順序保存ハッシュ関数による高速ジョインアルゴリズム†

大久保 英嗣†† 津田 孝夫††

本論文では，各次元でキーを有する多次元データを，1次元のアドレス空間へ写像する順序保存関数を提案する．提案する関数は，線形なハッシュ関数を基本としており，分布依存ハッシュ関数のクラスに属している．本論文では，まず，提案するハッシュ関数の諸性質について議論する．次に，関係データベースにおける代表的な関係代数演算であるジョイン操作へ適用する場合の方法について述べる．最後に，実験結果を示し，本ハッシュ関数の有効性について考察する．

1. はじめに

現在までに，多くのハッシュ法が研究され，さまざまな分野で使用されている[1),2)]．しかし，ハッシュすべきデータの分布を考慮した，分布に依存する関数の研究は少ない．すなわち，従来の方法では，データの分布とは独立に，あらかじめ定義された関数によりアドレスを計算している．本論文では，それらの関数とは異なった，単純な線形変換を利用した，複数キーによる分布依存 (distribution-dependent) ハッシュ関数を提案する．とくに，実際の場面で使用する際，多次元データの順序づけのための関数として構成したいので，多次元データの順序を形式的に定義し，その順序を保存するハッシュ関数を考えることとする．

われわれの問題は，k 次元キー空間 $X=\{\boldsymbol{x}^t=(x_1^t, x_2^t, \cdots, x_k^t) | 1\leqslant t\leqslant n\}$ の n 個の要素を1次元アドレス空間 $A=\{u | 1\leqslant u\leqslant n\}$ へ順序を保存して一様に写像する関数 h を求めることにある．ここで一様とは，衝突 (collision) が起こらないことを意味する．また，高次元順序を以下のように定義する．

[定義] 任意のキー $\boldsymbol{x}, \boldsymbol{y}\in X$ に関して，ある $i(1\leqslant i\leqslant k)$ が存在して，$x_i<y_i$ かつ $x_j=y_j(1\leqslant j\leqslant i-1)$ なるとき，$\boldsymbol{x}<\boldsymbol{y}$ である．

本定義は，ベクトル $\boldsymbol{x}$ の要素を $x_1, x_2, \cdots, x_k$ の順に重みづけを行っていることと同様である．また，たとえば複数アイテムに関するソーティングにおいて，ソートすべきアイテムの順序を指定することに対応している．

このような関数を構成するには，k 次元乱数 $\boldsymbol{z}$ に関する分布関数 $f(\boldsymbol{x})=P(\boldsymbol{z}\leqslant\boldsymbol{x})$ を使用すればよい（ここで，P は確率を表す）．このとき，$h(\boldsymbol{x})=\lfloor nf(\boldsymbol{x})\rfloor$ (ただし，$\lfloor\ \rfloor$ は floor を表す) を計算することによって，キー $\boldsymbol{x}$ をもつレコードのアドレスを求めることができる．種々の分布関数近似が考えられるが，本論文では，線形変換によるものを考える．このほか，多項式近似による方法[3),4)]があるが，多項式近似の誤差，多項式自体の計算量が無視できないので，なるべく簡単なものを選択した．

2. 単一キーハッシュ関数

m および M をそれぞれキー空間 X (ただし，1次元) におけるキーの最小値および最大値とする*．このとき，この二つの値を使用して，次のようにハッシュ関数を定義する．

$$h(x)\triangleq\lfloor(x-m)(n-1)/(M-m)\rfloor+1 \qquad (2.1)$$

この関数は，キー空間 X における区間 $[m, M]$ をアドレス空間 A へ線形に（すなわち，距離を保存して）写像する．本手法は，Dobosiewicz の分散ソート法[5),12)]，Deutscher らが提案している分布依存ハッシュ関数[6)]でも使用されている．Deutscher らは，ハッシュ関数を構成するために，次のような分布関数 f の線形近似を導入した．(2.1)式を使用して，各区間

$$I_i=\begin{cases}(m, m+(M-m)/n) & (i=1)\\ [m+(i-1)(M-m)/n, m+i(M-m)/n] & (2\leqslant i\leqslant n)\end{cases}$$

