データベース・システム 81－4
（1991. 1. 17）

複合オブジェクトの符号化

小濱 千恵　　中野 良平

ＮＴＴ情報通信処理研究所

データベースにおけるオブジェクトの管理には、値に基づく方法と識別子に基づく方法がある。値に基づく方法は、単一オブジェクトに対しては効率的であるが、集合やタプルを構成子とする複合オブジェクトを導入すると、効率的な管理が困難になる。そこで、複合オブジェクトを値と１対１に符号化して格納することによって、オブジェクトの管理を簡明化する。複合オブジェクトの１対１符号化を効率よく行なうために、１つの複合オブジェクトに対して簡単な計算により１つの特性数を割り当て、特性数を比較することで複合オブジェクトが異なることを検出する手法を提案する。また、高い確率で異なる複合オブジェクトを検出する関数について検討する。

Encoding of Complex Objects

Chie KOHAMA　　Ryohei NAKANO

NTT Communications and Information Processing Laboratories
1-2356, Take, Yokosuka-Shi, Kanagawa, 238-03, Japan

In database management systems, there are two ways of managing objects: value-based and identifier-based. In a value-based approach, expanding atomic objects into complex ones makes object management much more difficult. If we can provide one-to-one mapping from complex objects to codes, we will have a quite simple management system. To make one-to-one encoding efficient, we propose an approach to make use of characteristic numbers of a complex object. It is expected that complex objects having different values can be distinguished from each other by comparing their characteristic numbers. This paper proposes functions which generates characteristic numbers having high distinction ability.

1．はじめに

データベースにおけるオブジェクトの管理には、値に基づく(value-based)方法と識別子に基づく(id-based)方法がある。前者は、値が等しいオブジェクトを同一のものとみなして管理する方法であり、後者は、オブジェクトの値に関係なくオブジェクトの実体と1対1に識別子を付与して管理する方法である。[1][2][3] 数値や文字列などの集まりを表形式で表現した関係(relation)に対しては、値に基づく効率的な管理が可能である。しかし、多様なデータ表現を可能にするために、集合やタプルを構成子とする複合オブジェクトを関係の値として導入した入れ子型関係(nested relation)では、値に基づく管理を効率的に行なうことが難しくなる。

そこで、複合オブジェクトを値と1対1に符号化して格納することにより、複合オブジェクトの管理を簡明化する。符号化により、複合オブジェクトを数値や文字列などと同様に取り扱うことができる。1対1符号化では、符号化済みの複合オブジェクトとの等値判定の負荷が大きい。そのために、複合オブジェクトにその特性を表現する特性数を割り当て、特性数の比較によって複合オブジェクトが異なることを検出する手法を提案する。また、複合オブジェクトを特性数に割り当てる関数について検討する。

以下、2章に符号化と特性数の考え方、3章に特性数関数の検討、4章に特性数関数の評価法、5章に特性数関数の検証結果を示す。

2．複合オブジェクトの符号化

2．1　対象とする複合オブジェクト

対象とする複合オブジェクトを以下に示す。

〈単一ｵﾌﾞｼﾞｪｸﾄ〉 :=〈数値〉|〈文字列〉
〈集合〉 :={〈単一ｵﾌﾞｼﾞｪｸﾄ〉}
〈ｵﾌﾞｼﾞｪｸﾄ〉 :=〈単一ｵﾌﾞｼﾞｪｸﾄ〉
|〈複合ｵﾌﾞｼﾞｪｸﾄ〉
〈ﾀﾌﾟﾙ〉 :={[〈属性名〉:〈ｵﾌﾞｼﾞｪｸﾄ〉]}
〈入れ子型関係〉 :={〈ﾀﾌﾟﾙ〉}
〈複合ｵﾌﾞｼﾞｪｸﾄ〉 :=〈入れ子型関係〉|〈集合〉

2つの複合オブジェクトが等値であるとは、複合オブジェクトの構造が一致しているとともに、すべての値が一致していることを指す。入れ子型関係は関係の拡張なので、複合オブジェクトは関係を包含する。入れ子型関係からなるデータベースを入れ子型データベースと呼ぶ。

