Generalized Hypertree Decompositions: NP-Hardness and Tractable Variants

DBAI-TR-2007-55

Georg Gottlob  Zoltán Miklós  Thomas Schwentick

DBAI TECHNICAL REPORT
MARCH 21, 2007
Generalized Hypertree Decompositions: NP-Hardness and Tractable Variants

Georg Gottlob\textsuperscript{1} \hspace{1em} Zoltán Miklós\textsuperscript{2} \hspace{1em} Thomas Schwentick\textsuperscript{3}

Abstract. The generalized hypertree width $GHW(H)$ of a hypergraph $H$ is a measure of its cyclicity. Classes of conjunctive queries or constraint satisfaction problems whose associated hypergraphs have bounded $GHW$ are known to be solvable in polynomial time. However, it has been an open problem for several years if for a fixed constant $k$ and input hypergraph $H$ it can be determined in polynomial time whether $GHW(H) \leq k$. Here, this problem is settled by proving that even for $k = 3$ the problem is already NP-hard. On the way to this result, another long standing open problem, originally raised by Goodman and Shmueli in 1984 in the context of join optimization is solved. It is proven that determining whether a hypergraph $H$ admits a tree projection with respect to a hypergraph $G$ is NP-complete. Our intractability results on generalized hypertree width motivate further research on more restrictive tractable hypergraph decomposition methods that approximate general hypertree decomposition ($GHD$). We show that each such method is dominated by a tractable decomposition method definable through a function that associates a set of partial edges to a hypergraph. By using one particular such function, we define the new Component Hypertree Decomposition method, which is tractable and strictly more general than other approximations to $GHD$ published so far.

\textsuperscript{1}Institut für Informationssysteme, Technische Universität Wien, Favoritenstr. 9, A-1040 Wien, Austria. E-mail: gottlob@dbai.tuwien.ac.at
\textsuperscript{2}Institut für Informationssysteme, Technische Universität Wien, Favoritenstr. 9, A-1040 Wien, Austria. E-mail: miklos@dbai.tuwien.ac.at
\textsuperscript{3}Universität Dortmund, Lehrstuhl Informatik I; E-mail: thomas.schwentick@udo.edu

Acknowledgements: This paper was supported by the Austrian Science Fund (FWF) project: Nr. P17222-N04, Complementary Approaches to Constraint Satisfaction.

Copyright © 2007 by the authors
1 Introduction and Overview

Nearly Acyclic Hypergraphs and Hypergraph Decompositions. It is well-known that acyclic conjunctive queries, i.e. queries with an acyclic query hypergraph, are solvable in polynomial time [7]. A similar result holds for many other problems that can be structurally characterized through hypergraphs. 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 the 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 [26]. Secondly, there exists a very successful generalization of graph acyclicity, namely, bounded treewidth [24]. A large number of graph-based problems are tractable on instances of bounded treewidth [10, 4, 5, 21, 11]. 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 in the constraint satisfaction communities, various methods of hypergraph decompositions have been 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 [16]. 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)** [19, 3], also called acyclic guarded cover in [9].

**Generalized Hypertree Decompositions.** The concept of a GHD is intuitively explained by the following example, adapted from [2, 3].

Consider the Boolean conjunctive query 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 \(H_0 = (V_0, E_0)\) associated with the query, depicted in Figure 1, has the vertex set \(V_0 = \{v_1, v_2, \ldots, v_{10}\}\), where for each query variable \(X_i\) there is a vertex \(v_i\) and and an edge set
$E_0$ which consists of the following edges:

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

$GHD$s of width 2 and 3 of $H_0$ (and of query $Q_0$) are depicted in Figure 2.a and 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 $GHW(D)$, 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 $GHW(H)$ of $H$ is the minimum width over all possible $GHD$s of $H$. Note that a hypergraph $H$ is acyclic iff $GHW(H) = 1$. In [18] it was shown that $GHD$s satisfy the Polynomial Query Evaluation property. In particular, given a Boolean query $Q$, with 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 hypertree decompositions of smaller width leads to better query answering algorithms.

![Figure 1: Hypergraph $H_0$ of the query $Q_0$.](image-url)

**Is Bounded $GHW$ Polynomially Recognizable?** A major question was whether for a fixed constant $k$ and an input hypergraph $H$, it can be determined in polynomial time if $GHW(H) \leq k$, i.e. whether bounded $GHW$ is polynomially recognizable, and if so, whether a $GHD$ of $H$ of width $k$ can be computed in polynomial time. These questions were first posed as open problems in 2001 (in the PODS’01 conference version of [19]) and have since been re-posed several times by various authors, for example, in [9].

The analysis of generalized hypertree decompositions is combinatorially involved. Rather than attacking the $GHW$ recognition problem directly, we first dealt with a conceptually and combinatorially somewhat simpler related problem, namely, the problem of determining whether a hypergraph $H$ admits a tree projection with respect to another hypergraph $G$.

**Tree Projections.** For two hypergraphs $H_1$ and $H_2$ we write $H_1 \leq H_2$ iff each edge of $H_1$ is contained in at least one edge of $H_2$. Let $G$ and $H$ be hypergraphs such that $G \leq H$. A tree
A projection of $H$ with respect to $G$ is an acyclic hypergraph $H'$ such that $G \leq H' \leq H$. Tree projections were studied in [14, 23, 25] in the context of query optimization. In particular, in [14] the following Tree Projection Theorem was shown. A query program $P$ consisting of a sequence of projections, selections, or 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 Tree Projection Problem has as instance a pair $(G, H)$ of hypergraphs and asks whether $H$ has a tree projection with respect to $G$. If such a tree projection exist, it is also called an acyclic hypergraph sandwich, and the Tree Projection Problem is also referred to as the Acyclic Hypergraph Sandwich Problem [15]. For other types of “sandwich” problems, see [12, 13]. The complexity of the Tree Projection Problem has been repeatedly stated as an open problem for over twenty years [14].

Relating Tree Projections to GHW. As already pointed out in [19], there is an interesting connection between tree projections and generalized hypertree width. For a hypergraph $H = (V, E)$, denote by $H^k$ the hypergraph $(V, E^k)$, where $E^k$ are all unions of $k$ or less hyperedges from $H$. The following lemma, implicit in [19], follows directly from the definitions of $GHD$ and of tree projections:

Lemma 1.1 ([19]) For each hypergraph $H$, $GHW(H) \leq k$ if and only if $H^k$ has a tree projection with respect to $H$.

Lemma 1.1 can be seen as an easy polynomial-time reduction from the $GHW$ recognition problem to the tree projection problem. This means that if checking “$GHW(H) \leq k$” turned out to be NP-complete for some constant $k$, then the tree projection problem would be NP-complete, too. Conversely, if the tree projection problem is tractable, then so is the $GHW$ recognition problem. Given that, in addition, the tree projection problem appeared to be simpler, we first attacked this problem.
**Complexity of the Tree Projection Problem.** We are able to exhibit a polynomial-time transformation from 3SAT into the tree projection problem. We thus obtain the following result:

The tree projection problem is NP-complete.

This result is of independent interest. It entails hardness results for problems of query optimization discussed in [14, 23, 25].

**Complexity of GHW recognition.** By using a similar but noticeably more involved construction as for the tree projection problem, we are able to polynomially transform 3SAT into the problem of checking whether a hypergraph has generalized hypertree width at most 3. The additional difficulty arises from the fact that now it is no longer sufficient to polynomially transform a 3SAT instance into a pair of hypergraphs \((G, H)\) such that \(H\) has tree projection with respect to \(G\). Instead, according to Lemma 1.1, we shall transform 3SAT into an instance of the tree projection problem of the form \((G, G^3)\). To achieve this, we make use of involved coding and padding methods. We thus obtain the following result:

Deciding if \(GHW(H) \leq 3\) is NP complete.

