DBSJ Journal, Vol.8, No.3
December 2009

動的多次元データセットのコンパクトな実現方式の提案

A Proposal of a Compact Realization Scheme for Dynamic Multidimensional Datasets

都司 達夫♥ 水野 広治♦
松本 宗純♠ 樋口 健♣

Tatsuo TSUJI Hiroharu MIZUNO
Muneyoshi MATSUMOTO Ken HIGUCHI

本論文では関係テーブルに代表される単純型データのタプルの集合としての多次元データセットをコンパクトに記憶装置上に実現し，検索する方式を提案する．本方式は拡張可能配列の柔軟な拡張性に基づいており，現在我々が提案している経歴･オフセット法による実現手法と長所短所が相反しており，相補的である．本文では本方式の概要について述べる．

This paper proposes a new realization scheme of a multidimensional dataset which is a set of tuples whose element is of simple data type. The scheme is based on the flexible extendibility of an extendible array, and its advantages and disadvantages are mutually contradictory and complementary with the history-offset implementation scheme we are now developing. The outline of the new scheme will be described in this paper..

1. はじめに

関係テーブルのレコードに代表されるような複数の単純型データの組(タプル)からなるデータを，一般的に多次元データという．多次元データを処理対象とする情報処理分野は元来，極めて多岐に渡っているが，近年多次元情報処理の応用分野の拡大につれて，急激にその重要性を増大している．

応用分野として，多次元データベースを核とするデータウエアハウスや多次元オンライン分析（MOLAP）(たとえば[10])、データマイニング(たとえば[11])をはじめ，Web 空間の構造解析（たとえば[12]），移動オブジェクトの管理（たとえば[13]），マルチメディアデータ，文書データ，時系列データなどの特徴量のモデル化や類似検索（たとえば，[14]）などの分野において，きわめて活発に研究されている．

これらの応用分野の多くは，特に，データの大規模化への対応とともにシステム運用時の動的なデータ量増大や構造変化に効率よく対処するためのシンプルなスキームを必要としており、それに基づく汎用性の高い動的多次元データの処理基盤技術の開発が強く要請されている．

一般に n 次元データのタプルは各次元の属性値の定義域を多次元配列の添字に変換するデータ構造により，多次元配列中の一要素の n 次元座標 $(i_1,i_2,...,i_n)$ に一対一に変換することができる．さらに，この n 次元座標から当該配列のアドレス関数により，その要素の位置を表すオフセットを求めることができ，逆にこのオフセットから，一意に元の n 次元座標にデコードすることができる．すなわち，元の n 次元タプルはこのようにして求めたオフセットとして取り扱うことができる．しかし，あらかじめ各次元の属性のカージナリティが既知である場合にのみ有効であり，新たな属性値の出現に対しては，より大きい新たな多次元配列を確保して，ほとんどすべての既存要素を新しいアドレス関数にしたがって，この新たな配列に再配置する必要がある．

動的に増大する多次元データのエンコード方式として，我々が最近提案している経歴･オフセット法[7][8]は時間･空間効率ともに優れた方式である．経歴･オフセット法は拡張可能配列[3][6]の仕組みを取り入れることにより，固定サイズ配列のこのような欠点を回避している．[3][6]とは異なり，メモリの実アドレス参照を排除したより論理的なエンコード方式である．また，多次元データの上記の配列表現において問題となる過疎性の問題を解消している．

挿入したタプル中に新たな属性値が出現した場合，そのタプルを収容する部分配列が動的に確保され，現在の拡張可能配列の当該次元方向に付加される．経歴･オフセット法ではタプルは，拡張可能配列中，それが含まれる部分配列の拡張経歴値とその部分配列におけるオフセットの対にエンコードされる．この方式の基本的な利点は次の3点である．

(1) タプルの次元数nに関わらず，経歴値とオフセットの2つのスカラー値でタプルを表現でき，システム内部でコンパクトな固定長でタプルを扱える．
(2) 単純な四則演算でタプルのエンコード・デコードが可能．
(3) タプルの動的な追加・削除に対して，タプル集合全体を再配置することなく，柔軟に対応し得る．

本稿では，これらの利点を活かしつつ，経歴･オフセット法におけるタプルアクセス速度と記憶効率を基本レベルで向上させるための新たな方式を提案する．この方式では拡張可能配列のアドレス関数計算を行うことなく，シフトやマスク演算等のレジスタ命令のみでより高速にタプルのエンコードとデコードが可能である．また，必要な記憶量を削減することができる．ただし，従来の経歴･オフセット法に比べて，エンコード結果を収容する論理空間の飽和をより，早めてしまうことが本方式の欠点である．この欠点を克服する必要があるが，経歴・オフセット法と相補的に棲み分けを行う．以降では，本稿で提案する多次元データのエンコード方式とその実装方式を，それぞれ，経歴･パターン法およびHPMDと呼ぶが，これらは動的に変化する多次元情報のための汎用的な処理基盤システムを提供し得ると考える．

