) holds.

Similarly, an SCD of width 6 is depicted in Figure 5.5, therefore \( scw(H) \leq 6 \). Let us assume indirectly that \( chw(H) = 5 \). By Lemma 5.7, for all GHDs of width 5, one of the vertices \( E, F, E_1 \) or \( F_1 \) is in \( (\text{vertices}(\lambda(p)) \setminus \chi(p)) \). In this case the condition 2 in the definition of SCD (Def. 5.5) is violated. Contradiction. □

### 5.3.2 Hypergraphs with \( chw(H) = 2 \) vs. \( scw(H) = 3 \)

In this section we give another example for a hypergraph having component hypertree width strictly smaller than spread cut width. The example is a variant of Example 1.1.

**Example 5.4.** We define the hypergraph \( H_S \) as follows. Let \( H_0(V_0, E_0) \) be the hypergraph from Example 1.1.

\[
\begin{align*}
V(H_S) &= V_0 \cup \{v_{11}, v_{12}, \ldots, v_{18}\}, \\
E(H_S) &= E(H_0) \cup \{e_9 = \{v_1, v_{11}\}, e_{10} = \{v_2, v_{12}\}, e_{11} = \{v_3, v_{13}\}, e_{12} = \{v_4, v_{14}\}, e_{13} = \{v_5, v_{15}\}, e_{14} = \{v_6, v_{16}\}, e_{15} = \{v_7, v_{17}\}, e_{16} = \{v_8, v_{18}\}\}.
\end{align*}
\]

The hypergraph \( H_S \) is depicted in Figure 5.6.
Lemma 5.9. Let $D = \langle T, \chi, \lambda \rangle$ be a spread cut decomposition of $H_S$ of width 2 and let $e$ be an edge such that $e \in E(H_S) \setminus E(H_0)$. Then, there exists a spread cut decomposition $D' = \langle T', \chi', \lambda' \rangle$ of the same width as $D$, such that $e$ only occurs in the $\lambda'$ sets of leaf nodes.

Proof. We use the following notation: $V(H_S) = L \cup M \cup N$, where $L = \{v_9, v_{10}\}$, $M = \{v_1, \ldots, v_8\}$ and $N = \{v_{11}, \ldots, v_{18}\}$.

Let $e$ be an edge as stated in the lemma. Let $p_e$ be a node where $e$ is covered, i.e. $\text{vertices}(e) \subseteq \chi(p_e)$. If there are at least two such nodes, then “replace” the edge $e$ in the $\lambda$ set of these nodes (except at node $p_e$). By “replacing” an edge $e$ in $\lambda(p)$ we mean that we define $\lambda'(p)$ as $(\lambda(p) \setminus e) \cup f$, where $f$ is an edge containing the vertex in $e \cap M$. If there is a vertex $v \in \chi(p)$ such that
Figure 5.6: The hypergraph $H_S$ of the example 5.4

Figure 5.7: a) $GHD$ of width 2 and b) $HD$ of width 3 of the hypergraph $H_S$
v ∈ e ∩ N, then we define χ′(p) as χ(p) \ {v}. The decomposition D′ we get after this transformation is a GHD and b) it is also a spread cut decomposition. D′ is a GHD because, pe still covers the edge e, the connectedness condition remains valid for each vertex in V(HS) and the 3rd condition of GHD must also hold for D′. The decomposition D′ is a spread cut because by the transformation, if for a vertex v, v ∈ vertices(λ(p)) \ χ(p) holds at some node p of D′, then v ∈ N and in this case in a 2-width GHD at this node p the λ(p) set cannot contain any edge from E(HS) \ E(H0).

Thus, we can assume without loss of generality that in a spread cut decomposition of HS the edges of E_n only occur in the λ sets of leaf nodes. In other words, each spread cut decomposition of HS can be obtained from a spread cut decomposition of H0, by adding some child nodes, if necessary.

**Lemma 5.10.** GHW(HS) = 2, CHW(HS) = 2, SCW(HS) = 3.

**Proof.** A generalized hypertree decomposition of width 2 of the hypergraph HS is depicted in Figure 5.7. Since the hypergraph is cyclic, it has no generalized hypertree decomposition of width 1, therefore the generalized hypertreewidth of the hypergraph is 2.

The decomposition in Figure 5.7 a) is a component hypertree decomposition, therefore CHW(HS) = 2. At the root node r, the vertex v2 is adjacent to the edges of two [vertices(λ(r))]-components, C1 = {v12} and C2 = {v1, v8, v11, v18}. The decomposition is not a spread cut decomposition, as C1 and C2 meet one [χ(r)]-component, namely the component C = {v1, v8, v11, v12, v18}. The [vertices(λ(r))] components are: C1 = {v12}, C2 = {v1, v8, v11, v18}, C3 = {v17}, C4 = {v16}, C5 = {v4, v5, v14, v15} and C6 = {v13}.

The decomposition in Figure 5.7 b) of width 3 is a spread cut decomposition. We show that there is no spread cut decomposition of HS of width 2.

Let us suppose indirectly, that there is a generalized hypertree decomposition D of width 2 which is also a spread cut decomposition of HS. D can be obtained from a GHD of H0, by adding child nodes, by Lemma 5.9.
The generalized hypertree decompositions of $H_0$ are studied in [3], in Section 3.3.1. In each minimal generalized hypertree decomposition of $H$ of width at most 2, at each node $p$ of the decomposition tree, $\{v_9, v_{10}\} \subseteq \chi(p)$, see [3], Lemma 3.3.3. The hypergraph $H_0$ has no hypertree decomposition of width 2. ($HW(H_0) = 3$, by Proposition 3.3.2 in [3]). A $GHD$ of $H_0$ of width 2 in normal form must have a node $p$ with a vertex $v$, such that $v \in \text{vertices}(\lambda(p)) \setminus \chi(p)$, otherwise the decomposition satisfies the special condition, but $H_0$ has no $HD$ of width 2. The vertex $v$ must belong to the set $M$, but then at node $p$, one $[p]$-component meets two $[\text{vertices}(\lambda(p))]$-components. Contradiction.

\[\square\]

5.3.3 Hypergraph with $chd(H) = 2n + 1$ vs. $scw(H) = 3n$

The authors of [21] defined a class of hypergraphs $H_n$ for which $scw(H_n) = 2n + 1$, $chd(H_n) = 2n + 1$, but $hd(H_n) = 3n$. We present here a modified version of this example, such that the opposite holds, namely a class of hypergraphs $G_n$ such that $chd(G_n) = 2n + 1$, but $scw(G_n) = hw(G_n) = 3n$.

**Example 5.5.** The set of vertices of $G_n$ is the union of the disjoint sets i.e. $V(G_n) = K \cup K' \cup S \cup S' \cup W \cup W' \cup B$, which are defined below.

- $K = \{k_i \mid i = 1, \ldots, 2n\}$
- $K' = \{k'_i \mid i = 1, \ldots, 2n\}$
- $S = \{s_i \mid i = 1, \ldots, n\}$
- $S' = \{s'_i \mid i = 1, \ldots, n\}$
- $W = \{w_i \mid i = 1, \ldots, n\}$
- $W' = \{w'_i \mid i = 1, \ldots, n\}$
- $B = \{b_{i,j} \mid i, j = 1, \ldots, 2n\}$
The edges of $G_n$ are the following binary edges

$\{(b_{i,j}, k_l) \mid i, j = 1, \ldots, 2n; l = 1, \ldots, 2n\}$,
$\{(b_{i,j}, k'_l) \mid i, j = 1, \ldots, 2n; l = 1, \ldots, 2n\}$,
$\{(b_{i,j}, s_l) \mid i, j = 1, \ldots, 2n; l = 1, \ldots, n\}$,
$\{(b_{i,j}, s'_l) \mid i, j = 1, \ldots, 2n; l = 1, \ldots, n\}$,
$\{(b_{i,j}, b_{i',j'}) \mid i, j, i', j' = 1, \ldots, 2n; i \neq i', j \neq j'\}$,
$\{(k_i, k_j) \mid i \neq j, i, j = 1, \ldots, 2n\}$,
$\{(k'_i, k'_j) \mid i \neq j, i, j = 1, \ldots, 2n\}$,
$\{(k_i, s_j) \mid i = 1, \ldots, 2n; j = 1, \ldots, n\}$,
$\{(k'_i, s'_j) \mid i = 1, \ldots, 2n; j = 1, \ldots, n\}$,
$\{(s_i, w_i) \mid i = 1, \ldots, n\}$,
$\{(s'_i, w'_i) \mid i = 1, \ldots, n\}$.

and the edges

$e_j = (b_{i,j}, k_j, k_{n+j} \mid i = 1, \ldots, 2n)$, for $j = 1, \ldots, n$,
$e'_i = (b_{i,j}, k'_j, k'_{n+j} \mid j = 1, \ldots, 2n)$, for $i = 1, \ldots, n$,
$f_i = (b_{i+n,j}, s_j \mid j = 1, \ldots, 2n)$, for $i = 1, \ldots, n$,
$f'_j = (b_{i+n,j}, s'_j \mid i = 1, \ldots, 2n)$, for $j = 1, \ldots, n$.

The hypergraph is depicted in Figure 5.8.\(^4\)

We use the following short notation (introduced in [21]):

$F = \{e_1, \ldots, e_n, f'_1, \ldots, f'_n\}$ and $F' = \{e'_1, \ldots, e'_n, f_1, \ldots, f_n\}$.

Intuitively, our construction follows the same strategy as for the Example 5.4: we added some new vertices and edges to the example in [21], such that any $GHD$ of $G_n$ having smaller width than the hypertreewidth cannot be a $SCD$, because the condition of spread cut, which requires that a $[p]$-component must meet only one $[vertices(\lambda(p))]$-component, is violated.

