A Greedy Algorithm for Constructing a Low-Width Generalized Hypertree Decomposition

Kaoru Katayama
Tokyo Metropolitan University
6-6 Asahigaoka, Hino, Tokyo, Japan 191-0065
kaoru@tmu.ac.jp

Tatsuro Okawara
Tokyo Metropolitan University
6-6 Asahigaoka, Hino, Tokyo, Japan 191-0065
okawara-tatsuro@sd.tmu.ac.jp

Yuka Ito*
Rakuten,Inc.
4-12-3 Higashishinagawa, Shinagawa, Tokyo, Japan 140-0002
yuka.a.ito@mail.rakuten.co.jp

ABSTRACT

We propose a greedy algorithm which, given a hypergraph H and a positive integer k, produces a hypertree decomposition of width less than or equal to $3k-1$, or determines that H does not have a generalized hypertree-width less than k. The running time of this algorithm is $O(m^{k+2}n)$, where m is the number of hyperedges and n is the number of vertices. If k is a constant, it is polynomial. The concepts of (generalized) hypertree decomposition and (generalized) hypertree-width were introduced by Gottlob et al. Many important NP-complete problems in database theory or artificial intelligence are polynomially solvable for classes of instances associated with hypergraphs of bounded hypertree-width. Gottlob et al. also developed a polynomial time algorithm `det-k-decomp` which, given a hypergraph H and a constant k, computes a hypertree decomposition of width less than or equal to k if the hypertree-width of H is less than or equal to k. The running time of `det-k-decomp` is $O(m^{2k}n^2)$ in the worst case, where m and n are the number of hyperedges and the number of vertices, respectively. The proposed algorithm is faster than this. The key step of our algorithm is checking whether a set of hyperedges is an obstacle to a hypergraph having low generalized hypertree-width. We call such a local hypergraph structure a k-hyperconnected set. If a hypergraph contains a k-hyperconnected set with a size of at least $2k$, it has hypertree-width of at least k. Adler et al. propose another obstacle called a k-hyperlinked set. We discuss the difference between the two concepts with examples.

1. INTRODUCTION

The concepts of hypertree decomposition and hypertree-width were introduced by Gottlob et al. [5] Many important NP-complete problems in database theory and artificial intelligence such as the conjunctive query containment problem are polynomially solvable for classes of instances associated with hypergraphs of bounded hypertree-width [5]. Gottlob et al. [7] also introduced the concept of generalized hypertree decomposition and generalized hypertree-width.

*A part of this work was completed while the author was a student at Tokyo Metropolitan University.

ICDT 2010, March 22–25, 2010, Lausanne, Switzerland.

We propose a greedy algorithm which, given a hypergraph H and a positive integer k, produces a hypertree decomposition of width less than or equal to $3k-1$, or determines that H does not have generalized hypertree-width less than k. Since a hypertree decomposition is also a generalized hypertree decomposition by definition, our algorithm produces a generalized hypertree decomposition. The running time of our algorithm is $O(m^{k+2}n)$, where m is the number of hyperedges and n is the number of vertices. If k is a constant, the running time of our algorithm is polynomial. Gottlob et al. [9] also develped a polynomial time algorithm called `det-k-decomp` which, given a hypergraph H and a positive integer k as a constant, computes a hypertree decomposition of width less than or equal to k if the hypertree-width of H is less than or equal to k. If the hypertree-width of H is more than k, H is rejected. The running time of `det-k-decomp` is $O(m^{2k}n^2)$ in the worst case and our algorithm is faster than `det-k-decomp`.

The key step of our algorithm is checking whether a set of hyperedges is an obstacle to a hypergraph with low generalized hypertree-width. We call such a local hypergraph structure a k-hyperconnected set, where k is a positive integer. We show that, if a hypergraph contains a k-hyperconnected set of size $2k$, the generalized hypertree-width of the hypergraph is at least k. If a given set of hyperedges is not a k-hyperconnected set, our algorithm finds a set of hyperedges called a *separator*, which separates two different subsets of the given set of hyperedges. This follows the approach used by Kleinberg and Tardos [12] for designing an algorithm for constructing a low-width tree decomposition of a graph. The tree decomposition algorithm runs in $O(f(k)mn)$ time, where $f(k)$ is a function that depends only on a positive integer k, and m, n are the number of edges and vertices of a graph, respectively. In both algorithms, the running time is dominated by the time required to check whether a (hyper)graph contains an obstacle to a (hyper)graph having low (hyper)tree-width. In the tree decomposition algorithm, this can be done efficiently using an algorithm for network flow in $O(f(k)m)$ time. On the contrary, in our hypertree decomposition algorithm, it requires more time, $O(m^{k+1}n)$, because every possibility is checked.

Adler et al. [1] proposed another obstacle, a k-hyperlinked set, to a hypergraph with low generalized hypertree-width. A similar greedy algorithm to ours can be constructed with the concept of a k-hyperlinked set. We show the difference between a k-hyperconnected set and a k-hyperlinked set with examples. Although several algorithms for constructing a hypertree decomposition have already been proposed, as we mention in the next section, to our knowledge there is no other algorithm with the same approach to hypertree decomposition, which is trying to find an obstacle to a hypergraph having low generalized hypertree-width.

This paper is organized as follows: In Section 2, we discuss related work. In Section 3, we give definitions of hypergraphs and hypertree decompositions. In Section 4, we introduce the concept of a k-hyperconnected set as an obstacle to a low-width (generalized) hypertree decomposition and show the relation between the size of a k-hyperconnected set and the hypertree-width. We describe the algorithm `check_k-hyperconnected` which, given a hypergraph, a set of hyperedges and a positive integer k, checks whether the given set of hyperedges is a k-hyperconnected set. We also explain the difference between a k-hyperconnected set and a k-hyperlinked set with examples. Then, in Section 5, we introduce the algorithm `low-width-ghd` which, given a hypergraph and a positive integer k, constructs a (generalized) hypertree decomposition or reports that the hypergraph does not have the hypertree-width less than k. We also evaluate the running time of `low-width-ghd`. Finally, we conclude the paper in Section 6.