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

**Approximating GHDs.** The unfavorable complexity results related to generalized hypertree decompositions motivate the search for somewhat weaker hypergraph decomposition methods that in some sense approximate \(GHDs\), and that fulfill the criteria of polynomial query evaluation and polynomial recognizability. In this paper, we concentrate on decomposition methods \(M\) which associate with each hypergraph \(H\) a set \(M(H)\) of generalized hypertree decompositions of \(H\) and search for a \(GHD\) in \(M(H)\) of minimal width. Intuitively, we thus consider methods which “approximate \(GHD\) from above”. The width \(MW(H)\) of a hypergraph \(H\) according to some decomposition method \(M\) is the minimum \(GHW\) of a decomposition in \(M(H)\). 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 is some hypergraph \(H\) such that \(MW(H) < NW(H)\), then we write \(M < N\).

The only method we are aware of, that does not fit into this framework is the fractional hypertree decomposition method (FHD) [22]. This method is based on principles different from “near acyclicity”. It was shown in [22] that \(FHD < GHD\), but the computational properties of FHD are unexplored (we conjecture FHD is not polynomially recognizable unless \(P=NP\)).

The following well-known approximation methods will be considered here. **Query Decomposition** (QD)[8] with the associated notion of **query width** (QW), **hypertree decompositions** (HD) [18], with the associated notion of **hypertree width** (HW), and **spread cut decomposition** (SCD) [9] with the associated notion of **spread cut width** (SCW). All these decomposition methods explicitly restrict the sets \(\chi(p)\) that may appear at a decomposition node \(p\). In a QD, each set \(\chi(p)\) must coincide with the union of all edges in \(\lambda(p)\). This is a very strong restriction. The more general HD merely requires that any element \(v\) that appears in some edge of \(\lambda(p)\) but not in \(\chi(p)\), does not occur in \(\chi(p')\) of any descendant \(p'\) of \(p\) in \(T\) either. SCDs are defined through a similar condition (see Section 6). Of the three decompositions only HDs are polynomially recognizable; QDs and SCWs are not (unless \(P=NP\))
By results\(^1\) of [18, 3, 2, 9], \(GHD < HD < QD\), and \(GHD < SCD < QD\), while \(SCD\) and \(HD\) are incomparable. Note also that the term “approximation method” is also appropriate in the complexity theoretic sense. In fact, in [3] it was shown that for each hypergraph \(H\), \(GHW(H) \leq HW(H) \leq 3 \times GHW(H) + 1\), and thus both query width and hypertree width approximate \(GHW\) by a factor of 3.

**Subedge-Based Decomposition Methods.** Motivated by the goal to improve hypertree decompositions, 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 \(M \leq GHD\), 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 \(GHD < M < HD\), 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 \(V(H) - \text{vertices}(M)\), or all vertices that are not edge-connected to any component of \(V(H) \setminus M\), or all vertices from \(e \setminus (M \setminus \{e\})\) 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 \(SCD\) and found the following:

\[
CHD < HD \text{ and } CHD < SCD.
\]

\(^1\)The relation \(SC < QW\) follows from the definitions of [9].
In particular, for the hypergraph $H_0$ of Figure 1, we have $HW(H_0) = 3$ but $CHW(H_0) = SCW(H_0) = GHW(H_0) = 2$.

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

**Future Research** We think that the following questions are of particular interest for future research: (i) The best known approximation factor for $GHW$ is 3. Is it possible to define a decomposition method with a better approximation factor? (ii) Are fractional hypertree decompositions [22] polynomially recognizable? (iii) The best known upper bound for computing a hypertree decomposition of width $k$ is exponential in $2^k$. Can we do better?

**Structure of the Paper** In Section 2 we give some definitions. In Section 3 we show that the hypergraph projection problem is NP-complete. In Section 4 we show that determining whether $GHW(H) \leq 3$ is NP-complete. In Section 5 we introduce the concept of subedge-based decomposition and prove our general result about subedge-based decompositions. In Section 6 we define component hypertree decompositions and we compare $CHDs$ to $HDs$ and $SCDs$ and show that $CHDs$ are strictly more general than the others.

## 2 Preliminaries

A **hypergraph** is a pair $H = \langle V, E \rangle$ consisting of a set $V$ of vertices and a set $E$ of hyperedges. A hyperedge $e \in E$ is a subset of $V$. 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 $H(Q) = \langle V, E \rangle$ 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 $Q$. A query is acyclic if its associated hypergraph is acyclic. We refer to the standard notion of hypergraph acyclicity in database theory [1].

A join tree $JT(Q)$ for a conjunctive query $Q$ is a tree whose vertices are the atoms in the body of $Q$ such that whenever the same variable $X$ occurs in two atoms $A_1$ and $A_2$, then $A_1$ and $A_2$ are connected in $JT(Q)$ and $X$ occurs in each atom in the unique path linking $A_1$ and $A_2$. In other words, the set of nodes, where $X$ occurs induces a connected subtree of $JT(Q)$. Acyclic queries can be characterized in terms of join trees: A query is acyclic iff it has a join tree (see [6]).

A hypertree for a hypergraph $H = \langle V, E \rangle$ is a triple $\langle T, \chi, \lambda \rangle$, where $T = \langle N, F \rangle$ 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 2.1** ([18]) A hypertree decomposition of a hypergraph $H = \langle V, E \rangle$ is a hypertree $HD = \langle T, \chi, \lambda \rangle$, 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 \( p \in N \), vertices(\( \lambda(p) \)) \( \cap \) \( \chi(T_p) \) \( \subseteq \) \( \chi(p) \).

The width of a hypertree decomposition is defined as \( |HD| = \max_{p \in N} |\lambda(p)| \). The hypertree width of a hypergraph is the minimum width over all of its hypertree decompositions. We call a hypertree decomposition complete, if for all edges \( e \) of the hypergraph \( H \), there is a node \( p \in N \) such that \( e \in \lambda(p) \) and vertices(\( e \)) \( \subseteq \) \( \chi(p) \). We refer to the condition 4 of Definition 2.1 as the “special condition”. A hypertree \( \langle T, \chi, \lambda \rangle \) is called a generalized hypertree decomposition, if the conditions 1-3 of Definition 2.1 hold.

Let \( H = (V, E) \) be a hypergraph and let \( X, Y \in V \) be two vertices of \( H \) and \( S \subseteq V \) a subset of vertices. \( X \) and \( Y \) are \([S]\)-adjacent if there is an edge \( e \subseteq E \), such that \( \{X, Y\} \subseteq \text{vertices}(e) \setminus S \). The maximum \([S]\)-connected sets are called \([S]\)-components. We use the same short notation as in [18]: a \([p]\)-component denotes a \([\chi(p)]\)-component. In the case of hypertree decompositions, the \([p]\)-components and \([\text{vertices}(\lambda(p))]\)-components coincide (see lemma 5.8 in [18]). This does not hold for generalized hypertree decompositions, where a \([p]\)-component may have nonempty intersection with several \([\text{vertices}(\lambda(p))]\)-components.

Normal form hypertree decompositions play a crucial role in the proofs in [18].

Definition 2.2 ([18]) 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).

Proposition 2.3 (Gottlob et al.[18]) If \( H \) has a generalized hypertree decomposition of width \( k \), then \( H \) has a generalized hypertree decomposition of width \( k \) in normal form.