**Lemma 5.11.** $ghw(G_n) \leq 2n + 1$, $chw(G_n) \leq 2n + 1$.

\(^4\)Some edges, which are completely covered by others, are not depicted.
Proof. Figure 5.9 depicts the $\lambda(p)$ sets of a generalized hypertree decomposition of $G_n$ of width $2n + 1$. The $\lambda$ and $\chi$ sets (of an example node) are depicted in Figure 5.10. The $\langle \lambda, \chi \rangle$ labels of the nodes of the decomposition tree are

$$\langle \{F, (k_1, s_j)\}, \{B \cup K \cup \{s_j\}\} \rangle \quad \text{for } j = 1, \ldots, n$$
$$\langle \{F', (k'_1, s'_j)\}, \{B \cup K' \cup \{s'_j\}\} \rangle \quad \text{for } j = 1, \ldots, n$$
$$\langle \{F, (w_j, s_j)\}, \{B \cup K \cup \{s_j\} \cup \{w_i\}\} \rangle \quad \text{for } j = 1, \ldots, n$$
$$\langle \{F', (w'_j, s'_j)\}, \{B \cup K' \cup \{s'_j\} \cup \{w'_i\}\} \rangle \quad \text{for } j = 1, \ldots, n.$$ 

The decomposition is also a component hypertree decomposition. The $\chi$-sets of the decomposition consists of complete hyperedges and the subedges of the form $(\text{vertices}(f'_i) \setminus s'_i)$ for $i = 1, \ldots, n$ and $(\text{vertices}(f_i) \setminus s_i)$ for $i = 1, \ldots, n$ and these subedges form a subset for the set $f^C(H, 1)$. This follows from the definition
of $f^C$: $s_i \in f_i \cap \text{vertices}(edges(C))$ where $C$ is the component containing $w_i$ thus $f_i \setminus s_i$. Similarly, $f'_i \setminus s'_i$ is also obtained by the subedge function $f^C$.  

\[ \square \]

Figure 5.9: Component hypertree decomposition of width $2n+1$ of the hypergraph $G_n$ from Example 5.5

\[ \begin{array}{c}
F_i(k_1, s_n) \\
F_i(w_n, s_n)
\end{array} \hspace{1cm} \begin{array}{c}
F_i(k_1, s_1) \\
F_i(w_1, s_1)
\end{array} \hspace{1cm} \begin{array}{c}
F'_i(k'_1, s'_n) \\
F'_i(w'_n, s'_n)
\end{array} \hspace{1cm} \begin{array}{c}
F'_i(k'_1, s'_1) \\
F'_i(w'_1, s'_1)
\end{array} \]

Figure 5.10: The $\lambda$ and $\chi$ sets of the decomposition from Figure 5.9

On the other hand, the decomposition in Figure 5.9 is not a spread cut decomposition. For example, let us examine the node $p$ of the decomposition tree with the labels $\langle \{F, (k_1, s_1)\}, \{B \cup K \cup \{s_1\}\} \rangle$. The vertex $s'_1$ is adjacent to the edge $f'_1$ ($f'_1 \in F$), but $s'_1$ is not in the $\chi(p)$ set. Thus, the two $\{\text{vertices}(\lambda(p))\}$-components $C_1 = \{w'_1\}$ and $C_2 = \{k'_1, \ldots, k'_n\}$ meet only one $[p]$-component. We can formulate this observation even more generally.

**Lemma 5.12.** Let $D = \langle T, \chi, \lambda \rangle$ be a GHD of $H$ and let $p$ be a non-leaf node of $T$. If $F \subseteq \lambda(p)$ and there exists a vertex $s'_j \in S'$ (for some $j = 1, \ldots, n$) such that $s'_j \in \text{vertices}(F) \setminus \chi(p)$, then $D$ is not a spread cut decomposition.

\[ \text{It is also an extended component hypertree decomposition (ECHD). The decomposition method ECHD is defined in chapter 6.} \]
Proof. If a vertex $s'_j$ with the property $s'_j \in \text{vertices}(F) \setminus \chi(p)$ exists, then there is a $[p]$-component that meets more than one $[\text{vertices(\lambda(p))}]$-component, thus $D$ is not a spread cut decomposition. □

**Lemma 5.13.** In each GHD of $H$ there exists a node $p$, such that $K \cup B \subseteq \chi(p)$. Symmetrically, there exists a node $p'$, such that $K' \cup B \subseteq \chi(p')$, too.

Proof. The set $B \cup K$ forms a clique graph. In every tree decomposition of a hypergraph containing a clique, there exists a node which covers all of the vertices of the clique, see [11]. Therefore, there exists a node $p$, such that $B \cup K \subseteq \chi(p)$ holds. □

**Lemma 5.14.** If $p$ is a node of $T$ in a generalized hypertree decomposition $\langle T, \chi, \lambda \rangle$ in normal form and $B \not\subseteq \chi(p)$, then $p$ is a leaf node of $T$.

Proof. Every vertex in $G_n$ is directly connected to every vertex in $B \setminus \chi(p)$, thus there is at most one $[p]$-component. Since the decomposition is in normal form, this component corresponds to only one subtree. □

We need the following lemma from [21], which holds for both the class of hypergraphs $G_n$ (we defined here) and $H_n$ (defined in [21]).

**Lemma 5.15.** ([21]) Let $p$ be a node of $T$, and $B \subseteq \chi(p)$, then either $F \subseteq \lambda(p)$ or $F' \subseteq \lambda(p)$.

**Lemma 5.16.** $\text{scw}(G_n) = 3n$

Proof. A generalized hypertree decomposition of width $3n$ of $G_n$ is depicted in Figure 5.11. The decomposition consist of two distinguished nodes $p$ and $p'$ with

\[
\lambda(p) = \{F, (k_s) \mid j = 1, \ldots, n\} \quad \chi(p) = \{\text{vertices}(\lambda(p))\}
\]
\[
\lambda(p') = \{F', (k'_s) \mid j = 1, \ldots, n\} \quad \chi(p') = \{\text{vertices}(\lambda(p'))\}
\]

and the nodes

\[
\lambda(p_j) = \{F, (w_j, s_j)\} \quad \chi(p_j) = \{\text{vertices}(\lambda(p))\} \quad \text{for } j = 1, \ldots, n,
\]
\[
\lambda(p'_j) = \{F', (w'_j, s'_j)\} \quad \chi(p'_j) = \{\text{vertices}(\lambda(p'))\} \quad \text{for } j = 1, \ldots, n.
\]
Since the decomposition is edge defined, i.e. for all nodes \( q \) of the decomposition tree, \( \chi(q) = \text{vertices}(\lambda(q)) \), for each edge \( e_1 \) and \( e_2 \) (\( e_1 \neq e_2 \)) of \( \lambda(q) \), \( e_1 \cap e_2 \subseteq \chi(q) \) clearly holds. Furthermore, the decomposition is in normal form and each \([q]\)-component meets at most one \([\text{vertices}(\lambda(q))]\)-component. Thus, the decomposition is a spread cut decomposition, therefore the spread cut width of \( G_n \) is at most \( 3n \).

We have to show that \( G_n \) has no spread cut decomposition of width smaller than \( 3n \). Indirectly, let us assume, that there exists such a \( GHD \ D \), in normal form of width \( < 3n \). Because of Lemmas 5.14 and 5.15, for each (non-leaf) node \( p \) of the decomposition, either \( F \subseteq \lambda(p) \) or \( F' \subseteq \lambda(p) \). Let us assume without loss of generality that \( F \subseteq \lambda(p) \). Then, since \( p \) is non-leaf node, \( B \subseteq \chi(p) \). Furthermore, by Lemma 5.12, the relation \( \text{vertices}(F) \subseteq \chi(p) \) holds.

Let us assume that there exists an \( s_i \in S \ (i \in [1..n]) \) such that \( s_i \notin \chi(p) \). Then there must be a \([p]\)-component \( C \) –since \( D \) is in normal form, thus \( C \) corresponds to a subtree at \( p \)– containing \( s_i \). There exists another component at node \( p \), corresponding to a different subtree, containing the vertices of \( K' \). Since the vertices in \( B \cup K' \) form a clique, there must be a node in the decomposition, covering all of the vertices in \( K' \). If this node is the child node \( q \) of \( p \), which is a leaf node, then \( \lambda(q) = \{ F, e'_1, \ldots, e'_n \} \), thus the width of \( D \) is \( 3n \). Therefore we can assume that \( q \) in not a leaf node. Then, by Lemma 5.15, \( F' \subseteq \lambda(q) \) holds and by Lemma 5.12, also \( \text{vertices}(F') \subseteq \chi(q) \). Therefore, since \( s_i \notin \chi(p) \), but \( s_i \) is in the \( \chi \)-set of two different subtrees. Thus the connectedness condition does not hold. Contradiction. \( \square \)
Figure 5.11: Spread cut decomposition of width $3n$ of the hypergraph $G_n$ from Example 5.5

Figure 5.12: The $\lambda$ and $\chi$ sets of the decomposition depicted in Figure 5.11
Chapter 6

Extended component hypertree decomposition

In a very recent manuscript, Cohen et al. [21] gave a new definition for spread cut decomposition. ¹ In this chapter we study this new definition of spread cut decomposition and compare it with hypertree decomposition. We also define a decomposition method, called extended component hypertree decomposition and we compare it with spread cut decomposition.

6.1 New definition of spread cut decomposition

First, we recall here the new definitions from [21]. The authors introduce the following labeling functions.

Definition 6.1. ([21]) Let \( \langle T, \chi, \lambda \rangle \) be a generalized hypertree decomposition in normal form and let \( p \) be a node of \( T \). For a node \( v \) in \( \text{vertices}(\lambda(p)) \) the labels \( L_A \) are defined as follows:

¹Although the definitions in [22] and in [21] are different, the authors use the same name, spread cut decomposition, for the two different concepts. We denote the decomposition method defined in [22] as \( SCD \) and the decomposition method defined in [21] as \( SCD_{\text{new}} \). In this chapter, if we refer in the text to spread cut decomposition, we refer to \( SCD_{\text{new}} \).
\[ L_1[1](v) = \{ C \mid C \text{ is a}[\text{vertices}(\lambda(p))]\text{-component, } \exists e \in E, e \cap C \neq \emptyset, v \in e \}, \]
\[ L_2[2](v) = \{ e \in \lambda(p) \mid v \in \text{vertices}(e) \}. \]

Using the labeling of the vertices, the authors define the following special property of the labels.

**Definition 6.2.** ([21]) Let \( \langle T, \chi, \lambda \rangle \) be a generalized hypertree decomposition in normal form and let \( p \) be a node of \( T \). The \( \langle \lambda(p), \chi(p) \rangle \) sets at node \( p \) of \( T \) respect labels, if \( \forall v, w \in \text{vertices}(\lambda(p)), (v \in \chi(p) \text{ and } L_\lambda(v) = L_\lambda(w)) \implies w \in \chi(p) \).

The new definition of spread cut decomposition uses this special property of the labels.

**Definition 6.3.** ([21]) A generalized hypertree decomposition \( \langle T, \chi, \lambda \rangle \) of a hypergraph \( H \) in normal form is called spread cut decomposition if additionally the following conditions hold:

1. at each node \( p \) of \( T \), each \([p] \)-component meets at most one \([\text{vertices}(\lambda(p))]\)-component,

2. at each node \( p \) of \( T \), the \( \chi(p) \) and \( \lambda(p) \) sets respect labels.

**Example 6.1.** A spread cut decomposition of width 2 of the hypergraph of Example 1.1 is depicted in Figure 1.2 a). The decomposition is in normal form. We show that at the root node \( r \), the conditions of the Definition 6.3 hold.

There are two \([\text{vertices}(\lambda(r))]\)-components, namely \( C_1 = \{v_1, v_8\} \) and \( C_2 = \{v_4, v_5\} \). The \([r] \)-components are \( C_1^r = \{v_1, v_2, v_8\} \) and \( C_2^r = \{v_4, v_5\} \), so each \([r] \)-component meets one \([\text{vertices}(\lambda(r))]\)-component.

The labels of the vertices in \( \text{vertices}(\lambda(r)) \) are:

\[ L_\lambda(v_2) = \{ [C_1], [e_2] \}, \]
\[ L_\lambda(v_3) = \{ [C_2], [e_2] \}, \]
\[ L_\lambda(v_6) = \{ [C_2], [e_6] \}, \]
\[ L_\lambda(v_7) = \{ [C_1], [e_6] \}, \]
\[ L_\lambda(v_9) = \{ [C_1, C_2], [e_2] \}, \]
\[ L_\lambda(v_{10}) = \{ [C_1, C_2], [e_6] \}. \]
The guarded block $\langle \lambda(r), \chi(r) \rangle$ respects labels.

The following lemma gives a characterization of possible labels in a spread cut decomposition.