2. RELATED WORK

Gottlob et al. [5] proposed the alternating algorithm `k-decomp`, which, given a hypergraph H a positive integer k, constructs a hypertree decomposition of minimal width less than or equal to k, if the hypertree-width of H is less than or equal to k. If the hypertree-width of H is more than k, `k-decomp` rejects H. They also presented the algorithm `opt-k-decomp` [6], which is another algorithm for computing a hypertree decomposition of minimal width less than or equal to k, given a hypergraph and a positive integer k. The running time of `opt-k-decomp` is $O(m^{2k}n^2)$, where m is the number of hyperedges and n is the number of vertices. If k is a constant, it is polynomial. Gottlob et al. [9] developed the algorithm `det-k-decomp` which, given a hypergraph H and a positive integer k as a constant, computes a hypertree decomposition of width less than or equal to k if the hypertree-width of H is less than or equal to k. If the hypertree-width of H is more than k, H is rejected. The running time of `det-k-decomp` is $O(m^{2k}n^2)$ in the worst case, where m and n are the number of hyperedges and vertices in the hypergraph, respectively. Gottlob et al. [8] showed that deciding whether a hypergraph has generalized hypertree-width at most 3 is NP-complete.

Scarcello et al. [14] proposed modified versions of `opt-k-decomp` for computing a hypertree decomposition with cost functions. Dermaku et al. [2] used heuristics for generating tree decompositions and partitioning hypergraphs to produce hypertree decompositions. Harvey et al. [11] introduced the reduced normal form of a hypertree decomposition and improved `opt-k-decomp`.

Adler et al. [1] explored the relationship between hypertree width and various hypergraph invariants. Many structural decomposition methods of a hypergraph are proposed besides generalized hypertree decomposition. Grohe et al. [10] introduced the concept of *fractional hypertree decomposition* which is a generalization of generalized hypertree decomposition. Gottlob et al. [4] and Miklós [13] compared them.

3. PRELIMINARIES

We describe definitions of hypergraphs and (generalized) hypertree decompositions and introduce two properties of a (generalized) hypertree decomposition.

3.1 Hypergraph

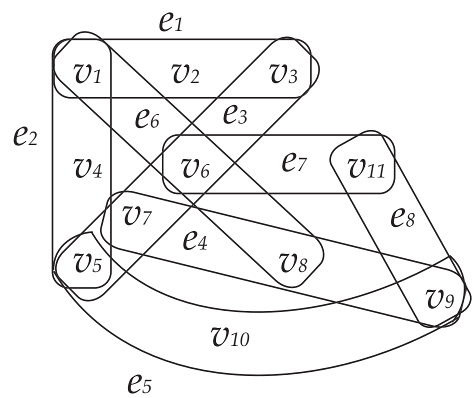

Figure 1: Connected hypergraph H

A hypergraph is a pair $H=(V(H),E(H))$, where $V(H)$ is a finite set of *vertices* and $E(H)$ is a set of *hyperedges*. A hyperedge is a subset of $V(H)$, which is not an empty set. We merely call a hyperedge an *edge*. For a set of edges $E \subseteq E(H)$, $ver(E)$ stands for $\bigcup_{e\in E} e$. We assume $ver(E(H)) = V(H)$.

Let a and b be two vertices in $V(H)$. a is *adjacent* to b if an edge $e \in E(H)$ exists such that $\{a,b\} \subseteq e$. A *path(a,b)* is a sequence $v_0(=a), v_1, v_2, \ldots, v_h(=b)$ of vertices such that v_i is adjacent to v_{i+1} $(0 \le i \le h-1)$. A hypergraph H is *connected* if, for any pair of two vertices $a,b \in V(H)$, a path(a,b) exists. We deal with only connected hypergraphs in this paper. Let W be a subset of $V(H)$. a is *[W]-adjacent* to b if an edge $e \in E(H)$ exists such that $\{a,b\} \subseteq e \setminus W$. A *[W]-path(a,b)* is a sequence $v_0(= a), v_1, v_2, \ldots, v_h(=b)$ of vertices such that v_i is $[W]$-adjacent to v_{i+1} $(0 \le i \le h-1)$. A set of vertices $C \subseteq V(H)$ is *[W]-connected* if, for any pair of two vertices $a,b \in C$, there is a $[W]$-path(a, b). A *[W]-component* is a maximal $[W]$-connected non-empty set of vertices. Let F be a subset of $E(H)$. A *[F]-fragment* is a maximal set of edges that share the vertices with a $[ver(F)]$-component, that is, $\{e \in E(H) | e \cap [ver(F)]\text{-component} \neq \emptyset\}$. For a set of vertices C, let a set of edges $cov(C)$ be $\{e \in E(H) | e \cap C \neq \emptyset\}$, and a family of subsets of $cov(C)$, $cov^*(C)$ be $\{F \subseteq cov(C) | \forall e \in F : e \not\subseteq ver(cov(C) \setminus e)\}$.

EXAMPLE 1. *Consider connected hypergraph H in Figure 1. The set of vertices $V(H)$ is $\{v_1, v_2, \ldots, v_{11}\}$ and the set of edges $E(H)$ is $\{e_1, e_2, \ldots, e_8\}$ where $e_3 = \{v_3, v_5, v_6, v_7\}$ and $e_6 = \{v_1, v_6, v_8\}$. For a set of vertices $W = \{v_3, v_5, v_6, v_7, v_8\}$, the $[W]$-components are $\{v_1, v_2, v_4\}$ and $\{v_9, v_{10}, v_{11}\}$. For a set of vertices $C = \{v_9, v_{10}, v_{11}\}$, a set of edges $cov(C)$ is $\{e_4, e_5, e_7, e_8\}$ and a family of subsets of $cov(C)$, $cov^*(C)$ is $\{\{e_5, e_7\}, \{e_5, e_8\}\}$. For a set of edges $F = \{e_3, e_6\}$, the $[F]$-fragments are $\{e_1\}$, $\{e_2\}$ and $\{e_4, e_5, e_7, e_8\}$.*

3.2 Hypertree Decomposition

A *hypertree decomposition* of a hypergraph H is a triple $\langle T, \chi, \lambda \rangle$. $T = (V(T), E(T))$ is a rooted tree, where $V(T)$ is a finite set of *nodes*, and $E(T)$ is a set of edges of T. $\chi : V(T) \to 2^{V(H)}$ and $\lambda : V(T) \to 2^{E(H)}$ are functions associating a set of vertices $\chi(t) \subseteq V(H)$ and edges $\lambda(t) \subseteq E(H)$ to each node t respectively. We call $v \in V(H)$ a vertex and $t \in V(T)$ a node. For any $t \in V(T)$, T^t denotes the maximal subtree of T rooted at t. For a subtree T' of T, we use $\chi(T')$ and $\lambda(T')$ to denote $\bigcup_{n\in V(T')} \chi(n)$ and $\bigcup_{n\in V(T')} \lambda(n)$, respectively.