2. 経歴･オフセット法

ここでは，多次元データの代表例として関係テーブルを取り上げ，筆者らが従来提案している多次元データのエンコード方式である経歴･オフセット法について概説する．詳しくは[7][8]を参照されたい．

♥ 正会員 福井大学大学院工学研究科情報・メディア工学専攻 tsuji@pear.fuis.fukui-u.ac.jp
♦ 非会員 福井大学工学部 mizuno@fuis.fukui-u.ac.jp
♠ 学生会員 福井大学大学院工学研究科情報・メディア工学専攻博士前期課程 matumoto@pear.fuis.fukui-u.ac.jp
♣ 正会員 福井大学大学院工学研究科情報・メディア工学専攻 higuchi@pear.fuis.fukui-u.ac.jp

2.1 拡張可能配列

プログラミング言語で用いられる従来の固定サイズ配列は，実行時に動的に配列の各次元サイズの変更を行うことができない[1]．これに対して，拡張可能配列[3][6]では，配列サイズの動的変更に対して，すでに確保している領域はそのまま使用し，新たな領域を差分領域として，必要な分だけ付加することができる(図 1)．以下では，メモリの動的割り付け環境にアダプトした拡張可能配列の実現モデル[6]に基づいて述べる．

一般に n 次元拡張可能配列は，n-1 次元の固定サイズの部分配列の集合で表される．拡張可能配列では，各部分配列の先頭アドレスおよび，部分配列が何番目の拡張の際に確保されたかを表す経歴値をそれぞれ，アドレステーブル（図 1 では L_1, L_2），経歴値テーブル（H_1, H_2）に格納している．また，部分配列の要素へ高速にアクセスできるように，n が 3 以上の場合には，各部分配列のアドレス関数を計算するための n-2 個の係数値を各部分配列ごとに係数ベクトルとして係数テーブルに記録している．n が 2 の時には，係数テーブルを持つ必要はない．2 次元拡張可能配列の場合，部分配列は 1 次元であり，座標 (i_1, i_2) の部分配列内オフセットは添字 i_1 または i_2 である．図 1 では 2 次元拡張可能配列を第 1 次元方向にサイズを一つ拡張している．第 2 次元方向のサイズが 3 であるので，サイズ 3 の一次元部分配列を確保する．次に，現在の経歴値を記録している経歴値カウンタを 1 つカウントアップして，新たに確保した部分配列の経歴値とする．また，新たに確保した部分配列の先頭アドレスを拡張次元方向のアドレステーブルに記録する．

要素 $(i_1, i_2, \ldots, i_n)$ が与えられたとき，各次元の添字に対応する経歴値の内，最大経歴値の次元を m とすると，次元 m のアドレステーブルの i_m番目の先頭アドレスの部分配列が，要素を含む．その部分配列の係数ベクトルを知って，要素のアドレスが計算できる．たとえば，図 1 の (3,2) 要素の場合，$H_1[3] > H_2[2]$ であるから，この要素は経歴値 5 の部分配列に含まれ，その先頭番地は $L_1[3]$である．したがって，求めるアドレスは $L_1[3]+2=78$ と計算される．このように補助テーブルのわずかな記憶コストの増加のみで，シンプルな方法で動的に拡張が可能な配列の要素のアドレス計算が可能であることは注目に値する．

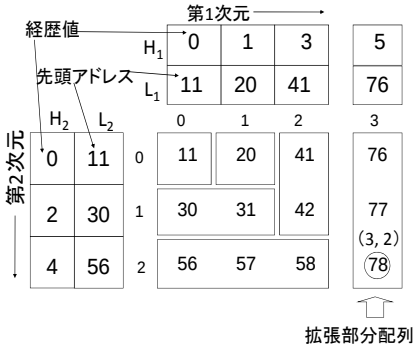


図 1　拡張可能配列

2.2 拡張配列を用いた多次元データの表現

関係テーブルの各属性を従来の多次元配列の各次元に，属性値を配列の添字に対応付けることによって，関係テーブルのタプルを，配列要素の位置で表現することができる．2.1 節で述べた，拡張可能配列を用いれば，関係テーブルに対して，動的なタプルの追加に対応することができる．図 2 に拡張可能配列を用いた，関係テーブルの表現例を示す．