**Lemma 6.1.** (Cohen et al. [21], Proposition 7.6)

Let $D = \langle T, \chi, \lambda \rangle$ be a spread cut decomposition. Then, for each node $p$ of $T$ and for each edge $e \in \lambda(p)$, there exists a vertices $\lambda(p)$-component $C_e$, such that the vertices $\lambda(p) \setminus \chi(p)$ are precisely the vertices with labels \{0, $l_2$ | $l_2 \subseteq \lambda(p) \} \cup \{C_e, l_2 | l_2 \subseteq \lambda(p), e \in l_2\}.

It follows from the above lemma that an analogous result to Lemma 5.1 does not hold: it is possible that $v$ and $w$ are two different vertices of an edge $e$ in $\lambda(p)$, such that both $v \notin \chi(p)$ and $w \notin \chi(p)$ and $L_1[v] = \emptyset$, while $L_1[w] = \{C_e\}$.

### 6.2 Comparing hypertree and spread cut decompositions

In this section we study the relation of spread cut decompositions to hypertree decompositions. The spread cut width of hypergraphs can be smaller than their hypertreewidth. For example, Adler [3] has shown that the hypertreewidth of the hypergraph of example 6.1 is 3. A spread cut decomposition of width 2 for this hypergraph is depicted in Figure 1.2 (see also [21]). The authors of [21] ask whether for all hypergraphs the spread cut width is smaller or equal to the hypertreewidth, i.e. whether $SCD_{new} \leq HD$ holds.

We answer this question here, by constructing a hypergraph which has hypertreewidth 2 and spread cut width 3.

**Example 6.2.** Let us consider the following hypergraph $H$:\(^2\)

$$H = \{a(S, X, X', C, F), b(S, Y, Y', C', F'), c(C, C', Z), d(X, Z), e(Y, Z),$$
$$f(F, F', Z'), g(X', Z'), h(Y', Z'), j(J, X, Y, X', Y')\}$$

---

\(^2\)This is the hypergraph of the query $Q_3$ in [43].
We define a hypergraph $H'$, based on $H$ as follows. For each vertex $W$ of $H$ we define two new vertices $W_1$ and $W_2$, and we add these vertices to the vertex set of $H$. For each vertex $W$ we add the two binary edges $e^1_W(W, W_1)$ and $e^2_W(W, W_2)$ to the edges set of $H$, i.e. $V(H') = V(H) \cup \{W_1, W_2 \mid W \in V(H)\}$ and $E(H') = E(H) \cup \{(W, W_1), (W_2) \mid W \in V(H)\}$, see Figure 6.1.

Lemma 6.2. $hw(H') = 2$, $scw(H') = 3$.

Proof. The authors of [43] show that $hw(H) = 2$ and $qw(H) = 3$, see section 3.3. in [43]. From a hypertree decomposition of width 2 for $H$ we can easily obtain a hypertree decomposition for $H'$, just by adding child nodes, for each binary edge in the straightforward way: add the node with $\langle \lambda, \chi \rangle = \langle \{e^1_W\}, \{W, W_1\}\rangle$ as a child node of one of the nodes where the vertex $W$ occurs in the $\chi$ set. In this way we get a hypertree decomposition of width 2. Since $H'$ is not acyclic, $hw(H') = 2$.

![Figure 6.1: Binary edges added to H](image)

If any of the edges in $E(H)$ is used partially at node $p$ in a decomposition of $H'$, and the vertex $V$ is in $\text{vertices}(\lambda(p)) \setminus \chi(p)$ then the two $[\text{vertices}(\lambda(p))]-$components belong to one $[p]$-component, therefore it is not a spread cut decomposition. Therefore each spread cut decomposition must contain the complete edges of $H$. The only edges which may be used partially are the binary edges, which cannot help to construct a decomposition of width 2. However, as it was shown in [43], $H$ does not have a decomposition with complete edges of width 2. A query decomposition of width 3 is depicted on Figure 5 in [43]. This decomposition is in normal form and at each node $p$, it holds that $\chi(p) = \text{vertices}(\lambda(p))$, in other terms, the decomposition is edge defined. Furthermore each $[\lambda(p)]$-component meets exactly one $[p]$-component, therefore is is also a spread cut...
decomposition. Thus, $scw(H') = 3$. □

Now we define a class of hypergraphs $G_n$, such that $hw(G_n) \leq 2n + 1$ and $scw(G_n) = 3n$. The example is an adaptation of the example 7.13 in [21] for our purposes. A similar class of hypergraphs having component hypertreewidth $2n+1$ and spread cut width $3n$ is presented in Chapter 5.

**Example 6.3.** The set of vertices of $G_n$ is the union of the disjoint sets i.e. $V(G_n) = K \cup K' \cup S \cup S' \cup W \cup W' \cup B$, which are defined below.

\[ K = \{k_i \mid i = 1, \ldots, 2n\}, \]
\[ K' = \{k'_i \mid i = 1, \ldots, 2n\}, \]
\[ S = \{s_i \mid i = 1, \ldots, n\}, \]
\[ S' = \{s'_i \mid i = 1, \ldots, n\}, \]
\[ B = \{b_{i,j} \mid i, j = 1, \ldots, 2n\}, \]
\[ W^1 = \{w^1 \mid w \in (K \cup K' \cup S \cup S' \cup B)\}, \]
\[ W^2 = \{w^2 \mid w \in (K \cup K' \cup S \cup S' \cup B)\}, \]

The edges of $G_n$ include the following binary edges

\[ \{(b_{i,j}, k_l) \mid i, j = 1, \ldots, 2n; l = 1, \ldots, 2n\}, \]
\[ \{(b_{i,j}, k'_l) \mid i, j = 1, \ldots, 2n; l = 1, \ldots, 2n\}, \]
\[ \{(b_{i,j}, s_l) \mid i, j = 1, \ldots, 2n; l = 1, \ldots, n\}, \]
\[ \{(b_{i,j}, s'_l) \mid i, j = 1, \ldots, 2n; l = 1, \ldots, n\}, \]
\[ \{(b_{i,j}, b_{i',j'}) \mid i, j, i', j' = 1, \ldots, 2n; i \neq i', j \neq j'\}, \]

and also the following binary edges

\[ \{(k_i, k_j) \mid i \neq j, i, j = 1, \ldots, 2n\}, \]
\[ \{(k'_i, k'_j) \mid i \neq j, i, j = 1, \ldots, 2n\}, \]
\[ \{(k_i, s_j) \mid i = 1, \ldots, 2n; j = 1, \ldots, n\}, \]
\[ \{(k'_i, s'_j) \mid i = 1, \ldots, 2n; j = 1, \ldots, n\}, \]

81
the binary edges

\{(w, w^1) \mid w \in (K \cup K' \cup S \cup S' \cup B)\}, \{(w, w^2) \mid w \in (K \cup K' \cup S \cup S' \cup B)\},

and the edges

\begin{align*}
e_j &= (b_{i, j}, k_j, k_{n+j} \mid i = 1, \ldots, 2n), \quad \text{for } j = 1, \ldots, n, \\
e'_i &= (b_{i, j}, k'_j, k'_{n+j} \mid j = 1, \ldots, 2n), \quad \text{for } i = 1, \ldots, n, \\
f_i &= (b_{i+n, j}, s_j \mid j = 1, \ldots, 2n), \quad \text{for } i = 1, \ldots, n, \\
f'_j &= (b_{i+n, j}, s_j \mid i = 1, \ldots, 2n), \quad \text{for } j = 1, \ldots, n.
\end{align*}

Furthermore, \(G_n\) contains each of the subedges of the edges \(e_i, e'_i, f_i, f'_i\) \((1 \leq i \leq n)\), i.e.

\[\{s \mid s \subseteq e_i \lor s \subseteq e'_i \lor s \subseteq f_i \lor s \subseteq f'_i, \text{ for } i = 1, \ldots, n\}.\]

The hypergraph is depicted schematically in Figure 6.2. The edges, which are covered by other edges are not depicted. The vertices of \(W_1\) and \(W_2\) and the edges incident to this vertices are also not depicted.

**Lemma 6.3.** \(h_w(G_n) \leq 2n + 1\).

**Proof.** In [21] the authors present a generalized hypertree decomposition (a spread cut decomposition) of width \(2n+1\) for their example. In each case when a subedge is used as \(\chi(p)\) set at some node \(p\) of the decomposition in [21] the subedge can be used also the \(\lambda(p)\) set for a decomposition for \(G_n\). Therefore a hypertree decomposition of width \(2n + 1\) for \(G_n\) exists.

The nodes of a possible tree decomposition of \(G_n\) are

- \(\{B \cup K \cup \{s_j\}\} \text{ for } j=1, \ldots, n\),
- \(\{B \cup K' \cup \{s'_j\}\} \text{ for } j=1, \ldots, n\),
- \(\{W, W^1 \mid W \in V(G_n)\}, \{W, W^2 \mid W \in V(G_n)\}\).
Some nodes of the tree decomposition are depicted in Figure 6.3.

Since we defined our example by adding the possible sub-edges of each non-binary hyperedge to the example in [21], we can conclude that $\text{hw}(G_n) \leq 2n + 1$ holds. Note that we have to add two child nodes for each vertex $W$ at an appropriate node of the decomposition, to cover the edges $(W, W^1)$ and $(W, W^2)$, but this does not affect the width. □

**Lemma 6.4.** $\text{scw}(G_n) = 3n$

Note that the decomposition in described in Lemma 6.3 is no spread cut decomposition. For example, let us examine the node $p$ of the decomposition tree with the guarded block $\langle \{F, (k_1, s_1)\}, \{B \cup K \cup \{s_1\}\} \rangle$. The vertex $s'_1$ is adjacent to the edge $f'_1$ ($f'_1 \in F$), but $s'_1$ is not in the $\chi(p)$ set. Thus, the two different
vertices(λ(p))]-components $C_1 = \{s_1^1\}$ and $C_2 = \{s_1^2\}$ meet one \(p\)-component.

We can formulate this observation more generally. We use the following short notation (introduced in [21]):

\[
F = \{e_1, \ldots, e_n, f_1', \ldots, f_n'\} \quad \text{and} \quad F' = \{e_1', \ldots, e_n', f_1, \ldots, f_n\}.
\]

**Lemma 6.5.** If generalized hypertree decomposition of $G_n$ uses a subedge of any edge from $F$ or $F'$, then the decomposition is not a spread cut.

**Lemma 6.6.** $scw(G_n) = 3n$

*Proof.* The decomposition of width $3n$ in [21] (extended with the child nodes covering the edges $(W, W_1)$ and $(W, W_2)$) is a spread cut decomposition. The $\lambda(p)$ sets of a possible decomposition for $G_n$.

\[- F, (k_1, s_1), \ldots, (k_1, s_n),\]

\[- F', (k'_1, s'_1), \ldots, (k'_1, s'_n),\]

\[- (W, W_1), (W, W_2) \text{ for each } W \in V(G_n).\]

The decomposition is of width $3n$, because both the set $F$ and $F'$ contains $2n$ hyperedges. Since the decomposition is *edge defined*, i.e. for all nodes $q$ of the decomposition tree, $\chi(q) = vertices(\lambda(q))$, all vertices, at all nodes respect labels. Furthermore, the decomposition is in normal form and each $[q]$-component meets at most one \(vertices(\lambda(q))\)-component. Thus, the decomposition is a spread cut decomposition, therefore the spread cut width of $G_n$ is at most $3n$. $G_n$ has no spread cut decomposition of width smaller than $3n$, because none of the edges in $F$ can be used partially in any spread cut decomposition. We can argue the same way as in [21] to show that any decomposition of $G_n$ of width strictly smaller then $3n$ uses subedges, thus it is not a spread cut decomposition, by Lemma 6.5.  