DEFINITION 1. ***(Hypertree Decomposition)*** *[5] A hypertree decomposition of a hypergraph H is a triple $\langle T, \chi, \lambda \rangle$, which satisfies all the following conditions:*

1. *for each edge $e \in E(H)$, $t \in V(T)$ exists such that $e \subseteq \chi(t)$;*
2. *for each vertex $v \in V(H)$, the set $\{t \in V(T) | v \in \chi(t)\}$ induces a connected subtree of T;*
3. *for each $t \in V(T)$, $\chi(t) \subseteq ver(\lambda(t))$;*
4. *for each $t \in V(T)$, $ver(\lambda(t)) \cap \chi(T^t) \subseteq \chi(t)$.*

The *width* of a hypertree decomposition $\langle T, \chi, \lambda \rangle$ is the largest size of $\lambda(t)$ over every node t of T. The *hypertree-width* of a hypergraph H is the minimum width over all hypertree decompositions of H. The hypertree-width of an acyclic hypergraph is 1.

A *generalized hypertree decomposition* of a hypergraph H is a triple $\langle T, \chi, \lambda \rangle$, which satisfies conditions 1,2, and 3 of Definition 1. The *width* of a generalized hypertree decomposition $\langle T, \chi, \lambda \rangle$ is the largest size of $\lambda(t)$ over every node t of T. The *generalized hypertree-width* of a hypergraph H is the minimum width over all generalized hypertree decompositions of H. The generalized hypertree-width of a hypergraph is less than or equal to the hypertree-width. [1].

DEFINITION 2. ***(Normal Form)*** *[8] A generalized hypertree decomposition $\langle T, \chi, \lambda \rangle$ of a hypergraph H is in normal form, if, for each vertex $t \in V(T)$ and each child s of t, all the following conditions hold:*

1. *there is exactly one $[\chi(t)]$-component C_t such that $\chi(T^s) = C_t \cup (\chi(s) \cap \chi(t))$;*
2. *$\chi(s) \cap C_t \neq \emptyset$, where C_t is the $[\chi(t)]$-component satisfying condition 1;*
3. *$ver(\lambda(s)) \cap \chi(t) \subseteq \chi(s)$.*

The hypertree decomposition constructed with our algorithm is in normal form, as shown later in Proposition 7.

EXAMPLE 2. *Figure 2 shows a normal form (generalized) hypertree decomposition of hypergraph H in Figure 1. The width of this (generalized) hypertree decomposition is 2.*

A hypergraph is separated by deleting vertices assigned to a node or common vertices assigned to two connected nodes in the (generalized) hypertree decomposition.

PROPOSITION 1. *Suppose that there are subtrees $T_1, T_2, \ldots, T_d$ when a node p is deleted from tree T of a (generalized) hypertree decomposition $\langle T, \chi, \lambda \rangle$ of a hypergraph H. Then for any pair $i, j \in \{1, 2, \ldots, d\} (i \neq j)$, $(\chi(T_i) \setminus \chi(p)) \cap (\chi(T_j) \setminus \chi(p)) = \emptyset$ and $\{e \in E(H) | \{u, v\} \subseteq e, u \in \chi(T_i) \setminus \chi(p), v \in \chi(T_j) \setminus \chi(p)\} = \emptyset$ (Figure 3).*

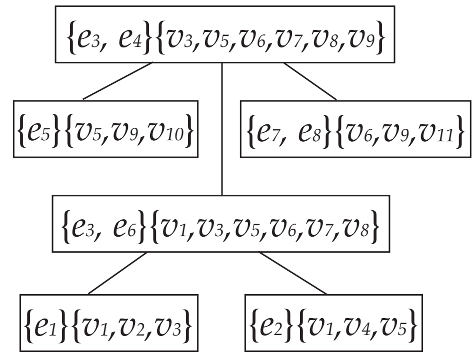

Figure 2: Normal form hypertree decomposition of *H*

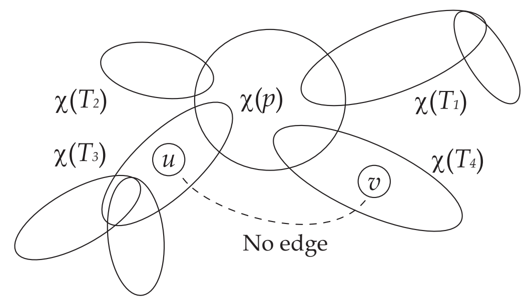

Figure 3: Subtrees $T_1, T_2, \ldots, T_d$ by deleting node p from a (generalized) hypertree decomposition. There is no edge which contains vertices u and v when $\chi(p)$ is deleted from hypergraph.

PROOF. Omitted. □

PROPOSITION 2. *Suppose that there are subtrees T_p and T_t when an edge $(p, t) \in E(T) (p, t \in V(T))$ is deleted from tree T of a (generalized) hypertree decomposition $\langle T, \chi, \lambda \rangle$ of a hypergraph H. Then by deleting $\chi(p) \cap \chi(t)$ from H, H is disconnected into two components, $\chi(T_p) \setminus (\chi(p) \cap \chi(t))$ and $\chi(T_t) \setminus (\chi(p) \cap \chi(t))$. That is, $(\chi(T_p) \setminus (\chi(p) \cap \chi(t))) \cap (\chi(T_t) \setminus (\chi(p) \cap \chi(t))) = \emptyset$ and $\{e \in E(H) | \{u, v\} \subseteq e, u \in \chi(T_p) \setminus (\chi(p) \cap \chi(t)), v \in \chi(T_t) \setminus (\chi(p) \cap \chi(t))\} = \emptyset$ (Figure 4).*

PROOF. Omitted. □

4. OBSTACLES TO LOW GENERALIZED HYPERTREE-WIDTH

The key step in designing our algorithm is trying to find an obstacle to a hypergraph having low generalized hypertree-width. We call such an obstacle a *k-hyperconnected* set, which is a set of edges of the hypergraph. The notion of a *k-hyperconnected* set is an adaptation of *k-connectedness* for a graph to our setting [3]. We show the relation between the size of a *k*-hyperconnected set in a hypergraph and the hypertree-width of the hypergraph. We propose the algorithm `check_k-hyperconnected` to decide whether a subset of edges of a hypergraph is a *k*-hyperconnected set, given a hypergraph and a positive integer *k*. The running time of *k*-hyperconnected set is $O(m^{k+1}n)$. If *k* is a constant, it is polynomial.