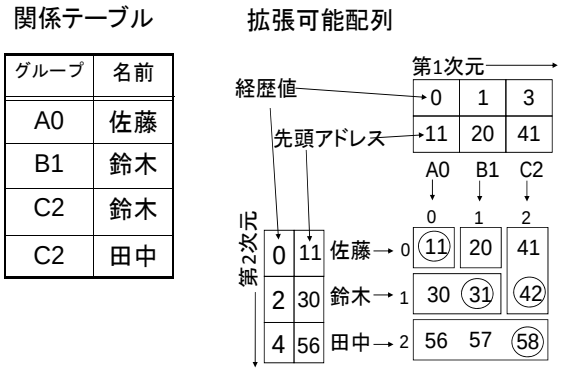
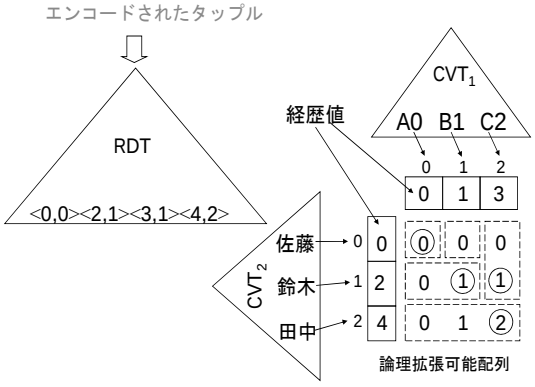


図 2　拡張可能配列を用いた関係テーブルの表現

2.3 経歴･オフセット法を用いた多次元データの実装

2.2 節で述べたように，多次元配列を用いて関係テーブルを表現することは可能であるが，通常，関係テーブルのタプルに対応しない配列要素が多く存在してしまうため，使用しない配列要素を多くメモリ上に確保しなければならないという過疎性の問題がある．そこで，タプルに対応しない無駄な配列要素を保持しないために，配列要素について，それが属する部分配列の経歴値とその部分配列内のオフセットを配列要素の位置情報として保持する．この位置情報は対応するタプルそのものを表すことになる．このような多次元データセットのエンコーディングの方法を，経歴･オフセット法(history-offset encoding)と呼ぶ．この方法では，配列の次元数 n に依存することなく，小さな固定長で n 次元タプルを表現することができる．図 3 に，経歴･オフセット法を用いた図 2 の関係テーブルの実装を示す．

図 3 では，関係テーブルの属性値から配列の添字に高速に変換できるよう，配列の各次元において添字変換用の B^+木（CVT : key-subscript Conversion Tree と呼ぶ）を利用している．また，経歴値とオフセットの対を B^+木（RDT : Real Data Tree と呼ぶ）を用いて管理している．このように，経歴･オフセット法を用いて多次元データを実装するための方式および，そのデータ構造を HOMD (History-Offset implementation for Multidimensional Datasets)と呼んでいる．



[1] Java や C++では一次元配列についてのみ可変長配列（ベクトル）が使用できる．

図 3　HOMD による関係テーブルの実装例

3. 経歴･パターン法とその実装

3.1 経歴･パターン法

前節で述べた経歴･オフセット法では n 次元タプル(n≧3)から配列要素のオフセットへの変換には，n-2 回の乗算と加算を必要とする．また，逆変換には 2(n-2)回の除算を必要とし，この計算コストは高い．そこで，このオフセットへの変換，逆変換のコストを削減可能なエンコード法を以下に提案する．

一般に，0 以上の整数 m の表現に要するビット数を $b(m)$ とする．経歴パターン法における拡張可能配列の各次元サイズには論理サイズと実サイズの 2 種類がある（図 5 参照）．実際に存在するタプルの次元 k の添字の内,最大添字を m(添字は 0 から始まるとする）とすると，次元 k の実サイズは m+1 であり，これは次元 k の属性のカージナリティである．また，論理サイズは $2^{b(m)}$である．ここで，n 次元拡張可能配列 A の現在の各次元の実サイズを $[s_1, s_2, ..., s_n]$ として経歴値カウンタの値を h とする．$<b(s_1), b(s_2), ..., b(s_n)>$ を経歴値 h における A の境界ベクトルと呼ぶ．新たなタプルの挿入により，ある次元 k（1≦k≦n）の方向に A のサイズが 1 つ拡張して，$b(s_k) < b(s_k+1)$となったとき A は次元 k の方向に拡張される．拡張分の部分配列 ΔA の各次元のサイズは元の A と同一であり，拡張後の配列の実サイズは

$[s_1, ..., s_{k-1}, s_k+1, s_{k+1}, ..., s_n]$ となる，また，論理サイズは $[2^{b(s_1)}, ..., 2^{b(s_{k-1})}, 2^{b(s_k+1)}, 2^{b(s_{k+1})}, ..., 2^{b(s_n)}]$ となる．このとき，現在の経歴値カウンタの値が 1 インクリメントされ，その値 h+1 が次元 k の経歴値テーブル H_k に格納される．また，この新たな経歴値における ΔA の境界ベクトル V は，