に関して頻度（キーの個数）N_i と累積分布関数 G_i を決定することができる．G_i は，$m+i(M-m)/n$ より小さいキーの数を表すもので，この G_i と N_i を使用して，区間 I_i における分布関数 f の近似を次のように定義している．

† A Fast Join Algorithm by Order-Preserving Hash Function by EIJI OKUBO and TAKAO TSUDA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学科

* 以下，本論文では，キーは整数であると仮定しておく．キーが実数あるいは文字列の場合でも，計算機内部では，整数と見なすことが可能であることによる．

$$f_i(x)\triangleq(G_i+(n(x-m)/(M-m)-i)N_i)/n \tag{2.2}$$

Deutscher らは，さらに，二つの衝突回避の手法も示している．しかし，5章で示すように，ハッシュ法が適用される環境に従って，特別の手法を使用することができるので，ハッシュ関数としてわれわれは(2.2)式よりも簡単な(2.1)式を採用する．

3. 多重キーハッシュ関数

キーが k 次元ベクトル $x=(x_1, x_2, \cdots, x_k)$ で表現される k 次元キー空間 X を考える．キーベクトル x の第 i 成分 x_i は，意味的には，x に対応するレコードによって表現される実体の特性あるいは属性を表現するものである．$X_i(1\leqslant i\leqslant k)$ を，i 番目のキーが取りうる値の空間とするとき，従来の多重キーハッシングでは，k 個の異なったハッシュ関数 $h_i: X_i\rightarrow\{0, 1, \cdots, b_i-1\}(1\leqslant i\leqslant k)$ を使用していた（ここで，b_i はキー空間 X_i に関するバケットの数を表す）．すなわち，ベクトル $(h_1(x_1), h_2(x_2), \cdots, h_k(x_k))$ を計算し，X におけるベクトルを一意にバケット番号0から $b_1b_2\cdots b_k-1$ へ写像する関数 h を計算していた．本章では，これらの方法とは異なり，前章で定義した単一キーによるハッシュ関数を複数キーの場合へ拡張する．

最初に，(2.1)式を変形した次の関数を導入する．

$$f*x=f(x)\triangleq\alpha(x-m)/(M-m), 0<\alpha<1 \tag{3.1}$$

ただし，$m=\min x, M=\max x$ とする．この関数は，区間$[m, M]$を区間$[0, \alpha]$へ線形に写像する．ここで，α は1に近いほど望ましいが，特別な基準はない．したがって，α には任意性があるが，実際の場合は，1に十分近い値であれば問題がないことが実験により確かめられている．むしろ，ここで注意しなければならないのは，上記関数がキー空間Xの要素をアドレス空間Aへ一様に写像するとは限らないということである．それは，Xにおける距離を保存するために作られたものであり，キー値の分布がそのままアドレス空間内の分布となってしまうことによる．Xの要素を一様に写像したければ，この関数を衝突が生じたバケットに対して再帰的に適用しなければならない．

さて，上記(3.1)式は，次のように k 次元へ拡張可能である．

$$h(x)\triangleq f*x_1+f*x_2+\cdots+f*x_k \tag{3.2}$$

(ただし，$f*x+y$ は $f(x+y)$ を，$f*x+f*y$ は $f(x+f(y))$ を意味する)．本定義は，次のような再帰的定義と置き換えることができる．

$$f_i^t=\begin{cases}\alpha(x_i^t+f_{i+1}^t-s_i)/(l_i-s_i) & (1\leqslant i\leqslant k)\\ 0 & (i\geqslant k+1)\end{cases} \tag{3.3}$$