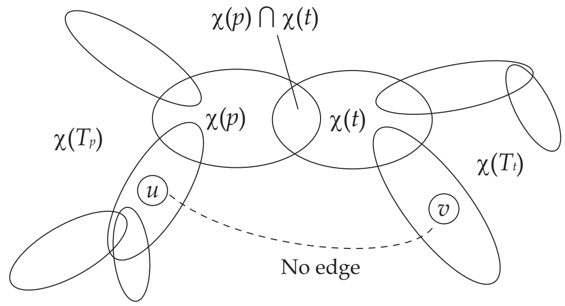

Figure 4: Subtrees T_p and T_t by deleting the edge between node p and node t from a (generalized) hypertree decomposition. There is no edge which contains vertices u and v when $\chi(p) \cap \chi(t)$ is deleted from hypergraph.

4.1 k-hyperconnected set

We give the definition of a k-hyperconnected set and prove a proposition for its algorithmic use.

DEFINITION 3. ***(separator)*** *Let Y and Z be a pair of subsets of $E(H)$ of a hypergraph H such that $|Y| = |Z|$ and $Y \neq Z$. A subset of $E(H)$, S is a* separator *for a pair of Y and Z if it satisfies all the following conditions:*

1. *$|S| < |Y| = |Z|$;*
2. *there is no $[ver(S)]$-path from $ver(Y)$ to $ver(Z)$.*

We say that S separates Y and Z, or that Y and Z are separable *with S.*

DEFINITION 4. ***(k-hyperconnected set)*** *Let X be a subset of $E(H)$ of a hypergraph H and k be a positive integer. Let Y and Z be an arbitrary pair of two subsets of X such that $|Y| = |Z|$. X is a k-hyperconnected set, if it satisfies all the following conditions:*

1. *$|X| \geq k$;*
2. *X does not contain separable subsets Y and Z, where $|Y| = |Z| \leq k$. In other words, there is no separator $S \subseteq E(H)$, which separates Y and Z such that $|S| < |Y| = |Z| \leq k$.*

We call an edge, which is included in X, an X-edge.

Intuitively, a k-hyperconnected set is highly self-entwined. It does not have any small parts that can easily split off from each other. A k-hyperconnected set cannot be separated by deleting less than k edges.

PROPOSITION 3. *If a hypergraph H contains a k-hyperconnected set with a size of at least $2k$, H has the generalized hypertree width of at least k.*

PROOF. Suppose that a hypergraph H contains a k-hyperconnected set X with a size of at least $2k$, and it has a generalized hypertree decomposition $\langle T, \chi, \lambda \rangle$ of a width less than k. There is a node t of T that satisfies the following conditions:

1. Let X^t be a subset of X-edges $\{x \in X | x \subseteq \chi(T^t)\}$. $|X^t|$ is more than or equal to $\lceil \frac{|X|}{2} \rceil$;
2. t is as far from the root of T as possible.

Clearly, $\chi(t)$ contains all vertices of at least one X-edge, and node t is not a leaf of T because the set of edges X with a size of at least $2k$ cannot be contained in a node of the generalized hypertree decomposition of a width less than k. Now we divide X into three distinct subsets, $X_p = X \setminus X^t$, $X_t = \{x \in X | x \subseteq \chi(t)\}$, and $X_c = X^t \setminus X_t$. There is no $[\chi(t)]$-path between any pair of vertices in X_p and X_c from Proposition 1. The size of X_p and X_c is less than or equal to k. Two subsets, Y and Z, of $E(H)$, where $|X_t| < |Y| = |Z| \leq k$, can be made from X_p and X_c by adding edges in X_t. Then X_t separates Y and Z. This means that X is not a k-hyperconnected set and contradicts the assumption. ☐

4.2 Comparing with k-hyperlinked set

Adler et al. [1] define the concept of a *k-hyperlinked* set for a set of edges of a hypergraph. *Hyperlinkedness* of a hypergraph is the largest integer k for which the hypergraph contains a k-hyperlinked set. It is an adaptation of the *linkedness* of a graph. A k-hyperlinked set also an obstacle to a hypergraph having low generalized hypertree-width. We show that the size of a k-hyperlinked set is also associated with the generalized hypertree-width of the hypergraph, and compare the two notions using examples. Adler et al. [1] prove that the hyperlinkedness of a hypergraph is less than or equal to the generalized hypertree-width of the hypergraph.

DEFINITION 5. ***(X-big)*** *[1] Let H be a hypergraph and X be a subset of $E(H)$. A subset of vertices $V(H)$, C is X-big, if it satisfies the following condition:*

$$|\{e \in X | e \cap C \neq \emptyset\}| > \frac{|X|}{2}.$$

An X-big component is a maximal set of X-big vertices in which each vertex is adjacent to another one.

DEFINITION 6. ***(k-hyperlinked set)*** *[1] Let H be a hypergraph and k be a positive integer. A subset of $E(H)$, X is* a k-hyperlinked set, *if the hypergraph $(V(H) \setminus ver(S), \{e \cap (V(H) \setminus ver(S)) | e \in E(H)\})$ has an X-big component for any set $S \subseteq E(H)$ where $|S| < k$. We call an edge, which is included in X, an X-edge as in Definition 4.*

PROPOSITION 4. *If a hypergraph H contains a k-hyperlinked set with a size of at least $2k$, H has a generalized hypertree width of at least k.*

PROOF. This proposition can be proven by the same idea of Proposition 3. ☐

We show the difference between a k-hyperlinked set and a k-hyperconnected set with the following examples.