3 Complexity of the tree projection problem

Theorem 3.1 Tree Projection is NP-complete.

Proof Clearly, Tree Projection is in NP. The proof of NP-hardness is by a reduction from 3SAT. Let \( \varphi = \bigwedge_{i=1}^{m} (L_{i1} \lor L_{i2} \lor L_{i3}) \) be a 3SAT formula with \( m \) clauses and variables \( x_1, \ldots, x_n \).

We construct hypergraphs \( H_1 = (V, E_1) \) and \( H_2 = (V, E_2) \) such that \( H_1 \) has a join tree with respect to \( H_2 \) if and only if \( \varphi \) is satisfiable.

\( V \) has the following elements:
- for each \( i \leq n \) there are \( y_i \) and \( y'_i \),
- for each \( i \leq m + 1 \) and \( j \leq 2n + 2 \) there is \( a_{ij} \).
In the following, $Y$ denotes \{$y_1, \ldots, y_n$\} and $Y'$ denotes \{${y'_1, \ldots, y'_n}$\}. Further, $Y_i$ is $Y - \{y_i\}$ and $Y'_i = Y' - \{y'_i\}$.

We will use the convention that hyperedges of $H_1$ are denoted by lower case letters $e$... and hyperedges of $H_2$ by upper case symbols $E$... The hyperedges of $H_1$ are the following.

- for each $i, j, 1 \leq i \leq m, 1 \leq j \leq 2n + 2$, there is a hyperedge $e^j_i = \{a^j_i, a^j_{i+1}\}$,
- for each $j, 1 \leq j \leq 2n + 1$, there is a hyperedge $e^j_{m+1} = \{a^j_{m+1}, a^{j+1}_1\}$,
- for each $i, 1 \leq i \leq n$, there is a hyperedge $e_i = \{y_i, y'_i\}$, and
- there are hyperedges $e = \{a^1_1\} \cup Y$ and $e' = \{a^{2n+2}_{m+1}\} \cup Y'$.

$H_2$ has the hyperedges $E = \{a^1_1\} \cup Y \cup Y'$, $E' = \{a^{2n+2}_{m+1}\} \cup Y \cup Y'$ and, for each $i, j, k$, $1 \leq i \leq m, 1 \leq j \leq 2n + 2, 1 \leq k \leq 3$, a hyperedge $E^j_{ik}$ depending on $L_{ik}$ as follows:

- if $L_{ik} = x_p$, for some $p$ then $E^j_{ik} = \{a^j_i, a^j_{i+1}\} \cup Y \cup Y_p$,
- if $L_{ik} = \neg x_p$, for some $p$ then $E^j_{ik} = \{a^j_i, a^j_{i+1}\} \cup Y_p \cup Y'$.

Finally, for each $j \leq 2n + 1$, there is a hyperedge $E^j_{m+1} = \{a^j_{m+1}, a^{j+1}_1\} \cup Y \cup Y'$.

We show next that $\varphi$ is satisfiable if and only if $H_1$ has a join tree with respect to $H_2$.

Let us assume first that $\varphi$ has a satisfying truth assignment $\rho$. Then $T$ can be chosen as follows (See Figure 3).

- for each $i, j, 1 \leq i \leq m + 1, 1 \leq j \leq 2n + 2$, $T$ has a node $v^j_i$. There are two further nodes, $v$ and $v'$,
- for each $i, j, 1 \leq i \leq m, 1 \leq j \leq 2n + 2$, there is an edge $\{v^j_i, v^j_{i+1}\}$, and
- for each $j, 1 \leq j \leq 2n + 1$, there is an edge $\{v^j_{m+1}, v^{j+1}_1\}$,
- there are two additional edges: between $v$ and $v^1_1$, and between $v^{2n+2}_{m+1}$ and $v'$.

Thus, $T$ is a line from $v$ to $v'$.

![Figure 3: Join tree](image)

In order to define $\chi$ and $\lambda$, we fix, for each $i, a k_i, 1 \leq k_i \leq 3$ such that the $i$-th clause of $\varphi$, $L_{i1} \lor L_{i2} \lor L_{i3}$, is satisfied by $L_{ik_i}$, i.e. $L_{ik_i} = 1$ under $\rho$. Let us choose $p_i$, such that $L_{ik_i} = x_{p_i}$ or $L_{ik_i} = \neg x_{p_i}$.
For each $i, j, 1 \leq i \leq m, 1 \leq j \leq 2n + 2$, we let $\lambda(v_i^j) = E_{ik_i}^j$, and, for each $j, 1 \leq j \leq 2n + 1$, we let $\lambda(v_{m+1}^j) = E_{m+1}^j$. Finally, $\lambda(v) = E$ and $\lambda(v') = E'$.

Let $Z$ be the set $\{y_i | \rho(x_i) = 1\} \cup \{y_i' | \rho(x_i) = 0\}$. We define $\chi$ as follows.

- $\chi(v) = \{a_i^1\} \cup Y \cup Z$, $\chi(v') = \{a_{m+1}^{2n+2}\} \cup Y' \cup Z$,

- for each $i, j, 1 \leq i \leq m, 1 \leq j \leq 2n + 2$, let $\chi(v_i^j) = \{a_i^j, a_i'^{j+1}\} \cup Z$, and

- for each $j, 1 \leq j \leq 2n + 1$, let $\chi(v_{m+1}^j) = \{a_{m+1}^j, a_1^{j+1}\} \cup Z$.

It is not hard to see that $(T, \chi, \lambda)$ is indeed a join tree for $H_1$ with respect to $H_2$. The crucial point is that, for each $i, j, 1 \leq i \leq m, 1 \leq j \leq 2n + 2, \lambda(v_i^j) \subseteq \lambda(v')$, since $L_{ik_i} = 1$ and $L_{ik_i} = x_{p_i}$ or $L_{ik_i} = -x_{p_i}$, therefore $Z$ contains the “right” element for $E_{ik_i}^j$.

It remains to show that the existence of a join tree implies satisfiability of $\varphi$. To this end, let $\langle T, \chi, \lambda \rangle$ be a join tree for $H_1$ with respect to $H_2$.

Let $v, v'$ be nodes of $T$ that cover the hyperedges $e$ and $e'$, i.e. $e \subseteq \chi(v)$ and $e' \subseteq \chi(v')$. Let $P = v_1, \ldots, v_l$ ($v_1 = v, v_l = v'$) be the path from $v$ to $v'$ in $T$. For each $1 \leq i \leq m + 1, 1 \leq j \leq 2n + 2$ let $P_i^j$ be the set of nodes $w \subseteq P$ with $a_i^j \in \chi(w)$. Clearly, each $P_i^j$ is a subpath of $P$, and for $j < j'$ and $i \leq m$, the subpaths $P_i^j$ are disjoint from the subpaths $P_i^{j'}$ and the former are closer to $v$ than the latter. We denote, for each $j, 1 \leq j \leq 2n + 2$, the node of $P_i^j$ which is closest to $v$ by $u_j$. Further, we set $u_{2n+3} = v'$. Clearly, for each $j \leq 2n + 2$, the nodes of $P$ covering the hyperedges of the form $e_i^j$ lie between $u_j$ and $u_{j+1}$.

Let, for each $j, 1 \leq j \leq 2n + 2, X_j$ be the set $\chi(u_j) \cap (Y \cup Y')$ and let $X_{2n+3} \cap Y = \chi(v') \cap (Y \cup Y')$. As $Y \subseteq \chi(v)$ and $Y' \subseteq \chi(v')$, the sequence $X_1 \cap Y, X_2 \cap Y, \ldots, Y_{2n+3} \cap Y$ is non-increasing and the sequence $X_1 \cap Y, \ldots, X_{2n+3} \cap Y$ is non-decreasing. Furthermore, as the hyperedges $e_i = \{y_i, y_i'\}$ of $H_1$ must be covered, for each $i$ and $j$ it holds $y_i \in X_j$ or $y_i' \in X_j$.

