Conjunctive Query の Hypertree Decomposition 構築のための貪欲アルゴリズム

伊藤　由花† 　片山　 薫††

† 東京都立大学工学部電子情報工学科
†† 首都大学東京大学院システムデザイン研究科

あらまし　関係データベースに対する基本的な問い合わせのクラスである Conjunctive Query におけるいくつかの重要な問題（containment 問題など）は NP 完全である．しかし，問い合わせを表す hypergraph が acyclic であれば，これらの問題を効率的に処理することができる．Gottlob らは hypergraph の Hypertree-Width を定義し，hypertree-width が定数である hypergraph では containment 問題などが多項式時間で処理できることを示した．本稿では，問い合わせと定数 w に対し，width $4w$+3 以下の Hypertree decomposition を出力するかまたは H の hypertree-width は w より大きいことを出力するアルゴリズムを与える．計算量は $O(m^{w+2}n)$ である（n:頂点数，m:枝数）．

キーワード　問い合わせ最適化，parametrized complexity，ハイパーグラフ，木分解

1. はじめに

conjunctive query は，選択・射影・ジョインによって表現される関係データベースへの問い合わせの基本的なクラスである．一般に，問い合わせを最適化するには時間がかかる．しかし，acyclic な問い合わせであれば，その問い合わせは多項式時間で処理できることが知られている [3]．Chekuri ら [3] は問い合わせの閉路の度合いを表す query width という数を定義した．また query decomposition を定義し，問い合わせを木構造に写像する方法を示した．木構造に分解された問い合わせに対して containment 問題（ある問い合わせが他の問い合わせに含まれるかどうかを判定する問題）などが多項式時間で処理できる．しかし，query-width がある定数以下であるかどうか判定することは NP 完全だということが Gottlob らによって示された [5]．彼らは query decomposition を改良した hypertree decomposition および hypertree-width を定義した．hypertree-width がある定数以下であるかは多項式時間で判定できる．Leone ら [8] は決定性アルゴリズムを用いて（定数 w 以下の）最適な hypertree decomposition を構築するアルゴリズムを示した．n を頂点数，m を枝数としたとき，計算量は $O(m^{2w}n^2)$ であった．

本稿では，conjunctive query と定数 w が与えられたとき，query を hypergraph で表し貪欲アルゴリズムを用いて分解する．hypertree decomposition の width は $(4w+3)$ 以下である．もしくは “obstruction set” を示すことにより hypergraph の hypertree-width は w より大きいことを示す．obstruction set とは hypertree-width が w 以下になるのを妨げる hyperedge の部分集合である．w-linked という概念を定義し，obstruction set を特徴づける．計算量は $O(m^{w+2}n)$ である．

2. Conjunctive Queries

conjunctive query は最も基本的な関係データベースへの問い合わせのクラスである．

［例 1］　以下のような問い合わせ Q_1 と Q_2 を考える

$$Q_1 : ans \leftarrow a(L,M,N) \wedge b(M,O) \wedge c(N,P)$$
$$Q_2 : ans \leftarrow a(L,M,N) \wedge b(M,O) \wedge c(N,O)$$

L, M などをそれぞれ **variable** とよび，$a(L,M,N)$ などのまとまりを **atom** とよぶ．

conjunctive query は hypergraph で表すことができる．Q_1, Q_2 に対応する hypergraph を図 1,2 に示す．hypergraph が **acyclic** であれば，その問い合わせを acyclic であるといい，そうでない場合を cyclic という．hypergraph が acyclic であるかどうかは，以下で定義する **join tree** が存在するかどうかで判定する．

［定義 1］　join tree JT は以下の 3 つの条件を満たす木である
（1）　JT は木
（2）　JT の頂点は atom と 1 対 1 で対応する
（3）　ある variable X が a_1 と a_2 に含まれるならば，JT 上で a_1 と a_2 をつなぐ道上の atom は X を含む

Q_1 の join tree を図 3 に示す．join tree が存在するので，Q_1 は acyclic である．一方，Q_2 は join tree を作ることができない．よって cyclic である．

データベースで acyclic は重要な概念である．acyclic な問い合わせであれば，一般には NP 完全である問題 (containment 問題など) が多項式時間で解けることが示されている [3].

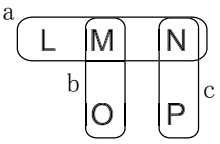

図 1　Q_1 に対応する hypergraph

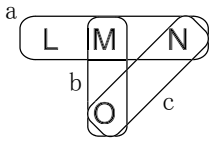

図 2　Q_2 に対応する hypergraph

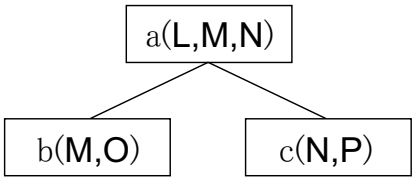

図 3　Q_1 の join tree

3. 関 連 研 究

acyclic な hypergraph は join tree を作ることにより効率的な処理が可能であった．これは木に対してボトムアップに処理することにより，中間のリレーションを小さく保つことができるからである．さらに，あるノードがほかのノードを気にせずに処理できることで，並列的な処理が可能だからである．cyclic なグラフは join tree を作ることができない．しかし，以下で定義する query-width がある定数以下であれば，いくつかの頂点や枝をまとめてひとつの木のノードに写像することにより acyclic と同様の処理を可能とする．

これまでにさまざまな hypergraph の分解方法が提案されている．以下ではいくつかの代表的な分解方法について述べる．

3.1 Tree Decomposition for graphs [9]

一般のグラフについての tree decomposition を示す．tree decomposition はグラフの頂点を木のノードに写像することにより，木と同様の構造を持たせたものである．以下で示すような hypergraph に対する木状分解はこの tree decomposition を応用している．

