Hyperlinkedness に基づく Hypertree Decomposition 構築アルゴリズム

大河原達郎† 片山 薫†

† 首都大学東京システムデザイン研究科

E-mail: †okawara-tatsuro@sd.tmu.ac.jp, ††kaoru@comp.metro-u.ac.jp

あらまし 関係データベースに対する基本的な問合せのクラスである conjunctive query に関する重要な問題（containement 問題など）の中には NP 完全であるものが存在する．ある conjunctive query を表現した hypergraph に対して制限を与えることにより，これらの問題が多項式時間で解けることが知られている．hypergraph を木構造に分解する手法である hypertree decomposition と，それに基づく hypergraph のパラメータである hypertree width の概念は Gottlob らにより提案された．伊藤らは hypergraph 内の枝集合の連結度を表す k-linked 性に基づく hypertree decomposition 構築アルゴリズムを提案した．本研究では枝集合の連結度を表す別の指標である k-hyperlinked 性に基づき，伊藤らの hypertree decomposition 構築アルゴリズムを利用したアルゴリズムを提案する．本アルゴリズムは hypergraph と定数 k を入力として与えたとき，width が $3k-1$ 以下の hypertree decomposition を出力するか，入力された hypergraph の hypertree width が k 以上であることを知らせる．hypergraph の枝数と頂点数をそれぞれ m，n としたとき，アルゴリズムの計算量は $O(m^{k+2}n)$ である．

キーワード Hypertree Decomposition，Hypertree Width, Hyperlinkedness

1. はじめに

関連データベースへの問合せを表す基本的なクラスである conjunctive query に関する重要な問題（containment 問題など）の中には一般に NP 完全であるものが存在する．しかし，conjunctive query を hypergraph で表現し，その hypergraph に制限を与えることによりその conjunctive query は多項式時間で解けることが知られている [2].

hypergraph の環状性（cyclicity）を定量的に表現する方法がいくつか知られている．Gottlob ら [4] は hypergraph の分解手法として hypertree decomposition を提案し，環状性を表す hypertree width を定義した．hypergraph から hypertree decomposition を構築するアルゴリズムはこれまでに提案されており，最適な hypertree decomposition を構築するものや，実行時間短縮のため最適とは限らないがある定数以下の width を持つ hypertree decopmosition が構築するものなどがある [6]．一般的に width が小さい hypertree decomposition の方が問題は解きやすくなる．伊藤ら [8] は hypergraph の枝集合の交わりの度合いを表す k-linked 集合 (k は正整数) を定義して hypertree width との関係を示し，これを基に width が $4k+3$ 以下の hypertree decomposition か，入力された hypergraph の hypertree width が k 以上であることを出力するアルゴリズムを提案した．このアルゴリズムは最適な hypertree decomposition を構築するとは限らないが，他のアルゴリズムと比べ十分に高速に実行できる．hypergraph 内の枝集合の連結度として，Adler ら [1] がグラフにおける linkedness の概念を hypergraph に適合させた k-hyperlinked な枝集合の概念がある．本論文ではこの k-hyperlinked な枝集合の概念と hypertree width の関係を示し，伊藤らの提案した hypertree decomposition の構築アルゴリズムに用い，新たに width が $3k-1$ 以下である hypertree decomposition を構築するアルゴリズムを提案する．

