Vol.2016-IFAT-122 No.1
Vol.2016-DC-101 No.1
2016/3/24

動的木グラフの分割と再構成

須藤篤志† 鈴木雄也† 都司達夫† 樋口 健†

概要： 本研究では，動的に成長するラベル付き木グラフを対象として、記憶負荷のバランスを考慮して部分木に動的に分割する方式を提案することを目的とする．ルートノードからのラベルパス式およびノード位置情報を効率よく扱うために、当研究室で開発中の経歴・パターン法と呼ぶ多次元データのエンコード方式を使用する．記憶量の削減と検索速度の向上を目的として，ノード位置情報のエンコードを動的分割に対応させて効率的に行う方式を提案するとともに，木グラフの動的分割と再構成を行う．このための経歴・パターン法によるエンコード方式改良の考え方を説明し、その問題点と対処の方法を述べ、評価する．

Partitioning Dynamic Growing Tree and Its Reorganization

ATSUSHI SUDOH† YUYA SUZUKI† TATSUO TSUJI† KEN HIGUCHI†

1. はじめに

木構造グラフ(以下木グラフ）はきわめて，応用性の高いグラフ構造であり，ソーティング・サーチング，情報検索，データマイニング，XML 文書処理，など，これまできわめて多くの応用分野において多くの研究が行われてきた．木グラフはこれらの分野において，データ表現・知識表現に用いられ，その簡便で強力な操作性により，効率良い分析・処理が可能であることが広く認識されている．さらに，XMLDB やマイニングなど，システム運用時のグラフ規模の動的な増大や構造の変化に効率良く対処する必要がある分野の増加に伴い，その強力な処理技術が強く求められている．

木グラフをクラスタリングして分割処理する手法は数多く提案されており，並列処理による性能向上(例えば，[4]）や問い合わせアルゴリズムの性能向上(例えば,[5]）を目的とするものが多い．一方，動的に成長する木構造を効率よく扱うためには木構造におけるノード位置を反映したルートノードから各ノードに至る経路式のエンコーディング（ラベル付け）が重要である．例えば，[4]，[5]，[7]などでは．XML 文書の構造を表す XML 木について，その動的な構造更新に対して再エンコードを行う必要のないノードのエンコード方式を提案している．

本研究では，動的に成長するラベル付き木グラフを対象として、記憶負荷のバランスを考慮して部分木に動的に分割する方式を提案することを目的とする．ルートノードからのラベルの経路式およびノード位置情報を効率よく扱うために、当研究室で開発中の経歴・パターン法と呼ぶ多次元データのエンコード方式[8]を使用する．記憶量を削減し，検索速度を向上させるために，ノード位置情報のエンコードを動的分割に対応させて効率的に行う方式を提案するとともに，木グラフの動的分割と再構成を行う．このための経歴・パターン法によるエンコード方式の問題点とその対処の方法を述べ、実装・評価する．

† 福井大学
University of Fukui

2. 経歴・パターン法とその実装

2.1 経歴・パターン法

経歴・パターン法[6]は拡張可能配列の概念を基にした多次元データセットのエンコード方式である．一般に n 次元データタプル t（n 2）の各次元を n 次元配列 A の次元に対応させ，t の各次元属性値を A の当該次元の添字に対応させれば，t は A の一つの配列要素を表す添字座標で表現できる．n 次元データタプルの集合 M のタプルが動的に増加し，新たな属性値が現れる場合には，A の各次元サイズは動的に拡張できる必要がある．

図 1 に示すように拡張可能配列 A の拡張は拡張する直前の配列と同形のサイズの n 次元部分配列が拡張次元方向に付加される．拡張時に付加される各部分配列はそれが何番目に拡張・付加されたものであるかを示す経歴値で識別される．

この経歴値は各次元毎に用意される“経歴値テーブル” Hi (i=1, …, n)に格納される（ 図 1)．部分配列中の要素は，その部分配列の経歴値と，A の配列空間におけるその要素の座標のエンコードの対で表される．座標のエンコードは各次元の添字の値のビット長を記録した境界ベクトルと呼ぶベクトルを当該部分配列の経歴値により引当て，このビット長にしたがって，各次元添字を結合することにより得られる座標パターンと呼ぶビット列にエンコードされる．この境界ベクトルおよび，拡張された次元は“境界ベクトルテーブル”と呼ぶ単一の配列 B に格納される．図 1 において，○は M のタプルに対応する拡張可能配列の要素を表わしている．