$V = <b(s_1), ..., b(s_{k-1}), b(s_k)+1, b(s_{k+1}), ..., b(s_n)>$

となり，経歴値 h+1 の下に，V が拡張次元 k とともに境界ベクトルテーブル B に記録される（図 4）．

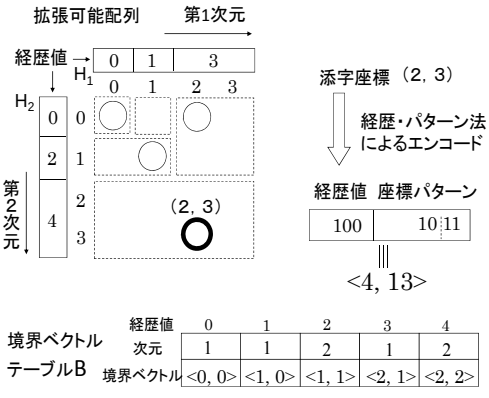


図 4　経歴･パターン法の説明図

配列の要素位置を指定する n 次元座標 $I = (i_1, i_2, ..., i_n)$ は以下のように経歴値 h と座標パターン p の対にエンコードされる．境界ベクトル V はこのエンコードに使用されるとともにエンコードされた対を元の n 次元座標にデコードするのにも使用される．経歴値 h は I の各次元座標のビットパターンサイズ $[b(i_1), b(i_2), ..., b(i_n)]$ について，各次元の経歴値テーブル H_k を参照し，$H_k[b(i_k)]$（1≦k≦n）の最大値を求めればよい．この h に対して，$B[h]$ の境界ベクトルは各次元の添字のビットパターンのサイズを与える．このサイズにしたがって，座標パターン p は次元の高い添字パターンから座標パターン格納域 (たとえば 1 long 値長 64 ビット) の下位ビットから順に配置して得られる．このエンコード結果は RDT に格納される．なお，経歴値は座標パターンそのもののサイズを表していることに注意されたい．

例えば，図 4 の拡張可能配列において，配列要素(2,3)をエンコードする場合, (2,3) が属している部分配列の経歴値は 4 であり，B[4]を参照して，境界ベクトルは<2,2>であることを知る．よって座標パターンは最上位ビットから 2 ビットで 1 次元目の 2，次の 2 ビットで 2 次元目の 3 の添字パターンの連接となる．したがって，経歴値とこのパターンとを対とした $<4,1011_2>$ にエンコードされる．例えば，この対の格納サイズを 16 ビットの short 型とする場合，経歴値より座標パターンのほうが多数のビットを使うので上位 4 ビットを経歴値, 下位 12 ビットを座標パターンとすることが考えられる．

逆に, エンコード結果 ＜経歴値 h, 座標パターン p＞ から元の n 次元座標 $I = (i_1, i_2, ..., i_n)$ へのデコードは，まず，$B[h]$より要素 I が属する部分配列の次元 k を求める．つづいて，$B[h]$の境界ベクトルにしたがって p 中の各次元の座標値 i_k（1≦k≦n）を分離して取り出せばよい．

3.2 経歴パターン法の実装（HPMD）

n 次元 HPMD (History-Pattern implementation of Multidimensional Datasets) は経歴･パターン法の実装方式であり，以下のデータ構造から構成される．

(1) n 個の CVT_i（1≦i≦n）と RDT の n+1 個の B+木. CVT_i は次元 i の属性値をキーとして，対応する拡張可能配列の添字を記録する．また，RDT のキーとして経歴･パターン法によるタプルのエンコード値が格納される．

(2) 各次元毎に経歴値を格納する n 個の経歴値テーブル H_i（1≦i≦n），および，経歴値を添字として，経歴値に対応する拡張次元と境界ベクトルを格納した境界ベクトルテーブル B.

(3) 各次元の添字ごとにその添字に対応する属性値およびその属性値を持つすべてのタプルの数を記録する属性値テーブル C_i（1≦i≦n）.

例として，表 1 に 2 次元タプルが順次，追加されたときの当該タプルが属する所属部分配列の経歴値，座標パターン，境界ベクトルを示し, 図 5 に表 1 に対応する(1), (2), (3)の HPMD の基本データ構造を示す．

[例 1]　HPMD によるエンコード/デコード例

表 1 および図 5 において，タプル $<e, s>$ の場合を例として，元のタプルデータからのエンコード/デコード例を示す．$CVT_1(e)$, $CVT_2(s)$を検索してタプルの座標 (4,5) を引当てる．$4=100_2$, $5=101_2$ であり，$b(4)=3$, $b(5)=3$ である．$H_1[3]=4$ と $H_2[3]=6$ を比較して，$H_1[3] < H_2[3]$．したがって，(4,5)は経歴値 6 の部分配列に含まれる．B[6]の境界ベクトルは<3,3>であり，(4,5) のエンコードは $<6,100.101_2> = <6,37>$ となる．座標パターン 100.101_2 の ‘.’ は各次元の座標値パターンの境界位置を表す．逆にこのエンコードより，境界ベクトルテーブル B を参照して，経歴値 6 の部分配列の所属次元 2 と境界ベクトル <3,3> を引当てる．これより，$37 = 100101_2$ の上位 3 ビット分が 1 次元目の添字，下位 3 ビット分が 2 次元目の添字になることがわかる．したがって，$<6,100.101_2>$と境界を定めることができ，(4,5) にデコードされる．このとき，$C_1[4] = e$, $C_2[5] = s$ より，元のタプル $<e, s>$ が得られる．