2. 関連研究

データベースにおける conjunctive query に関する問題や，人工知能の分野における制約充足問題など，一般的に効率的な解法が見つかっていない NP 完全である問題の中には，問題を hypergraph 化して制限を与えることにより多項式時間で解けるようになるものがある．hypergraph の環状性（枝の交わり具合）はその問題の複雑さを表し，これまでに定量的に表現する方法がいくつか知られている．Chekuri ら [2] は hypergraph の分解手法である query decomposition と，hypergraph の環状性を表す query width を定義した．しかし，ある hypergraph の query width が定数以下であることを判定することは NP 完全であるということが Gottlob ら [4] により証明された．そこで Gottlob らは新たな hypergraph の分解手法として hypertree decomposition を提案し，環状性を表す hypertree width を定義した．彼らは与えられた hypergraph の hypertree width が k 以下であることを判定することは一般に NP 完全であるとしたが，k が定数ならば hypertree width が k 以下であることの判定は多項式時間で行え，かつ並列化可能であることを証明した [4]．Adler ら [1] は hypergraph の持つ不変量（hyperlinkedness，hyperbramble など）と hypertree width の関係について比較した．また Gottlob ら [5] は hypertree decomposition より一般化された generalized hypertree decomposition と generalized hypertree width に関する考察も行った．

hypergraph から hypertree decopmosition を構築するアルゴリズムに関する研究も行われている．Gottlob ら [4] は，入力された hypergraph と定数 k に対し，もし存在するならば width

が k 以下の hypertree decomposition を出力する非決定的アルゴリズム `k-decomp` を提案した．その後 Gottlob ら [3], [7] は同じく width が k 以下の hypertree decomposition を出力する決定的アルゴリズム `opt-k-decomp` を提案し，その計算量は m と n をそれぞれ hypergraph の枝数との頂点数としたとき $O(m^{2k}n^2)$ である．この `opt-k-decomp` は多項式時間で処理できるアルゴリズムであるが，入力が小さい hypergraph であっても膨大なメモリ量と時間を要するという欠点がある．そこで Gottlob ら [6] はそれらの問題を解消するアルゴリズム `det-k-decomp` を提案した．このアルゴリズムの計算量は，c を最も大きい枝が含む頂点の数としたとき $O(m^{k+1}\min(m,ck)n^2)$ である．

3. 諸 定 義

3.1 Hypergraph

hypergraph は頂点の有限集合 $V(H)$ と，頂点の部分集合族 $E(H)$ の対で構成され，$H=(V(H),E(H))$ もしくは単に $H=(V,E)$ と表す．一般に $e\in E(H)$ を *hyperedge* というが，以下では単に**枝**と呼ぶ．ある頂点を含む枝の数を，その頂点の**次数**と言う．枝 $e\in E(H)$ に含まれる頂点を $\mathbf{ver}(e)$ で表す．すべての枝について $|ver(e)|=2$ が成り立つものを一般にグラフと呼ぶ．また，枝集合 $F\subseteqq E(H)$ について，$\mathbf{ver}(F)$ は $\bigcup_{e\in F} ver(e)$ を意味する．枝集合 F に含まれる枝 $(e\in F)$ のことを F**-edge** とよぶ．

頂点 $a,b\in V(H)$ について，a と b が**隣接**しているとは，$a,b\subseteqq ver(e)$ となるような枝 $e\in E(H)$ が存在することである．**path(a,b)** とは，頂点の列 $v_0(=a),v_1,v_2,\ldots,v_h(=b)$ において v_i と $v_{i+1}(0\leqq i\leqq h-1)$ が隣接しているものである．hypergraph H が**連結**であるとは，全ての $a,b\in V(H)$ について path(a,b) が存在することである．本稿では連結な hypergraph のみを扱うこととする．

$W\subseteqq V(H)$，$a,b\in V(H)$ としたとき，a と b が *[W]-adjacent* であるとは，$a,b\subseteqq(ver(e)-W)$ となる枝 $e\in E(H)$ が存在することである．*[W]-path(a,b)* は，頂点の列 $v_0(=a),v_1,v_2,\ldots,v_h(=b)$ において v_i と $v_{i+1}(0\leqq i\leqq h-1)$ が $[W]$-adjacent であるものである．$C\subseteqq V(H)$ は，すべての $a,b\in C$ について，$[W]$-path(a,b) が存在するとき *[W]-conncected* であるという．極大な $[W]$-conncected な C を $[W]$-**component** と呼ぶ．