Now we turn our attention to another problem, namely whether an $SCD_{new}$ can be described as a hypertree decomposition. It turns out that spread cut decompositions of $H$ can be seen as hypertree decompositions of an another hypergraph
$H'$, which can be obtained from $H$ by adding subedges in a controlled way as described below, such that the hypertree decomposition satisfies some additional conditions. We will use this characterization later in Chapter 7.

We need later the following technical definition. Let us define the set $\Delta_k$ as follows.

**Definition 6.4.** Let $H$ be a hypergraph and let $k$ be a positive integer. We define $\Delta_k(H)$ as

$$\Delta_k(H) = \{ e \cap \chi(p) \mid D = (T, \chi, \lambda) \text{ is a spread cut decomposition of width at most } k, p \in \text{nodes}(T), e \in \lambda(p) \}.$$

Now we are ready to describe the relation between spread cut decomposition and hypertree decomposition.

**Lemma 6.7.** Let $H$ be a hypergraph and let $D = (T, \chi, \lambda)$ be a GHD of $H$ of width $k$. The decomposition $D$ is a spread cut decomposition of $H$ if and only if

1. $D' = (T, \chi', \lambda')$, where for each $p$ of $T$, $\lambda'(p) = \{ \chi(p) \cap e \mid e \in \lambda(p) \}$, is a hypertree decomposition of the hypergraph $H \cup \Delta_k(H)$, in normal form,

2. for each node $p$ of $T$, the guard $(\lambda(p), \chi(p))$ has unbroken components,

3. for each node $p$ of $T$, the guard $(\lambda(p), \chi(p))$ respects labels.

**Proof.** If $D$ is a spread cut decomposition of $H$, then by definition 6.3 it satisfies condition (2) and (3). Note that in condition (2) and (3) we refer to the $\lambda(p)$ sets and not to $\chi(p)$. Furthermore, since for all nodes $p$ in $T$, the special condition of hypertree decomposition is satisfied, since $\chi(p) = \text{vertices}(\lambda(p))$, thus $D'$ is a hypertree decomposition of the hypergraph $H \cup \Delta_k(H)$. Since by Definition 6.3, the decomposition $D$ is in normal form, we can conclude that $D'$ is also in normal form, as the $\chi$-sets of the two decompositions are identical.

On the other hand, let $D'$ be a hypertree decomposition of $H \cup \Delta_k(H)$. Let $D$ be a generalized hypertree decomposition of $H$ of width $k$, such that $D$ and $D'$ have the same $\chi$-sets, at each node $p$ of $T$. By the construction of the $\Delta_k(H)$ set, such a decomposition must always exist. As the two decompositions share the $\chi$-sets...
and \( D' \) is in normal form, also \( D \) is in normal form. Furthermore, by conditions \( 2 \) and \( 3 \), the decomposition \( D \) is also a spread cut decomposition. \( \square \)

### 6.3 Extended component hypertree decomposition

As the following example shows, there are spread cut decompositions, such that the subedges corresponding to the \( \chi \)-sets are not in the set of subedges defined by the subedge function \( f^C \) of the component hypertree decomposition.

**Example 6.4.** Let \( H \) be a hypergraph containing the following edges:

\[
\begin{align*}
&e(A, B, C, E, F), \quad f(C, D, E, F), \quad g(E, F, G, H).
\end{align*}
\]

Let us study the guarded cover \( \langle \lambda, \chi \rangle \), where \( \lambda \) contains exactly these three edges. Let us assume, that there is a \( \{\text{vertices}(\lambda)\} \)-component, whose edges meet the vertices \( \{B, C, D, E\} \), and let us assume that these vertices are only adjacent to the edges of this component. Assume that the guarded block is \( \langle \{e, f, g\}, \{A, B, C, D, F, G, H\} \rangle \).

The only vertex, which is not in the \( \chi \) set is \( E \). Thus, the guarded block uses a subedge \( e'(A, B, C, F) \). This subedge is not in the set \( f^C(H, 3) \), according to the Definition 5.4. On the other hand, this guarded cover is valid according to Definition 6.3.

![Figure 6.4: Guarded cover of example 6.4.](image)

This motivates us to define a new subedge-based decomposition, which generalizes \( SCD_{\text{new}} \). We call this decomposition extended component hypertree de-
composition ($ECHD$). Before we define the subedge function $f^E$ for $ECHD$, we need some technical definitions.

**Definition 6.5.** Let $M$ be a set of hyperedges of the hypergraph $H$. Let $N$ be a subset of $M$, $N \subseteq M$. We define $\text{slice}(N, M)$ as

$$\text{slice}(N, M) = \{ v \mid \forall e \in N, v \in \text{vertices}(e) \text{ and } \forall e' \in (M \setminus N), v \notin \text{vertices}(e') \}.$$ 

Thus, $\text{slice}(N, M)$ is a set of vertices in $\text{vertices}(M)$, which are adjacent to the edges of $N$ but to no other edges in $M$. Clearly, $\text{vertices}(M) = \bigcup_{N \subseteq M} \text{slice}(N, M)$.

**Example 6.5.** Let $M$ be a following set of hyperedges, $M = \{ a(A, B, C, D, E, F), b(B, C, D, E, F, G, H, I), c(D, E, F, G, H, J, K, L, O), d(C, D, E, F, G, H, O, N, M) \}$ and let $N$ be the subset $N = \{ b, c, d \}$. Then $\text{slice}(N, M) = \{ G, H \}$.

![Figure 6.5: Slice.](image)

**Definition 6.6.** Let $M$ be a set of edges of the hypergraph $H$ and let $N$ be a subset of $M$. We define the set $\text{internal}(N, M) = \{ v \mid v \in \text{slice}(N, M) \text{ and there exists no [\text{vertices}(M)]-component } C, \text{ such that } v \in \text{vertices}(\text{edges}(C)) \}$.

**Definition 6.7.** Let $H$ be a hypergraph, let $M$ be a set of edges of $H$ and let $C$ be a [\text{vertices}(M)]-component. Let $N_1, \ldots, N_r$ ($1 \leq r \leq 2^{|M|-1}$) be subsets of $M$.

The function $\text{elim}(M, C, N_1, \ldots, N_r)$ associates a set containing the following subedges to a tuple $(M, C, N_1, \ldots, N_r)$

1. $\bigcup_{1 \leq i \leq r} (\text{slice}(N_i, M) \cap \text{vertices}(\text{edges}(C)))$,
2. $\bigcup_{1 \leq i \leq r} \text{internal}(N_i, M)$.
Definition 6.8. Let $H$ be a hypergraph and let $k$ $(0 \leq k)$ be an integer. We define the subedge function $f^E$ as:

$$f^E(H,k) = \{ e \setminus e' \mid M \text{ is a set of } \leq k \text{ hyperedges of } H, e \in M, C \text{ is a } [\text{vertices}(M)]-\text{component}, 1 \leq r \leq 2^{k-1}, N_1, \ldots, N_r \text{ are sets of hyperedges, for each } i (1 \leq i \leq r), N_i \subseteq M, e' \in \text{elim}(M, C, N_1, \ldots, N_r) \}.$$ 

The decomposition method $M_{f^E}$ is referred to as extended component hypertree decomposition (ECHD).

### 6.4 Comparing spread cut and extended component hypertree decompositions

We compare here extended component hypertree decompositions and spread cut decompositions. We proceed in a similar way, as we did in Section 5.2.

Lemma 6.8. $M_{f^E} \leq S_{C_{new}}$.

Proof. The lemma follows from Corollary 4.1 and the definition of $f^E$ (Definition 6.8). We have to show that a possible $\chi(p)$ set in a spread cut decomposition is a subset of the subedges defined by $f^E$.

The possible $\chi(p)$ sets in a spread cut decomposition are characterized by Lemma 6.1. All of these subsets are obtained by the subedge function of the extended component hypertree decomposition (Definition 6.8).

In fact, for a vertex $v$ in $\text{vertices}(\lambda(p)) \setminus \chi(p)$, such that $L_\lambda(v) = \{[C], [N_i]\}$, the relevant subedges are associated to a tuple $(\lambda(p), C, \ldots, N_i, \ldots)$, as $\cdots \cup \text{slice}(N_i, M) \cap \text{vertices}(\text{edges}(C)) \cup \ldots$. Similarly, for a vertex $w$ such that $L_\lambda(w) = \{0, [N_w]\}$ the subedge-function $f^E$ constructs the corresponding subedges.
The *elim* function associates to a tuple \((\lambda(p), C, \ldots, N_w, \ldots)\) the sets
\[
\cdots \bigcup \text{internal}(N_w, M) \bigcup \ldots.
\]

\[\square\]

**Theorem 6.1.** For the new definition of spread cut, \(ECHD < SCD_{\text{new}}\).

**Proof.** By Lemma 6.8, \(ECHD \leq SCD_{\text{new}}\). The hypergraph of Example 5.4 has spread cut width 3 and extended component hypertreewidth 2. The decomposition in Figure 5.7 a) is also a component hypertree decomposition, therefore \(CHW(H) = 2\). Since the hypergraph is cyclic, it has no \(GHD\) of width 1. The decomposition in Figure 5.7 b) is a spread cut decomposition of width 3.

The arguments that the hypergraph has no spread cut decomposition of width 2 do not change for the new definitions. This is because the new decomposition also has the condition that at a node \(p\) of a decomposition tree, a \([p]\)-component meets at most one \([\lambda(p)]\)-component.

The class of hypergraphs defined in Example 5.5 has also strictly smaller extended component hypertreewidth than spread cut width.\[\square\]

Finally, let us compare \(CHD\) and \(ECHD\). Although it is not clear at all, whether \(SC_{\text{new}} \leq SC\) holds, it is very easy to compare \(CHD\) and \(ECHD\).

**Lemma 6.9.** \(ECHD \leq CHD\).

**Proof.** The lemma follows immediately from Corollary 4.1 and the definitions of \(f^E\) and \(f^C\).\[\square\]

Actually, we designed \(ECHD\), such that the relation \(ECHD \leq CHD\) holds.

### 6.5 Approximating GHDs

In this section we introduce a parametrized version of extended component hypertree decomposition. Extended component decomposition, as defined in Section 6.3, corresponds to the parameter value 1.
Definition 6.9. Let $H$ be a hypergraph, let $M$ be a set of edges of $H$. Let $N_1, \ldots, N_r$ ($1 \leq r \leq 2^{\left|M\right|}$) be subsets of $M$. Furthermore, let $d$ be a constant number ($1 \leq d \leq |E(H)|$) and let $C_1, \ldots, C_d$ be $[\text{vertices}(M)]$-components.

The function $\text{elim}(M, C_1, \ldots, C_d, N_1, \ldots, N_r)$ associates a set containing the following subedges to a tuple $(M, C_1, \ldots, C_d, N_1, \ldots, N_r)$:

1. $\bigcup_{1 \leq i \leq r} \text{slice}(N_i, M) \cap \bigcup_{1 \leq j \leq d} \text{vertices}(\text{edges}(C_j))$,
2. $\bigcup_{1 \leq i \leq r} \text{internal}(N_i, M)$.