Thus, there is a $j \leq 2n + 2$ such that $X_j = X_{j+1}$. And for all nodes $u$ between $u_j$ and $u_{j+1}$ it holds $X_j \subseteq \chi(u)$.

We derive a truth assignment for $x_1, \ldots, x_n$ from $X_j$ as follows. For each $i \leq n$, we set $\rho(x_i) = 1$ if $y_i \in X_j$ and otherwise $\rho(x_i) = 0$. Note that in the latter case $y_i' \in X_j$.

We claim that $\rho$ is a satisfying assignment for $\varphi$. Indeed, for each $i$, there must be a node $u$ between $u_j$ and $u_{j+1}$ which covers the hyperedge $e_i^j$. The only candidates are $E_{ik_i}^j, E_{ik}^j$ and $E_{ik}^j$. Thus, there must be a $k$ such that $X_j \subseteq \chi(u) \subseteq E_{ik_i}^j$. Consequently, if $L_{ik_i} = y_p$ then $y_p$ must be in $X_j$ and if $L_{ik_i} = -y_p$ then $y_p'$ must be in $X_j$. In either case $L_{ik_i}$ is satisfied by $\rho$ on $x_p$. Therefore, $\rho$ satisfies $\varphi$. \qed

4  Generalized hypertree decomposition

In this section we show the following result.

**Theorem 4.1** Testing whether a hypergraph has generalized hypertree width at most 3 is NP-complete.
The proof uses the same basic idea as the proof of Theorem 3.1. Nevertheless, the construction is considerably more complicated as, opposed to that proof, we can not choose \( H_2 \) freely but rather are forced to choose \( H_2 = H_1^3 \). Here, \( H_1^3 \) denotes the hypergraph with the same elements as \( H_1 \) whose hyperedges are all unions of three hyperedges of \( H_1 \).

Before we present the complete proof of Theorem 4.1, we describe the construction of a sub-hypergraph of \( H_1 \) with a particular property.

To this end, let \( V_0 = \{b_1, b_2, b_3, c_1, c_2, c_3, d\} \). Let \( A_1, A_2, A_3 \) be further sets of elements, pairwise disjoint and disjoint from \( V_0 \). We write \( A \) for \( A_1 \cup A_2 \cup A_3 \). Let \( H_1 = (V, E) \) be a hypergraph with \( V_0 \cup A_1 \cup A_2 \cup A_3 \cup \{a\} \subseteq V \) such that the only hyperedges containing elements from \( V_0 \) as follows.

- \{a, b_1\} \cup A_1, \{b_1, c_1\} \cup A_1, \{c_1, d\} \cup A_1,
- \{a, b_2\} \cup A_2, \{b_2, c_2\} \cup A_2, \{c_2, d\} \cup A_2,
- \{a, b_3\} \cup A_3, \{b_3, c_3\} \cup A_3, \{c_3, d\} \cup A_3,
- \{b_1, c_2\}, \{b_1, c_3\}, \{b_2, c_1\}, \{b_2, c_3\}, \{b_3, c_1\}, \{b_3, c_2\}.

The set containing these hyperedges is denoted by \( E_0 \).

**Claim 1.** Every join tree \( T \) of \( H_1 \) with respect to \( H_1^3 \) has nodes \( v_1, v_2, v_3 \) with the following properties:

- \{a, b_1, b_2, b_3\} \subseteq \chi(v_1)
- \{b_1, b_2, b_3, c_1, c_2, c_3\} \subseteq \chi(v_2)
- \{c_1, c_2, c_3, d\} \subseteq \chi(v_3)
- \( v_2 \) is on the path from \( v_1 \) to \( v_3 \)

The proof of the above claim is included in the full version of this paper [20]. We are now prepared to present the proof of Theorem 4.1.

**Proof** [of Theorem 4.1] The problem is clearly in NP. The lower bound is again by a reduction from 3SAT. Let \( \varphi \) be a propositional formula in conjunctive normal form with \( m \) clauses \( \varphi_i \) of the form \( L_{i1} \lor L_{i2} \lor L_{i3} \) and variables \( x_1, \ldots, x_n \). For convenience and without loss of generality we assume that \( \varphi_1 = \neg x_1 \land \neg x_2 \land \neg x_3 \) and \( \varphi_m = x_4 \land x_5 \land x_6 \). This can always be accomplished by adding 2 new clauses and 6 new variables without affecting the satisfiability.

We describe next the construction of a hypergraph \( H_1 = (V_1, E_1) \) that has a join tree with respect to \( H_1^3 \) if and only if \( \varphi \) is satisfiable.

In a nutshell, \( H_1 \) consists of two copies \( C, C' \) of the hypergraph of the above claim plus additional hyperedges connecting \( C \) and \( C' \) in a similar fashion as in the proof of Theorem 3.1. To this end, we use the same sets \( Y, Y' \) related to the variables of \( \varphi \) and elements of the form \( a^j \). In order to control (and restrict) the ways in which hyperedges are combined in \( T \) we use an additional large set \( S \) of further elements.

We then make sure that \( C \) contains \( S \) as well as \( Y \) and that \( C' \) contains \( S \) and \( Y' \) and that each pair \( \{y_i, y'_i\} \) occurs in some node. Thus, all nodes on the path of \( T \) which connects \( C \) with \( C' \) must contain \( S \) and, just as in Theorem 3.1, for each \( i \), one of \( y_i \) and \( y'_i \).
We now describe the construction of $H_1$ more formally. Let $l = (2n + 2)(m + 1)$. Let $S$ be \{1, \ldots, l\}^{3} \times \{0, 1\}^{5}$. The elements of $H_1$ are

- $a, b_1, b_2, b_3, c_1, c_2, c_3, d,$
- $a', b'_1, b'_2, b'_3, c'_1, c'_2, c'_3, d',$
- $y_1, \ldots, y_n,$
- $y'_1, \ldots, y'_n,$
- all elements of the form $a_i^j$ with $1 \leq i \leq m + 1,$
  $1 \leq j \leq 2n + 2$
- all elements from $S$.

Let again $Y$ denote \{y_1, \ldots, y_n\} and $Y'$ denote \{y'_1, \ldots, y'_n\}.

We introduce some notation for subsets of $S$ next. We write elements of $S$ in the form $(i_1, i_2, i_3, j_1, j_2, j_3, k_1, k_2)$, thereby splitting the 8 components into 3 groups. The wildcard * indicates that the respective component can carry arbitrary values. E.g., $(*, *, *, 1, *, *, *, *)$ denotes the set of tuples with $j_1 = 1$. If the wildcard occurs in all components of a group we replace by one wildcard *. Thus we can denote the above set also by $(*, 1, *, *, *)$.

For $i, j, k$ with $k = (j - 1)(m + 1) + i$, we write $S_{i,j}$ for the set $(k, *, *, *, *, *) \cup (*, k, *, *, *, *) \cup (*, *, k, *, *, *)$.