$C\subseteqq V(H)$ において，$cov(C)=\{e\in E(H)|ver(e)\cap C\neq\emptyset\}$ とする．$cov(C)$ に含まれる各枝 $c\in cov(C)$ について $ver(c)\nsubseteqq ver(cov(C)-c)$ を満たす枝集合の集合を $cov^*(C)$ と表す．

［例 1］ 図 1 の hypergraph H は連結である．頂点集合 $W\subseteqq V(H)$ を $W=\{c,e,f,g,h\}$ とすると，$[W]$-component は $C_1=\{a,b,d\}$ と $C_2=\{i,j,k\}$ の二つである．$cov(C_2)=\{e_5,e_6,e_7,e_8\}$ であり，$cov^*(C_2)=\{\{e_6,e_7\},\{e_6,e_8\}\}$ となる．■

$F\subseteqq E(H)$，$c,d\in E(H)$ としたとき，c と d が *[F]-neighbor* であるとは，$(ver(c)-ver(F))\cap(ver(d)-ver(F)$ が空でないことである．*[F]-path(c,d)* は，枝の列 $e_0(=c),e_1,e_2,\ldots,e_h(=d)$ において e_i と $e_{i+1}(0\leqq i\leqq h-1)$ が $[F]$-neighbor であるものである．$D\subseteqq E(H)$ は，すべての $c,d\in D$ について，$[F]$-path(c,d) が存在するとき *[F]-part* であるという．極大な $[F]$-part な D を $[F]$-**fragment** と呼ぶ．$F\subseteqq E(H)$ について，H_F は $(ver(F),F)$ から成る H の部分 hypergraph を示す．

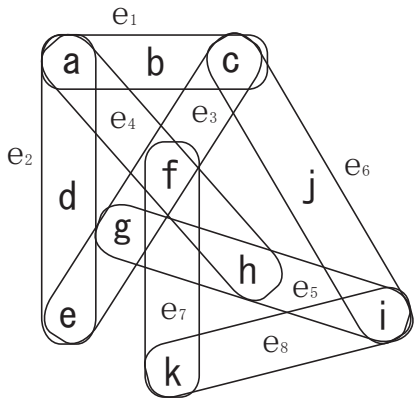

図 1　Hypergraph H

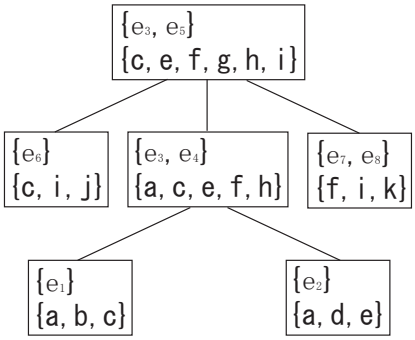

図 2　H の hypertree decomposition

3.2 Hypertree Decomposition

hypergraph H の **hypertree decomposition HD[H]** は，$\langle T,\chi,\lambda\rangle$ の組で表される．ここで T は根付木 $T=(V(T),E(T))$，χ と λ はそれぞれ H の頂点と枝を木 T のノードに写像する関数 $\chi:V(T)\to 2^{V(H)}$，及び $\lambda:V(T)\to 2^{E(H)}$ である．本稿では $v\in V(H)$ を**頂点**，$t\in V(T)$ を**ノード**と呼び，$e\in E(H)$ を**枝**，$e\in E(T)$ を**辺**と呼ぶ．あるノード t と隣接してるノードのうち，t から見て根側にあるノードを t の**親ノード**，葉側にあるノードを t の**子ノード**とよぶ．$t\in V(T)$ について，T_t は t を根とする T の極大な部分木を示す．部分木 T_t について，$\chi(T_t)$ と $\lambda(T_t)$ はそれぞれ $\bigcup_{n\in V(T_t)}\chi(n)$ と $\bigcup_{n\in V(T_t)}\lambda(n)$ を表す．