EXAMPLE 3. *A hypergraph H and a subset $X = X_1 \cup X_2$ of $E(H)$ are defined as in Figure 5. In this case X is a 1-hyperconnected*

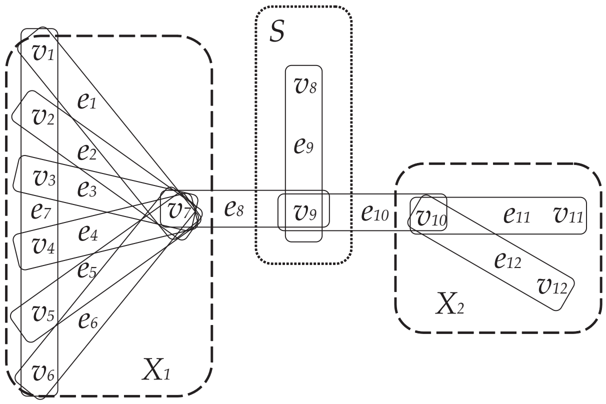

Figure 5: Hypergraph for example 3. $X = X_1 \cup X_2$ **is a 1-hyperconnected set and a 2-hyperlinked set.**

set and a 2-hyperlinked set. Let two sets of edges, Y and Z, such that $|Y| = |Z| = 2$ be subsets of X_1 and X_2, respectively. Since there is no [ver(S)]-paths(y,z) for any pair of vertices, $y \in Y$ and $z \in Z$, X is a 1-hyperconnected set. On the other hand, in a hypergraph $(V(H) \setminus e, \{e \cap (V(H) \setminus e) | e \in E(H)\})$ constructed by deleting any edge $e \in E(H)$ from H, the number of remaining edges in X is larger than $|X|/2 = 9/2$. But, in a hypergraph $(V(H) \setminus (e_1 \cup e_7), \{e \cap (V(H) \setminus (e_1 \cup e_7)) | e \in E(H)\})$ constructed by deleting two edges, e_1 and e_7, from H, the number of remaining edges in X is 2 and less than $|X|/2 = 9/2$. This means that X is a 2-hyperlinked set.

EXAMPLE 4. *A hypergraph H and a subset X of $E(H)$ are defined as in Figure 6. In this case X is a 3-hyperconnected set and a 2-hyperlinked set. Any two subsets Y and Z each of size 3 of X cannot be separated by deleting any set of edges of size 2. This means that X is a 3-hyperconnected set at least. Since there are no two different subsets each of size 4 of X, we cannot choose separable subsets for a separator of size 3. Therefore X is not a 4-hyperconnected set. On the other hand, in a hypergraph $(V(H) \setminus e, \{e \cap (V(H) \setminus e) | e \in E(H)\})$ constructed by deleting any edge $e \in E(H)$ from H, the number of remaining edges in X is larger than $|X|/2 = 2$. But, in a hypergraph $(V(H) \setminus (e_2 \cup e_3), \{e \cap (V(H) \setminus (e_2 \cup e_3)) | e \in E(H)\})$ constructed by deleting two edges, e_2 and e_3 from H, the number of remaining edges in X is $|X|/2 = 2$. This means that X is a 2-hyperlinked set.*

4.3 Finding Separator

We describe the algorithm `check_k-hyperconnected` which, given a hypergraph H, a subset X of edges of H and a positive integer k, determines whether X is a k-hyperconnected set. If X is not a k-hyperconnected set, `check_k-hyperconnected` returns a set of edges as a separator for a pair of two separable subsets in X. We can develop a similar algorithm using the notion of a k-hyperlinked set.

A simple way to do this is to check whether there is a separator for every pair of subsets of each size less than or equal to k of X. However, it is not easy to find such a separator. Therefore, we check whether a pair of separable subsets, Y and Z, of X exists for every subset with a size of less than k of $E(H)$ conversely. If the size of X is more than or equal to $2k-1$, it is necessary to check it

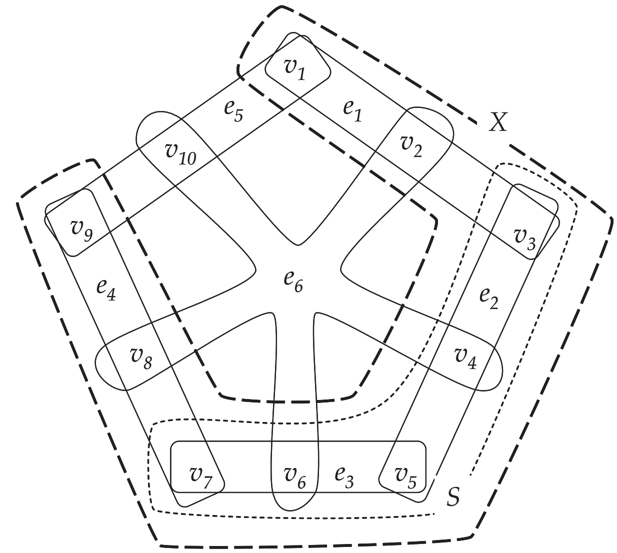

Figure 6: Hypergraph for example 4. X **is a 3-hyperconnected set and a 2-hyperlinked set.**

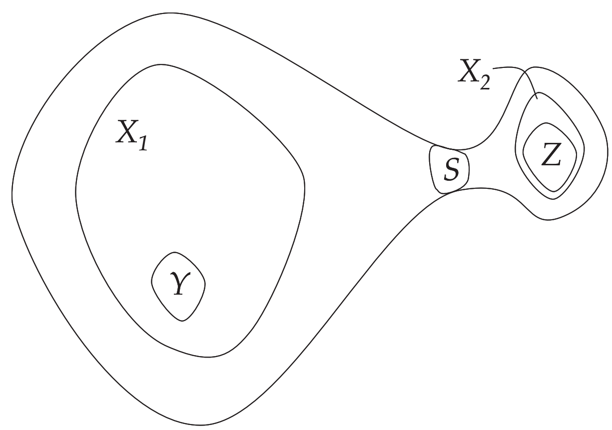

Figure 7: X**-edges in** $X \setminus (Y \cup Z \cup S)$ **are added to** S **to make the size of** S **be** $k-1$**. Edges in** $S \cap X$ **are added to** Y **and** Z **to make their size be** k**.**

only for every subset of size $k-1$ of $E(H)$ because if a separator with a size of less than $k-1$ is found, we can make it be $k-1$ by adding edges in $X \setminus (Y \cup Z \cup S)$ (Figure 7). That is, every separator is contained in subsets of size $k-1$ of $E(H)$. In the case where the size of a separator S is $k-1$, each size of separable subsets Y and Z must be more than $k-1$ to satisfy the first condition in Definition 3. When candidate sets Y and Z for separable subsets are found for a subset S of size $k-1$ of $E(H)$, but each size of Y and Z is less than or equal to $k-1$, we may increase the size by adding the same edges in $S \cap X$ to Y and Z (Figure 7). If we can make the size be k, the set of edges, Y, Z, and S become separable subsets and the separator.

`check_k-hyperconnected` repeats the following steps for every subset S of size $k-1$ of $E(H)$, as shown in Algorithm 1, unless it finds a separator of size $k-1$ or that a given X is a k-hyperconnected set, that is, X does not contain separable subsets Y and Z of each size less than or equal to k. We do not check whether the size of X is more than or equal to $2k-1$ since the size of X is always more than $2k$ in the algorithm using `check_k-hyperconnected`, which constructs a low-width (generalized) hypertree decomposition. In Algorithm 1, we use variables Y, Z, and S to denote candidates of two separable subsets of X and a separator for the subsets, respectively.