The hyperedges of $H_1$ are as follows:

- all hyperedges as mentioned before Claim 1 in this section with $A_1 = Y$, $A_2 = (\ast ; \ast ; 0, \ast ),$ A$_3 = (\ast ; \ast ; 1, \ast )$;

- all hyperedges as mentioned before Claim 1 with $a', b'_1, b'_2, b'_3, c'_1, c'_2, c'_3, d$ in place of $a, b_1, b_2, b_3, c_1, c_2, c_3, d$ and with $A_1 = Y'$, $A_2 = (\ast ; \ast ; *, 0), A_3 = (\ast ; \ast ; *, 1);$  
- $e_1^i = \{a, a_i^1\} \cup (S - S_{11}), e_{m+1}^{2n+2} = \{a', a_{m+1}^{2n+2}\} \cup (S - S_{m+1,2n+2});$
- for each $i, j, 1 \leq i \leq m, 1 \leq j \leq 2n + 2$, the hyperedge $c_i^j = \{a_i^j, a_i^{j+1}\} \cup (S - S_{i,j});$
- for each $j, 1 \leq j \leq 2n + 1$, the hyperedge $e_{m+1}^{j+1} = \{a_{m+1}^j, a_{m+1}^{j+1}\} \cup (S - S_{m+1,j});$
- for each $i, 1 \leq i \leq n$, there is a hyperedge $e_i = \{y_i, y'_i\};$
- finally there are, for each $i, j, (1 \leq j \leq 2n + 2, 1 \leq i \leq m + 1)$ six special hyperedges as follows.
  - If $L_{i,1}$ is $x_p$, for some $p$, then $H_1$ has the hyperedges $Y \cup (S_{i,j} \cap (\ast ; 0, \ast , \ast , \ast ))$ and $(Y' - \{y'_p\}) \cup (S_{i,j} \cap (\ast ; 1, \ast , \ast , \ast ))$.
  - If $L_{i,1}$ is $\neg x_p$, for some $p$, then $H_1$ has the hyperedges $(Y - \{y_p\}) \cup (S_{i,j} \cap (\ast ; 0, \ast , \ast , \ast ))$ and $Y' \cup (S_{i,j} \cap (\ast ; 1, \ast , \ast , \ast ))$. 


- If $L_{i,2}$ is $x_p$, for some $p$, then $H_1$ has the hyperedges $Y \cup (S_{i,j} \cap (\ast,\ast,0,\ast))$ and $(Y' - \{y_p\}) \cup (S_{i,j} \cap (\ast,\ast,1,\ast))$.
- If $L_{i,2}$ is $-x_p$, for some $p$, then $H_1$ has the hyperedges $(Y - \{y_p\}) \cup (S_{i,j} \cap (\ast,\ast,0,\ast))$ and $Y' \cup (S_{i,j} \cap (\ast,\ast,1,\ast))$.
- If $L_{i,3}$ is $x_p$, for some $p$, then $H_1$ has the hyperedges $Y \cup (S_{i,j} \cap (\ast,\ast,0,\ast))$ and $(Y' - \{y_p\}) \cup (S_{i,j} \cap (\ast,\ast,1,\ast))$.
- If $L_{i,3}$ is $-x_p$, for some $p$, then $H_1$ has the hyperedges $(Y - \{y_p\}) \cup (S_{i,j} \cap (\ast,\ast,0,\ast))$ and $Y' \cup (S_{i,j} \cap (\ast,\ast,1,\ast))$.

Now we show that $H_1$ has a join tree with respect to $H^3_1$ if and only if $\varphi$ is satisfiable.

To this end, let us first assume that $\varphi$ is satisfiable. Let $\rho$ be a satisfying truth assignment. Let $Z$ be the set
\[ \{y_i \mid \rho(x_i) = 1\} \cup \{y'_i \mid \rho(x_i) = 0\}. \]

We construct $T$ as a path $v_c$, $v_b$, $v_a$, $v$, $v'_1$, $v'_4$, $v'_5$, $v'_6$, $v'_b$, $v'_a$, see Figure 4. Here $\lambda(v_c)$ is composed by the hyperedges with $\{c_1, d\}$, $\{c_2, d\}$ and $\{c_3, d\}$ and $\chi(v_b) = \{d, c_1, c_2, c_3\} \cup S \cup Y$. Analogously, $\chi(v_b) = \{c_1, c_2, c_3, b_1, b_2, b_3\} \cup S \cup Y$ and $\chi(v_a) = \{b_1, b_2, b_3, a\} \cup S \cup Y$.

The nodes $v'_b$, $v'_a$, $v'_c$ are defined analogously with $Y'$ instead of $Y$.

The remaining nodes are defined such that the following holds.

- $\chi(v) = \{a, a^1\} \cup S \cup Z \cup Y$,
- $\chi(v') = \{a', a^2_{m+2}\} \cup S \cup Z \cup Y'$,
- for each $1 \leq j \leq 2n + 1$, $\chi(v^j) = \{a^j_{m+1}, a^{j+1}\} \cup S \cup Z$, and
- for each $i, j, 1 \leq i \leq m, 1 \leq j \leq 2n + 2$, $\chi(v_i^j) = \{a_i^0, a_i^{j+1}\} \cup S \cup Z$.

It is not hard to see, that $\lambda$ (and $\chi$) can be chosen in this way and that all hyperedges of $H_1$ are covered by $T$. It should be noted here that each special hyperedge is either covered by $v_c$ or $v'_c$.

It remains to show that $\varphi$ is satisfiable if $H_1$ has a join tree with respect to $H^3_1$. To this end, let $T$ be such a join tree. Let $C$ denote the subtree it has because of Claim 1 (with nodes $v_1, v_2, v_3$) and let $C'$ denote the corresponding subtree for the $a', b'_i, c'_i, d$ elements (with nodes $v'_1, v'_2, v'_3$). It is not hard to show that each node $v$ on the path $P$ of $T$ from $v_1$ to $v'_1$ has the following properties.

- $S \subseteq \chi(v)$
Furthermore, for each \(i \leq n\), there is a node \(v\) in \(P\) with \(\{y_i, y'_i\} \subseteq \chi(v)\).

It is easy to see that there can be no node in \(P\) which is composed by two or more hyperedges of the form \(e^i_1\): indeed there is no way to cover all of \(S\) by only one additional hyperedge.

Thus, \(P\) consists of disjoint subpaths \(P_0, P_1, \ldots, P_{2n+2}^j, P_{0}^j\) such that each node \(v\) in \(P_i^j\) fulfills \(e^i_1 \subseteq \lambda(v)\), for some \(i, j\). To cover all of \(S\), \(\lambda(v)\) must also contain two corresponding special hyperedges.

We fix a node \(v_1\) with \(\{a, a^1_1\} \subseteq \chi(v_1)\) and, for each \(j, 2 \leq j \leq 2n + 2\), we fix a node \(v_j\) with \(\{a^{j-1}_{m+1}, a^j_1\} \subseteq \chi(v_j)\). Similar to the proof of Theorem 3.1 we define, for each \(j\), \(X_j = \chi(v_j) \cap (Y \cup Y')\). Again, there must be a \(j\) such that \(X_j = X_{j+1}\). Just as in that proof, we obtain a truth assignment \(\rho\) by taking, for each \(i \leq n\), \(\rho(x_i) = 1\) if \(y_i \in X_j\) and otherwise \(\rho(x_i) = 0\). And again it is easy to show that \(\rho\) is actually a satisfying assignment for \(\varphi\).