Definition 6.10. Let $H$ be a hypergraph and let $k$ ($0 \leq k$) be an integer. We define the parametrized subedge function $f^E[d]$ as:

$$f^E(H, k) = \{e \setminus (f_0 \cup f_1) \mid M \text{ is a set of } \leq k \text{ hyperedges of } H,$$
$$e \in M,$$
$$C_1, \ldots, C_d \text{ are } [\text{vertices}(M)]\text{-components},$$
$$N_1, \ldots, N_r (1 \leq r \leq 2^{k-1}) \text{ are sets of hyperedges},$$
$$\text{for each } i (1 \leq i \leq r), N_i \subseteq M,$$
$$f_0 \in \text{elim}(M, C_1, \ldots, C_d, N_1, \ldots, N_r),$${}
$$f_1 \in \text{elim}(M, C_1, \ldots, C_d, N_1, \ldots, N_r)\}.$$}

The decomposition method $M_{f^E}[d]$ is referred to as parameterized extended component hypertree decomposition ($\text{ECHD}[d]$).

Lemma 6.10. $\text{ECHD}[1] = \text{ECHD}$.

Proof. The lemma follows directly from definition 6.10. □

Lemma 6.11. If $d_1 \leq d_2$, then $\text{ECHD}[d_2] \leq \text{ECHD}[d_1]$.

Proof. The lemma follows from Corollary 4.1 and the definitions of $f^E[d]$. □

By definition 6.10, $\text{ECHD}[1] = \text{ECHD}$. By Lemma 6.11, if we choose larger parameter values, we get closer to $\text{GHD}$.

We conclude the chapter with a summary of the comparison results between the various decomposition methods discussed, depicted on Figure 6.6.
Figure 6.6: Comparison results in Chapters 5 and 6.
Chapter 7

On the parallel complexity of CSP decompositions

7.1 CSP decomposition methods and alternating logspace

In [40], Gottlob et al. compare tractable CSP decomposition methods and study which of the decomposition concepts capture larger classes of hypergraphs. Figure 7.1 is adapted from [40], and also reflects the latest development in the field.\(^1\)

We recall that an arrow on the figure has the following meaning: if there is an arrow from the decomposition method \(A\) to the method \(B\), then there exists a structure with bounded width defined by \(B\), but unbounded width defined by \(A\), in other words, \(A\) strongly generalizes \(B\). Arrows with dotted line have a different semantics: if there is an arrow form \(A\) to \(B\), then \(A < B\) holds. As we explained in Section 3.1, this is a weaker condition.

Generalized hypertree decomposition is depicted with a bold borderline on the Figure, because –as it was shown recently in [45]– testing whether a hypergraph has generalized hypertreewidth at most 3 is NP-complete. Fractional hypertree

\(^1\)For the sake of simplicity, we do not depict the methods \(SCD_{new}\) and \(ECHD\) on the figure.
decompositions were introduced by Grohe and Marx [51]. It is an open problem whether recognizing bounded fractional hypertreewidth structures is tractable, therefore we depicted the class with a dotted borderline. Given a fractional hypertree decomposition of a CSP, it can be solved in polynomial time. The polynomial time algorithm for constraint solving defined in [51] however is different from the algorithms in [87] and [41], and it is not known to be parallelizable.

In this chapter we study the recognition problem for bounded width structures and we design alternating logspace recognition algorithms for each of the methods. Furthermore, accepting computations have a witness tree of polynomial size, thus by Ruzzo’s results [74] we can conclude that the recognition problems are in LogCFL. For some of the classes we can give an even better upper bound.

The recognition algorithms follow the same scheme as the recognition algorithm for hypertree decomposition, in [43]. Intuitively, this common scheme is the following: the algorithms make a non-deterministic guess, in an implementation
this step corresponds to an existential state of the ATM, and the components generated by this guess correspond to universal states. Figure 7.2 depicts the relation between a decomposition and a witness tree of an ATM schematically.

Figure 7.2: A decomposition and a corresponding witness tree of an ATM.

Cohen et al. [21] give a unified theory for structural CSP decompositions, they show that all of the decomposition methods follow the same definition scheme and they only differ in one condition, characteristic for the particular method. They also give a scheme for a recognition algorithm, such that the recognition algorithms for each method can be derived from this scheme. Cohen et al. [21] do not point out any connection to ATMs.
7.2 Biconnected components

Biconnected components were introduced in [32]. In a hypergraph \( H = (V, E) \) a vertex \( v \) is separating, if removing \( v \) from \( H \), the number of connected components of \( H \) increases. A biconnected component \( C \) is a maximal set of vertices which has no separating vertex.

![Biconnected components decomposition](image)

Figure 7.3: Biconnected components decomposition.

A biconnected decomposition \( \langle T, \chi \rangle \) of \( H \) is a tree with a labeling function that associates a biconnected component of \( H \) or a singleton vertex to the nodes of \( T \), such that there is an edge between \( p \) and \( q \) in \( T \), if \( \chi(p) \) is a biconnected component, and \( \chi(q) \) is a separating vertex contained in \( \chi(p) \). Without loss of generality we can assume that the root node of a biconnected decomposition corresponds to a biconnected component. The minimal width over all possible biconnected decompositions of a hypergraph \( H \) is the biconnected width of \( H \).

In Figure 7.4 we give a high level description of the algorithm \( k \)-biconnected that decides whether a hypergraph has \( k \)-bounded biconnected component width. The algorithm can be effectively implemented on an alternating Turing machine, as we show later (Lemma 7.3). The following lemmas show that the algorithm \( k \)-biconnected indeed recognizes hypergraphs with biconnected width \( k \). Note that the description of the algorithm refers to two different types of components: to biconnected components (a set of vertices without separating vertex) and to connected components of \( H[V \setminus S] \) (the connected components of the induced subgraph on vertices \( V \setminus S \)).

\(^2\)In [32], the concept was defined for graphs, whereas here we concentrate on hypergraphs.
ALTERNATING ALGORITHM $k$-biconnected

**Input:** hypergraph $H$, (non-empty)

**Result:** Accept, if the biconnected width of $H$ is at most $k$, Reject otherwise.

**Procedure** $k$-biconn($C_R : ConnectedComponent, W : Vertex$) begin

1) **Guess** a set $S$ of size at most $k$
2) **Check** whether $S$ is a biconnected component and whether $W \in S$ is a separating vertex
3) If the check above fails Then Halt and Reject; Else Let $C = \{(C, Z) \mid C$ is a connected component of $H[V \setminus S], Z$ is a separating vertex such that $Z \in S$ and $C \subset C_R\}$
4) If, for each $(C, Z) \in C$, $k$-biconn($C, S, Z$)
then Accept else Reject
end;

begin (**Main**)  
Accept if $k$-biconn($V, \emptyset, \emptyset)$
end.

Figure 7.4: Recognizing $k$-bounded biconnected component width hypergraphs
Lemma 7.1. For any hypergraph $H$, such that biconnected-width of $H$ is at most $k$, $k$-biconnected accepts $H$.

Proof. Let $⟨T,χ⟩$ be a biconnected decomposition of $H$. We show that there exists an accepting computation tree $τ$ for $k$-biconnected. The accepting computation and the tree $τ$ can be constructed using the decomposition $⟨T,χ⟩$ as follows.

- For the initial call of $k$-biconn($V,∅,∅$), we choose the set $S$ in Step 1 as $χ$(root($T$)),
- for a call $k$-biconn($C_R,R,W$), where $R$ corresponds to a biconnected component at some vertex $r$ of $T$, and $s$ is a descendant node of $r$ such that the only node on the unique path from $r$ to $s$ in $T$ is $p$ that corresponds to a vertex $P$ separating the biconnected components $R$ and $S$, where $S$ is the biconnected component corresponding to $s$, then we choose $S$ in Step 1.

We use induction to show that in this way we defined an accepting computation.

Basis: If the tree contains only a root node, then clearly we define an accepting computation.

Induction step: Assume that the computation we defined above reaches the vertex $r$. Let $s$ be a node of $T$ such that $s$ is a descendant node of $r$ such that the only node on the unique path from $r$ to $s$ in $T$ is $p$ that corresponds to a vertex $P$, separating the biconnected components $R$ and $S$. By the choice of $S$ in Step 1, and by the choice of the third parameter at the procedure call 4, the checks in Step 2 do not fail, as $S$ is a biconnected component of size at most $k$ and $P ∈ S$ is a separating vertex. Thus, the computation also reaches the vertex $s$. Therefore, by induction, the defined computation is accepting. □

Lemma 7.2. For any hypergraph $H$, such that $k$-biconnected accepts $H$, the biconnected-width of $H$ is at most $k$. 

97
Proof. Let $\langle F, \rho \rangle$ be a witness tree of an accepting computation of $k$-biconnected with input $H$. We can construct a biconnected decomposition $\langle T, \chi \rangle$ of with $k$ as follows.

- For the root node of $F$ we define a node of $T$ and we label it with $\rho(\text{root}(F))$. This node is the root of $T$.
- For each node $r$ in $F$ we add a node $p_r$ with label $\chi(p_r) = \rho(r)$ in $T$.
- For each node $r$ of $F$, let $\text{children}(r)$ denote the child nodes of $r$ in $F$. For each vertex in $\{v | v \in \rho(r) \cap \rho(s), \text{where } s \in \text{children}(r)\}$ we define a vertex $p_v$ with label $\chi(p_v) = \{v\}$ in $T$, and add it as a child node of $p_r$ in $T$.
- If $s$ is a child node of $r$ and $v \in \rho(r) \cap \rho(s)$, then $p_s$ is a child node of $p_v$ in $T$.

We prove that the tree constructed in this way is indeed a biconnected decomposition of width at most $k$. We have to show the following.

1. Since $\langle F, \rho \rangle$ is a witness tree of an accepting computation, both $\rho(r)$ and $\rho(s)$ are biconnected components of $H$ and there is a separating vertex $Z \in \rho(r)$, which is also in $\rho(s)$, since the test in Step 2 does not fail. Indirectly, if for two different vertices $Z_1$ and $Z_2$, the containment $\{Z_1, Z_2\} \subseteq \rho(r) \cap \rho(s)$ holds, then neither of them is a separating vertex. Contradiction.

2. We constructed $T$ in a way that each node has exactly one parent node, therefore it is a tree.

3. The tree $T$ is labeled with biconnected components and singleton vertices. By the construction, if $\chi(p)$ is a biconnected component and $q$ is a child of $p$ then
\( \chi(q) \) is a separating vertex contained in \( \chi(p) \). The width of the decomposition is clearly at most \( k \). Thus the tree is indeed a biconnected decomposition of \( H \). \hfill \Box

**Lemma 7.3.** \( k \)-biconnected can be implemented on a logspace alternating Turing machine with polynomially bounded tree size.

*Proof.* The same implementation technique as described in [43] (Lemma 5.15.) for hypertree decompositions can be used. We can represent biconnected components of size at most \( k \) in logarithmic space. For a biconnected component \( S \) (\( \leq k \)), also the connected components of \( V \setminus S \) can be represented in logspace: we need to store the set \( S \), which is of size \( k \) and one vertex in the component. For each vertex of \( H \), we can test in logarithmic space whether a vertex is in a connected component. Before the procedure call at Step 4. we have to test whether a vertex is a separating vertex in \( H \). This test can also be performed by a logspace algorithm, by Theorem 2.9. \hfill \Box

**Theorem 7.1.** Deciding whether a hypergraph \( H \) has biconnected width at most \( k \) is feasible in \( \text{LogCFL} \).