1. Choose a subset S of size $k-1$ from $E(H)$.
2. Let L be $\{e \in X | e \subseteq ver(S)\}$. If $|L| \geq k$, return the subset S as a separator and stop.

   Any edge e in L can belong to both Y and Z because $e \setminus ver(S)$ is an empty set, and there is no $[ver(S)]$-path from e to other vertices. Therefore, if $|L| \geq k$, we can construct Y, Z, and S, which satisfy the conditions of Definition 3, by choosing k edges from L as Y, k edges from X as Z, and the subset S as a separator.
3. Divide the set of the $[S]$-fragments into two subsets, Y and Z.
4. If there are more than or equal to k X-edges in each of $Y \cup L$ and $Z \cup L$, return the subset S as a separator and stop.

   If each $Y \cup L$ and $Z \cup L$ includes more than or equal to k X-edges, we can make separable subsets Y' and Z', which satisfy the conditions of Definition 3, by choosing k edges from L as Y', and k edges from X as Z'. In this case, the subset S separates Y' and Z'.

PROPOSITION 5. *The running time of* `check_k-hyperconnected` *is* $O(\binom{m}{k-1} m^2 n)$.

PROOF. Let k be a positive integer as a constant, and m, n be the number of edges $|E(H)|$ of a hypergraph H and the number of vertices $|V(H)|$, respectively. The number of subsets of $E(H)$, where each of their sizes is $k-1$, is $\binom{m}{k-1}$. For each subset S of size $k-1$ of $E(H)$, we enumerate the number of edges $\{e \in X | e \subseteq ver(S)\}$. This takes $O(mn)$ time. Finding the set of $[S]$-fragments and dividing it into two subsets take $O(m^2 n)$ time. Thus, the whole running time of `check_k-hyperconnected` is $O(\binom{m}{k-1} m^2 n)$. Since $O(\binom{m}{k-1})$ is $O(m^{k-1})$, a less accurate but more readable upper bound of the running time is $O(m^{k+1} n)$. □

Algorithm 1 `check_k-hyperconnected`

Input: a hypergraph $H = (V(H), E(H))$, a subset X of $E(H)$, and a positive integer k

Output: a separator $S \subseteq E(H)$ for a pair of subsets of X, or a message "X is a k-hyperconnected set".

```
for each subset S of size k−1 of E(H) do
    let L be {e ∈ X | e ⊆ ver(S)}
    if |L| ≥ k then
        return S
    end if
    find all [S]-fragments F_1, F_2, ..., F_d in H
    arrange F_1, F_2, ..., F_d in descending order of the number of
        X-edges contained in each [S]-fragment
    Y ← F_1
    Z ← F_2
    for i = 3 to d do
        if |{e ∈ X | e ∈ Y}| ≤ |{e ∈ X | e ∈ Z}| then
            Y ← Y ∪ F_i
        else
            Z ← Z ∪ F_i
        end if
    end for
    if (|L| + |{e ∈ X | e ∈ Y}| ≥ k) and (|L| + |{e ∈ X | e ∈ Z}| ≥ k)
    then
        return S
    end if
end for
return "X is a k-hyperconnected set"
```

5. CONSTRUCTING A LOW-WIDTH HYPERTREE DECOMPOSITION

We propose an algorithm for constructing a (generalized) hypertree decomposition of H of width less than or equal to $3k-1$ or detemines that H does not have a generalized hypertree-width less than k, where k is a positive integer as a constant. The following procedure repeatedly decomposes a hypergraph by deleting a set of edges and constructs a (generalized) hypertree decomposition $\langle T, \chi, \lambda \rangle$. The proposed algorithm is described formally in Algorithm 2 and 3. Figure 8 and 9 show decomposed components of a hypergraph, and Figure 10 shows the constructed hypertree decomposition corresponding to Figure 8 and 9.

1. Arbitrarily select a set of edges less than or equal to $2k-1$ from $E(H)$ and make the root r of T.

   For the root r of T, the selected set of edges is assigned to $\lambda(r)$, and all vertices included in the edges are assigned to $\chi(r)$. In Figure 10, a set of edges E and a set of vertices $\chi(E)$ are assigned to $\lambda(r)$ and $\chi(r)$, respectively.
2. For each $[\chi(r)]$-component C_r, make a child node t of the root r.

   By Proposition 1 and 2, we can deal with each $[\chi(r)]$-component C_r independently. Figure 8 shows that there are two $[\chi(r)]$-components C_{r_1} and C_{r_2}. To decompose a $[\chi(r)]$-component C_r further, we choose an arbitrary edge e_t from $cov(C_r)$. For child node t corresponding to C_r, we add the edge e_t and e_t to $\lambda(t)$ and $\chi(t)$ respectively. To ensure that condition 2 of Definition 1 is satisfied when some vertices in $B_r = ver(cov(C_r)) \cap \chi(r)$ are included in a child node of t in the later process, we add vertices B_r to $\chi(t)$. Figure 8 shows that there are four vertices in B_{r_1}. We also add a set of edges $E_{A_r} \in cov^*(A_r)$ where $A_r = B_r \setminus e_t$, and a set of vertices $ver(E_{A_r})$ to $\lambda(t)$ and $\chi(t)$, respectively, to satisfy condition 3 of Definition 1. Figure 8 shows that there are three vertices in A_{r_1} and two edges in $E_{A_{r_1}}$. In Figure 10, B_{r_1} is not contained in $\chi(t)$ since it is included in $e_t \cup ver(E_{A_{r_1}})$. Since $ver(\lambda(t))$ is equal to $\chi(t)$, condition 4 of Definition 1 is also satisfied.
3. For each $[\chi(r) \cup \chi(t)]$-component C_t formed from a $[\chi(r)]$-component C_r, make a child node s of t in the same way to step 2 above.

   Figure 9 shows that there are three $[\chi(r) \cup \chi(t)]$-component $C_{t_1}, C_{t_2}, C_{t_3}$ formed from an $[\chi(r)]$-component C_{r_1}.

The tree $\langle T, \chi, \lambda \rangle$ constructed from the above procedure satisfies all the conditions of Definition 1 and is a (generalized) hypertree decomposition.