This completes the proof of the theorem. \(\square\)

\section{Subedge-based GHDs}

Our intractability results on generalized hypertree width motivate further research on tractable decomposition methods that approximate generalized hypertree decompositions. In this section 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 proposition, which is merely a simple observation, sets up the stage for the considerations in this section.

\begin{proposition}
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 \(\chi'(p) = \{e \cap \chi(p) \mid e \in \lambda(p)\}\) 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 \(D\).
\end{proposition}

The proposition follows basically from the definitions of \(GHD\) and \(HD\). Nevertheless, it 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 that, in this paper, a decomposition method \(M\) associates with each hypergraph \(H\) a set \(M(H)\) of allowed \(GHDs\). In principle, we would be interested in methods that can be implemented by tractable algorithms. But as the experience from \(HD\) (and from tree decompositions in the case of graphs) shows, we cannot expect algorithms whose running time is polynomial, independent of the parameter \(k\). Thus, we say an algorithm \(A\) implements a decomposition method \(M\) if \(A\) on input \((H, k)\) outputs a \(GHD\) from \(M(H)\) of width \(\leq k\), if it exists, otherwise “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, each function \(f\) mapping a hypergraph \(H\) to a set of subedges of \(H\) induces a decomposition method: (1) Compute \(f(H)\), (2) compute a minimal \(HD\) \(D\) of \(H \cup f(H)\). As (2) is only feasible, for each fixed \(k\), it makes sense, to allow
If \( f \) to depend on the given \( k \) as well. Thus, a subedge function is a function \( f \), mapping each pair \((H, k)\) to a set of subedges of \( H \). To avoid technical complications, we further require that subedge functions be monotone in the following sense: for each \( i < j \),

\[ f(H, i) \subseteq f(H, j). \]

If \( D = (T, \chi, \lambda) \) is a HD of a hypergraph \( H \cup f(H, k) \) and \( D' = (\mathcal{T}, \chi', \lambda') \) is a GHDo of \( H \), we say that \( D' \) covers \( D \) if, for each \( p \), each \( e \in \lambda(p) \) is a subedge of some \( e' \in \lambda'(p) \).

With each subedge function \( f \) we associate the decomposition method \( M_f \) as follows: \( M_f(H) \) is the set of all \( \text{GHDo} \)s \( D' \) 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 \( D \). We call a decomposition method of the form \( M_f \) subedge-based.

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

The latter example shows that, in general, \( f(H, k) \) does not need to be of polynomial size. Nevertheless, as 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 5.2** (a) If \( f \) is polynomially computable, then, for each fixed constant \( k \), whether \( M_f(W(H) \leq k) \) can be decided in polynomial time, 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 [18] to compute a HD \( D \) of width \( i \leq k \) 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, by replacing each such \( e \) by the respective \( e' \) yields a HD \( D' \) of width \( i \) for \( H \).

Note that \( D' \) might not be in \( M_f(H) \) as \( |D'| \geq k \) 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\(^2\) 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 HD 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 computation to compute \( f(H, k) \) with the LOGCFL check [18, 17] whether the hypertree width of \( H \cup f(H, k) \) is \( \leq k \) (in the standard way known from complexity theory).

From the proof of Lemma 5.2 we can conclude:

**Corollary 5.3** For a hypergraph \( H \), \( M_f(W(H)) \) 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.

\(^2\)If no such edges exist, then \( H \) has less than \( k \) hyperedges and \( M_f(H) \leq k \) holds trivially by definition of \( M_f \).
Theorem 5.4 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 \( BW(H) \leq AW(H) \).

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 5.5 For each decomposition method \( M \) which can be implemented by a polynomial algorithm \( A \) there is a polynomial subedge function \( f \) such that \( M_f \leq M \).

Proof Let \( M \) and \( A \) as stated. Given a hypergraph \( H \) and a number \( k \), let \( D = \langle T, \chi, \lambda \rangle \) be the GHD of width \( k \) for \( H \) computed by \( A \). Let \( D' = \langle T, \chi', \lambda' \rangle \) be defined as in Proposition 5.1. Then we define \( f(H, k) = \bigcup_p \lambda'(p) \), where \( p \) ranges over all nodes of \( T \). As \( A(H, k) \) can be computed in polynomial time, \( f(H, k) \) is polynomial.

Furthermore, \( D' \) has width \( \leq k \) and is in \( M_f(H) \). As this holds, for every \( MW(H) \leq k \) we can conclude \( M_f W(H) \leq MW(H) \). \( \square \)

Of course, the function \( f \) in the proof of Theorem 5.5 depends on the ability of already computing a GHD. Thus, the reader might get the impression that the detour through \( f \) is not very useful. Nevertheless, in the next section we exhibit a polynomial subedge function \( f \) which is defined entirely in terms of \( H \) and does not involve the construction of a decomposition.

6 Component hypertree decomposition

In this section we give an example of a subedge defined decomposition, called “component hypertree decomposition”, that strictly generalizes both hypertree decomposition [18] and spread cut decomposition [9] and it is also tractable.

6.1 Definitions

Definition 6.1 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 \( \text{prop}(e, M) = e \setminus \bigcup_{e' \in M, e \neq e'} e' \).

Definition 6.2 Let \( M \) be a set of edges of the hypergraph \( H \) and let \( e \) be an edge in \( M \). We define the set \( \text{internal}(e, M) = \{ v \mid v \in \text{vertices}(e), e \in M \text{ and there exists no } [\text{vertices}(M)]- \text{component } C, \text{ such that } v \in \text{vertices}(\text{edges}(C)) \} \).

Definition 6.3 Let \( H \) be a hypergraph, let \( M \) be a set of edges of \( H \) 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. \( \text{internal}(e, M) \).

**Definition 6.4** Let \( H \) be a hypergraph, let \( M \) be a set of edges of \( H \) and let \( C \) be a \([\text{vertices}(M)]\)-component and \( e \in M \). We define the subedge function \( f^C \) as:

\[
 f^C(H, k) = \{ e \setminus e' \mid \text{M is a set of} \leq k \text{ hyperedges of } H, \\
e \in M, \\
C \text{ is a}[\text{vertices}(M)]-\text{component}, \\
\text{and } e' \in \text{elim}(M, C, e) \}.
\]

The decomposition method \( M_{f^C} \) referred as component hypertree decomposition (CHD).

According to our definition, the generalized hypertree decomposition of figure 2 a) is a component hypertree decomposition. A hypertree decomposition of the hypergraph \( H \cup f(H, 2) \) is depicted on figure 5.

![Figure 5: Hypertree decomposition of width 2 of the hypergraph \( H \cup f^C(H, 2) \), \( e'_2(v_3, v_9) \subseteq e_2 \), \( e'_3(v_3, v_{10}) \subseteq e_3 \). Note that \( \{e'_2, e'_3\} \subseteq f^C(H, 2) \).](image-url)

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 \([\text{vertices}(\lambda(p))]\)-components can be represented also in logarithmic space, and all of the required computations (computing connected components, intersections and difference of sets) is feasible in logspace, see e.g. [17]. Therefore, by lemma 5.2, deciding whether for a fixed constant \( k \), a given hypergraph \( H \) has component hypertree width at most \( k \), is feasible in \( \text{LOGCFL} \).

### 6.2 Comparison with other tractable decompositions

**Definition 6.5** ([9]) A normal form \(^3\) generalized hypertree decomposition \(<T, \chi, \lambda>\) of a hypergraph \( H \) is called spread cut decomposition (SCD) if additionally the following conditions hold:\(^4\)

\(^3\)Personal communication from the authors of [9]: the original definition in [9] does not ensure the existence of a tractable recognition algorithm. While finishing the present paper, we learnt that the authors of [9] have recently defined a new tractable variant of spread cut decomposition. We plan to compare this new decomposition method to subedge defined decompositions in the full version of this paper.