ここで，経歴・パターン法による座標値のエンコード例

を示す．例えば，座標(4, 1)の要素は　図 1 に示すように経歴値が 5, 座標パターンが 100.01(2)にエンコードされる(“.”は各次元添字の境界位置を表す)．このエンコードの手順は，まず座標(4, 1)の要素がどの部分配列に属するかを求める．これには各次元の座標値を表すのに必要なビット数を求めるが，これは，それぞれ 3 ビット，1 ビットである．経歴値テーブル H1 および，H2 を参照し，H1[3] = 5，H2[1] = 1 であり，H1[3] > H2[1]であるから，(4, 1)の要素は経歴値 5 の部分配列に属することがわかる．

次に境界ベクトルテーブルよりその経歴値 5 に対応する境界ベクトルを求め，これをもとに座標値の結合処理を行う．この例では経歴値 5 の境界ベクトルは<3, 2>であるから，1 次元目の添字に 3 ビット，2 次元目の添字に 2 ビットずつ用いて座標値の結合を行う．また，座標値は左から 1 次元目，2 次元目として結合を行っている．1 次元目の座標 $100_{(2)}$と 2 次元目の座標 $01_{(2)}$をビットシフトとマスク処理によって結合して座標パターン $100.01_{(2)}$が生成される．以上より，座標(4, 1)のエンコードは<5,17>となる．

逆にエンコードされた＜経歴値，座標パターン＞の対をエンコード結果の例<5, 17>により，座標にデコードする手順を説明する，経歴値が 5 であるから，境界ベクトルテーブルより，B[5] = <3, 2>を引き当てる．座標パターンが 17 = $10001_{(2)}$であり，これを 3 ビットと 2 ビットに分け座標(4, 1)を得る．

境界ベクトルの作成方法より，次の性質が成り立つ．

“経歴値 h は座標パターンのビット長を表す．“

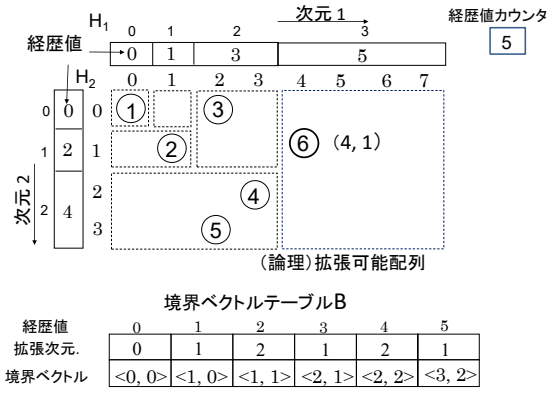


図 1 経歴・パターン法の説明図

Figure 1 Data structures for history-pattern encoding

2.2 タプルエンコード用データ構造

図 2 のテーブルはタプルが順に挿入されたときに，タプルがマッピングされる拡張可能配列の座標，エンコードに使用される境界ベクトル，および，エンコード結果の経歴・パターンを表している．これは図 1 に示した拡張可能配列の成長を表している．タプルが順次挿入されるのに従って，配列が拡張され，エンコードされる．タプルのエンコードに必要なデータ構造として，図 1 に示したもののほか，以下が必要である．

(1) “CVT_i (i=1, …, n)”：各次元について，その属性値を拡張可能配列の対応する次元の添字にマッピングするためのデータ構造（B^+木）．

(2) “属性値テーブル”と呼ぶ属性値の一次元配列 A_i (i=1, …, n)．エンコードされた経歴・パターンを属性値の組としてタプルにデコードするのに必要．