［定義 1］ (**Hypertree Decomposition**) hypergraph $H=(V,E)$ の hypertree decomposition は下記の条件を満たす $HD[H]=\langle T,\chi,\lambda\rangle$ の組である．

（1） 各枝 $h\in E(H)$ について，$h\subseteqq\chi(p)$ となる $p\in V(T)$ が存在する (これを p が h をカバーするという);

（2） 各頂点 $v\in V(H)$ について，T のノード集合 $p\in V(T)|v\in\chi(p)$ が T の連結部分木を形成する;

（3） 各ノード $p\in V(T)$ について，$\chi(p)\subseteqq ver(\lambda(p))$ が成り立つ;

（4） 各ノード $p \in V(T)$ について，$ver(\lambda(p)) \cap \chi(T_p) \subseteqq \chi(p)$ が成り立つ．

■

上記の条件のうち，(4) を **special condition** と呼び，special condition 以外の条件を満たすような $\langle T, \chi, \lambda \rangle$ の組を **generalized hypertree decomposition** という．

hypertree decompsition $HD[H]$ の **width**，**Width(HD)** は $\max_{t \in V(T)} |\lambda(t)|$ で定義される．hypergraph H の **hypertree width HW**$[H]$ は，$\min_{HD[H]} Width(HD)$ で定義される．$Width(HD) = HW[H]$ のとき，HD は**最適**であるという．

［例 2］ 図 2 は図 1 の hypergraph H の hypertree decomposition であり，その width は 2 である．

■

木 T には以下のような二つの性質がある．

（1） T からノード t を除いたとき，T は t の次数個の部分木に分離される．

（2） T から辺 e を除いたとき，T は二つの部分木に分離される．

木構造である hypertree decomposition にもこのような性質がある [8]．

［命題 1］ hypergraph H の hypertree decomposition $\langle T, \chi, \lambda \rangle$ の木 T からノード p が除かれたとき，複数の部分木 $T^1, T^2, \ldots, T^d$ に分離される．このとき，

$$\chi(T^1) - \chi(p), \chi(T^2) - \chi(p), \ldots, \chi(T^d) - \chi(p)$$

は互いに共通の頂点を含んでおらず，またこれら同士を結ぶ枝も H には存在しない．

証明 略 □

［命題 2］ hypergraph H の hypertree decomposition $\langle T, \chi, \lambda \rangle$ の木 T から辺 $(p,t)(p,t \in V(T))$ が除かれたとき，部分木 T_p と T_t が生じる．hypergraph から $\chi(p) \cap \chi(t)$ を除くと，その hypergraph は $\chi(T_p) - (\chi(p) \cap \chi(t))$ と $\chi(T_t) - (\chi(p) \cap \chi(t))$ の二つの連結成分に分離される．これらの連結成分は互いに共通の頂点を含んでおらず，またこれら同士を結ぶ枝も H 内に存在しない．

証明 略 □

3.3 Normal Form

Gottlob ら [4] は hypertree decomposition から冗長性を排除するため，normal form となる hypertree decomposition の条件を提案した．

［定義 2］ (**Normal Form** [4])

hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ の各ノード $r \in V(T)$ と，r の子ノード s が以下の条件を満たしているとき，*normal form* であるという．

（1） $\chi(T_s) = C_r \cup (\chi(s) \cap \chi(r))$ となるような $[\chi(r)]$-component C_r がただ一つ存在する

（2） 条件 (1) を満たす $[\chi(r)]$-component C_r において，$\chi(s) \cap C_r \neq \emptyset$

（3） $ver(\lambda(s)) \cap \chi(r) \subseteqq \chi(s)$

■

normal form の条件を満たす hypertree decomposition を**正規形**と呼ぶ．

4. k-hyperlinked 性判定アルゴリズム

4.1 k-hyperlinked 集合の概要

一般のグラフにおいて，あるグラフの連結度を示す概念として linkedness がある．Adler ら [1] はこれを hypergraph に適応し，hyperlinkedness を定義した．