\(^4\)Condition 3 follows from the “canonical form” theorem, proven in [9] (Theorem 7.6). We find it convenient to include this condition in the definition.
1. For each node $p$ of $T$, each $[p]$-component meets at most one $[\text{vertices}(\lambda(p))]$-component.
2. For each node $p$ of $T$, for all pairs of edges $e_1, e_2 \in \lambda(p)$, $(e_1 \neq e_2)$, $e_1 \cap e_2 \subseteq \chi(p)$.
3. For each node $p$ of $T$, for each edge $e \in \lambda(p)$,
   
   (a) either $\forall \ v \in \text{internal}(e, M), v \in \chi(p)$,
   
   (b) or $\ \forall \ v \in \text{internal}(e, M), v \notin \chi(p)$ and for all $\ [\\text{vertices}(\lambda(p))]\$-components $C$,
   
   $\text{vertices}(\text{edges}(C) \cap e) \subseteq \chi(p)$.

**Lemma 6.6** Let $(T, \chi, \lambda)$ 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 $e \setminus \chi(p) = \text{prop}(e, M) \cap \text{vertices}(\text{edges}(C_e))$.

**Proof** (sketch) Assume that $e \setminus \chi(p)$ contains a vertex from $\text{internal}(e, \lambda(p))$. Then, by condition 3 of definition 6.5, $e \setminus \chi(p) = \text{internal}(e, \lambda(p))$. 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. □

**Definition 6.7** For $M$, $H$, $C$ and $e$ as in definition 6.3 let $\text{elim}^*(M, C, e)$ be defined as in definition 6.3 except that we only associate two subedges to a triple $(M, C, e)$, namely those mentioned in points 2 and 3 in Def. 6.3.

**Definition 6.8** 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 6.9** $M_f, \leq SC$

**Proof** (sketch) 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 6.8) and Corollary 5.3. □

**Theorem 6.10** $CHD < HD$ and $CHD < SCD$. 
Proof (sketch) Clearly, by Corollary 5.3, \( CHD \leq HD \). For hypergraph \( H_0 \) in the introduction, see figure 1, \( HW(H) = 3, CHW(H) = 2 \), therefore \( CHD < HD \).

Let us first prove first 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 5.3 and Lemma 6.9, \( CHD = M_{f^C} \leq M_{f^*} \leq SCD \).

For the hypergraph \( H \) on Figure 6, which is an adaptation of an example from Adler [2] for our purposes, \( CHW(H) = 5, SCW(H) = 6 \), therefore \( CHD < SCD \). The hypergraph is defined as follows. The vertices of \( H \) 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 on the figure. Each balloon vertex is connected by an edge to each ground vertex. All other edges are depicted in the figure.

![Figure 6](image)

It is easy to see that \( CHW(H) = 5 \). (A \( CHD \) of \( H \) of width 5 is included in the full version of this paper [20].) Similarly, an \( SCD \) of width 6 can be found therefore \( SCW(H) \leq 6 \). Assume that \( SCW(H) = 5 \). By using the Robber and Marshals game described in [19], it can be shown that for every \( CHD D \) of width 5, there must exists at least one decomposition node \( p \), such that one of the vertices \( E, F, E_1 \) or \( F_1 \) is in \( (vertices(\lambda(p)) \setminus \chi(p)) \). However, this is forbidden by condition 2 in the definition of \( SCD \) (Def. 6.5). Contradiction. □

Acknowledgements

The authors are very grateful to Isolde Adler for her comments and suggestions on earlier versions of this paper. The research was partially supported by the Austrian Science Found (FWF), under Project P17222-N04.
References


Appendix

The appendix contains proofs and examples, which have been excluded from the paper because of space limitations.

Section 4

For a tree $T$ and subtrees $s, t$ of $T$ we write $s \cap t$ for the set of nodes that $s$ and $t$ have in common.

We will need two results about subtrees of trees from [?] which are summarized in the following proposition.

Proposition 6.11 Let $T$ be a tree.

$(a)$ If $t_1, \ldots, t_k$ are subtrees of $T$, $k \geq 4$, such that, for each $i < k$, $t_i \cap t_{i+1} \neq \emptyset$ and $t_k \cap t_1 \neq \emptyset$, then there are $i$ and $j$, $i \neq j \pm 1$ (modulo $k$) with $t_i \cap t_j \neq \emptyset$.

$(b)$ If $t_1, \ldots, t_k$ are subtrees of $T$ such that $t_i \cap t_j \neq \emptyset$, for each $i, j \in \{1, \ldots, k\}$, then $T$ contains a node $v$ which is in every $t_i$.

Property (a) can be stated in more general terms. A tree $T$ and subtrees $t_1, \ldots, t_n$ induce a graph $G$ in the following way: the vertices of $G$ are $t_1, \ldots, t_n$, $(t_i, t_j)$ is an edge if $t_i \cap t_j \neq \emptyset$. It is shown in [?] that a graph is chordal if and only if it can be obtained in such a way.

Claim 1. Every join tree $T$ of $H_1$ with respect to $H_1^2$ has nodes $v_1, v_2, v_3$ with the following properties:

- $\{a, b_1, b_2, b_3\} \subseteq \chi(v_1)$
- $\{b_1, b_2, b_3, c_1, c_2, c_3\} \subseteq \chi(v_2)$
- $\{c_1, c_2, c_3, d\} \subseteq \chi(v_3)$
- $v_2$ is on the path from $v_1$ to $v_3$

Proof

For an element $z$ of $V_0$, we denote by $t_z$ the subtree of $T$ induced by the nodes of $T$ containing $z$.  

![Figure 7: Hypergraph](image)
We define the following hypergraph \( \mathcal{H} = (\mathcal{V}, \mathcal{E}) \), which is an adaptation of an example from Adler [2], for our purposes.

\[
B = \{ G_{ij} | i, j \in \{1, 2, 3, 4\} \} \cup \{ F_{ij} | i, j \in \{1, 2, 3, 4\} \} \\
\mathcal{V} = B \cup \{ A, B, C, D, E, F, A_1, B_1, C_1, D_1, E_1, F_1 \} \\
\mathcal{E} = \{ (g, p) | g \in B, p \in V \setminus B \} \cup \{ a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4 \} \cup \{ (A, A_1), (A, B), (B, C), (A, D), (C, D), (D, E), (D, F) (A_1, B_1), (B_1, C_1), (A_1, D_1), (C_1, D_1), (D_1, E_1), (E_1, F_1) \}
\]

where

\[
a_1 = (G_{11}, G_{12}, G_{13}, G_{14}, F_{11}, F_{12}, F_{13}, F_{14}, E_1) \\
a_2 = (G_{21}, G_{22}, G_{23}, G_{24}, F_{21}, F_{22}, F_{23}, F_{24}, E_1) \\
a_3 = (G_{31}, G_{32}, G_{33}, G_{34}, F_{31}, F_{32}, F_{33}, F_{34}, D) \\
a_4 = (G_{41}, G_{42}, G_{43}, G_{44}, F_{41}, F_{42}, F_{43}, F_{44}, D) \\
b_1 = (G_{11}, G_{21}, G_{31}, G_{41}, F_{11}, F_{21}, F_{31}, F_{41}, E) \\
b_2 = (G_{12}, G_{22}, G_{32}, G_{42}, F_{12}, F_{22}, F_{32}, F_{42}, E)
\]

Section 6

We define the following hypergraph \( \mathcal{H} = (\mathcal{V}, \mathcal{E}) \), which is an adaptation of an example from Adler [2], for our purposes.