$$0<\alpha<1 \tag{3.4}$$

$$l_i=\max_t\{x_i^t+f_{i+1}^t\} \tag{3.5}$$

$$s_i=\min_t\{x_i^t+f_{i+1}^t\} \tag{3.6}$$

ここで，$x^t=(x_1^t, x_2^t, \cdots, x_k^t)(1\leqslant t\leqslant n)$ は k 次元のキー値を表す．さらに，(3.5)と(3.6)において，$l_i=M_i+\alpha, s_i=m_i$(ただし，$M_i=\max_t x_i^t, m_i=\min_t x_i^t$) とおのおの置くと，次の関数を得る．

$$h(x)=\sum_{i=1}^{k-1}\alpha^i(x_i-m_i)/\prod_{j=1}^{i}(M_j-m_j+\alpha)+\alpha^k(x_k-m_k)/(M_k-m_k)\prod_{j=1}^{k-1}(M_j-m_j+\alpha) \tag{3.7}$$

3次元の場合の例を**表1**に示す．表1の例においては，式(3.3)は式(3.7)と等価である（ただし，表1では，$\alpha=0.5$ としている）．

上記ハッシュ関数を使用して，順序づけ関数 $\bar{h}$ を以下のように定義する．

$$\bar{h}(x)=\lfloor h(x)(j-1)/\alpha\rfloor+1 \tag{3.8}$$

ここで，j はアドレス空間におけるバケットの数である．表1の例では，$j=n$ と置いているが，衝突を避

表1 線形変換による多重キーハッシング
Table 1 Multikey hashing by a liner transformation.

t	(x_1, x_2, x_3)	$h(x_1, x_2, x_3)$	$\bar{h}(x_1, x_2, x_3)$
1	(0, 0, 0)	0.000	0
2	(0, 0, 1)	0.017	0
3	(0, 0, 2)	0.033	1
4	(0, 1, 0)	0.067	2
5	(0, 1, 1)	0.083	2
6	(0, 1, 2)	0.100	3
7	(1, 0, 0)	0.200	7
8	(1, 0, 1)	0.217	7
9	(1, 0, 2)	0.233	8
10	(1, 1, 0)	0.267	9
11	(1, 1, 1)	0.283	10
12	(1, 1, 2)	0.300	10
13	(2, 0, 0)	0.400	14
14	(2, 0, 1)	0.417	14
15	(2, 0, 2)	0.433	15
16	(2, 1, 0)	0.467	16
17	(2, 1, 1)	0.483	17
18	(2, 1, 2)	0.500	18

けるためには，jを十分大きくしなければならない．

4. 多重キーハッシュ関数の性質

本章では，提案したハッシュ関数の二，三の性質について述べる．

（1） 順序保存性 (order-preserving)

以下に，1章における定義の意味で，提案した関数が順序を保存することを示す．

[定理 1] 任意のキー $\boldsymbol{x},\boldsymbol{y}\in X$ に関して，$\boldsymbol{x}<\boldsymbol{y}$ のとき $h(\boldsymbol{x})<h(\boldsymbol{y})$ となる．

(証明) ここでは，式(3.3)に関して証明する．式(3.7)に関しては，省略する．

仮定より，あるiが存在して $x_j=y_j(1\leqslant j\leqslant i-1)$ であり，関数fが単調増加であるから，

$$x_i+\sum_{j=i+1}^{k} f*x_j<y_i+\sum_{j=i+1}^{k} f*y_j$$

を証明すればよい．ここで，

$$S^x_{i+1}=x_{i+1}+\sum_{j=i+2}^{k} f*x_j,$$

$$S^y_{i+1}=y_{i+1}+\sum_{j=i+2}^{k} f*y_j$$

とおくと，式(3.3)から式(3.6)により，$\boldsymbol{x},\boldsymbol{y}\in X$ に関して，

$$f(S^x_{i+1})=f_{i+1}=\alpha(x_{i+1}+f_{i+2}-s_{i+1})/(l_{i+1}-s_{i+1})$$
$$<\alpha(l_{i+1}-s_{i+1})/(l_{i+1}-s_{i+1})=\alpha<1$$
$$f(S^y_{i+1})=f_{i+1}=\alpha(y_{i+1}+f_{i+2}-s_{i+1})/(l_{i+1}-s_{i+1})$$
$$>\alpha(s_{i+1}-s_{i+1})/(l_{i+1}-s_{i+1})=0$$