グラフは $G = (V(G), E(G))$ と表すことにする．

［定義 2］ **(Tree Decomposition)**

tree decomposition $TD[G] =< T, \chi >$ とは，木 $T = (V(T), E(T))$ と関数 $\chi : V(T) \rightarrow 2^{V(G)}$ の組である．ただし，以下の 3 つの条件を満たす．

（1） $\bigcup_{t \in V(T)} \chi(t) = V(G)$

（2） 各枝 $e \in E(G)$ について，$e = \{u, v\}$ ならば，$\{u, v\} \subseteqq \chi(t)$ となる t が存在する

（3） 各頂点 $v \in V(G)$ について，$\{t \in V(T) | v \in \chi(t)\}$ は T の連結な部分木を構成する

$TD[G]$ の **width** を $\max_{t \in V(T)}\{|\chi(t)|\} - 1$ で定義する．1 を引くのは任意の木の tree-width を 1 にしたいためである．グラフ G の **tree-width tw[G]** を $\min_{TD[G]}\{width(TD)\}$ で定義する．tw[G] はすべての G の木分解のなかで width の最小値を表す．tree-width が小さいほどグラフは木に近い形をしていることを表す．

3.2 Query Decomposition [3]

ある conjunctive query Q が acyclic な形に近ければ Q は“扱いやすい”とされる．扱いやすさの指標を query-width とよぶ．また acyclic のような形にすることを query decomposition という．

［定義 3］ **(Query Decomposition)**

Query Decomposition QD[Q] とは，以下の 3 つの条件を満たす $< T, X >$ の組のことである．$T = (V(T), E(T))$ は木であり，X は $V(T)$ から atom と variable の集合への写像である．

（1） 各 atom $a \in Q$ に対して，ある $t \in V(T)$ が存在して，$a \in X(t)$

（2） 各 atom $a \in Q$ に対して，$\{t \in V(T) | a \in X(t)\}$ が T の連結な部分木を構成する

（3） 各 variable $L \in Q$ について，$\{t \in V(T) | L \in X(t)\} \cup \{t \in V(T) | L$ を含む $atoms \in X(t)\}$ が T の連結な部分木を構成する

QD[Q] の **width** を $\max_{t \in V(T)} |X(t)|$ で定義する．また，Q の **query-width** $qw(Q)$ を $\min_{QD[Q]}\{width(QD)\}$ で定義する．

3.3 Cycle Cutset and Hypercutset [2]

cycle cutset S とは，hypergraph H の頂点の部分集合 $S \subseteqq V(H)$ で，$V(H) - S$ が acyclic となるものである．cutset-width とはすべての cycle cutset の大きさの最小値である．

4. Hypertree Decomposition

本章では後で使う命題を証明する．

4.1 Hypergraph に関する諸定義

hypergraph は $H = (V(H), E(H))$ と表す．$V(H)$ は頂点，$E(H)$ は頂点の部分集合の集合である．一般に $E(H)$ を **hyperedge** というが，以下では単に**枝**とよぶ．枝の次数がすべて 2 のものが（一般の）グラフである．枝 $e \in E(H)$ に含まれる頂点を **var(e)** と表す．hypergraph ではすべての $e \in E(H)$ について $var(e) \neq \phi$ である．また，枝集合 $E \subseteqq E(H)$ について，$\bigcup_{e \in E} var(e)$ を **var(E)** と略記する．

頂点 $a, b \in V(H)$ が隣接しているとは，$\{a, b\} \subseteqq var(e)$ を満たす $e \in E(H)$ が存在することである．頂点 $a, b \in V(H)$ 間の path(a,b) とは，頂点の列 $v_0(= a), v_1, v_2, \ldots, v_h(= b)$ で 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，すべての $a, b \in V(H)$ について path(a,b) が存在することである．以下，連結な hypergraph のみを扱うこととする．

$W \subseteqq V(H)$ とする．頂点 $a, b \in V(H)$ が [W]-adjacent であるとは，$\{a, b\} \subseteqq (var(e) - W)$ を満たす $e \in E(H)$ が存在すること

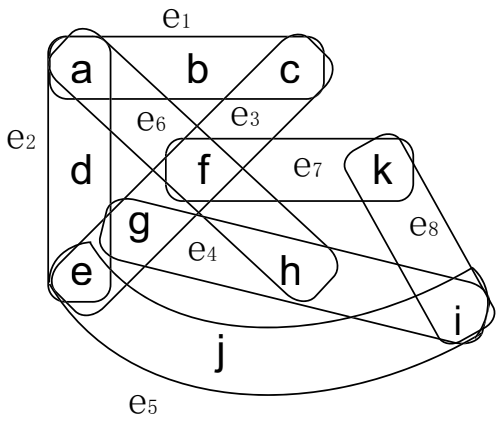

図 4　hypergraph H

である．[W]-path(a,b) とは頂点の列 $v_0(=a), v_1, v_2, \ldots, v_h(=b)$ で v_i と v_{i+1} $(0 \leqq i \leqq h-1)$ が [W]-adjacent であるものである．$C \subseteqq V(H)$ が [W]-connected であるとは，すべての $a, b \in C$ について，[W]-path(a,b) が存在することである．極大な [W]-connected な C を **[W]-component** と呼ぶ．ある [W]-component C に対して，$\mathbf{E}(\mathbf{C})$ を $E(C) = \{e | var(e) \cap var(C) \neq \phi\}$ で定義する．

［例 2］ 図 4 のような hypergraph H を考える．H は連結である．$var(e_2) = \{a, d, e\}$ である．$W = \{c, e, f, g, h\}$ とすると，[W]-component は $\{a, b, d\}$ と $\{i, j, k\}$ である．$C = \{i, j, k\}$ とすると，$E(C) = \{e_4, e_5, e_7, e_8\}$ である．

4.2　Hypertree Decomposition とその性質

hypertree decomposition は根付木 $T = (V(T), E(T))$ と関数 $\chi : V(T) \to 2^{V(H)}$ と関数 $\lambda : V(T) \to 2^{E(H)}$ からなる．以下，$V(H)$ を頂点，$V(T)$ をノードとよぶこととする．T_p は p を根とする極大な部分木を表す.