To determine whether the hypertree decomposition of the required size can be constructed, for each child node of r, we check the size

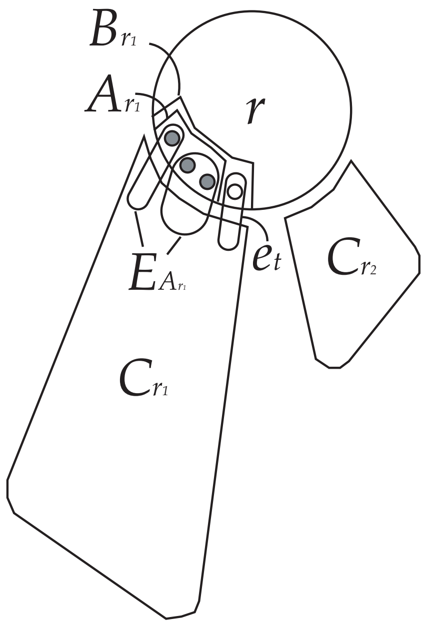

Figure 8: $[\chi(r)]$-components C_{r1} and C_{r2}. B_{r_1} is vertices in $\chi(r)$ that are also contained in $ver(C_{r_1})$. A_{r_1} is vertices in B_{r_1} that is not included in e_t. $E_{A_{r_1}}$ is element of $cov^*(A_{r_1})$.

of $\lambda(t) = \{e_t\} \cup E_{A_r}$ after the above step 2. Here, there is clearly a set of edges $E_{A_r} \in cov^*(A_r)$ less than or equal to $2k-1$ because $A_r \subseteq ver(\lambda(r))$ and $|\lambda(r)| \leq 2k-1$. If the size of $\lambda(t)$ is less than or equal to $2k-1$, node t can be treated the same as root r, and we go through the procedure. If the size of $\lambda(t)$ is $2k$, we check whether $\lambda(t)$ is a k-hyperconnected set with `check-k_hyperconnected` described in Section 4. There are the following two cases.

- $\lambda(t) = \{e_t\} \cup E_{A_r}$ is a k-hyperconnected set

  The hypergraph does not have a generalized hypertree-width less than k by Proposition 3. `hd-decomp` returns the message and halts.

- $\lambda(t) = \{e_t\} \cup E_{A_r}$ is not a k-hyperconnected set

  There is a separator $S \subseteq E(H)$ of size $k-1$ and two separable sets of edges $Y, Z \subseteq \lambda(t)$ of size k each. Figure 11 shows this situation in a $[\chi(r)]$-component. To decompose the $[\chi(r)]$-component, we add $S \cap cov(C_r)$ to $\lambda(t)$ and $ver(S \cap cov(C_r))$ to $\chi(t)$. Since the size of $S \cap cov(C_r)$ is less than the size of S, the size of $\lambda(t) = E_{A_r} \cup \{e_t\} \cup (S \cap cov(C_r))$ is less than or equal to $3k-1$, which is the width we want.

  To continue to the same process further for each $[\chi(r) \cup \chi(t)]$-component C_t, the size of $E_{A_t} \in cov^*(A_t)$ needs to be less than or equal to $2k-1$ as the size of E_{A_r}. Since there is no $[\chi(S)]$-path between $[\chi(r) \cup \chi(t)]$-components, a set of vertices $ver(cov(C_t))$ has common vertices with either $ver(Y \cup S)$ or $ver(Z \cup S)$ (Figure 11). A_t is a subset of $ver(cov(C_t))$. Therefore $cov^*(A_t)$ is included in a subset of either $Y \cup S$ or $Z \cup S$. Since both size of $Y \cup S$ and $Z \cup S$ is less than or equal to $2k-1$, the size of $E_{A_t} \in cov^*(A_t)$ is also less than or equal to $2k-1$.

Algorithm 2 `low-width-ghd`

Input: a hypergraph $H = (V(H), E(H))$, a positive integer k
Output: a hypertree decomposition $\langle T, \chi, \lambda \rangle$ of H, which has a width less than or equal to $3k-1$, or a message "H does not have generalized hypertree-width less than k"

arbitrarily select a set of edges E with the size less than or equal to $2k-1$ from $E(H)$
create root node r of tree T
$\lambda(r) \leftarrow E$
$\chi(r) \leftarrow ver(E)$
for each $[\chi(r)]$-component C_r **do**
  `create_node`(r, $cov(C_r)$, k, $\langle T, \chi, \lambda \rangle$)
end for
return $\langle T, \chi, \lambda \rangle$

Algorithm 3 `create_node`

Input: a hypergraph H, a node r, a set of edges $cov(C_r)$, a positive integer k and a tree $\langle T, \chi, \lambda \rangle$
Output: a tree $\langle T, \chi, \lambda \rangle$ or a message "H does not have generalized hypertree-width less than k"

$B_r \leftarrow ver(cov(C_r)) \cap \chi(r)$
select an edge e_t from $cov(C_r)$
$A_r \leftarrow B_r \setminus e_t$
find a set of edges $E_{A_r} \in cov^*(A_r)$
create a child node t of r in T
$\lambda(t) \leftarrow \{e_t\} \cup E_{A_r}$
$\chi(t) \leftarrow e_t \cup ver(E_{A_r})$
if $|\lambda(t)| = 2k$ **then**
  if `check_k-hyperconnected` $(H, \lambda(t), k)$ = "$\lambda(t)$ is a k-hyperconnected set" **then**
    return "H does not have generalized hypertree-width less than k"
  else
    $S \leftarrow$ `check_k-hyperconnected` $(H, \lambda(t), k)$
    $\lambda(t) \leftarrow \lambda(t) \cup (S \cap cov(C_r))$
    $\chi(t) \leftarrow \chi(t) \cup ver(S \cap cov(C_r))$
  end if
end if
for each $[\chi(r) \cup \chi(t)]$-component C_t **do**
  `create_node`(t, $cov(C_t)$, k, $\langle T, \chi, \lambda \rangle$)
end for
return $\langle T, \chi, \lambda \rangle$

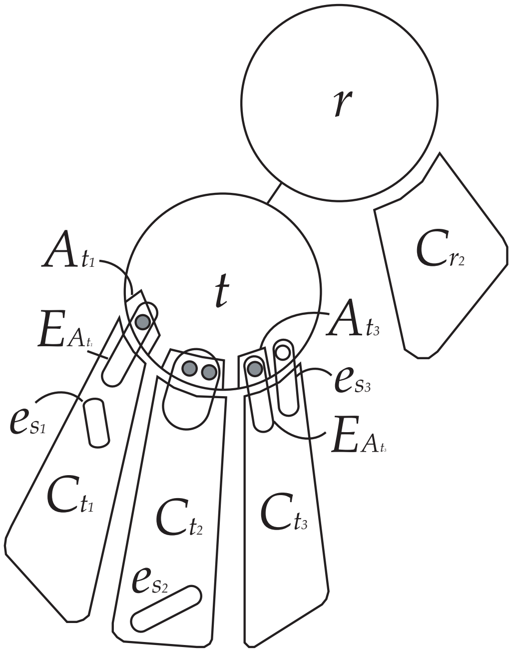

Figure 9: C_{t_1}, C_{t_2}, and C_{t_3} are $[\chi(r) \cup \chi(t)]$-components formed from C_{r_1}.