(3) エンコードされた経歴・パターンの出力用ファイル．RDT と呼ぶ B+木ファイルまたは ETF と呼ぶ逐次ファイル

(3)の出力用ファイルが RDT の時には(1)~(3)による経歴パターン法の実装データ構造を HPMD，ETF の時には HPRD と呼ぶ．

	会社	商品		座標	境界ベクトル	<経歴値,パターン>
1	D社	鉛筆	→	(0, 0)	<0,0>	<0, .> = <0,0>
2	B社	ハサミ	→	(1, 1)	<1,1> ○	<2, 1.1> = <2,3>
3	C社	鉛筆	→	(2,0)	<2,1> ○	<3,10.0> = <3,4>
4	A社	定規	→	(3,2)	<2,2> ○	<4,11.10> = <4,14>
5	C社	消しゴム	→	(2,3)	<2,2>	<4,10.11> = <4,11>
6	E社	ハサミ	→	(4,1)	<3,2> ○	:5,100.01> = <5,17>

[注] ○はタプルの挿入により拡張可能配列の当該次元サイズが拡張されることを表す．

図 2 図 1 に対応したタプルのエンコーディング例

Figure 2 A tuple encoding example for Fig.1

HPMD(HPRD)では新しい属性を低コストで追加することができる．例えば，図 2 のテーブルに新しく「価格」という属性を追加する場合，HPMD は 2 次元から 3 次元に次元拡張が行われる．このとき，「価格」次元用に経歴値テーブルが設けられ，新しく CVT が追加される(図 3)．また，境界ベクトルは以後，3 次元ベクトルとなる．次元拡張以前にエンコード済の＜経歴値，座標パターン＞は再エンコードの必要はなく，また，境界ベクトルテーブルを再構成する必要もない．次元拡張以前の境界ベクトルの次元 3 の値は次元拡張後に 0 として扱うことで，次元 3 の添え字のビット列は空列となるからである．

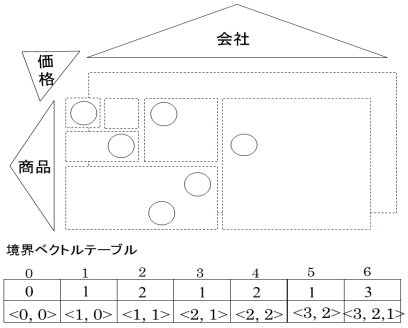


図 3 図 2 の HPMD を 3 次元に拡張した後の状態

Figure 3 Dimensional extension of HPMD for Fig.2

3. ラベル付き木グラフの格納

3.1 木グラフの拡張可能配列へのマッピング

図 4 に示すように高さ n の木グラフ T の各高さレベルは n 次元拡張可能配列の次元に対応させ，T の各ノードを拡張可能配列の座標に対応させる．経歴･パターン法により，T の各ノードは＜経歴値，座標パターン＞にエンコードすることができる．ノードの挿入により木の各高さレベルにおける最大枝分かれ数が増加したときに，対応する次元方向に配列が拡張する．また，木の高さが 1 増加したときには配列の次元拡張(図 4 参照）が起こり n+1 次元となる．拡張可能配列の特性により，木の高さ n までの既存のノードの<経歴値，座標パターン>は再計算する必要はない．

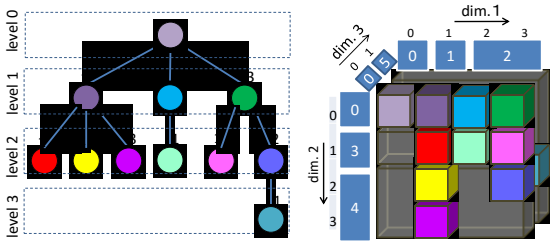


図 4 木グラフの拡張可能配列へのマッピング
Figure 4 Mapping Dynamic Tree to an Extendible Array

3.2 構造木と経路木

対象とするラベル付き木グラフモデルとして

(a) 辺にはラベルが付され，同じ親ノードから出る二つ以上の兄弟ノードへの辺のラベルは同一でもよい．

(b) また，ノードや辺は動的に付加･除去できる．

と定める．図 5 の左にこのような木グラフの例を示す．

上記のモデルによる木グラフを拡張可能配列への次の二種類の埋め込みで実現する．一つは木グラフ T から辺のラベルを捨象した木構造 T1 の埋め込みである．T1 の各ノードを拡張可能配列の座標に対応させ，経歴・パターン法により，＜経歴値，座標パターン＞にエンコードする．このエンコード結果は当該ノードの祖先の各ノードおよび兄弟ノードの位置情報を保持しており，この位置情報に基づいて，木グラフの構造検索を行う．他の一つは，辺のラベルの連接である経路式による検索のために木グラフ T の同一ノードから出る辺のラベル重複を排して，経路式の一意性を保証したグラフ T2 を構成して拡張可能配列に埋め込む．T2 のルートノードから各ノードに至る経路を，同様に経歴・パターン法により，エンコードする．