［定義 4］ **(Hypertree Decomposition)**

hypergraph H の hypertree decomposition HD[H] とは，以下の 4 つの条件を満たす $< T, \chi, \lambda >$ の組のことである

（1） 各枝 $e \in E(H)$ について，ある $p \in V(T)$ が存在して，$var(e) \subseteqq \chi(p)$

（2） 各頂点 $v \in V(H)$ について，集合 $\{p \in V(T) | v \in \chi(p)\}$ が T の連結な部分木を構成する

（3） 各ノード $p \in V(T)$ について，$\chi(p) \subseteqq var(\lambda(p))$

（4） 各ノード $p \in V(T)$ について，$var(\lambda(p)) \cap \chi(T_p) \subseteqq \chi(p)$

HD[H] の **width(HD)** を，$\max_{p \in V(T)} |\lambda(p)|$ で定義する．また，H の **hypertree-width hw[H]** を $\min_{HD[H]}\{width(HD)\}$ と定義する．任意の acyclic な hypergraph の hypertree-width は 1 である．$HD[H] = hw(H)$ のとき，HD は**最適**であるという．

［例 3］ 図 4 の hypergraph の hypertree decomposition を図 5 に示す．width は 2 である．また $hw[H] = 2$ である．

木の性質として，あるノードをグラフから削除すると，削除したノードの次数ぶんだけ連結成分に分かれる．また，ある枝を削除するとグラフは 2 つの連結成分に分かれる．hypertree

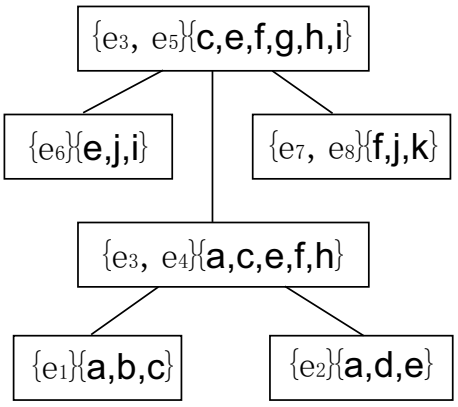

図 5　H の hypertree decomposition

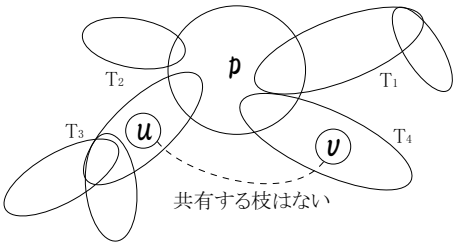

図 6　命題 1 の概観図

decomposition もこのような性質を生かしたグラフの分解方法である．

［命題 1］ 木 T からノード p を削除する．このとき木はいくつかの部分木 $T_1, T_2, \ldots, T_d$ に分かれる．このとき，

$$\chi(T_1) - \chi(p), \chi(T_2) - \chi(p), \ldots, \chi(T_d) - \chi(p)$$

は互いに頂点を共有しておらず，また共有する枝もない（図 6 参照）．ただし $\chi(T_i)$ は $\bigcup_{t \in T_i} \chi(t)$ をあらわす.

証明　証明には背理法を用いる．まず $\chi(T_i) - \chi(p)$ と $\chi(T_j) - \chi(p)$ $(i \neq j)$ がともに頂点 v を共有していたとする．T_i に含まれるノードから T_j に含まれるノードへのどんな path も p を含んでいるので $\chi(p)$ は v を含むことになる．これは $\chi(T_i) - \chi(p)$ と $\chi(T_j) - \chi(p)$ $(i \neq j)$ がともに頂点 v を共有することに矛盾する．

次に $v \in (\chi(T_i) - \chi(p))$ と $u \in (\chi(T_{\mathrm{j}}) - \chi(p))$ の間に枝が存在するとする $(i \neq j)$．このとき定義から $(v, u) \subseteqq \chi(q)$ となるノード q が存在する．q は T_i に含まれていないとする．v は $\chi(T_i)$ と q に含まれている．T_i と q を結ぶ道上には p があり，$\chi(p)$ は v を含むことになる．これは $\chi(T_i) - \chi(p)$ に v があることに矛盾する．また q が T_i に含まれているときには u が $\chi(T_i)$ と $\chi(T_j)$ に含まれることになり，矛盾が生じる． □

［命題 2］ 木 T から枝 (p, t) を削除する．このときできる 2 つの部分木をそれぞれ T_p,T_t とする．このとき，$\chi(p) \cap \chi(t)$ をグラフから削除すると，グラフは $\chi(T_p) - (\chi(p) \cap \chi(t))$ と $\chi(T_t) - (\chi(p) \cap \chi(t))$ の 2 つのコンポーネントに分かれる．この 2 つは頂点を共有しておらず，また共有する枝もない．(図 7 参照)