配列の拡張に対して柔軟に効率よく対応できる2節で述べた拡張可能配列の仕組みは，各次元の添字のビットパターンサイズを一律に固定することなく，各次元の座標値のビットパターンの境界を次元拡張の状況に応じて，柔軟に設定していることに活かされているといえる．ただし，各次元の添字座標値のビットパターンのサイズの総和は定められた固定長(たとえば1 long 値長 64 ビット)以下である．このような，境界設定の柔軟性を確保した上で，機械語のレジスタ命令のみで，エンコードとデコードが可能である．すなわち，エンコード時の各次元の添字パターンの連接およびデコード時の各次元添字の取り出しは，シフト演算および論理和，論理積演算のレジスタ命令のみの実行により可能であり，経歴・オフセット法のように乗除算を含むアドレス関数の計算は不要である．このことはタプルのアクセス速度および検索速度が HOMD よりも高速であることを意味している．

2.1 節で説明した拡張可能配列の係数ベクトルテーブルの記憶コストは $O(n^2)$であった．本方式の境界ベクトルテーブルはこの係数ベクトルテーブルに対応するがその記憶コストは $O(n)$となり，大幅に削減できる．また，各次元の経歴値テーブルサイズは各次元長を$[s_1, s_2, \ldots\ldots, s_n]$とすると，2.1 節の方式では，$s_1+s_2+\ldots\ldots+s_n$であるのに対して，

$$\lceil \log_2 s_1 \rceil + \lceil \log_2 s_2 \rceil + \cdots\cdots + \lceil \log_2 s_n \rceil$$

となる．以上より，経歴値テーブルの記憶コストは CVT, RDT および属性値テーブルなどの HPMD の他の構成要素に比べて，ほとんど無視できる．また，経歴値データのサイズは経歴オフセット法の場合より，かなり小さい．

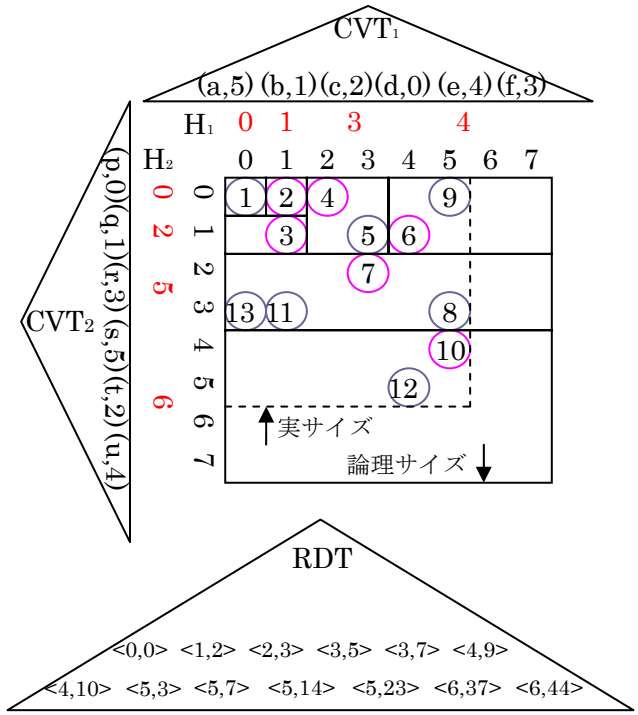
属性値テーブルの属性値はタプルへのデコードと検索に必要であり，またタプル数は検索の最適化プランを立てるのに必要とされる．2.3 節で述べた HOMD と同様 RDT は多次元データセットに実際に登録されているタプルのみのエンコード値を格納しており，登録されていないタプルは記憶域を占めない．これにより，疎配列性の問題を解消している，

図 5 の拡張可能配列の論理サイズは [8,8] であり，また，各次元のカージナリティはそれぞれ 6, 6 であるから，実サイズは [6,6] である（図 5 の拡張可能配列中，点線の範囲）．なお，HOMD の場合には，拡張可能配列の論理サイズと実サイズは常に一致している（図 7 参照）．経歴・パターン法では，n 次元データセットの場合，実サイズ空間の論理サイズ空間に対する大きさの割合は $1/2^n$ ～1 の範囲である．

表 1　2 次元タプル集合とそのエンコード

追加順	タプル	経歴値 h	座標パターン p	境界ベクトル b
1	<d, p>	0	.	<0, 0>
2	<b, p>	1	1.	<1, 0>○
3	<b, q>	2	1.1	<1, 1>○
4	<c, q>	3	10.1	<2, 1>○
5	<f, q>	3	11.1	<2, 1>
6	<e, q>	4	100.1	<3, 1>○
7	<f, t>	5	011.10	<3, 2>○
8	<a, r>	5	101.11	<3, 2>
9	<a, p>	4	101.0	<3, 1>
10	<a, u>	6	101.100	<3, 3>○
11	<b, r>	5	001.11	<3, 2>
12	<e, s>	6	100.101	<3, 3>
13	<d, r>	5	000.11	<3, 2>

(○はそのタプルの挿入により拡張可能配列の論理サイズの拡張が起こることを示す.)