［定義 3］ hypergraph H において，$M \subseteqq E(H)$，$C \subseteqq V(H)$ とする．

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

となるとき，C は M-*big* であるという．

■

$S \subseteqq E(H)$ であるとき，$H - \bigcup S$ に M-big な連結成分は多くても一つである．

［定義 4］ k を正の整数であるとする．$|S| < k$ を満たすあらゆる枝集合 $S \subseteqq E(H)$ について，$H - \bigcup S$ が M-big 成分を持つならば，枝集合 $M \subseteqq E(H)$ は k-hyperlinked である．

■

M が k-hyperlinked であるなら，M は $(k-1)$-hyperlinked でもある．H に含まれる k-hyperlinked 集合の k の最大値を H の *hyperlinkedness* と呼び，hlink(H) と書く．

枝集合 $S \subseteqq E(H)$ は，$M \subseteqq E(H)$ について $H - \bigcup S$ が M-big 連結成分を持たないとき，*balanced separator* であるという．hypergraph H において hlink$(H) \leqq k$ である必要十分条件は，あらゆる $M \subseteqq E(H)$ について k 以下の大きさの balanced separator が存在することである．

［定理 1］ hypergraph H が，$2k$ 以上の大きさで k-hyperlinked である枝集合を持つならば，HW$[H] \geqq k$ である．

証明 hypergraph H が k-hyperlinked である枝集合 $X(|X| \geqq 2k)$ を持ち，かつ HW$[H] < k$ であることを仮定する．HD$[H]$ は正規形であっても一般性を失わない．

- $ver(x) \subseteqq \chi(T_t)$ となる X-edge が $\lceil \frac{|X|}{2} \rceil$ 個以上
- t ができるだけ根ノードから離れている

以上の 2 条件を満たすノード t には必ず X-edge が 1 本以上含まれており，かつ葉ではない．t より親側の部分木に属するノードには $|X| - \lfloor |X|/2 \rfloor - 1$ 本以下，t より子側に存在するどの部分木も，属するノードには $\lfloor |X|/2 \rfloor$ 本以下の X-edge が存在する．以上より，t を取除くと X-big な連結成分は存在しないため，X が k-hyperlinked であるという仮定に矛盾する． □

4.2 アルゴリズム

本研究では伊藤らが提案した枝集合の k-linked 性（k は正の整数）を判定するアルゴリズムを前述の k-hyperlinked である枝集合の判定に適応した．本アルゴリズムは入力として hypergraph H，判定の対象となる枝集合 X，そして正整数 k が与えられ，以下に示す手順で枝集合が k-hyperlinked かどうかを判定する．

Algorithm 1 check_k-hyperlinked

Input: hypergraph H, 枝集合 X，定数 k
Output: メッセージ “X は k-hyperlinked である”, もしくは balanced separator S

for $\forall S \subseteqq E(H)$,$|S| = k-1$ **do**
  $l \leftarrow \{e \subseteqq E(H) | ver(e) \subseteqq ver(S)\}$
  if $l \geqq |X|/2$ **then**
    S を出力
  end if
  H 内の $[S]$-fragment を全て見つける
  各 $[S]$-fragment のうち，X-edge を一番多く含むものを K とする
  if $|K$ 内の X-edge$| \leqq |X|/2$ **then**
    S を出力
  end if
end for
メッセージ “X は k-hyperlinked である”を出力

（1） $E(H)$ から，$S(|S| = k-1)$ を任意に選ぶ．

（2） H から $\bigcup S$ を除くことにより，複数の $[S]$-fragment が生じる．

（3） これらの fragment のうち X-edge を最も多く含んでいるものが X-big 連結成分を持つかを判定し，持たなければ X は k-hyperlinked ではないことがわかる．

（4） X が k-hyperlinked であることを示すためには，H 内のあらゆる $S(|S| = k-1)$ について X-big 連結成分の存在を確認する．