証明　命題 1 と同様 □

分解されたグラフを扱う際には，冗長性が排除されているこ

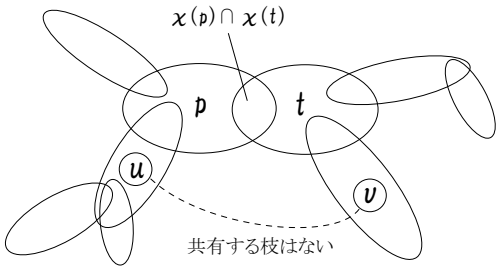

図 7　命題 2 の概観図

とが望ましい．以下で定義する nomal form は冗長性を極力排除した hypertree decomposition である．

［定義 5］ **(nomal form)**

以下の 4 つの条件を満たすとき，hypertree decomposition $HD =< T, \chi, \lambda >$ は nomal form である．ただし $r \in V(T)$ の子を s とし，C_r は $[\chi(r)]$-component を表す．

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

（2） 条件 1. を満たす C_r について，$\chi(s) \cap C_r \neq \phi$ が成り立つ

（3） 条件 1. を満たす C_r について，各 $h \in \lambda(s)$ に対し $var(h) \cap var(E(C_r)) \neq \phi$ が成り立つ

（4） 条件 1. を満たす C_r について，$\chi(s) = var(E(C_r)) \cap var(\lambda(s))$ が成り立つ

条件 (4) から，nomal form であれば，各 s について，$\lambda(s)$ と条件 (1) を満たす $[\chi(r)]$-component から $\chi(s)$ を求めることができる．

［定理 1］ [8]

H の k-width hypertree decomposition が存在すれば，H の normal form な k-width hypertree decomposition が存在する

証明　略　□

5. Hypertree Decomposition の構築

一般に hypergraph H の最適な hypertree decomposition を構築することは困難である．しかしある定数 w が与えたとき，H の hypertree-width が w 以下であれば多項式時間で最適な hypertree decomopsition を構築できる．本論文では定数 w に対して width が $(4w + 3)$ 以下の hypertree decomposition を構築するアルゴリズムを示す．最適ではないが，Leone らの最適な hypertree decomposition を求めるアルゴリズムより高速に動作する．

5.1 Obstruction Set

アルゴリズムを構築するためにまず，hypergreph H が定数 w 以下の hypertree-width をもつのに “障害”となるような H の構造を見つける．以下で w-linked という概念を導入し障害となる構造を特徴づける．

［定義 6］ 大きさの等しい 2 つの枝集合 $Y, Z \subseteqq E(H)$ が **separable** であるとは，以下の 2 つの条件 $(\diamondsuit)$ を満たす $S \subseteqq E(H)$ が存在することである．

$$(\diamondsuit) = \begin{cases} (1)|S| < |Y| = |Z| \\ (2)var(Y) \text{ から } var(Z) \text{ への} \\ [var(S)]\text{-path(Y,Z) が存在しない} \end{cases}$$

(2) の条件を満たすとき，S は Y と Z を**分離**するという．

［定義 7］ 枝集合 $X \subseteqq E(H)$ が w**-linked** であるとは，以下の 2 つの条件を満たすときをいう

（1） $|X| \geqq w$

（2） $Y, Z \subseteqq X$, $|Y| = |Z| \leqq w$ を満たすすべての Y と Z の組について，Y と Z は separable でない

直感的に，w-linked な枝集合は密接に関連していて，簡単には 2 つのコンポーネントに分離できないことがわかる．ある $p \in V(T)$ に対して $\chi(p) \subseteqq var(\lambda(p))$ なので，命題 1 から $var(\lambda(p))$ は hypergraph を分離することがわかる．しかし w-linked な集合は w 個以上の枝がないと分離することができないので hypertree-width が大きくなることが予想できる．hypergraph H と $X \subseteqq E(H)$ が与えられたとき X が w-linked かどうか判定するアルゴリズムは 6. 章で述べる．

［定理 2］ Hypergraph H が w-linked である枝集合 X ($|X| \geqq 3w$) を含むならば， H の hypertree-width は w 以上である

証明　背理法を用いる．すなわち，H が w-linked である集合 X ($|X| \geqq 3w$) をもち，かつ $hw(H) < w$ と仮定する．HD は nomal form とする．以下 X に含まれる枝を X-edge とよぶ．

はじめに以下の条件を満たす $t \in V(T)$ を探す．

- $var(x) \subseteqq \chi(T_t)$ となる X-edge が $2w$ 個以上
- 根からできる限り離れている

$var(x) \subseteqq \chi(T_t)$ となるとき T_t は x をカバーしているという．明らかに条件を満たす t は T の葉ではない．したがって t はいくつかの子ノード $t_1, t_2, \ldots, t_d$ をもつ．これらの子ノードは根から離れた位置にあるので $2w$ 個未満の X-edge をカバーしている．いま， T_{t_i} がカバーする X-edge の数が多いほうから $1, 2, \ldots, d$ と添え字を付け直す．以下を終了するまで繰り返す．

$i \leftarrow 2$
if t_1 がカバーする X-edge が w 個以上
then 終了
else $t_1 \leftarrow t_1 \cup t_i$
$i \leftarrow i + 1$

t_1 がカバーする X-edge は w 個以上 $2w$ 個以下である．ここで t_1 がカバーする X-edge から w 個を選び Y とする．また，t_1 に含まれていない X-edge から w 個を選び，Z とする．命題 1 よりノード t に含まれている頂点 $\chi(t)$ 削除することによってグラフは分離される．定義より $\chi(t) \subseteqq var(\lambda(t))$ であるから，$var(\lambda(t))$ を削除してもグラフは分離されるはずである．枝 $\lambda(t)$ が含む頂点 $var(\lambda(t))$ を削除することによって Y と Z は分離される．しかし仮定より Y と Z は w-linked であり定義から $(w - 1)$ 個以下の枝で分離されない．これは矛盾である．　□