H₁(次元 1 の経歴値テーブル)

0	1	2	3
0	1	3	4

H₂(次元 2 の経歴値テーブル)

0	1	2	3
0	2	5	6

B(境界ベクトルテーブル)

経歴値	0	1	2	3	4	5	6
次元	1	1	2	1	1	2	2
境界ベクトル	<0,0>	<1,0>	<1,1>	<2,1>	<3,1>	<3,2>	<3,3>

C₁(次元 1 の属性値テーブル)

	0	1	2	3	4	5
属性値	d	b	c	f	e	a
タプル数	2	3	1	2	2	3

C₂(次元 2 の属性値テーブル)

	0	1	2	3	4	5
属性値	p	q	t	r	u	s
タプル数	4	3	1	3	1	1

図 5　HPMD の基本データ構造

3.3 HPMD における検索

n 次元の HPMD について，あるタプルの次元 i の添字を k とする．次元 i の添字 k に対する基部分配列とは次元 i 以外の添字はすべて 0 の配列要素(0, 0, …, 0, k, 0, …, 0) を含む部分配列である．このとき，次の顕著な性質がある．

［性質 1］タプルの次元 i の添字 k が指定されたとき，その

基部分配列の経歴値を h とする．次元 i の添字が k であるタプルはその基部分配列および，h より大きい経歴値の部分配列で次元が i でない部分配列にのみ存在する．この性質を図 5 の 2 次元 HPMD について，図 6 に示す．点線は拡張可能配列の実サイズを表す．

HPMD における検索では一般に検索対象の部分配列中のタプルは全検索を行うが，検索範囲が肥大化し，無駄な検索を多く行う必要がある(図 6)．また，図 6 に見るように検索対象の属性の次元や属性値が所属する部分配列の経歴値によって，検索範囲が大きく異なる．この依存性をそれぞれ，次元依存性，経歴値依存性と呼ぶ．このような依存性は，図 7 に見るように HOMD の場合より，HPMD の場合の方がより強い．

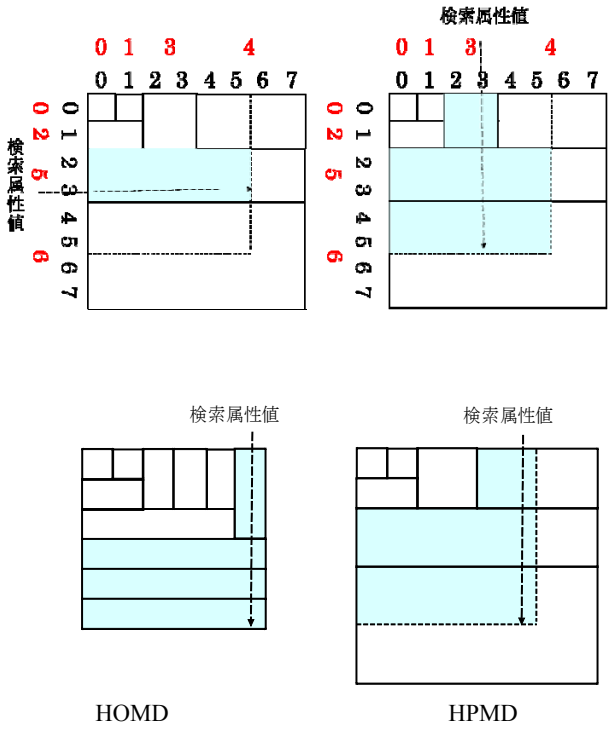


図 7　HOMD と HPMD における検索範囲

4. チャンク化 HPMD（C-HPMD）

本節では，3.3 節で述べた HPMD における検索範囲の肥大化の問題点を改善するため，[8]と同様に HPMD のチャンク化を行い，これを C-HPMD と呼ぶ．C-HPMD では 3.3 節で述べた，次元依存性や経歴値依存性を同時に軽減しつつ拡張可能配列のランダムアクセス性を活かした検索が可能である．また，タプルの論理空間を格段に拡げて，より，大規模な多次元データセットを収容することができる．