*Proof.* The theorem follows from Ruzzo’s characterization of \( \text{LogCFL} \) [74], and from the Lemmas 7.1, 7.2, 7.3. \hfill \Box

Tarjan [84] describes an algorithm for finding the biconnected components of a graph. The algorithm runs in linear time, but it also uses linear space. We gave an upper bound for the \( k \)-bounded biconnected width recognition problem in Theorem 7.1. Here we further improve this bound and pinpoint the exact complexity of the problem.

**Theorem 7.2.** Let \( H \) be a hypergraph and let \( \text{bw}(H) \) denote the biconnected width of \( H \). Deciding, whether \( \text{bw}(H) \leq k \), for a fixed positive \( k \), is complete for \( \text{L} \) under \( \text{NC}^1 \) reductions.

*Proof.* Containment. A biconnected component decomposition of \( H \) never “breaks up” a biconnected component of \( H \), by the definition of a biconnected component,
i.e. if for a set of vertices $C'$ the containment $C' \subset C$ holds, then $C'$ does not occur as a label in any biconnected decomposition. Thus, the biconnected width of a hypergraph equals to the size of its maximal biconnected component. Therefore, we can test $bw(H) \leq k$, by the following algorithm.

First test each set of vertices $S$ of size $k$, whether $S$ is a biconnected set. If $S$ is biconnected, then test whether there is a vertex $v$, such that $v \notin S$ and $S \cup \{v\}$ is biconnected. If there exists a vertex $v$ with this property, then output “REJECT” and halt. If each set of size $k$ is tested, then output “ACCEPT”.

As $k$ is a fixed number, this algorithm can be implemented using only logarithmic space. The hypergraph $H$ has at most $\binom{|E|}{k}$ sets of size $k$. Testing, whether a set of vertices (of size $k$) is biconnected is feasible in logspace, see 2.9. therefore, the algorithm can be realized using a logspace Turing machine with a logspace oracle. Thus, since $L^L = L$, the test is feasible in logspace.

**Hardness.** Let us assume, that $H$ is connected, otherwise we consider each connected component of $H$ separately. The problem $bw(H) \leq k$ is hard for deterministic logspace under NC$^1$ reductions already for $k = 2$. Testing acyclicity of graphs is hard for L, under NC$^1$ reductions, see [24]. A hypergraph which has biconnected width 2 is an acyclic graph, see 7.4. □

**Lemma 7.4.** Let $H$ be a hypergraph. $bw(H) \leq 2$ if and only if $H$ is an acyclic graph.

**Proof.** We assume that $H$ is connected, otherwise we consider its connected components.

If $bw(H) \leq 2$, then $H$ does not have any edge of size larger than 2, since vertices of the same hyperedge of $H$ form a biconnected component. If a graph has a cycle of length at least 3, then it must be contained in one biconnected component, which has size at least 3. Therefore the graph must be acyclic. If a graph is acyclic, then it has a biconnected decomposition of width 2. □

The following lemma shows, that not only the recognition, but also the construction of a biconnected decomposition can be done very efficiently.
Lemma 7.5. Let $H$ be a hypergraph and let $k$ be a positive integer. There exists a logspace transducer $R$, which outputs a biconnected decomposition of $H$ of width at most $k$, if there exists one.

Proof. We describe here how to realize such a logspace transducer. Our construction uses the fact, that the composition of two logspace transducers can be realized also as a logspace transducer, even if the output of the first machine cannot be accommodated in logarithmic space, see [68].

$R$ can be realized as a composition of two logspace transducers $R_1$ and $R_2$. The transducer $R_1$ outputs the biconnected components of size at most $k$. This can be realized by a logspace transducer, for each $k$, by the algorithm described in the proof of Theorem 7.2. The transducer $R_2$ first outputs a node for each biconnected component $C_i$, labeled with $C_i$. Then it tests each pairs of components $C_i$ and $C_j$, whether they have a nonempty intersection. In this case, $R_2$ outputs a node $n$ labelled with the vertex in $C_i \cap C_j$ (if such a node exists), and connects $n$ with the nodes corresponding to $C_i$ and $C_j$. The composition of $R_1$ and $R_2$ outputs a biconnected decomposition of $H$ and can be realized in logspace. □

7.3 Cycle cutset and hypercutset

A cycle cutset [26] (a.k.a feedback vertex set) of a hypergraph $H = (V, E)$ is a set of vertices, such that the induced subhypergraph on vertices $V \setminus S$ is acyclic. The cycle cutset width of $H$ is 0 if $H$ is acyclic, otherwise the minimal size over all of its cycle cutsets.

Theorem 7.3. Deciding whether a hypergraph $H = (V, E)$ has cycle cutset width at most $k$ is complete for L under $NC^1$ reductions.

Before we prove theorem 7.3, we study the complexity of the recognition problem for acyclic hypergraphs, which is also of independent interest. Using deep results from complexity theory, we can obtain the following theorem.
Theorem 7.4. Recognizing acyclic hypergraphs is complete for deterministic logspace under NC$^1$ reductions.

Proof. Gottlob et al. [41] have shown, that recognizing acyclic hypergraphs is in $SL$, therefore in $L$, since $L = SL$. Recognizing acyclic graphs is complete for deterministic logarithmic space [24], under NC$^1$ reductions. In the special case, when a hypergraph is a graph, the definition of acyclicity coincides with the usual definition of acyclic graphs. □

Proof. (Of Theorem 7.3)

Containment. A logspace algorithm can go through all subsets of $V$ of size $k$ and test whether the removal of the subset makes the hypergraph acyclic. Testing hypergraph acyclicity can be realized by a logspace oracle, because of Theorem 7.4. Thus, by Theorems 2.7 and 2.8, testing bounded cycle cutset width of a hypergraph is feasible in logspace.

Hardness. Testing, whether a hypergraph $H$ is acyclic, i.e. whether $H$ has cycle cutset width 0, is hard for $L$ under NC$^1$ reductions, see Theorem 7.4. □

A simple modification of the concept of cycle cutset is a cycle hypercutset, where we remove hyperedges, instead of vertices. A cycle hypercutset of a hypergraph $H = (V, E)$ is a set of edges $S$, such that the hypergraph induced by the vertices of $V \setminus \text{vertices}(S)$ is acyclic. The cycle hypercutset width of a hypergraph is the minimum cardinality over all of its possible cycle hypercutsets.

Theorem 7.5. Deciding whether a hypergraph $H = (V, E)$ has cycle hypercutset width at most $k$ is complete for $L$ under NC$^1$ reductions.

Proof. Analogous to the proof of Theorem 7.3. □

7.4 Hinge decomposition

Hinge decompositions are defined in [53]. We recall here the definitions.
Definition 7.1. ([53]) Let \( H = (V,E) \) be a hypergraph, \( G \subseteq E(H) \) and \( F \subseteq E(H) \setminus G \). The edge set \( F \) is connected with respect to \( G \), if for any two edges \( e, f \in F \), there exists a sequence of edges \( e_1, \ldots, e_n \) in \( F \) such that

- \( e_1 = e \),
- for \( i = 1, \ldots, n-1 \), \( e_i \cap e_{i+1} \) is not contained in \( \bigcup_{g \in G} g \),
- \( e_n = f \).

The maximal connected sets of \( E(H) \setminus G \) with respect to \( G \) are the connected components with respect to \( G \).

Definition 7.2. ([53]) Let \( H = (V,E) \) be a hypergraph and let \( G \) be a set of (at least two) edges of \( H \), i.e. \( G \subseteq E(H) \) and let \( C_1, \ldots, C_m \) be the connected components w.r.t. \( G \). Then \( G \) is a hinge, if for \( i = 1, \ldots, m \) there exists an edge \( g_i \in G \), such that \( \text{vertices}(C_i) \cap \text{vertices}(G) \subseteq g_i \).

A hinge is minimal if it does not contain any other hinge.

Definition 7.3. ([53]) A hinge decomposition of a hypergraph \( H \) is a tree with a labeling function \( \lambda \) that associates a set of hyperedges of \( H \) to the nodes of \( T \), such that the following conditions hold.

1. \( \forall p \in \text{nodes}(T), \lambda(p) \) is a minimal hinge of \( H \),
2. \( \forall e \in E(H), \exists p \in \text{nodes}(T), \) such that \( e \in \lambda(p) \),
3. if \( p \) and \( q \) are two nodes of \( T \), such that \( (p,q) \) is an edge of \( T \), then \( \exists e, e \in \lambda(p) \cap \lambda(q) \). Moreover, \( e = \text{vertices}(\lambda(p)) \cap \text{vertices}(\lambda(q)) \).
4. if \( s \) and \( r \) are two nodes of \( T \), then on the unique path between \( s \) and \( r \) in \( T \) each node contains the vertices of \( \text{vertices}(r) \cap \text{vertices}(s) \).

There is an interesting connection between hinge decompositions and biconnected component decompositions [48]: hinge decompositions of a hypergraph
correspond to biconnected decompositions of minimal reducts of its dual hypergraph. This, however, does not lead directly to design an efficient recognition algorithm for \(k\)-bounded hinge width as we do not know whether the optimal reducts are efficiently computable.

**Lemma 7.6.** Let \(H\) be a hypergraph, let \(k\) be a fixed constant number (\(2 \leq k \leq |E|\)) and let \(G\) be a subset of at most \(k\) hyperedges of \(H\). Testing whether \(G\) is a minimal hinge is feasible in logspace.

**Proof.** A logspace algorithm can be realized in the following way. First we test whether for each component \(C_i\) with respect to \(G\) there is an edge \(e_i \in G\), such that \(G \cap C \subseteq e_i\). If this holds, we do the same test for all subsets of \(G\). If none of the subsets of \(G\) is a hinge, then we can conclude that \(G\) is a minimal hinge. Since the set \(G\) has only \(2^k\) (a constant number of) subsets, the algorithm can be implemented using only logarithmic space. \(\square\)

Figure 7.5 gives a high level description for the algorithm \(k\)-hinge. We apply the same proof technique as for bounded-width hypertree decompositions in [43] and the proof is similar to the proof presented in section 7.2.

**Lemma 7.7.** Let \(H\) be a hypergraph. The algorithm \(k\)-hinge accepts \(H\) if and only if \(H\) has hinge width at most \(k\).

**Proof.** Analogous to the proof of Lemma 7.2 and 7.1. \(\square\)

**Theorem 7.6.** Deciding whether a hypergraph \(H\) has hinge width at most \(k\) is feasible in LogCFL.

**Proof.** The algorithm \(k\)-hinge can be implemented on a logspace ATM such that its witness tree is of polynomial size. \(\square\)
**ALTERNATING ALGORITHM** \( k \)-hinge

**Input:** hypergraph \( H \), (non-empty)

**Result:** \textbf{Accept}, if the biconnected width of \( H \) is at most \( k \), \textbf{Reject} otherwise.

**Procedure** \( k - \text{hingeDecomp}(C_R : \text{ConnectedComponent}, R : \text{SetOfEdges}) \)

\begin{verbatim}
begin
1) \textbf{Guess} a set \( S \) of edges of size at most \( k \)
2) \textbf{Check}
\hspace{1em}a) whether \( S \) is a minimal hinge
\hspace{1em}b) whether there exists an edge \( e \), such that \( e = \text{vertices}(R \cap S) \).
3) If the check above fails \textbf{Then Halt} and \textbf{Reject}; \textbf{Else}
\hspace{1em}Let \( C = \{ C \mid C \text{ is a component w.r.t. } S \text{ and } C \subseteq C_R \} \)
\hspace{1em}4) If, for each \( C \in C, k - \text{hingeDecomp}(C, S) \)
\hspace{1em}then \textbf{Accept} else \textbf{Reject}
end;
\end{verbatim}

\begin{verbatim}
begin (\textbf{*Main*})
\textbf{Accept} if \( k - \text{hingeDecomp}(V, \emptyset) \)
end.
\end{verbatim}

Figure 7.5: Recognizing hypergraph with hinge width at most \( k \).
7.5 Spread cut decomposition

In this section we use the new definition of spread cut from Cohen et al. [21]. We show that testing whether a hypergraph $H$ has bounded spread cut width, i.e. testing whether $scw(H) \leq k$, is feasible in LogCFL.