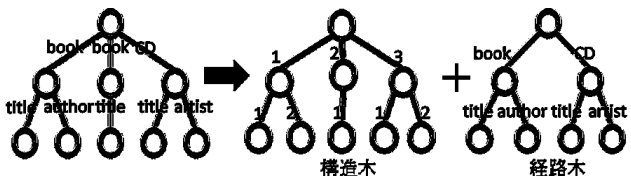


図 5 木グラフの構造木と経路木への分離
Figure 5 Separation of a tree graph to two kinds of tree

T の各ノードの構造情報および経路情報がそれぞれ，＜経歴値，座標パターン>の 2 値に集約されることが記憶コスト，検索コストの点で有効に機能し得る．ここでは T1 を構造木，T2 を経路木という．図 5 の右部分に構造木と経路木への分離を示す．経路木は構造要約の手法[6]によっている．以後，構造木の各ノードの座標のエンコード結果を nID(node ID)，経路木の各ノードの座標のエンコード結果を pID(path ID)と呼ぶ．

4. 構造木のエンコード上の問題点

構造木を経歴・パターン法によりエンコードする際の問題点として，以下の 2 つの問題が考えられる．

一つ目の問題として，木グラフであるために祖先パターンの重複問題が挙げられる．図 6 の例では，座標が (2,3,2,1,1), (2,3,2,1, 2), (2,3,2,1,3) であるレベル 5 のノードが 3 個ある．これらの 3 個のノードの座標のうち，祖先の座標パターン (2, 3, 2, 1) までの部分が重複している．子孫の数が多いほど，また，レベルが高くなるほど重複した祖先パターンの占める記憶量が大きくなる．

二つ目の問題として，経歴･パターン法を用いた木グラフのマッピングにおいて，最大レベルより低いレベル位置においてノードの追加が行われると，追加したノードのレベルに対応する次元より後の次元の添字はすべて 0 となる．例えば，論理拡張可能配列の次元数が 6 であるときに，レベル 2 の位置にノードの追加が行われると，そのノードの座標は(1, 2, 0, 0, 0, 0)のように次元 3 からの添字はすべて 0 となる．また，この座標における経歴値が 24，境界ベクトルが< 1, 2, 6, 6, 5, 4 >であるとすると，座標パターンは 110000000000000000000000(2)となり，24 ビット(3 バイト)を占める．そこで，全次元の添字を座標パターンにエンコードするのではなく，ノードのレベル位置に対応する次元までの添字を座標パターンにエンコードするようにする．上記の例では，ノードのレベル位置に対応する次元 2 までの添字を座標パターンに含める．従って，座標パターンは 110(2)となる．このとき，座標パターンの記憶コストは 3 ビット(1 バイト)となる．

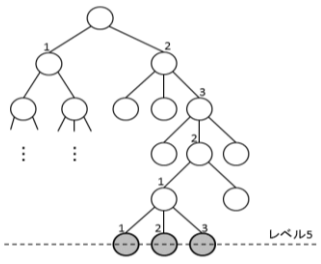


図 6 祖先パターンの重複
Figure 6 Duplication of ancestor pattern

5. 構造式の分割

3 節で述べた 2 つの問題点を解決するために，ここでは構造木を複数の部分木に分割する．経路木は図 5 のように

構造要約を行っており，一般的に非常に小さいものとなるため，分割は構造木に対してのみ行う．構造木の分割に伴って，ルートノードから任意のノードに至る辺の順序数(次元添字)の連接である構造式を 2 つに分割する必要がある．