枝集合の k-hyperlinked 性を判定するアルゴリズム `check_k-hyperlinked`(Algorithm 1) を示す．このアルゴリズムは対象となる枝集合が k-hyperlinked ではなかった場合，それを示す balanced separator を出力する．

$|E(H)| = m$, $|V(H)| = n$ としたとき，`check_k-hyperlinked` の計算量は $O(m^{k+1}n)$ である．

5. Hypertree Decomposition を構築するアルゴリズム

本章で示す hypertree decomposition 構築アルゴリズム `main`(Algorithm 2) と `HT-decomp`(Algorithm 3) は，hypergraph H と定数 k を入力とし，width が $3k-1$ 以下の hypertree decomposition か，“H の hypertree width は k 未満ではない”というメッセージを出力する．`HT-decomp` は hypertree decomposition の正規形を出力させるために，伊藤らのアルゴリズムを基に大河原ら [9] が提案したアルゴリズムを利用する．

彼ら [8], [9] の提案した hypertree decomposition の構築アルゴリズムにおいて鍵になる k-linked 性の判定アルゴリズムは，判定の対象となる枝集合の大きさが $3k+3$ になったときに呼び出されるが，本稿で提案するアルゴリズムでは判定の対象となる枝集合の大きさが $2k$ になった時点で呼び出される．これは H 内の枝集合の k-linked 性と HW$[H]$ の関係を判定するには，対象となる枝集合が $3k$ 以上の大きさが必要であったのに対し，本アルゴリズムでは定理 1 より，H 内の大きさ $2k$ 以上の枝集合があれば k-hyperlinked 性と HW$[H]$ の関係が判定できるためである．

Algorithm 2 main

Input: hypergraph $H = (V(H), E(H))$, 定数 k
Output: width が $3k-1$ 以下である H の hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$

枝集合 $E_r \subseteqq E(H)$ を任意に選ぶ $(1 \leqq |E_r| \leqq 2k-1)$
新しいノード $r \in V(T)$ を作る
$\lambda(r) \leftarrow E_r$
$\chi(r) \leftarrow ver(E_r)$
$[\chi(r)]$-component を全て見つける
for $\forall [\chi(r)]$-component C_r **do**
  HT-decomp(H, r, C_r, k)
end for
HD を出力

提案する hypertree decomposition 構築アルゴリズムは以下の手順で進行する．

（1） `main` では入力された hypergraph H の hypertree decomposition の木 T の根となるノード r を生成する．まず $E(H)$ から $2k-1$ 個以下の大きさの枝集合 E_r を任意に選ぶ．定義 1 の条件 (1) を満たすため，枝集合 E_r に含まれる頂点 $ver(E_r)$ を全て $\chi(r)$ に割り当てる．次に定義 1 の条件 (3) を満たすために，E_r を $\lambda(r)$ に割り当てる．ここで，r から $[\chi(r)]$-component C_r が複数生じる．命題 1 と 2 により，これらの連結成分はそれぞれ独立しているため，個別に処理することが出来る．各 C_r に対し，それぞれ `HT-decomp`(H, r, C_r, k) を呼び出す．

（2） `HT-decomp` は与えられたノードの子ノードを作る．ここでは r の子ノード t を作ることにする．`HT-decomp` はまず，与えられたノード r と C_r から $B_r = ver(cov(C_r)) \cap \chi(r)$ を計算する．B_r は，C_r と枝を共有する $\chi(r)$ の部分集合である．定義 1 の条件 (2) を満たすため，B_r は $\chi(t)$ に含まれている必要がある．次に枝 $e \in cov(C_r)$ を一つ任意に選ぶ．この e を $\lambda(t)$ に，$ver(e)$ を $\chi(t)$ にそれぞれ後のステップで加える．これらは定義 1 の条件 (1) と (3) を満たすために必要な操作である．このように `HT-decomp` が呼び出されるたびに新しい枝を一つずつ加えていくことによって入力された hypergraph は処理されていく．B_r を $\chi(t)$ に加えることから，定義 1 の条件 (3) を満たすために $A_r = B_r - ver(e)$ を全て含むような枝集合 $cov^*(A_r)$ を $\lambda(t)$ に加える必要がある．選ばれた $cov^*(A_r)$ を e とともに $\lambda(t)$ に，$ver(cov^*(A_r))$ を $ver(e)$ とともに $\chi(t)$ にそれぞれ加えることにより条件 (2) と (3) と (4) は満たされる．