\[
B = \{ G_{ij} | i, j \in \{1, 2, 3, 4\} \} \cup \{ F_{ij} | i, j \in \{1, 2, 3, 4\} \} \\
\mathcal{V} = B \cup \{ A, B, C, D, E, F, A_1, B_1, C_1, D_1, E_1, F_1 \} \\
\mathcal{E} = \{ (g, p) | g \in B, p \in V \setminus B \} \cup \{ a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4 \} \cup \{ (A, A_1), (A, B), (B, C), (A, D), (C, D), (D, E), (D, F) (A_1, B_1), (B_1, C_1), (A_1, D_1), (C_1, D_1), (D_1, E_1), (E_1, F_1) \}
\]

where

\[
a_1 = (G_{11}, G_{12}, G_{13}, G_{14}, F_{11}, F_{12}, F_{13}, F_{14}, E_1) \\
a_2 = (G_{21}, G_{22}, G_{23}, G_{24}, F_{21}, F_{22}, F_{23}, F_{24}, E_1) \\
a_3 = (G_{31}, G_{32}, G_{33}, G_{34}, F_{31}, F_{32}, F_{33}, F_{34}, D) \\
a_4 = (G_{41}, G_{42}, G_{43}, G_{44}, F_{41}, F_{42}, F_{43}, F_{44}, D) \\
b_1 = (G_{11}, G_{21}, G_{31}, G_{41}, F_{11}, F_{21}, F_{31}, F_{41}, E) \\
b_2 = (G_{12}, G_{22}, G_{32}, G_{42}, F_{12}, F_{22}, F_{32}, F_{42}, E)
\]
The vertices in $B$ 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 on Figure 8.

Figure 8: $HW(H) = 6$, $SCW(H) = 6$, $GHW(H) = 5$, $CHW(H) = 5$

Lemma 6.12 $GHW(\mathcal{H}) = 5$, $CHW(\mathcal{H}) = 5$, $HW(\mathcal{H}) = 6$, $SCW(\mathcal{H}) = 6$.

Proof (sketch) The proof is essentially the same as in Adler [2]. We changed her example only to construct a hypergraph whose component hypertreewidth is strictly smaller than its spread cut width. We show that in our example, none of the generalized hypertree decompositions of $\mathcal{H}$ of width 5 is a spread cut decomposition, therefore its spread cut width is at most 6.

Gottlob et al. [19] gave a characterization of $k$-hypertreewidth hypergraphs in terms of a “Robber and Marshal” games played on hypergraphs. They show that $k$ monotone marshals have a winning strategy on $H$ iff the hypertreewidth of $H$ is at most $k$. They relate the game trees of the monotonic robbers and marshals games to hypertree decompositions. The systematic construction of hypergraph examples in [2] made it possible to identify, exactly at which vertices a non-monotonic step can occur. No decomposition corresponding to a non-monotonic game tree with 5 marshals is a spread cut decompositions, because it violates condition 2 of definition 6.5.

We use the notation $(M, Comp)$ for a R&M game position, where $M$ denotes the set of hyper-edges occupied by the marshals, and $C$ is the escape space for the robber.

Claim 6.13 Let $(M, Comp)$ be a game position such that $|M| \leq 5$. Then, a) there exists a ground vertex in $V \setminus \text{vertices}(M)$ and b) if $B \not\subseteq \text{vertices}(M)$, where $B$ is the set of balloon vertices, then $V \setminus \text{vertices}(M)$ is connected.
Proof: \( a \) There are 12 ground vertices and 5 marshals can occupy only at most 10 at the same time. \( b \) Analogous to Claim 3.1, \( b \) in [2].

Claim 6.14 Let \((M, \text{Comp})\) be a game position, such that \(|M| \leq 5\). If \( B \subseteq \text{vertices}(M) \), then \( \\{a_1, a_2, a_3, a_4\} \subseteq M \) or \( \{b_1, b_2, b_3, b_4\} \subseteq M \).

Proof Suppose that \( \{a_1, a_2, a_3, a_4\} \not\subseteq M \) or \( \{b_1, b_2, b_3, b_4\} \not\subseteq M \). Then \( M \) contains at most 4 edges from \( \{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4\} \). They cover at most \( 3 \times 8 + 2 = 26 \) vertices from \( B \). Each of the remaining edges covers at most one vertex from \( B \). Contradiction.

Claim 6.15 There is no winning strategy for < 5 marshals on \( \mathcal{H} \).

Proof Analogous to Claim 3.4 in [2].

Claim 6.16 There is no monotone winning strategy for \( \leq 6 \) marshals on \( \mathcal{H} \). There exists a non-monotone winning strategy for 5 marshals. Furthermore, for all winning strategies with 5 marshals on \( \mathcal{H} \), the escape space of the robber is extended by one of the vertices \( \{E, F, E_1, F_1\} \).

Proof Let us argue indirectly. Because of Claim 6.14, in a game position \((M, \text{Comp}), \{a_1, a_2, a_3, a_4\} \subseteq M \) or \( \{b_1, b_2, b_3, b_4\} \subseteq M \). The balloon vertices must be covered in each step of the game, see Claim 3.2 in [2]. Let us assume without loss of generality, that \( \{a_1, a_2, a_3, a_4\} \subseteq M \) holds. Then, the robber may move into the circle \((A_1, B_1, C_1, D_1, E_1)\). This is possible, because at least one vertex of the circle is not occupied by the marshals, but then the 5th marshal alone cannot capture the robber in the circle. Now, the only possible choice for the marshals if they want to capture the robber, to move to \( \{b_1, b_2, b_3, b_4\} \), but in this case the escape space of the robber is extended either by \( E \) or by \( F_1 \). Contradiction.

The \( \lambda \) sets of a generalized hypertree decomposition of \( \mathcal{H} \) of width 5 are:

\[
\{a_1, a_2, a_3, a_4, \{A, A_1\}\}, \\
\{a_1, a_2, a_3, a_4, \{A, B\}\}, \\
\{a_1, a_2, a_3, a_4, \{B, C\}\}, \\
\{a_1, a_2, a_3, a_4, \{C, D\}\}, \\
\{a_1, a_2, a_3, a_4, \{D, F\}\}, \\
\{b_1, b_2, b_3, b_4, \{A_1, B_1\}\}, \\
\{b_1, b_2, b_3, b_4, \{B_1, C_1\}\}, \\
\{b_1, b_2, b_3, b_4, \{C_1, D_1\}\}, \\
\{b_1, b_2, b_3, b_4, \{D_1, F_1\}\}. \\
\]

This decomposition of \( \mathcal{H} \) of width 5 is at the same time also a component hypertree decomposition, as one can construct a hypertree decomposition of \( \mathcal{H} \cup f^C(\mathcal{H}, 5) \) using the following subedges: \((g_{11}, g_{12}, g_{13}, g_{14})\), \((g_{21}, g_{22}, g_{23}, g_{24})\), \((g_{31}, g_{32}, g_{33}, g_{34})\), \((g_{41}, g_{42}, g_{43}, g_{44})\).
The $\lambda$ sets of a generalized hypertree decomposition of $\mathcal{H}$ of width 6 are:

$\{a_1, a_2, a_3, a_4, (A, A_1)\}$,
$\{a_1, a_2, a_3, a_4, (A, B)\}$,
$\{a_1, a_2, a_3, a_4, (B, C)\}$,
$\{a_1, a_2, a_3, a_4, (C, D)\}$,
$\{a_1, a_2, a_3, a_4, (D, F)\}$,
$\{a_1, a_2, a_3, a_4, (A_1, E_1), (C_1, D_1)\}$,
$\{a_1, a_2, a_3, a_4, (A_1, B_1), (B_1, C_1)\}$.

This decomposition is at the same time also a spread cut decomposition. This completes the proof of lemma 6.12. □