したがって，キーは整数であると仮定しているから，$x_i+1\leqslant y_i$ となり，

$$x_i+f(S^x_{i+1})<x_i+1\leqslant y_i<y_i+f(S^y_{i+1})$$

(証明終り)

（2） 1対1対応 (one-to-one correspondence)

(3.2)の逆関数 h^{-1} が構成可能であることを示す．簡単のために，2次元の場合を示す．(3.1)式において，関数fの逆関数 f^{-1} は，

$$f^{-1}*x=f^{-1}(x)=(M-m)f(x)/\alpha+m,\quad 0<\alpha<1$$

であるから，

$$f^{-1}*h(x_1,x_2)=x_1+f*x_2$$

となる．したがって，x_1 は整数であるから，$f*x_2<1$ となることにより，

$$x_1=\lfloor f^{-1}*h(x_1,x_2)\rfloor$$
$$x_2=\lfloor f^{-1}*(f^{-1}*h(x_1,x_2)-\lfloor f^{-1}*h(x_1,x_2)\rfloor)\rfloor$$

したがって，次の定理を得る．

[定理 2] 任意のキー $\boldsymbol{x}\in X$ に関して，$h(\boldsymbol{x})=k$ ならば $h^{-1}(k)=\boldsymbol{x}$ となる．

（3） クラスタリングの性質(clustering property)

提案したハッシュ関数は，似通ったレコードをグループ化するといったクラスタリングの特性をもっている．たとえば，キー $(x_1,x_2,*)=(1,0,*)$ をサーチするとき（ここでは，＊はキーが指定されていないことを示す），その範囲を知るための不等式

$$h(1,0,\min x_3)\leqslant h(1,0,*)\leqslant h(1,0,\max x_3)$$

の最左辺と最右辺を計算するだけでよい．表1では，

$$0.200=h(1,0,0)\leqslant h(1,0,*)\leqslant h(1,0,2)=0.233$$

であるから，ただちに $(1,0,*)=\{(1,0,0),(1,0,1),(1,0,2)\}$ を得る．しかし，$(x_1,*,x_3)=(1,*,2)$ のときは，条件 $x_1=1, x_3=2$ を満足するキーをすべてサーチしなければならない．したがって，キーが指定されている次元を優先させてハッシュ関数を再構成する必要が生じる．これに関しては，これ以上言及しないが，本特性は，範囲検索による部分マッチの一つの手法としても使用可能であるといえよう．

5. 高速ジョインアルゴリズム

提案するアルゴリズムは，次元に関係なく，大きく分けて2段階の処理より構成される．最初に，ジョインすべきカラムの一つに前章までに述べたハッシュ関数を適用し，順序を保存してそのカラムをいくつかのグループに分割する（これを分配分割 (distributive partitioning) のフェーズと呼ぶ）．次に，もう一方のジョインカラムを同一の関数を使用してハッシュし，グループ番号を求める（これを補間サーチ (interpolation search) のフェーズと呼ぶ）．この二つの操作によって，同一グループ内だけでジョイン条件を満足するタプル識別子 (tuple identifiers) の組を求めることに帰着し，高速化が可能となる．なお，本章では，キーおよびデータを同一視して扱う．

5.1 アルゴリズム

ジョイン演算そのものに関しては，他文献[7),8)]を参照してもらうとして，ここでは，自然結合 (natural join) の場合だけを考える．他の場合へも容易に拡張することができる．さらに，関係のタプル数をnとしておく．

（1） 再帰的なジョイン (RJoin)

関係RおよびSのジョインカラムをそれぞれ R_1, S_1 とする．このとき，アルゴリズムは以下のようになる．

（a） カラム R_1 におけるアイテムの最大値 max

と最小値 min を求める.