n 次元の C-HPMD の拡張可能配列において，チャンクとは 1 辺が 2^r（r≧1）の n 次元超立方体であり，r をチャンクのランクという．HPMD は配列要素からなる部分配列を単位として拡張されるのに対して C-HPMD はチャンクからなる部分配列を単位として拡張される．HPMD におけるタプルのエンコードは<経歴値，座標パターン>の対であったのに対して，C-HPMD ではタプルは<チャンク番号，チャンク内座標パターン>の対にエンコードされる．チャンク番号の決定には 2.1 節述べた拡張可能配列の要素アドレスの計算方法が使用される．図 8 にランクが 1 のチャンク化した 2 次元拡張可能配列の例を示す．配列の中の数字はチャンク番号を表す．HPMD の各配列要素をチャンクに対応させると C-HPMD は HPMD とほぼ同等のデータ構造である．また，HPMD に関する 3 節の説明はタプルのエンコード/デコード方式と検索方式以外はほぼそのまま C-HPMD にも適用できる．マシン語長が 64 ビットの場合，C-HPMD では，チャンク内座標パターンとして 64 ビット，チャンク番号として 64 ビットの計 128 ビットの空間を確保でき，元のタプルの論理空間を格段に拡げることができる．

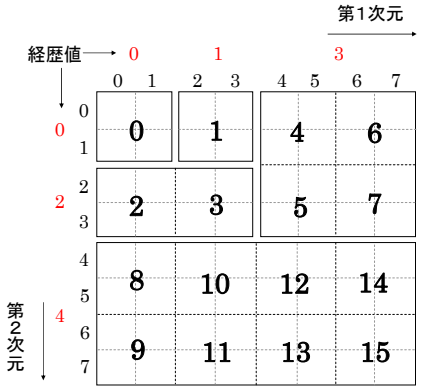


図 8

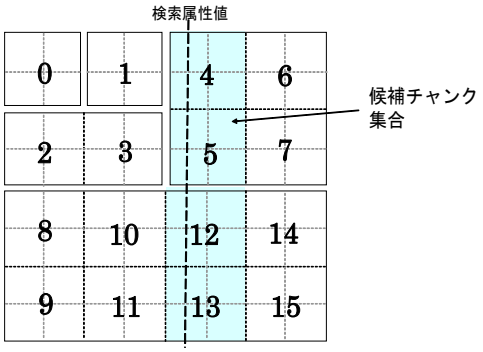


図 9　C-HPMD における検索範囲

5. C-HPMD における検索

以下，ユーザ指定の次元（検索次元）の属性値（検索属性値）の検索の概要を述べる．まず，検索次元の CVT について，検索属性値を検索する．なければ，該当タプルが存在しない．あれば，得た添字から検索次元における検索属性値のチャンク座標とチャンク内座標を求める．チャンク座標は得た添字をチャンクのランク値分のビット数だけ右シフトすることにより求められる．チャンク内座標は得た添字をランク値分のビット数だけオールビット 1 の値でマスクすればよい．次に検索属性値が所属しているチャンク部分配列の経歴値を得る．この経歴値を持つチャンク部分配列の中から検索条件に該当するタプルが含まれている可能性のあるチャンク（以下“候補チャンク”）のチャンク番号を計算する．候補チャンクの番号を一つ求める度にそのチャンク番号を持つ RDT のキーを検索し，検索属性値の検索次元におけるチャンク内座標が合致するタプルを論理積演算し，検索結果として取り出せばよい．RDT のキーはチャンク番号，チャンク内座標パターンの順に上位ビットから配置されるので，RDT

のシーケンスセット上ではチャンク番号でソートされている．したがって，同じチャンク内のタプルはシーケンスセット上で連続して配置されている．

基チャンク部分配列の検索が終わったら，3.3 節の［性質1］により，基チャンク部分配列よりも経歴値が大きく，かつ拡張方向が検索次元でないチャンク部分配列について，同じように候補チャンクを計算し，チャンク内を全検索する．

このような検索方法により検索範囲を図9のように大幅に限定できると同時に，次元依存性および経歴値依存性を軽減できる．

6. 関連研究

多次元データの実現方式について，現在まで，種々のアプローチにより膨大な研究がなされているが，ここでは，まず，本研究におけるような拡張可能配列に基づいた研究について，2,3 紹介する．Rosenberg[2]ではハッシュ技法に基づいた，また，Otoo[3]ではインデックス配列に基づいた拡張可能配列の実現方式を提案している．筆者等は拡張可能配列について時間・空間コストの改善方式を提案している[6]，Otto[3][4], Rotem[5] 等の研究は，高密度の拡張可能配列の二次記憶装置上の表現を論じているが，タプルエンコードや圧縮の考えには至っていない．経歴・オフセット法の応用として，[9]では，データキューブの差分構築方式を，また，[15]では XML 木のラベル付けの方式を提案している．

また，多次元データの圧縮格納方式としては，大規模多次元配列を扱う MOLAP では，chunk-offset 圧縮[1]と呼ぶ方法が，多く用いられるが，各次元サイズが固定であるために，新たなカラム値の出現に対して，柔軟に対処できない．