5.2 提案アルゴリズム

hypertree decomposition を構築する貪欲アルゴリズムを述べる．hypergraph H と定数 w の入力に対して，width が $(4w+3)$

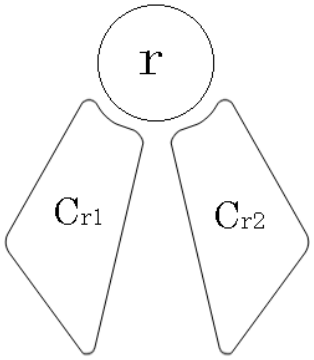

図 8　$[\chi(r)]$-component

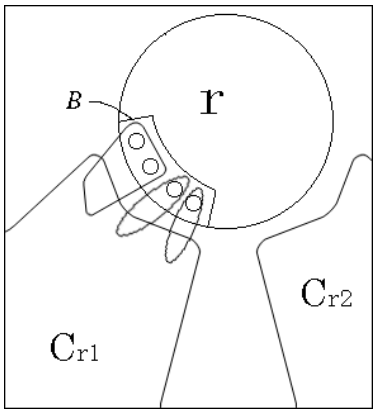

図 9　$B = var(E(C)) \cap \chi(r)$

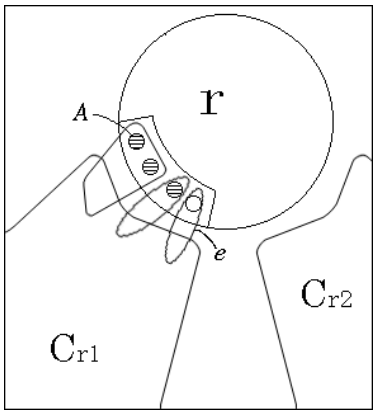

図 10　$A = B - e$

以下の hypertree decomposition または H の hypertree-width は w より大きいことを出力する．アルゴリズムはまず枝を任意に選ぶことからはじめる．枝の数は 1 以上 $(3w + 2)$ 以下とする．これらの枝を木の根 r に割り当てる．またこれらの枝に含まれる頂点すべてを根に割り当てる．このときいくつかの $[\chi(r)]$-component ができる (図 8 参照)．hypertree decomposition の性質から，ある component はほかの component を気にすることなく処理をすることができる．アルゴリズムは，親ノード r と 1 つの $[\chi(r)]$-component から子ノード t をつくり，今度は t を親ノードとして新たに子をつくり，と再帰的な構造になっている．

親ノード r と 1 つの $[\chi(r)]$-component C_r が与えられたとき，

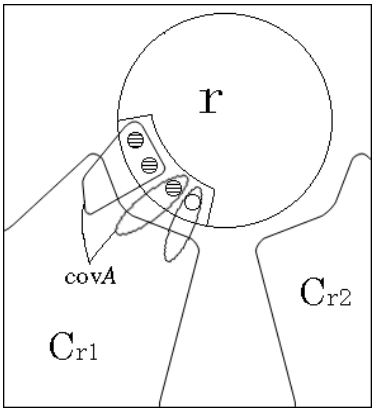

図 11　cov$A$

まず $B = var(E(C_r)) \cap \chi(r)$ を計算する．B は $\chi(r)$ のうち C_r へ枝が伸びている頂点の集合である (図 9 参照)．r の子を t とすると，定義 4 の条件 (2) から，$B \subseteqq \chi(t)$ となる．次に $E(C_r)$ から枝 e を任意に 1 つ選ぶ．この操作によって hypertree decomposition をさらに進めることができる．$var(e)$ を $\chi(t)$ に含め，e を $\lambda(t)$ に含める．これは定義 4 の条件 (1) および条件 (3) を満たすために必要な操作である．定義 4 の条件 (3) を満たすためには，あと $A = B - var(e)$ をカバーする枝を $\lambda(t)$ に含めなければならない．A をカバーする枝集合を cov$A$ と表す (図 10, 図 11 参照)．A をカバーするとは，$A \subseteqq var(\mathrm{cov}A)$ となることである．ただし冗長性を排除するため cov$A$ はさらに条件，1. 任意の $a \in \mathrm{cov}A$ に対して $a \cap A \neq \phi$，$2.var(a) \not\subseteqq var(\mathrm{cov}A - a)$，を満たしているとする．cov$A$ を $\lambda(t)$ に含めることにより定義 4 の条件 (3) を満たす．$a \in E(C_r)$ となる $a \in \mathrm{cov}A$ に対しては，$var(a)$ を $\chi(t)$ に含めることによって条件 (1),(4) を満たす．以上の操作により親ノード r と 1 つの $[\chi(r)]$-component C_r が与えられれば hypertree decomposition の定義に合致するように，r の子を作ることができる．さらに t と $[\chi(t)]$-component について同様の操作を行えば H の hypertree decomposition が構築できる (図 12 参照)．

width を $(4w + 3)$ 以下に制限するために，条件 (♯) がいつでも成り立つようにする．

(♯) $|\mathrm{cov}A| \leqq 3w + 2$

$(4w + 3)$ に対して $(w + 1)$ 足りないのは，選んだノード e と，component 内の枝を含めるための “空き”である．

アルゴリズムの cov$A$ を求めた段階で，$|\mathrm{cov}A| < 3w + 2$ ならばこのまま次の段階へ進むことができる．なぜなら，まずノード r の子を t とし，r,t での cov$A$ をそれぞれ covA_r,covA_t とする．$\lambda(r) = \mathrm{cov}A_r \cup e$ であり，$|\lambda(r)| \leqq 3w + 3$ である．$A_t \subseteqq var(\lambda(r))$ となることから，$|\mathrm{cov}A_t| \leqq |\lambda(r)| \leqq 3w + 3$ とすることができ，必ず (♯) を満たすからである．しかし $|\mathrm{cov}A| = 3w + 2$ のときは次の，子の段階で条件 (♯) を満たさない可能性がある．