(b) カラム R_1 における各アイテム x を以下の関数により，グループ j へ分配する.

$$j=1+\lfloor(x-\min)(n-1)/(\max-\min)\rfloor \quad (5.1)$$

さらに，max, min および n (アイテムの個数) は，ステップ(d)のために保持しておく.

(c) 各グループに分配されたアイテムの個数を数える．値の異なったアイテムが同一のグループに分配される場合は，ステップ(a)から(c)を再帰的に繰り返す.

(d) カラム S_1 に関して，ステップ(b)と同一の変換を施し，グループ番号を求める．さらに，当該グループにおいて，ジョイン条件を満足するタプル識別子の組を求める.

ステップ(d)は，補間サーチ法[9)-11)]と同様の考え方であり，カラム R_1 に関する順序づけられたリストをカラム S_1 におけるアイテムで補間的にサーチすることに対応している．ステップ(a)から(c)の再帰的な処理は，各グループに分配されるアイテムの個数が少なければ，終了するようにしたほうがよい（実験では10個以下とした).

(2) 2進木によるジョイン (DPJoin)

上記 RJoin は，最大値，最小値およびアイテムの個数を再帰処理の各段階で保持しなければならない．これは，RJoin の上記ステップ(c)および(d)を以下のように変更することで避けることができる.

(c) 異なるアイテムが同一グループに分配される場合は，当該グループで2進木を構成する.

(d) カラム S_1 に関して，ステップ(b)と同一の変換を施してグループ番号を求め，当該グループにおける2進木をたどり，ジョイン条件を満足するタプル識別子の組を求める.

(3) N-way ジョイン

関係 $R_i(1\leqslant i\leqslant N)$ のカラム $C_i(1\leqslant i\leqslant N)$ に関するジョイン，すなわち N-way ジョイン $C_1*C_2*\cdots*C_N$ は，上記 (2-way) ジョインの拡張と見ることができる．すなわち，ある関係のジョインカラムの一つを(5.1)式によりグループに分割し，次に他の関係のカラムに対して，同一の変換を行うということが考えられる．しかし，アルゴリズムは，(1)および(2)と同様であるので省略する.

(4) 複数カラムに関するジョイン

関係 R と関係 S の $k>1$ 個のカラムに関するジョインのアルゴリズムを以下に示す.

(a) 関係 R のジョインカラムの k 次元目のカラムの最大値および最小値を求め，$M=\max_t x_t^k$, $m=\min_t x_t^k$ とする．同時に，$W_t=x_t^k(1\leqslant t\leqslant n)$ とする.

(b) 以下の操作を i が $k-1$ から1まで繰り返す.

$$W_t=x_t^i+\alpha(W_t-m)/(M-m)$$
$$M=\max_t W_t,\quad m=\min_t W_t$$

(c) ステップ(b)で得られたハッシュ値 W_t によって，グループ番号 j を次式により計算する.

$$j=\lfloor(W_t-m)(n-1)/(M-m)\rfloor+1$$

(d) 関係 S のジョインカラムに関して，関係 R の k 次元目のカラムと共通の最大値および最小値から出発し，ステップ(b)および(c)を適用し，グループ番号とハッシュ値を得る．次に，当該グループ内で，関係 R に関するハッシュ値と比較し，結果を得る.

5.2 時間計算量

本節では，各ジョインアルゴリズムの計算の複雑さ，とくに，時間計算量について解析する.

[定理 3] RJoin および DPJoin の時間計算量は，最悪の場合 $O(n^2)$ である.