さらに，応用として，たとえば，画像や映像の内容に基づく検索では，データの特徴量を表現するベクトルは 100 次元を超えることもあり[14]，データマイニングにおいても高次元データの効率的な処理が重要になっている[11]．HPMD のチャンク化によって，タプルの存在空間を拡げることができるが，なお十分ではない．[8]ではタップル集合としてのテーブルを垂直に分割し，より規模の小さい複数の HOMD に分割することにより，この問題を扱っている．

7. おわりに

本論文では多次元データセットをコンパクトに記憶装置上に実現し，検索する新たな方式を提案した．今後，実装を完成させて，本方式の評価を行いたい．高次元データの扱いに関連して，特に，動的にデータ量や構造が変化し得る環境において，高次元データの表現と処理の方式を本研究のアプローチにおいて検討することは今後の課題の一つである．また，HOMD や HPMD は多次元データ処理の多岐に渡る応用分野において，利用可能な汎用基盤を提供し得るが，応用分野固有の動的特性を反映した構造チューニングも今後の興味ある課題である．

[謝辞]

本研究の一部は科学研究費補助金基盤研究(課題番号 8200005) による．

[文献]

[1] Zhao, Y．, Deshpande, P． M． and Naughton, J．F． : An array based algorithm for simultaneous multidimensional aggregate, Proc of the ACM SIGMOD Conference, pp. 159-170, 1997.

[2] A.L.Rosenberg, L.J.Stockmeyer, "Hashing Schemes for Extendible Array's", JACM, Vol.24, pp.199-121,1977.

[3] E. J. Otoo, T. H. Merrett, “A Storage Scheme for Extendible Arrays”, Computing, Vol.31, pp.1-9, 1983.

[4] E．J．Otoo, D． Rotem, A Storage Scheme for multidimensional databases using extendible files, Proc. of STDBM2006, pp.67-76, 2006.

[5] D. Rotem, J. L. Zhao, “Extendible Arrays for Statistical Databases and OLAP Applications”, Proc. of SSDBM1996, 1996.

[6] 都司達夫，水野剛，宝珍輝尚，樋口健，“拡張可能配列の遅延割付け方式”，電子情報通信学会論文誌 D-I, Vol. J86-D-I, No.5, pp.351-356 , 2003.

[7] K. M. A. Hasan, T. Tsuji, K. Higuchi, “An Efficient Implementation for MOLAP Basic Data Structure and Its Evaluation”, Proc. of DASFAA2007, pp. 288-299, 2007.

[8] T．Tsuji, M． Kuroda, K． Higuchi, “History offset implementation scheme for large scale multidimensional data sets”, Proc. of SAC2008, pp.1021-1028, 2008.

[9] D. Jin, T. Tsuji, T. Tsuchida, K. Higuch, “An Incremental Maintenance Scheme of Data Cubes”, Proc. of DASFAA 2008, pp.172-187, 2008.

[10] R. Zhang, P. Kalnis, B.C. Ooi, K.L. Tan, “Generalized multidimensional data mapping and query processing”, ACM Transactions on Database Systems (TODS), pp. 661-697, 2005.

[11] J. Han, M. Kamber, “Data Mining: Concepts and Techniques”, Second Edition, Morgan Kaufmann , 2005.

[12] 林和宏，大森匡，山下由展，星守，“多次元データマイニングによる Web 空間の構造解析の評価”，DBSJ Letters, Vol.6, No.1, pp.141-144, 2007.

[13] 西川嘉人，辻俊明，金子裕良，阿部茂，“移動オブジェクトの更新に適した領域分割形木構造: kdm-tree” 電子情報通信学会論文誌 D, vol. J92-D, no. 1, pp.1-11, 2009.

[14] 片山紀生，佐藤真一，“類似検索のための索引技術”，情報処理，vol..42, no.10, pp.958-964, 2001.

[15] Bei Li，川口勝也，都司達夫，樋口健，“構造更新に対応した XML 木のラベル付け方式とその評価“，DEIM フォーラム 2009，論文番号 B5-3，2009.

都司 達夫 Tatsuo TSUJI

福井大学大学院工学研究科教授．1978 大阪大学基礎工学研究科博士課程修了，工学博士．データベースに関する研究に従事．電子情報通信学会，情報処理学会，日本データベース学会，各会員．

水野 広治 Hiroharu MIZUNO

福井大学工学部技術部勤務．

松本 宗純 Muneyoshi MATSUMOTO

福井大学大学院工学研究科博士前期課程在学中．2009 同大工学部情報・メディア工学科卒業．多次元データベースシステムの研究・開発に従事．情報処理学会，日本データベース学会，各学生会員

樋口 健 Ken HIGUCHI

福井大学大学院工学研究科准教授．1997 電気通信大学電気通信学研究科博士課程終了，工学博士．分散データベースの研究に従事．電子情報通信学会，情報処理学会，日本データベース学会，各会員．