（3） 新たに生じる $[\chi(r) \cup \chi(t)]$-component を入力として `HT-decomp` を呼び出すことにより，同様の方法で t の子ノードを作ることができる．

以上の操作により H の hypertree decomposition を構築することが出来る．しかしこの作業だけでは構築される hypertree decomposition の width は制限されないので，以下の条件 (♯) が常に満たされるようにする．

(♯) $|cov^*(A_t)| \leqq 2k-1$

Algorithm 3 HT-decomp

Input: hypergraph H, ノード r, $[\chi(r)]$-component C_r, 定数 k
Output: width が $3k-1$ 以下である C_r の hypertree decomposition$HD = \langle T, \chi, \lambda \rangle$, もしくはメッセージ "$H$ の hypertree width は k 未満ではない"

1: 頂点集合 $B_r \leftarrow ver(cov(C_r) \cap \chi(r))$
2: 枝 $e \in cov(C_{r_i})$ を任意に選ぶ
3: $A_r \leftarrow B_r - ver(e)$
4: $cov^*(A_r)$ を見つける
5: 新しいノード $t \in V(T)$ を作り，r の子とする
6: $\lambda(t) \leftarrow \{e\} \cup cov^*(A_r)$
7: $\chi(t) \leftarrow ver(e) \cup ver(cov^*(A_r))$
8: **if** $|\lambda(t)| = 2k$ **then**
9: `check_k-hyperlinked` $(H, \lambda(t), k)$
10: **if** メッセージ "X は k-hyperlinked である"が返ってきた **then**
11: メッセージ "H の hypertree width は k 未満ではない"を出力し，停止する
12: **else**
13: (S が balanced separator である)
14: $\lambda(t) \leftarrow \lambda(t) \cup (S \cap cov(C_r))$
15: $\chi(t) \leftarrow \chi(t) \cup ver(S \cap cov(C_r))$
16: **end if**
17: **end if**
18: C_r から $[\chi(t)]$-component を全て見つける
19: **for** $\forall[\chi(t)]$-component C_t **do**
20: `HT-decomp`(H, t, C_t,k)
21: **end for**
22: HD を出力

width が $3k-1$ 以下の hypertree decomposition を構築するのに対し，(♯) では $|cov^*(A_t)|$ は $2k-1$ 以下になるように定めてある．残りの k は，component から任意に選択される一つの枝と，後の段階で足し合わされる大きさ $k-1$ の枝集合のために用意されている．

$|cov^*(A_t)| < 2k-1$ であるならば，t の子ノード s では $\lambda(s) = cov^*(A_t) \cup e$ であるから，$|\lambda(s)| \leqq 2k-1$ であり，次のステップに進むことが出来る．

$|cov^*(A_t)| = 2k-1$ である場合，任意に枝 e を加えるとその子ノード s は (♯) に反してしまい，小さい width の hypertree decomposition を構築できなくなる．ここで枝集合 $X = cov^*(A_t) \cup e$ に対し `check_k-hyperlinked` を呼び出し，hypergraph の hypertree width が k 未満かどうかを調べる．今，X の大きさは $2k$ に等しく，定理 1 より X が k-hyperlinked であるなら，H の hypertree width は k 以上であると判定され，アルゴリズムは停止する．X が k-hyperlinked でないなら，それを示す balanced separator $S \subseteqq E(H)(|S| = k-1)$ が存在する．$S \cap cov(C_t)$ を $\lambda(s)$ に，そして $ver(S \cap cov(C_t))$ を $\chi(s)$ にそれぞれ加えることにより，新しい $\lambda(s)$ が出来る．この $\lambda(s)$ の大きさは $|cov^*(A_t)| + |e| + |S|$ であり，$3k-1$ である．また，同時に複数の $[\chi(r) \cup \ldots \cup \chi(s)]$-component も生じる．これらの component に対し再び `HT-decomp` を個別に呼び出すことを考える．B_s をカバーする枝集合の大きさは最大で $|cov(A_s) \cup e \cup S| - |X|/2 = ((2k-1)+1+(k-1)) - k = 2k-1$ であるから，(♯) に違反せずにアルゴリズムを進行することが出来る．