(証明) RJoin の最悪の場合は，再帰処理の各段階で1個のアイテムが n 番目のグループへ分配され，残り $n-1$ 個のアイテムが1番目のグループへ分配される場合である．したがって，変換の時間計算量 $T(n)$ は，

$$T(n)\leqslant c_1n+T(1)+T(n-1)$$
$$T(1)=c_0 \quad (5.2)$$

となる．ここで，c_1n は再帰処理の各段階におけるステップ(a)から(c)で使用される時間を表している．不等式(5.2)の解は，

$$T(n)\leqslant c_0+\frac{1}{2}c_1(n-1)(n+2)=O(n^2)$$

となる．補間サーチのフェーズの計算の複雑さは，分配分割のフェーズと同様であるから，結局全体として $O(n^2)$ となる.

DPJoin の場合は，2進木構成の時間計算量が最悪の場合 $O(n^2)$ となることにより明らかである.

(証明終り)

[定理 4] RJoin の平均時間計算量は，$O(n\log n)$ である.

(証明) 分配分割のフェーズにおける時間計算量は，

$$T(n)\leqslant c_1n+T(i)+T(n-i)\quad(1\leqslant i\leqslant n-1)$$
$$T(1)=c_0$$

である．したがって，平均時間計算量は，

$$T(n)\leqslant c_1 n+\frac{1}{n-1}\sum_{i=1}^{n-1}[T(i)+T(n-i)]$$

$$T(1)=c_0$$

となり，この不等式は以下の解をもつ．

$$T(n)\leqslant cn\log_e n\leqslant cn\log n$$

（証明終り）

[定理 5] 複数カラムに関するジョインアルゴリズムの最悪の場合の時間計算量は，$O(n^2)$ である．

（証明） ステップ(c)，(d)は，1次元の場合のアルゴリズムをそのまま適用することができるので，最悪の場合の時間計算量は $O(n^2)$ となる．一方，ステップ(a)，(b)は次元数を k とするとき $O(kn)$ であるから，$k<n$ のとき $O(n^2)$ となる．

（証明終り）

5.3 再帰処理の段数

本節では，RJoin の最悪の場合の再帰処理の回数について解析する．

相異なる整数の最小距離は1であるから，一つのグループに分配されたアイテムの分布区間が1より小さくなるとき，当該グループのアイテムの値はすべて等しくなる．したがって，当該グループにおけるアイテムの最大値と最小値の差が，グループ内のアイテムの個数より小さくなるとき処理は終了する．いま，r 段目でこの条件が満足されると仮定すると

$$\begin{aligned}&(\max-\min)(n-r+1)!/n!>1,\\&(\max-\min)(n-r)!/n!\leqslant 1\end{aligned}\qquad(5.3)$$

なる二つの不等式を得る．最悪の場合，$r=n-1$ であるから，これを上式に代入すると，

$$\max-\min\leqslant n!<2(\max-\min)$$

となる．さらに，キーの長さを4バイトと仮定すると，$\max-\min\leqslant 2^{32}$ であるから，上式より $n=13$ となってしまう．この値は現実的ではないので，n が r より十分大きい場合を考える．不等式(5.3)より

$$n^{r-1}<\max-\min\leqslant n^r$$

を得る．ここで，$\max-\min=2^p$ および $n=2^q$ と置き，上式に代入すると

$$p/q\leqslant r<p/q+1$$

となる．したがって，n が十分大きいとき $r=p/q$ となる．q は n の単調増加関数であるから，r は n の単調減少関数となる．これは，あらかじめデータの最大値と最小値の差がわかれば，あるいは，データ長を固定することによって，再帰処理の段数を減少させることが可能であることを意味する．

いま，再帰的ステップが必要ない場合の時間計算量を $S(n)$ とすると，$S(n)$ は1段の計算時間を示すから，全体の時間計算量は $T(n)=rS(n)$ となる．$S(n)$ は，各データのハッシュに要する時間計算量であるから $O(n)$ となり，結局 $T(n)=O(rn)$ となる．したがって，r が小さくなれば時間計算量は $O(n)$ に近づく．