5.1 問題点に対する対策と構造式分割の考え方

本研究では構造木を使用環境(例えば，物理スレッドの最大数)に見合う分割数 k をあらかじめ設定し，k 個の逐次ファイル{ $ETF_1, ETF_2, \cdots, ETF_k$ }を用意する(図 7)．各 ETF は構造式の分割に伴って分割されるノード座標をエンコードして得られる経歴パターンを順次格納する逐次ファイルである．分割点を含まないノードの経歴・パターンはキーとして，RDT とよぶ B+木に格納する．分割点に設定したノード n 以降の経路は，ノード n をルートとする経歴・パターンを上記 ETF のいずれかに順次格納する．

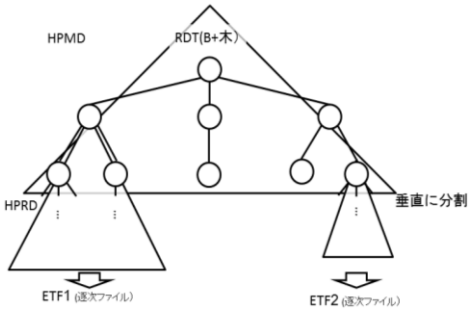


図 7 分割点による垂直分割

Figure 7 Vertical partitioning of a tree via split points

例として図 8 の左に示したの木グラフを分割点(2, 3, 2)で分割する．座標パターン分割の基本的な考え方として，分割点以下のノードに関しては境界ベクトルテーブルより経歴パターンにエンコードし RDT に格納する．図 8 の(2, 3, 2)は分割点であるため，境界ベクトルテーブルより<6, $101110_{(2)}$>と変換でき，これを RDT に格納する．

分割レベルより高いレベルのノードに関しても，境界ベクトルテーブルより経歴・パターンにエンコードするが，分割点までのパターンと，分割点以降のパターンに分けて格納する．このとき自身より高いレベルである座標 0 を省略して格納する．こうすることにより記憶コストを下げることができる．図 8 の(2, 3, 2, 1, 9, 0)は分割点を通るためパターンの分割を行う．分割点までの座標パターンと自身より高いレベルである座標 0 を省略し，パターンは<14, $11001_{(2)}$>となる．これを ETF に格納する．分割レベルまでの座標パターンに関しては，すでに RDT に格納されている．

しかし読み込み時に，自身のレベル以降の 0 のパターンを省略して格納しているため経歴値と分割点のレベルだけではパターンサイズが分からない．そこでパターンの先頭部分に分割以降のレベル数をメタ情報として一つの次元として扱い，経歴パターンにエンコードする．一般的に分割以降レベル数は大きくなく，高々数ビットの記憶コストで済む．こうすることで読みだしてくるときに経歴値，分割点のレベル，分割以降レベルを用いて境界ベクトルテーブルよりから ETF に格納されている実際のパターンサイズを知ることができる．

図 では図 と同様に分割点を(2, 3, 2)としている．座標(2 , 3 , 2, 1, 9)のノードを登録する際，分割点以降のレベル数は 2 となるため，ETF に格納する座標は(2 , 2 , 3 , 2, 1, 9, 0)となる．これをそのまま経歴パターンにエンコードすると< 16, $1010111011001000_{(2)}$>となるが，分割点までのパターンと自身のレベル以降の座標 0 を省略し，パターンは<16, $1011001_{(2)}$>となり，これを ETF に格納する．

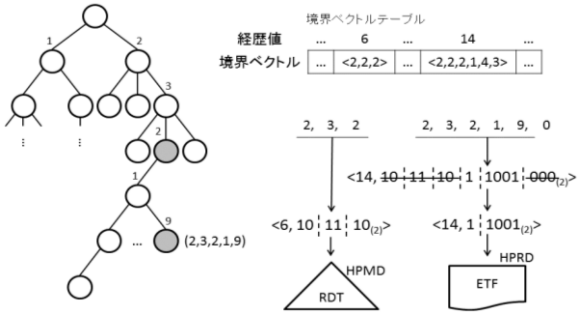


図 8 経歴･パターンの分割例

Figure 8 An example of splitting history pattern

分割レベルまでのパターンに関しては，分割以降レベル数の代わりに自身のレベル数を一つの次元として扱う．そのため，ETF に登録した際は分割以降レベル数であったため次元 1 の添字は 2 であったが，RDT に登録する際には次元 1 の添字を分割レベルである 3 にし，経歴パターンにエンコードする．こうして得られた経歴パターン<8, $11101110_{(2)}$>を RDT のキーとして登録する．