$|\mathrm{cov}A| = 3w + 2$ となったとき，H が本当に w 以下の hypertree-width を持つか確かめることとする．cov$A$ と新たに追加した枝 e を $\lambda(t)$ とする．$|\lambda(t)| = 3w + 3$ である．定理 2

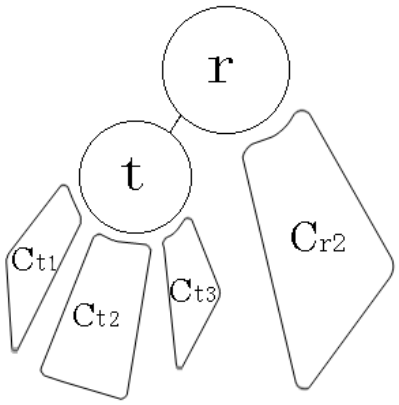

図 12　C_t component

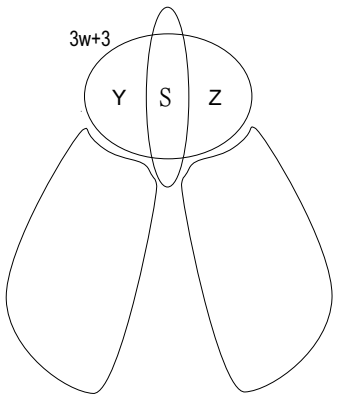

図 13　S を加えて新たなノードをつくる

から，$(w+1)$-linked な枝集合 X($|X| \geqq 3(w+1)$) を含むならば，H の hypertree-width は $(w+1)$ 以上であるので，$\lambda(t)$ が $(w+1)$-linked であればここで処理を中断する．

もし $(w+1)$-linked でなければ，$\lambda(t)$ のある枝集合 Y, Z に対して，条件 ($\diamondsuit$) を満たす S が存在する．これは，hypertree decomposition をさらに拡張できることを意味している．$S' = (E(C_r) \cup Y \cup Z) \cap S$ とし，$\lambda(t)$ に $S' \cap E(C_r)$ を加える．$\lambda(t)$ の大きさは $|\lambda(t)| \leqq 4w+3$ となる．ここで各 $[\chi(t)]$-comopnent C_t に対して HT-decomp(t, C_t) 実行時のことを考える．$B = var(E(C_t)) \cap \chi(t)$ は $(\mathrm{cov}A_t + e_t - Z) \cup S'$ (または $(\mathrm{cov}A_t + e_t - Y) \cup S'$) でカバーできる．もしカバーできないなら，$E(C_t)$ が $Z - S$ (または $Y - S$) と共有点をもつことになる．これは S が Y と Z を分離することに矛盾する．$|\mathrm{cov}A| \leqq |(\mathrm{cov}A_t + e_t - Z) \cup S'| \leqq (3w+3) - (w+1) + w = 3w+2$ であるので，条件 ($\sharp$) を満たしている (図 13 参照)．

以上のアルゴリズムにより，width $(4w+3)$ 以下の hypertree decomposition を構築することができる．アルゴリズムをまとめたものを図 14 に示す．関数 check_w-linked は入力 hypergraph H,$X \subseteqq E(H)$, 定数 w に対して，X が w-linked ならば “w-linked”というメッセージを，w-linked でないならある $Y, Z \subseteqq X$ に対して条件 ($\diamondsuit$) を満たす S を出力する．

ALGORITHM hypertree-decomp
INPUT: Hypergraph H=(V(H), E(H)), 定数 w
OUTPUT: width $(4w+3)$ 以下の A Hypertree Decomposition $HD =< T, \chi, \lambda >$ または “H の hypertree width は w より大きい” というメッセージ

HT-decomp(parent_node r, one_of_[$\chi(r)$]-component C_r) {
  頂点集合 $B \leftarrow var(E(C_r)) \cap \chi(r)$ とする
  枝 $e \in E(C_r)$ を 1 つ選ぶ
  頂点集合 $A \leftarrow B - var(e)$ とする
  A をカバーする枝集合 Cov$A$ をみつける
  新たなノード $t \in V(T)$ をつくり，r の子とする
    $\lambda(t) \leftarrow \{e\} \cup \mathrm{Cov}A$
    $\chi(t) \leftarrow B \cup var(e)$
  for $a \in \mathrm{Cov}A$ **do**
    if $a \in E(C_r)$ **then**
      $\chi(t) \leftarrow \chi(t) \cup var(a)$
    end if
    if $|\lambda(t)| = 3w+3$ **then**
      check_w-linked$(H, \lambda(t), w+1)$
      if “$(w+1)$-linked” **then**
        “H の hypertree-width は w より大きい”と出力して終了
      else
        ($\diamondsuit$) を満たす S に対して
        $\lambda(t) \leftarrow \lambda(t) \cup (S \cap E(C_r))$
        $\chi(t) \leftarrow \chi(t) \cup var(S \cap E(C_r))$
      end if
    end if
    C_r 内での $[\chi(t)]$-component をすべて見つける
    for 各 $[\chi(t)]$-component C_t **do**
      HT-decomp(t, C_t)
    end for
  end for
}

MAIN{
  枝集合 $\{e_i\} \subseteqq E(H)$ $(1 \leqq i \leqq 3w+2)$ を任意に選ぶ
  新たなノード $r \in V(T)$ をつくる
    $\lambda(r) \leftarrow \{e_i\}$
    $\chi(r) \leftarrow \bigcup_{i \in I} var(e_i)$
  $[\chi(r)]$-component をすべて見つける
  for 各 $[\chi(r)]$-component C **do**
    HT-decomp(r, C)
  end for
}