**Theorem 7.7.** Testing whether a hypergraph $H$ has bounded spread cut width, i.e. $scw(H) \leq k$, is feasible in LogCFL.

**Proof.** Because of Lemma 6.7, instead of testing $scw(H) \leq k$, we can test whether $hw(H') \leq k$, where $H' = H \cup \Delta_{k}(H)$, with some additional tests described by Lemma 6.7. The $k$-spreadCut depicted on Figure 7.6 is a variant of the algorithm $k$-decomp [43] for testing $k$-bounded hypertreewidth. The difference is that $k$-spreadCut performs more tests in Step 3, and in Step 2, the Set $ChiS$ is computed and passed as a procedure parameter.

The data structures needed for an efficient implementation are depicted in Figure 7.7.

The set $\Delta_{k}(H)$ can clearly be computed in logspace. Since both the $\lambda(p)$- and $\lambda(p)$-components can be represented in logspace, testing whether a pair $\langle \lambda(p), \chi(p) \rangle$ satisfies condition (1) of the Definition 6.3 can be tested with the following simple logspace algorithm. For each $\lambda(p)$-component $C$, go through the possible $\lambda(p)$-components and if $C$ meets more than one $\lambda(p)$-component then stop and output false, otherwise proceed until all $\lambda(p)$-components are checked.

Testing whether a pair $\langle \lambda(p), \chi(p) \rangle$ respects labels is also feasible in logspace. Note that in general logspace is not sufficient to store the labels of a vertex, as a vertex may be adjacent to the edges of many, i.e. a linear number of $\lambda(p)$-components, but we can test whether two labels are equal using only logspace. If a vertex $v$ is adjacent to the edges of a component $C$, we can test whether any other vertex $w$ is also adjacent. We can perform this test for all components (there are at most linearly many of them).

Thus, each of the above two tests can be performed using a logspace oracle. By
**ALTERNATING ALGORITHM** *k-spreadCut*

**Input:** hypergraph \( H \) (non-empty).

**Result:** *Accept*, if the spread-cut width of \( H \) is at most \( k \), *Reject* otherwise.

**Procedure** \( k-sCut(C_R : SetOfVertices, R : SetOfHyperedges, \ Chi_R : SetOfVertices) \)

`begin`

1) *Guess* a set \( S \) of \( k \) hyperedges from \( H \cup \Delta_k \)

2) *Calculate* \( \Chi_S \)

3) *Check* that all of the following conditions hold:

3.a) \( \forall e \in \text{edges}(C_R) \ vertices(e) \cap \text{vertices}(R) \subseteq \text{vertices}(S) \)

3.b) \( \text{vertices}(S) \cap C_R \neq \emptyset \)

3.c) each \( [\Chi_S] \)-component has nonempty intersection with at most one \( [\text{vertices}(S)] \)-component

3.d) \( (S, \Chi_S) \) respects labels

4) *If* the check above fails *Then Halt* and *Reject*; *Else*

Let \( C = \{ C \subseteq V \mid C \text{ is a } [\text{vertices}(S)]\text{-component and } C \subseteq C_R \} \)

5) *If, for each* \( C \in C \), \( k-sCut(C, S, \Chi_S) \)

*then Accept*

*else Reject*

`end;`

`begin (*Main*)`

*Accept* if \( k-sCut(V, \emptyset, \emptyset) \)

`end.`

Figure 7.6: Alternating algorithm for recognizing bounded spread-cut width

Theorem 2.7, the algorithm \( k\text{-spreadCut} \) can be implemented on an alternating logspace Turing machine. The ATM has a polynomial size witness tree, since each \( \langle \lambda(p), \chi(p) \rangle \) set is used only once, therefore by Ruzzo’s characterization of LogCFL [74], testing \( k \)-bounded spread cut width is feasible in LogCFL. \( \Box \)
7.6 Bounded-dimension hypergraphs

The *dimension* of a hypergraph $H(V,E)$ is the maximum size of its edges, i.e. $\text{dim}(H) = \max_{e \in E} |e|$. In this section we analyze the tree projection problem, with input hypergraphs $H$ and $H'$, denoted as $\text{TPP}(H,H')$, where the hypergraph $H'$ has bounded dimension. The TPP is NP-hard in general, see [45], but it can be solved in polynomial time in the bounded-dimension case. As a consequence, for this class of hypergraphs, the generalized hypertree recognition problem also becomes tractable.

By the close connection between TPP and decompositions, the existence of a solution for $\text{TPP}(H,H')$ can be formulated as the existence of a tree with two labeling functions as follows.

**Lemma 7.8.** The $\text{TPP}(H,H')$ has a solution if and only if there exists a triple $(T,\chi,\lambda)$ such that $T$ is a tree with two labeling functions, $\chi$ and $\lambda$ associating a set of vertices of $H$ and hyperedges of $H'$ to the nodes of $T$, such that the following conditions hold.

1. For each edge $e$ of $H$, there exists a node $p$ of $T$, such that $\text{vertices}(e) \subseteq \chi(p)$.

---

The results of this section were presented in a TU Wien Technical Report [38] in 2005 and were partially motivated by the results of Isolde Adler in [2]. We learned via personal communication from Isolde Adler (in 2007) that she independently obtained similar generalizations of her results on bounded dimension hypergraphs. She presented them in her PhD thesis [3], submitted in 2006. The proofs presented here are different from [3].
(2) for each vertex \( v \in V(H) \), the nodes in \( \{ p \mid v \in \chi(p) \} \) induces a connected subtree of \( T \).

(3) for each node \( p \) of \( T \), \( \chi(p) \subseteq \text{vertices}(\lambda(p)) \).

Proof. The lemma is a simple reformulation of the existence condition for a solution for TPP with input hypergraphs \( H \) and \( H' \), in terms of a join tree. □

We say that a triple \( \langle T, \chi, \lambda \rangle \) satisfies the special condition, if \( \text{vertices}(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p) \), where \( T_p \) denotes the subtree rooted at node \( p \). In the following lemma we formulate a simple observation. This formulation of the lemma opens a clear way to design algorithms for the tree projection problem.

Lemma 7.9. Let \( H \) and \( H' \) be hypergraphs, such that \( H \leq H' \) and \( H' \) has bounded dimension. Then, \( \text{TPP}(H, H') \) has a solution if and only if there exists a hypergraph \( H'' \), such that \( H \leq H'' \leq H' \) and \( \text{TPP}(H, H'') \) has a solution, which also satisfies the special condition.

Proof. Let \( D = \langle T, \chi, \lambda \rangle \) be a solution for \( \text{TPP}(H, H') \). Let us define a hypergraph \( H_D \) as follows. \( V(H_D) = V(H) \) and \( E(H_D) = E(H) \cup \{ \chi(p) \mid p \in \text{nodes}(T) \} \). Clearly, \( H \leq H_D \leq H' \). We define \( F = \langle T, \chi, \delta \rangle \), where for each node \( p \) of \( T \), \( \delta(p) = \{ e \mid e = \chi(p) \} \). Note, that by the definition of \( H_D \), for each \( p \), \( \delta(p) \neq \emptyset \). Then \( F \) is clearly a solution of \( \text{TPP}(H, H_D) \).

Furthermore \( F \) satisfies the special condition. We can show this indirectly: for each node \( p \) in \( F \), by our definition, \( \text{vertices}(\delta(p)) = \chi(p) \). If the special condition does not hold then the solution violates the connectedness condition (the condition 2 of Definition 7.8).

On the other hand, if there exists a hypergraph \( H'' \), such that \( H \leq H'' \leq H' \) and there is a solution for \( \text{TPP}(H, H'') \) satisfying the special condition, then there exists a solution for the problem \( \text{TPP}(H, H') \), too. □

Theorem 7.8. Let \( H \) and \( H' \) be hypergraphs, such that \( H' \) has bounded dimension, i.e. \( \dim(H') \leq d \). Deciding whether the \( \text{TPP}(H, H') \) has a solution is feasible in LogCFL.
ALTERNATING ALGORITHM treeProjection
Input: $H, H'$, such that $H < H'$.
Result:
Accept, if $TPP(H, H')$ has a solution,
Reject otherwise.

Procedure $TPP_{solvable}(C_R : SetOfVertices, ChiR : SetOfVertices)$

(*SetOfVertices $\subseteq V(H), Hyperedge \subseteq E(H')* )$

begin
1) Guess a set of vertices $ChiS$ of $H$
2) Check that all of the following conditions hold:
   2.a) $\forall e \in edges(C_R), \text{vertices}(e) \cap \text{vertices}(R) \subseteq \text{ChiS}$
   2.b) $\text{ChiS} \cap C_R \neq \emptyset$
   2.c) whether there exists an edge $S$ in $H'$, such that $\text{ChiS} \subseteq \text{vertices}(S)$
3) If the check above fails Then Halt and Reject; Else
   Let $C = \{ C \subseteq V \mid C \text{ is a } [\text{ChiS}] \text{-component and } C \subseteq C_R \}$
4) If, for each $C \in (C), TPP_{solvable}(C, \text{ChiS})$
   then Accept
   else Reject
end;

begin (*Main*)
Accept if $TPP_{solvable}(V(H), \emptyset)$
end.

Figure 7.8: Alternating algorithm for $TPP(H, H')$, where $H'$ has bounded dimension

Proof. The high level description of a decision algorithm treeProjection is depicted in Figure 7.8. The algorithm is a variant of the k-decomp algorithm from [43], and the containment in LocCFL can be proven in an analogous way, we only explain here the differences. Because of Lemma 7.9, instead of designing algorithms directly for bounded dimension $TPP(H, H')$, we can concentrate on problems $TPP(H, H'')$, with $H \leq H'' \leq H'$, for which the solution satisfies the
additional “special condition”. We do not know the hypergraph $H''$ in advance, but since $H \leq H'' \leq H'$, the sets ChiS—which correspond to the edges of $H''$—must also have bounded size, thus we can represent them in logarithmic space.

The witness tree of the algorithm treeProjection corresponds to a solution of $TPP(H, H'')$, satisfying the special condition. By the test at step 2.c), the hypergraph $H''$ satisfies the $H \leq H'' \leq H'$ condition. □

The following result was already known, see e.g. [86]. Nevertheless, we give here a surprisingly simple proof in our framework.