検索時には，目的のノードが分割点未満の場合は HPMD の RDT によりランダムアクセスで高速に検索できる．分割点以降の場合は，RDT は目的のノードの経歴・パターン値が格納されている ETF を絞り込む索引として機能する．ETF では，コストのかかるディスクシークを避けシーケンシャルに検索する．

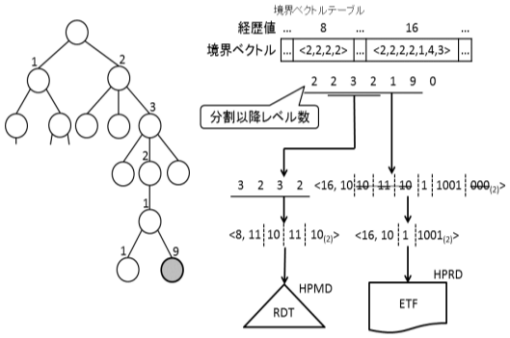


図 9 経歴･パターンの分割

Figure 9 Splitting history pattern

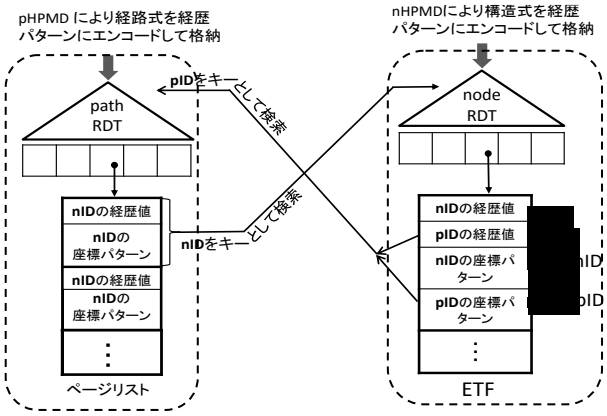
5.2 部分木の格納と nID，pID の相互参照

図 10 に n ID と pID を相互参照するためのデータ構造を

示す．5.1 節で説明した部分木のノードの nID は対応する pID と対にして対応する ETF に格納される．

pID に対応する nID 集合にアクセスするために図 10 の path RDT の B+木のキー部に pID，データ部にページリスト管理した pID に対応する nID 集合への参照を格納する．経路式をエンコード/デコードする HPMD(pHPMD という）により，エンコードされた pID を path RDT で検索して得られた nID をキーとして node RDT で親子，兄弟などの構造検索を行う．

また，構造式をエンコード/デコードす HPMD(nHPMD という）により，5.1 節に従って分割エンコードされた nID をキーとして node RDT を検索して ETF への参照を得てから ETF を逐次検索して，対応する pID を得る．相互に対応する pID と nID はこのように二つの RDT により相互参照することができる．



(a)　pID 索引　　(b)　nID 索引

図 10　　pID と nID の相互参照

Figure 10　Cross referencing between pID and nID

6. 分割点の決定と再構成

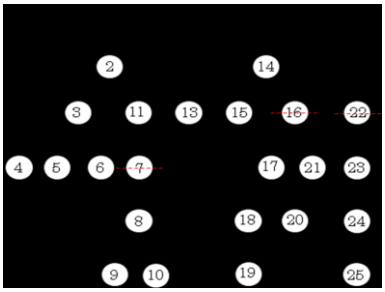
6.1 初期分割点の決定

分割点集合を決定するために，木グラフ中の各ノードについて子孫数をカウントする必要がある．第一段階として，本システムでは，pre-order 順に木グラフデータが格納された入力ファイルを逐次読み込みながら，分割せずにそのまま単一の逐次ファイル F に各ノードの経歴・パターンを格納していく．同時に，別途，葉ノードのみ取り出して，その経歴・パターンを葉ノード B+木に格納する．木グラフの葉ノード B+木のキーとしての経歴・パターンの大小比較は，経歴・パターンを添字座標にデコードしたものについて行う．