図 14　Hypertree Decomposition 構築アルゴリズム

6. w-linked 判定アルゴリズム

6.1 諸 定 義

[E]-fragment を定義する．これは [V]-comoponent の対となる考えである．

枝集合 $E \subseteqq E(H)$ に対して，枝 $e_1, e_2 \in E(H)$ が [E]-neighbor であるとは，$(var(e_1) - var(E)) \cap (var(e_2) - var(E)) \neq \phi$ を満たすことをいう．[E]-path(c,d) とは，枝の列 e_0(=

$c), e_1, e_2, \ldots, e_h(=d)$ で e_i と e_{i+1} $(0 \leqq i \leqq h-1)$ が [E]-neighbor であるものである．$D \subseteqq E(H)$ が [E]-part であるとは，$D \cap E = \phi$ かつすべての $c, d \in D$ について [E]-path(c,d) が存在することである．極大な [E]-part な D を **[E]-fragment** とよぶ．

$F \subseteqq E(H)$ に対して $\mathbf{H_F}$ を，頂点 var(F) と枝 F からなる H の部分グラフとする．

6.2 提案アルゴリズム

ある hypergraph H の枝集合 X と定数 w が与えられたときに，X が w-linked であるかどうかを判定するアルゴリズムを与える (ただし $|X| \geqq w \geqq 2$)．判定方法として通常考えられるのは，定義 7 から，すべての $Y, Z \subseteqq X$ に対して Y と Z を分離する $|S| < |Y| = |Z|$ となる S が存在するかどうかを調べる方法である．しかし Y と Z を分離するような S を “うまく” 見つけるのは容易ではない．

そこで，まず S の候補を列挙し，それらについて Y,Z となる集合が存在するかどうかを調べることにする．S の候補となる枝集合を K とする．アルゴリズムの基本的な考え方は，まず適当に K を選ぶ ($|K| = k < w$ とする)．そうするといくつかの [K]-fragment がつくられる．これらの fragment を 2 つのグループに分ける．このとき，ふたつのグループに含まれている X-edge の数がともに k より大きければ，それぞれのグループに含まれている X-edge から $(k+1)$ 個ずつ選んで Y, Z とし，K を S とすれば，定義から X は w-linked でないことがわかる．w-linked であると判定するには，K を $1 \leqq |K| \leqq w-1$ の範囲のすべてを調べ，K は条件 $(\diamondsuit)$ を満たす S ではないことを示す必要がある．

しかし $|X| \geqq 2w$ であれば，$|K| = w-1$ のすべての K を調べるだけでよい．たとえば，集合 Y, Z に対して，条件 $(\diamondsuit)$ を満たす S が存在したとする．$|Y| = |Z| < w$ ならば，適当な枝 e $(e \in X, e \notin Y, Z)$ をとって Y, Z, S それぞれに加えることを繰り返すことにより，条件 $(\diamondsuit)$ を保ったまま $|Y| = |Z| = w$ とすることができる．さらに $|S| < w-1$ のとき，適当な枝 $e(e \in E(H), e \notin S)$ を S に加えることにより，$|S| = w-1$ とすることができる．このようにして拡大された Y, Z, S は条件 $(\diamondsuit)$ を満たしている．よって，条件 $(\diamondsuit)$ を満たすすべての S は，$|S| = w-1$ となる S のどれかに含まれているため，$|K| = w-1$ となる K のみを調べればよいことがわかる．

図 15 は集合 X が w-linked かどうか判定するアルゴリズムである．

このアルゴリズムは H_X を調べる部分と H 全体を調べる部分に分けられる．実は H_X を調べる部分がなくてもプログラムは正常に動作する．むしろ，X が H 内でばらばらに存在していると H_X を調べる部分と H 全体を調べる部分とで同じような操作をするので二度手間である．しかし hypertree decomposition のアルゴリズムと組み合わせて使うことを考えると，hypertree decomposition の目的とアルゴリズムから X にはある程度の局所性が期待できる．よって H 全体を調べる前の段階で S の候補を十分ふるい落とすことが期待できる．

アルゴリズムをもう少し詳しくみていく．$var(e) \subseteqq var(K)$ となる $e \in X$ は Y と Z 両方に含まれてもよい枝である．なぜなら $var(e) - var(K) = \phi$ より $var(e)$ からの [K]-path が存在しないからである．$l \geqq w$ であれば l 個の中から w 個選んで Y とし，X の中から適当に Z を選ぶことにより $(\diamondsuit)$ を満たす $S(=K), Y, Z$ となる．

H_X を調べる部分と H 全体を調べる部分はほぼ同じ操作である．[K]-fragment を 2 つのグループに分けるときは X-edge が均等に含まれるようにした．K が $(\diamondsuit)$ を満たす S であるための十分必要条件は X-edge の数が 2 つのグループに w 個以上