6. 実験結果

本章では，前章に示したアルゴリズムの実験結果を示す．アルゴリズムはすべて PL/I で記述し，実験は京都大学大型計算機センターの FACOM M-200 を利用させていただいた．

データとしては，一様分布，正規分布，指数分布，およびポアソン分布の乱数を発生させ，あらかじめ指定した区間に分布するよう発生した乱数を変換した．また，α の値はすべて 0.9 とした．

表 2 ジョインアルゴリズムの比較
Table 2 Comparison of the join algorithms. (Computed times are in msec.)

(a) 一様分布の場合
(a) The case of the uniform distribution.

n	No. of result tuples	min	max	RJoin (ms)	DPJoin (ms)	Merge scans (ms)	Nested loops (ms)
200	22	1	2,000	8	14	19	88
500	42	1	5,000	19	20	41	536
1,000	82	1	10,000	35	35	78	2,151
2,000	160	1	20,000	66	60	152	8,571
3,000	255	1	30,000	99	86	224	19,245
4,000	386	1	40,000	133	114	296	34,265
5,000	472	1	50,000	164	140	381	55,018

(b) 正規分布の場合
(b) The case of the normal distribution.

n	No. of result tuples	min	max	RJoin (ms)	DPJoin (ms)	Merge scans (ms)
200	47	1	2,000	10	14	19
500	77	1	5,000	19	24	41
1,000	178	1	10,000	35	36	76
2,000	375	1	20,000	69	66	152
3,000	361	1	45,000	103	93	232
4,000	554	1	60,000	135	123	310
5,000	730	1	80,000	177	156	386

表 3 二つのカラムに関するジョイン
Table 3 Joins for two columns. (Computed times are in msec.)

n_1	n_2	Distributive partitioning (ms)	Interpolation search (ms)	Total CPU time (ms)
1,000	1,000	16	9	25
	5,000	16	55	71
	10,000	16	103	119
5,000	1,000	76	12	88
	5,000	77	57	134
	10,000	76	106	182
10,000	1,000	153	13	166
	5,000	154	59	213
	10,000	154	119	273

例 1. 各種ジョインアルゴリズムの比較

RJoin, DPJoin, 単純なループ法 (nested loops), およびソートマージによる方法 (merge scans) の比較を表 2 に示す. merge scans におけるソーティングには, 分散ソート法を用いた. また, nested loops に関しては, データが一様分布の場合だけ示してある. この方法は, データの分布に依存しないことによる. RJoin, DPJoin および merge scans は, この例の場合計算の複雑さは $O(n)$ となっているが, nested loops は $O(n^2)$ である. また, 本論文で提案している手法が merge scans よりも CPU コストが少なくなっているのは, この場合データが各グループに均等に分配されるためである. すなわち, RJoin では再帰処理の回数が, DPJoin では木の深さが小さくなり, ハッシングの効果が現れていることを示している.

例 2. 複数カラムに関するジョイン

表 3 に, 二つのカラムに関するジョインの結果を示してある. また, カラム数の影響を見るために, データ数が 5,000 の場合の分配分割のフェーズの時間を表 4 (a) に, データの分布の影響を見るために, 1 次元目のカラムが各分布の場合の 3 次元の結果を表 4 (b) に示す. 表 4 (b) においては, ポアソン分布 (相異なるデータの数が少ない場合に対応する) の場合は, 他の場合よりも計算時間が大きくなっている. これは, 分配分割のフェーズで, データが分配されるグループに偏よりが生じるためである. したがって, 複数カラムの場合は, 一様分布に近いカラムに重みをつける必要がある. しかし, 実際的場面では, ジョインすべきカラムは関係の主キーである場合がほとんどで, データの分布としては一様分布となり, あまり問題はないと考えている.