最後まで入力ファイルを読み終えたら，葉ノード B+木のシーケンスセットから経歴・パターンを逐次取り出しながら，各ノードについてその子孫数をボトムアップで計算し，子孫数表を作成し，子孫数の多い順にソートする．この子孫数表を元に分割点を決定し，上記逐次ファイル F より図 11 で示した木グラフの初期分割を行う．初期分割にあたっては，F 中の経歴・パターンのうち，経路に分割点を含まないものを RDT に，分割点を含むものを部分木ごとに用意した ETF に格納する．この時 5.1 節に示した構造式の分割アルゴリズムを用いて，記憶コストの削減を行う．

6.2 木グラフの成長に伴う再構成

初期分割後の木グラフの動的成長により部分木の記憶負荷が再びアンバランスになることが考えられる．そのためにこのアンバランスを検知して木グラフを再構成する必要がある．新たなノードの追加，もしくはノードの削除の際は葉ノード B+木の更新を行う．木グラフの構築の際に，初期ノード数の何%の変更が行われたら再構成を行うかを決めておき，指定数の変更があれば再構成を行う．この時，初期分割とは異なり，RDT の一部，再編成の対象になっている ETF のみ再構成すればよい．再構成に際しては，木グラフの任意のノードについてその子孫数を記録するテーブルを作成するが，このため，葉ノードのみの経歴・パターンを格納する B+木をメンテナンスしている．この B+木のシーケンスセットからその子孫数表をボトムアップに作成している．



nID	hp_1	hp_{14}	hp_2	hp_3	hp_{16}	hp_7	hp_{22}	・・・
子孫数	24	11	9	8	5	3	3	・・・

図 11　子孫数表と初期分割

Figure 11　Number of descendents table and the initial partition

7. 水平分割

本節では，本システムの比較評価用に構築したシステムについて述べる．このシステムは木グラフを経歴・パターン法を使用してエンコードした結果を格納する ETF を構造木のレベルごとに水平分割してそれぞれのレベルの ETF に格納する方式である(図 12)．このレベルによって検索対象の ETF を絞り込むことができる．

水平分割においても 3.2 節のように木グラフを経路木と構造木に分け，それぞれのノードを経歴・パターン法でエンコードする．エンコード結果の nID とそれに対応する pID の対を ETF に格納する．水平分割では 4 節で述べたようなメタ情報の必要なしに，葉ノードについて自身のレベル以降のパターン(冗長な 0)を省略して当該の ETF に格納できる．また，水平分割では，構造式の座標の次元数(レベル)によって探索対象の ETF が絞り込めるので，記憶コストとのトレードオフを考慮して，図 10 の左側の pID 索引は保

持していない.

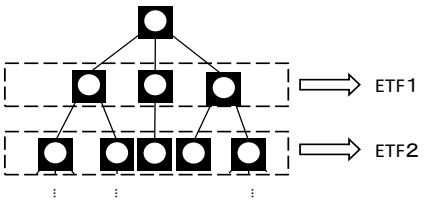


図 12 水平分割
Figure 12 Horizontal partition

8. 評価

本システムの性能評価として，本システムの垂直分割に対し，7 節で述べたレベルごとに ETF を用意し格納する水平分割システム[9]との比較評価を行った．評価に用いた計算環境を以下に示す.

CPU: Intel Core i7 920(2.67GHz) RAM:48GB,
OS: Cent OS 5.7

評価に使用した入力の木グラフは XMark[10]により生成した XML ファイルからテキストデータと属性を除去して得た次のようなファイルである.

ファイルサイズ:332,807,472[B],
ノード数: 16,703,210, 最大レベル:12

表 1 にデータベースの構築サイズと構築時間の測定結果を示す．ただし，本システムの構築時間は分割点の決定に要する時間は除外している．両システムとも経歴・パターン法によるエンコードを用いることにより低記憶コストで上記の木グラフを格納することができた．水平分割では図 10 における ETF をレベルごとに保有しているのみであるのに対して，本システムでは構造検索用，経路式検索用にそれぞれ HPMD を構築しているため，構築サイズは増加している．また，特にこれらの HPMD において，B+木の構築時間コストが高いために，構築時間は水平分割に比べて 5.7 倍程度となっている.