ALGORITHM check_w-linked
INPUT: Hypergraph H=(V(H), E(H)), 枝集合 $X \subseteqq E(H)$, width w
OUTPUT: X が w-linked のとき “w-linked” というメッセージまたは 条件 $(\diamondsuit)$ を満たす $S \subseteqq E(H)$

check_w-linked {
すべての $K \subseteqq E(H)$,$|K| = w-1$ について
  $var(e) \subseteqq var(K)$ となる $e \in X$ の個数を l とする
  if $l \geqq w$ **then**
    $S = K$ を出力して終了
  end if
  H_X での [K]-fragment をすべて求める
  H_X の各 [K]-fragment K_i について，$|K_1| \geqq |K_2| \geqq \ldots \geqq |K_d|$ とする
  $Y \leftarrow |K_1|$
  $Z \leftarrow |K_2|$
  for $i = 3$ to d **do**
    if $Z \leqq Y$ **then**
      $Z \leftarrow Z + |K_i|$
    else
      $Y \leftarrow Y + |K_i|$
    end if
  end for
  if $(l + Y < w)$ or $(l + Z < w)$ **then**
    この K は $(\diamondsuit)$ を満たす S ではない
  end if
  H での [K]-fragment をすべて求める
  各 [K]-fragment K_i について，X-edge を多く含む方から，$K_1, K_2, \ldots, K_d$ とする
  $Y \leftarrow (K_1$ に含まれる X-edge の数)
  $Z \leftarrow (K_2$ に含まれる X-edge の数)
  for $i = 3$ to d **do**
    if $Z \leqq Y$ **then**
      $Z \leftarrow Z + (K_i$ に含まれる X-edge の数)
    else
      $Y \leftarrow Y + (K_i$ に含まれる X-edge の数)
    end if
  end for
  if $(l + Y \geqq w)$ and $(l + Z \geqq w)$ **then**
    $S = K$ を出力して終了
  end if
すべての K を調べ終わったら，“w-linked” と出力
}

図 15　Hypertree Decomposition 構築アルゴリズム

あることである．逆に H_X を調べる段階でどちらかのグループの X-edge の数が w 個未満だと，hypergraph 全体を調べても w 個以上になることはない．

7. 提案アルゴリズムの計算量の評価

アルゴリズム hypertree-decomp の計算量を評価する．H の頂点数を n，枝数を m とする．

まず入力された枝集合が w-linked かどうか調べる関数 check_w-linked の計算量を評価する。1 つの K に対しアルゴリズムはグラフを全探索するので計算量は $O(mn)$ となる．$|K| = w-1$ となる K は全部で ${}_mC_{w-1}$ 個ある．$O({}_mC_{w-1})$ は $O(m^{w-1})$ でおさえることができるので全体の計算量は $O(m^w n)$ である．

関数 HT-decomp では枝 $e \in E(C)$ をひとつ選ぶが，H のある枝が選ばれるのは高々1 回である．よって関数 HT-decomp が呼び出されるのは高々m 回である．HT-decomp 中で計算コストが一番かかるのは $\lambda(s)$ が $(w+1)$-linked であるかどうかを調べることであり，計算コストは $O(m^{w+1}n)$ である．全体の計算コストは $O(m^{w+2}n)$ である．しかし $\lambda(s)$ が $(w+1)$-linked であるかどうか調べるのは $\lambda(s) = 3w+3$ になったときだけであり毎回呼び出されるわけではない．$(w+1)$-linked であるかどうか調べなかった場合，HT_decomp 1 回の実行にかかるコストは $O(mn)$ である．$(w+1)$-linked であるかどうか 1 度も調べなければ，計算量は $O(m^2 n)$ である．

Leone らが提案した定数 w 以下の最適な hypertree decompositon を求めるアルゴリズムの計算量は $O(m^{2w}n^2)$ であったが，より厳密に書けば $O(\Psi^2 n^2)$ である．Ψ は，

$$\Psi = \sum_{i=1}^{w} {}_mC_i = \sum_{i=1}^{w} \frac{m!}{i!(m-i)!}$$

を表している．本稿で提案したアルゴリズムも，より厳密に書けば $O({}_nC_w m^2 n)$ である．よって本稿で提案した hypertree decomposition のアルゴリズムの計算量は，最適な hypertree decompositon を求めるアルゴリズムの計算量に比べて十分速いことがわかる．

8. ま と め

hypergraph H と定数 w の入力に対して width $(4w+3)$ 以下の hypertree decomposition または hypertree-width は w より大きいことを出力するアルゴリズムを示した．また，w-linked という概念を導入しグラフの一部の構造から hypertree-width は w より大きいこと判定することができた．計算量は $O(m^{w+2}n)$ であった．

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 (B)(2)(課題番号:16300030) による．

文 献

[1] Hans L.Bodlaender. *A Liner Time Algorithm for Finding Tree-Decompositions of Small Treewidth.* SIAM Journal on Computing Vol.25,No.6,pp.1305-1317,1996.

[2] R.Dechter.*Constant Networks.* Encyclopedia of Artificial Intelligence, second edition, Wiley & Sons, pp.276-285,1992.

[3] Chandra Chekuri, Anand Rajaraman *Conjunctive Query Containment Revisited.* Springer LNCS, vol.1186, pp.56-70, 1997.

[4] George Gottlob, Martin Grohe, Nysret Musliu, Marko Samer, and Francesco Scarcello.*Hypertree Decompositions: Structure, Algorithms, and Applications.* WG 2005.

[5] Georg Gottlob, Nicola Leone, Francesco Scarcello. *Hypertree Decompositions and Tractable Queries.* Journal of Computer and System Science 64(3),pp.579-627, 2002.

[6] Georg Gottlob, Nicola Leone, Francesco Scarcello.*Robbers, Marshals, and Guards: Game-theoric and Logical Characterizations of Hypertree Width.* Proc. PODS'01,pp195-206,2001.

[7] Jon Kleinberg, Eva Tardos. *Algorithm Design* Addison Wesly,2006.

[8] Nicola Leone, Alfredo Mazzitelli, Francesco Scarcello.*Cost-Based Query Decompositions.* Proc. of SEBD 2002.

[9] N.Robertson, P.D.Seymour.*Graph minors.II.Algorithmic aspects of tree-width.* J.Algorithms, 7:309-332,1986.