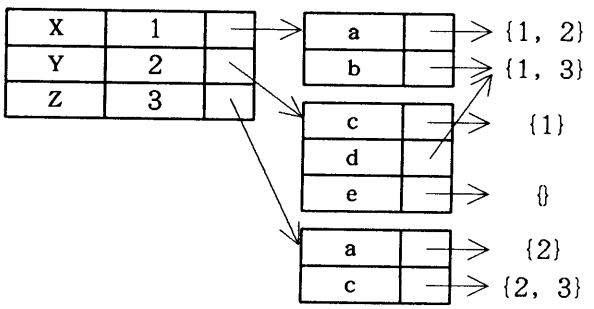
2．2　入れ子型データベースにおける1対1符号化

1対1符号化では、複合オブジェクトの値と符号が1対1に対応しており、等値である複合オブジェクトは同じ符号を持ち、同じ符号を持つ複合オブジェクトは等値であることが保証される。そのため、複合オブジェクトのすべての構成要素についての一致判定（以下、一致判定と呼ぶ。）を行なわずに、符号の比較だけで複合オブジェクトが等しいかどうか判定できる。1対1符号化の利点を以下に示す。

(1) 入れ子型データベースにおいて関係データベースと同様にオブジェクトを扱える。
(2) 関係データベースの格納構造に符号化機構を加えるだけで、入れ子型データベースの格納構造が実現できる。

X	1	a	{1, 2}
		b	{1, 3}
Y	2	c	{1}
		d	{1, 3}
		e	{}
Z	3	a	{2}
		c	{2, 3}

［図1(a)］ 入れ子型関係



［図1(b)］ 入れ子型関係の1対1符号化による格納

(3) データの圧縮効果があり、主記憶データベースなど格納領域が制限されるデータベースへの適用に有利である。[4]

入れ子型関係の例を[図1(a)]に、1対1符号化による格納構造の例を[図1(b)]に示す。

符号はシステムで一意とする。つまり、そのシステムの中で等しい複合オブジェクトにはすべて同じ符号が付与される。また、どのような符号を用いるかについては言及しないが、単一オブジェクトおよび符号は固定長とする。その理由として、以下があげられる。

(1) 固定長符号化に基づいた関係データベースシステム MACH1 に適用する。[5]

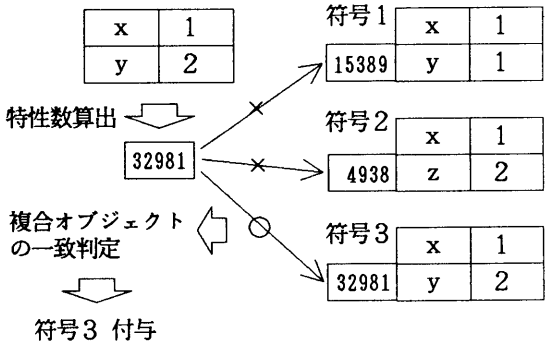
(2) 特性数関数の検討において、簡単化のためにとりあえず固定長を対象とする。

2．3　特性数を用いた等値判定

特性数導入の位置付けと特性数の概念について述べる。1対1符号化することによって、複合オブジェクトの等値判定が必要なのは符号化時のみとなる。しかし、複合オブジェクトでは、符号化時だけであっても等値判定の処理負荷は大きいと予想され、十分効率よく行なわれる必要がある。データベースにおける等値判定の高速化のためには、ハッシングやB木を用いた索引づけ(indexing)が利用されることが多い。複合オブジェクトを扱うデータベースでも、索引づけによる高速化が主流である。[6] しかし、索引づけによらずに複合オブジェクトの等値判定を効率的に行なう手法についても検討されつつある。[7] ここでは、索引づけなしで複合オブジェクトの高速な等値判定を試みる。