上記の手順により，width が $3k-1$ 以下の hypertree decomposition を構築することが出来る．

［命題 3］ `HT-decomp` が構築する hypertree decomposition は正規形である．[9]

証明 定義 2 の条件 (1) と (2) は明らかに満たされている．なぜなら本アルゴリズムは全ての $[\chi(t)]$-component を見つけ，各連結成分 C_t に対して `HT-decomp`(H, t, C_t, k) を呼び出しているからである．

$\lambda(s)$ は e と $cov^*(A_r)$(そして $S \cap cov(C_r)$) から構成されている．そして $\chi(s)$ は $ver(e)$ と B と $ver(cov^*(A_r))$(そして $ver(S \cap cov(C_r))$) から構成されている．これより定義 2 の条件 (3) も満たされている． □

$|E(H)| = m$，$|V(H)| = n$ として全体の計算量を評価する．本アルゴリズムにおいて最も計算量の多いステップは枝集合 X が k-hyperlinked であるかを調べる箇所であり，その計算量は $O(m^{k+1}n)$ である．`HT-decomp` が呼び出されるのは高々m 回なので，全体の計算量は $O(m^{k+2}n)$ である．

6. ま と め

我々は hypergraph 内の枝集合の k-hyperlinked 性を判定するアルゴリズム `check_k-hyperlinked` を提案し，これを伊藤ら [8] の提案した hypertree decomposition 構築アルゴリズムに用いた．本アルゴリズムは hypergraph H と定数 k が与えられたときに width が $3k-1$ 以下の hypertree decomposition を出力するか，H の hypertree width が k 以上であることを知らせるアルゴリズムを示した．アルゴリズムの計算量は，H の枝数と頂点数をそれぞれ m，n としたとき，$O(m^{k+2}n)$ である．

謝辞 本研究の一部は，(独) 日本学術振興会科学研究費補助金基盤研究 (C)(課題番号:19500089) による．

文 献

[1] Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *European Journal of Combinatories*, Vol. 28, pp. 2167–2181, 2007.

[2] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Springer LNCS*, Vol. 1186, pp. 56–70, 1997.

[3] Georg Gottlob, Nicola Leone, and Francesco Scarcello. On tractable queries and constraints. In *10 th International Conference and Workshop on Database and Expert Systems Applications(DEXA)*, pp. 1–15, 1999.

[4] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Science*, Vol. 64(3), pp. 579–627, 2002.

[5] Georg Gottlob, Zoltan Miklos, and Thomas Schwentick.

Generalized hypertree decompositions: Np-hardness and tractable variants. In *PODS'07*, 2007.

[6] Georg Gottlob and Marko Samer. A backtracking-based algorithm for computing hypertree-decompositions. In *arXiv.org:cs/0701083*, 2007.

[7] Nicola Leone, Alfredo Mazzitelli, and Francesco Scarcello. Cost-based query decompositions. In *SEBD*, pp. 390–403, 2002.

[8] 伊藤由花, 片山薫. Conjunctive query の hypertree decomposition 構築のための貪欲アルゴリズム. In *DEWS2007*, 2007.

[9] 大河原達郎, 片山薫. Hypertree decomposition の正規形を構築するためのアルゴリズム. In *DEWS2008*, 2008.