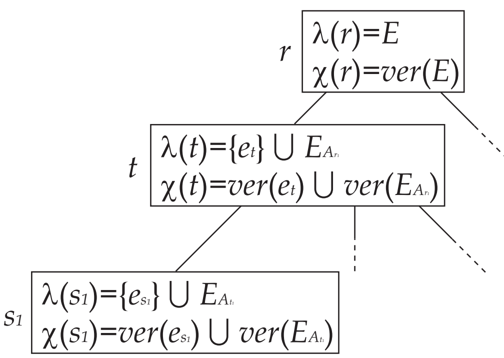

Figure 10: Hypertree decomposition for Figure 8 and 9.

PROPOSITION 6. *The running time of* `low-width-ghd` *is* $O(m^{k+2}n)$.

PROOF. Let k be a positive integer, m be the number of edges in a hypergraph H, and n be the number of the vertices. The most costly operation in `low-width-ghd` is `check_k-hyperconnected` in `create_node`. Since one edge $e \in E(H)$ is selected at most once in `create_node`, `create_node` is called at most m times. From Proposition 6, `check_k-hyperconnected` takes $O(m^{k+1}n)$. Thus, the entire running time of `low-width-ghd` is $O(m^{k+2}n)$. □

PROPOSITION 7. *A hypertree decomposition constructed by* `low-width-ghd` *is in normal form.*

PROOF. `low-width-ghd` creates a child node s of $t \in V(T)$ for each $[\chi(t)]$-component and assigns $e_s \cup ver(E_{A_t})$ to $\chi(s)$ in `create_node`, where e_s is selected arbitrary from $cov(C_t)$ and A_t is $(ver(cov(C_t)) \cap \chi(t)) \setminus e_s$. Thus, conditions 1 and 2 of Definition 2 are clearly satisfied. Since $\chi(s)$ contains all vertices of $\lambda(s)$ which consists of $\{e_s\}$, E_{A_t} and, if $\lambda(t)$ of size $2k$ is not a k-hyperconnected set, $S \cap cov(C_t)$ where S is a separator of $\lambda(t)$, condition 3 of Definition 2 is also satisfied. □

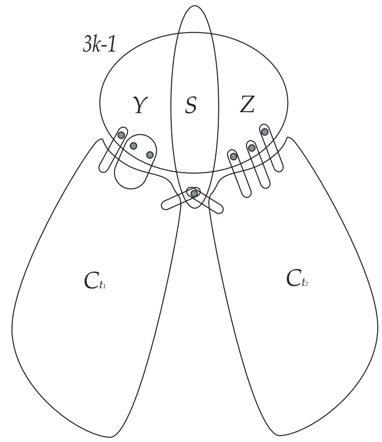

Figure 11: $[\chi(r) \cup \chi(t)]$-component C_{t_1} and C_{t_2} have common vertices with $ver(Y \cup S)$ and $ver(Z \cup S)$, respectively, where S separates Y and Z.

6. CONCLUSIONS

We have presented a greedy algorithm which, given a hypergraph H and a positive integer k as a constant, produces a hypertree decomposition of a width less than or equal to $3k-1$, or reports that H does not have a generalized hypertree-width of less than k. The key step of this algorithm is trying to find a k-hyperconnected set, which is an obstacle to a hypergraph having a low generalized hypertree-width. The entire running time is $O(m^{k+2}n)$ where m is the number of edges and n is the number of vertices in a hypergraph. If k is a constant, it is polynomial. This algorithm is faster than `det-k-decomp` developed by Gottlob et al. in the worst case.

7. ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers for their helpful feedback. This work was supported by Grant-in-Aid for Scientific Research(C)(21500104).

8. REFERENCES

[1] I. Adler, G. Gottlob, and M. Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007.

[2] A. Dermaku, T. Ganzow, G. Gottlob, B. J. McMahan, N. Musliu, and M. Samer. Heuristic methods for hypertree decomposition. In *7th Mexican International Conference on Artificial Intelligence(MICAI)*, pages 1–11, 2008.

[3] R. Diestel. *Graph Theory Second Edition*. Springer, 2000.

[4] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural csp decomposition methods. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 394–399, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[5] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. In *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 21–32, New York, NY, USA, 1999. ACM.

[6] G. Gottlob, N. Leone, and F. Scarcello. On tractable queries and constraints. In *10 th International Conference and Workshop on Database and Expert Systems Applications(DEXA)*, pages 1–15, 1999.

[7] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 195–206, New York, NY, USA, 2001. ACM.

[8] G. Gottlob, Z. Miklós, and T. Schwentick. Generalized hypertree decompositions: Np-hardness and tractable variants. *J. ACM*, 56(6):1–32, 2009.

[9] G. Gottlob and M. Samer. A backtracking-based algorithm for hypertree decomposition. *J. Exp. Algorithmics*, 13:1.1–1.19, 2009.

[10] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 289–298, New York, NY, USA, 2006. ACM.

[11] P. Harvey and A. Ghose. Reducing redundancy in the hypertree decomposition scheme. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 474–481. IEEE Computer Society, 2003.

[12] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, 2006.

[13] Z. Miklós. On the parallel complexity of structural CSP decomposition methods. In H. Broersma, S. Dantchev, M. Johnson, and S. Szeider, editors, *Algorithms and Complexity in Durham 2007, Proceedings of the third ACiD Workshop*, volume 9 of *Texts in Algorithmics*, pages 107–118. College Publications London, 2007.

[14] F. Scarcello, G. Greco, and N. Leone. Weighted hypertree decompositions and optimal query plans. *J. Comput. Syst. Sci.*, 73(3):475–506, 2007.