**Theorem 7.9.** Deciding whether a hypergraph $G$ has bounded treewidth at most $k$, i.e. $tw(G) \leq k$, is in LogCFL.

*Proof.* The theorem follows directly from theorem 7.8 and the relation between TPP and bounded treewidth (see Chapter 2). □

**Theorem 7.10.** Let $H$ be a hypergraph such that $\dim(H) \leq d$. Deciding whether the generalized hypertreewidth of $H$ is at most $k$, i.e. $ghw(H) \leq k$ is in LogCFL.

*Proof.* The theorem is a direct consequence of Theorem 7.8. □

### 7.6.1 Path decomposition

The above techniques can also be used to study recognition algorithms for bounded pathwidth hypergraphs: we can design an alternating algorithm for this class. This algorithm can be implemented even more efficiently than the algorithm for recognizing treewidth, by exploiting the special structure of the decomposition tree.

**Theorem 7.11.** Testing whether a hypergraph $H$ has bounded pathwidth, i.e. $pw(H) \leq k$ is feasible in NL.

*Proof.* As we have shown, the algorithm depicted in Figure 7.8 with inputs $H, H_k$ recognizes hypergraphs with treewidth at most $k$. If we extend the algorithm and at step 3 we also test whether the set $C$ contains at most 1 component, then we get
an algorithm for deciding $k$-bounded pathwidth. This additional test is feasible
in logspace by theorem 2.9, thus we can implement the algorithm on a logspace
ATM. Since the witness tree of the ATM has a special shape – it is a path – we can
implement the test using a nondeterministic Turing machine (without alternation).

\[\square\]

Analogously to (generalized) hypertree decomposition, we can introduce the
concept of (generalized) hyperpath decomposition: (generalized) hyperpath de-
composition of a hypergraph $H$ is a (generalized) hypertree decomposition
$\langle T, \chi, \lambda \rangle$, such that $T$ is a path. Let $\text{gpw}(H)$ denote the generalized hyperpathwidth
of $H$. Our motivation to introduce this concept is to possibly find a decomposition
concept with tractable recognition. In the case of graphs there exists a better upper bound for the complexity, in the case when the decompositions have a special shape, namely a path, so we could hope that such improvements can be made also in the case of hypergraphs. Unfortunately, the recognition problem for this class of hypergraphs is NP-hard.

**Theorem 7.12.** Let $H$ be a hypergraph. Testing whether $H$ has generalized hyperpathwidth at most $k$, i.e. $\text{gpw}(H) \leq k$ is NP-complete.

*Proof.* The reduction presented in [45] shows that $\text{ghw}(H) \leq k$ is NP-hard, even if $T$ is a path. \[\square\]
Chapter 8

Conclusion and future work

8.1 Conclusion

Solving Constraint Satisfaction Problems is NP-hard, therefore it is important to study tractable subclasses. Such subclasses can be found by associating a hypergraph to the problem and imposing structural restrictions on this hypergraph. We were interested in finding structures for which solving CSPs is feasible in polynomial time such that also the structures can also be recognized in polynomial time.

We studied the properties of a particularly important class, hypergraphs with bounded generalized hypertreewidth, which arise in a very natural way, by generalizing the concept of hypergraph acyclicity. Although the recognition problem for this class was shown to be NP-complete [45], we were able to define new tractable classes.

We introduced a novel way for defining new decomposition methods, using subedges. Using subedge functions we could define a particularly interesting new decomposition method, which improves all tractable methods known so far. Additionally, we got a very clear picture of tractable decomposition methods.

We studied the parallel complexity of the recognition problem for known decomposition methods, and showed --using a technique similar to testing bounded
hypertreewidth, presented in [43]—that these decision problems are feasible in a low complexity class $\text{LogCFL}$. In some cases, we found even better upper bounds.

### 8.2 Future work

The decomposition techniques for CSPs have been intensively studied, but we think that there is a lot of space for improvements. We think that the following areas are of particular interest for future research:

- Further improve the existing decomposition algorithms and methods.
- Understand and give a complete classification, if possible, for structural tractability.
- Understand the parallel complexity of the recognition of bounded width structures.
- Study decompositions related to hereditary hypergraph properties.

We think that the following open questions are particularly important in these areas, thus they should be further investigated.

**Improving the algorithms for existing decomposition methods** Many algorithms have been studied in the literature for hypertree decompositions.

- The best known upper bound for computing a hypertree decomposition of width $k$ is exponential in $2^k$. Can we do better?
- The best known approximation factor for $\text{GHW}$ is $3$, via hypertree decompositions. It is not clear whether this factor is tight, or if there exists a better bound for hypertree decompositions. Is it possible to define a decomposition method with a better approximation factor?
Towards a more complete understanding of tractable constraints  The results of this thesis contribute to understanding the tractable cases of CSPs. The following questions are particularly important to have a more complete picture of tractability.

- Are fractional hypertree decompositions [51] polynomially recognizable? Can we approximate FHDs?
- Are there even more general classes of hypergraphs for which the CSP problem becomes tractable?
- Can we give a classification of tractable constraints, in a similar fashion as in [52], but in terms of the associated hypergraphs instead of the primal graphs?

Better understanding of parallel complexity of recognition problems  As we discussed in Chapter 7, the recognition problem for the bounded width structures defined corresponding to various decomposition methods is in a low parallel complexity class \( L \subseteq \text{CFL} \). However, it is not clear whether this result can be further improved.

- Because of its central role we think that it would be important to better understand the parallel complexity of bounded treewidth recognition. It is known that recognizing hypergraphs with bounded treewidth is feasible in \( \text{LooCFL} \). Is there a better upper bound? Can we give a lower bound for this problem? What is the exact (parallel) complexity of this problem?

- A closely related problem is bounded pathwidth recognition. We have shown that bounded pathwidth recognition is in \( \text{NL} \). (Theorem 7.11) Can we improve this result?

- Similarly, we can ask whether it is possible to improve the known upper and lower bounds for the bounded hypertreewidth recognition problem?
If one could show that the recognition problem for any of the studied bounded width classes was in LogDCFL, this could lead to more efficient parallel algorithms. Problems in LogDCFL can be realized on more realistic models of parallel computation [29]. We can also turn the question around and ask whether it is possible to define a decomposition method, such that the recognition problem for the corresponding bounded width class is in LogDCFL.

The tractable decomposition methods could have a close connection to the context-free hyperedge replacement grammars, as presented e.g. in [57], but this connection needs to be further explored.

Let $H$ be a hypergraph. We have shown that testing $\gamma(H) \leq 1$, i.e. recognizing acyclic hypergraphs, is complete for L, under $\text{NC}^1$ reductions (see Theorem 7.4). The authors of [45] prove that testing $\gamma(H) \leq 3$ is $\text{NP}$-complete. What is the complexity of testing $\gamma(H) \leq 2$?

**$\beta$-hypertreewidth** There are many ways to define hypergraph acyclicity and there has been a long discussion in the database community, as to which acyclicity concept should be used. A detailed comparison of acyclicity concepts can be found in a seminal paper by Fagin [30], who showed that we can speak about degrees of acyclicity.

Theorem 8.1. (Fagin [30])

Let $H$ be a hypergraph. Then, $H$ is $\gamma$-acyclic $\implies$ $H$ is $\beta$-acyclic $\implies$ $H$ is $\alpha$-acyclic. None of the reverse implications hold.

Database theory adopted the most general acyclicity concept, the $\alpha$-acyclicity. Also in this thesis, if we referred to acyclic hypergraphs without specifying the degree, we meant $\alpha$-acyclicity. The concept of $\alpha$-acyclicity, however, has a counter-intuitive property: edge-subhypergraphs of $(\alpha)$-acyclic hypergraphs can be $(\alpha)$-cyclic.
Example 8.1. The hypergraph depicted in Figure 8.1 is clearly $\alpha$-acyclic, but if we remove the edge $(A, B, C)$, then the resulting edge-subhypergraph (containing the edges $\{(A, B, E), (B, C, F), (A, C, D)\}$) is $\alpha$-cyclic.

![Figure 8.1: $\alpha$-acyclic, but $\beta$-cyclic hypergraph](image)

Definition 8.1. (Fagin [30])

A hypergraph $H$ is $\beta$-acyclic, if $H$ is $\alpha$-acyclic and all of its edge-subhypergraphs are $\alpha$-acyclic.

Thus, Figure 8.1 depicts an $\alpha$-acyclic hypergraph, which is not $\beta$-acyclic.

Decomposition concepts studied in this thesis and generalized hypertree decompositions of bounded width can be seen as generalizations of the concept of $\alpha$-acyclicity. Indeed, hypergraphs with generalized hypertreewidth 1 are exactly the $\alpha$-acyclic hypergraphs [44]. Thus, the above mentioned counterintuitive properties are inherently present in the defined, more liberal classes of hypergraphs as well.

The research on decompositions in recent years has focused on finding larger and larger classes of hypergraphs with tractable recognition algorithms. We think that this goal can partially be given up, for the prize of having additional useful properties of decompositions. In particular, generalizations of $\beta$-acyclicity could result in such useful concepts. Gottlob et al. [46] defined a possible generalization of the concept of $\beta$-acyclicity in this way, using hypertree decompositions.
The $\beta$-hypertreewidth of a hypergraph $H$ is at most $k$ if $hw(H) \leq k$, and for all edge-subhypergraphs $H'$ of $H$, $hw(H') \leq k$.

It is not known, whether hypergraphs with bounded $\beta$-hypertreewidth can be recognized in polynomial time. As this question can be difficult to answer, our experience with subedges suggests that it can simplify the problem, if we restrict our attention to hypergraphs, which do not have subedges.

**Example 8.2.** Let $H_0$ be the hypergraph of Example 1.1. Let us define $H$ as $H = H_0 \cup (X_3, X_9)$.

The hypertreewidth of $H$ is 2, a $HD$ of width 2 is depicted on Figure 5.3. The edge subhypergraph we get after removing the hyperedge $(X_3, X_9)$ has hypertreewidth 3.
Bibliography


# Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>biconnected components</td>
<td>95</td>
</tr>
<tr>
<td>component hypertree decomposition</td>
<td>59</td>
</tr>
<tr>
<td>cycle cutset</td>
<td>101</td>
</tr>
<tr>
<td>extended component hypertree decomposition</td>
<td>88</td>
</tr>
<tr>
<td>generalized hypertree decomposition</td>
<td>23</td>
</tr>
<tr>
<td>guarded block</td>
<td>60</td>
</tr>
<tr>
<td>guarded cover</td>
<td>61</td>
</tr>
<tr>
<td>hinge decomposition</td>
<td>102</td>
</tr>
<tr>
<td>hypertree decomposition</td>
<td>22</td>
</tr>
<tr>
<td>join tree</td>
<td>16</td>
</tr>
<tr>
<td>normal form decomposition</td>
<td>39</td>
</tr>
<tr>
<td>$[S]$-connected vertices</td>
<td>24</td>
</tr>
<tr>
<td>spread cut</td>
<td>60</td>
</tr>
<tr>
<td>spread cut (new)</td>
<td>78</td>
</tr>
<tr>
<td>subedge function</td>
<td>49</td>
</tr>
<tr>
<td>subedge-based decomposition</td>
<td>49</td>
</tr>
<tr>
<td><code>treecomp(s)</code></td>
<td>43</td>
</tr>
<tr>
<td>tree decomposition</td>
<td>20</td>
</tr>
<tr>
<td>Tree Projection Problem</td>
<td>25</td>
</tr>
<tr>
<td>unbroken component</td>
<td>61</td>
</tr>
</tbody>
</table>