表 4 カラム数とデータの分布の影響
Table 4 Effect of the number of columns and the distribution of data. (Computed times are in msec.)

(a) カラム数の影響
(a) Effect of the number of columns.

(ms)

Number of columns	1	2	3	4	5	6
Distributive partitioning	38	84	131	178	224	271

(b) データの分布の影響
(b) Effect of the distribution of data.

n	Uniform (ms)	Normal (ms)	Exponential (ms)	Poisson (ms)
1,000	32	32	35	90
3,000	88	91	100	290
5,000	153	157	172	498
10,000	299	312	340	1,005
15,000	460	477	540	1,448

7. おわりに

本論文では, 順序保存ハッシュ関数とそのジョイン操作への応用を示した. とくに, 複数カラムに関するジョインの高速化が達成されたといえよう.

従来のアルゴリズムは, 各次元のアイテムをネストさせて処理するものが大半である. すなわち, ある次元に関するソートフェーズがあり, 同一の値をもつデータに対してその他の次元でソートを施すための比較フェーズがある. これが交互に同一回数現れるが, 本論文で提案している手法は, まずデータを, 順序を保存するハッシュ関数で 1 次元値に射影し, その値によりジョインを行うもので, フェーズが二つに分かれるのは同様であるが, おのおのただ一度だけ実行されるという点で大きく異なっている. したがって, 本手法のほうが, 各次元で繰り返しソーティングを行う従来の方法よりも高速であることが期待できる.

関係操作に関しては, 結合操作しか述べなかったが, このほか, 射影操作, 除算操作, 集合演算のインタセクションにも利用可能である.

本手法の問題点としては, ハッシュ関数におけるパラメータ α の値の任意性と, ハッシュ関数の計算のまるめ誤差の問題があるが, 実用上は, 再帰的処理を導入することで解決できると考えている.

参考文献

1) Knuth, D. E.: *The Art of Computer Programming*, Vol. 3, *Searching and Sorting*, Addison-Wesley, Reading, Mass. (1973).
2) Knott, G. D.: Hashing Functions, *Comput. J.*, Vol. 18, No. 3, pp. 265-278 (1975).
3) Tarter, M. E. and Kronmal, R. A.: Estimation of the Cumulative by Fourier Series Methods and Application to the Intersection Problem, Proc. 23rd Natl. ACM Conf., pp. 491-497 (1968).
4) Lefons, E. et al.: Evaluation of an Addressing Technique for Large Volumes of Data, AICA '79 Conf., 10-13 Oct., pp. 305-311 (1979).
5) Dobosiewicz, W.: Sorting by Distributive Partitioning, *Inf. Process. Lett.*, Vol. 7, No. 1, pp. 1-6 (1978).
6) Deutscher, R. F. et al.: Distribution-Dependent Hashing Functions and Their Characteristics, Proc. ACM SIGMOD Conf., pp. 224-236 (1975).
7) Date, C. J.: *An Introduction to Database Systems*, 2nd ed., Chap., 6, Addison-Wesley, Reading, Mass. (1977).
8) Gotlieb, L. R.: Computing Joins of Relations, Proc. ACM SIGMOD Conf., pp. 55-63 (1975).
9) Perl, Y. et al.: Interpolation Search—A Log Log *N* Search, *CACM*, Vol. 21, No. 7, pp. 550-553 (1978).
10) Nat, M.: On Interpolation Search, *CACM*, Vol. 22, No. 12, p. 681 (1979).
11) Burton, F. W. and Lewis, G. N.: A Robust Variation of Interpolation Search, *Inf. Process. Lett.*, Vol. 10, No. 4, pp. 198-201 (1980).
12) Nat, M.: A Fast Sorting Algorithm, a Hybrid of Distributive and Merge Sorting, *Inf. Process. Lett.*, Vol. 10, No. 3, pp. 163-176 (1980).

(昭和57年10月21日受付)
(昭和58年 7 月19日採録)