表 1 構築サイズ，構築時間
Table 1 Storage cost and construction time

	構築サイズ[MB]	構築時間[s]
本システム	266.8	57.45
水平分割	156.3	9.91

データベースの構築後，全ノードの 1%をランダムに抽出し，各種構造検索を行った結果を図 13 に示す．また，構造検索で抽出したノードの経路式を用いて経路式検索を行い，平均検索時間を測定した結果を表 3 に示す．構造検索では垂直分割により，構造に沿った検索と検索範囲の限定が効率的に行えることにより，本システムは水平分割に比べて有利になっている． また，経路式検索については，本システムでは経路式検索用の HPMD を構築しているため，水平分割に比べて非常に高速に検索できている.

9. むすび

本研究では負荷バランスを均一にするために，構造木を RDT, ETF とで等しく分担することを目指したが, 実際は部分木以降のノードは全体の 27%にしかならなかった．複数の部分木をまとめて 1 つの ETF に格納する等の工夫が必要である．また，動的再構成の方式を示したが，今後，再構成のコスト評価とよりよい再構成アルゴリズムを検討する必要がある.

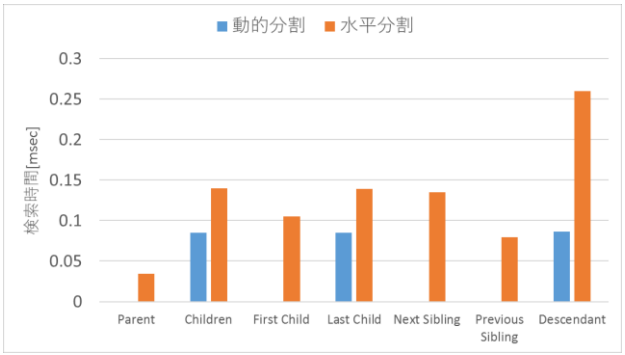


図 13 構造検索時間
Figure 13 Retrieval times for structural search

表 2 経路式検索時間
Table 2 Retrieval time for path expression

	平均検索時間[ms]
本システム	0.0023
水平分割	0.132

謝辞

本研究はJSPS科研費26330132の助成を受けたものである.

参考文献

[1]Extensible Markup Language: http://www.w3.org/XML/
[2]城戸，天笠，北川，PC クラスタを用いた XML データ処理方式の提案，Proc. of DEWS2006, 6B-oi3, 2006
[3]蒲原，上島，空間索引木を用いた範囲問合せ手法，情報処理学会論文誌：データベース,Vol.5, No.1, pp.1-17, 2012.
[4]O'Neil, P.E., O'Neil, E.J., Pal, S., Cseri, I., Schaller, G., Westbury, N.: ORDPATHs: Insert-friendly XML node labels, Proc. of the ACM SIGMOD, pp. 903-908, 2004.
[5]Li, C., Ling, T.W, : QED: a novel quaternary en-coding to completely avoid re-labeling in XML updates, Proc. of CIKM'05, pp. 501-508, 2005.
[6]Goldman, R., Widom, J., Dataguides: Enabling query formulation and optimization in semistrucutured databases, Proc. of VLDB, pp. 436-445, 1997.
[7]Li B., Kawaguchi K., Tsuji T., Higuchi K.: A Labeling Scheme for Dynamic XML Trees Based on History-offset Encoding, 情報処理学会論文誌：データベース (TOD), Vol.3, No.1, pp.1-17, 2010.
[8]Makino, M., Tsuji, T., Higuchi, K. : History-pattern encoding for large-scale dynamic multidimensional datasets and its evaluations, IEICE Transactions Vol.E99-D, No.4, 2016. (to appear)
[9]渡部，都司，樋口，：大規模木構造グラフの効率的な格納・検索方式，電気関係学会北陸支部連合大会予稿集，F-3, 2012.
[10]XMark- An XML Benchmark Project http://www.xml-benchmark.org/