複合オブジェクトが不等値であることを効率よく検出できれば、あとは等値の可能性がある複合オブジェクトに対してのみ一致判定を行なえばよい。できるだけ高い確率で不等値であることを検出できることが望まれる。そこで、不等値検出のための特性数の概念を提案する。

特性数とは、複合オブジェクトの特性を表現した固定長の数値または数値群であり、複合オブジェクトごとに割り当てられデータベース中に格納される。特性数が異なれば対応する複合オブジェクトが一致しないことが保証されるが、



[図2] 特性数による符号化の例

特性数が等しくても対応する複合オブジェクトが一致する保証はない。特性数による複合オブジェクトの符号化の例を[図2]に示す。

以下、簡単な計算で算出でき、高い確率で複合オブジェクトが不等値であることを検出する特性数関数について検討する。

3．望ましい特性数関数とは

3．1　前提条件と検討方針

特性数関数を検討するにあたっての前提条件および検討方針を述べる。特性数関数は、複合オブジェクトの値を引数とし、特性数または特性数群を出力とする関数である。構成要素の持つ情報量を考え、ここでは集合に対しては1つの特性数、入れ子型関係に対しては特性数群を割り当てるものとする。

特性数関数には簡単な計算からなる汎用の関数を用い、複合オブジェクトの値の分布を解析してそれに合った関数を利用することはしない。なぜなら、データベースは更新が起こるため、複合オブジェクトの値の分布はあらかじめわからず、更新の度に値の分布を調べ直すことは計算量が大きすぎる。簡単な計算で求められる関数という立場から、和、積、排他的論理和などの組み合わせで関数を構成する。

複合オブジェクトの特性数関数として集合と入れ子型関係の2種類の特性数関数が必要である。そこで、以下の関数について検討を行なう。

(1) 集合 $S=\{e_1, e_2, \ldots, e_n\}$ の特性数関数 $Fs(S)$

(2) タプルにおける要素と属性の組 $[a_i : x_i]$ の特性を表現する関数 $Fa(a_i, x_i)$

関数 Fs と Fa を組み合わせて、入れ子型関係の特性数関数を構成する。つまり、タプル T= {[a_1:x_1], ..., [a_n:x_n]} の特性を

Ft(T) = Fs(Fa(a_1, x_1), ..., Fa(a_n, x_n))

で表現し、入れ子型関係

R = {{[a_{11}:x_{11}], ..., [a_{1n}:x_{1n}]},
...
{[a_{n1}:x_{n1}], ..., [a_{nn}:x_{nn}]}}

の特性数関数

Fr(R)= Fs(Fs(Fa(a_{11}, x_{11}),...,Fa(a_{1n}, x_{1n})),
...
Fs(Fa(a_{n1}, x_{n1}),...,Fa(a_{nn}, x_{nn})))

を導く。

集合の特性数関数 Fs(S)が Fs1(S)と Fs2(S)という関数の組み合わせからなる時、入れ子型関係の特性数関数は、以下のような特性数群) [Fr1(R),Fr2(R)] を割り当てる。

Fr1(R)= Fs1(Fs(Fa(a_{11}, x_{11}),...,Fa(a_{1n}, x_{1n})),
...,
Fs(Fa(a_{n1}, x_{n1}),...,Fa(a_{nn}, x_{nn}))
Fr2(R)= Fs2(Fs(Fa(a_{11}, x_{11}),...,Fa(a_{1n}, x_{1n})),
...
Fs(Fa(a_{n1}, x_{n1}),...,Fa(a_{nn}, x_{nn}))

以下、基本的な用語を定義する。

衝突 -- 等値でない複合オブジェクトが同じ特性数に割り当てられてしまうこと。

衝突なしマッピング -- 衝突が起きずに、すべての複合オブジェクトが特性数に割り当てられること。

特性数総数 -- 複合オブジェクトを割り当てることができる特性数の個数。

オブジェクト数 -- 対象とする複合オブジェクトの個数。

衝突数 -- 衝突の起きた回数。

特性数割り当て数 -- 割り当てが行なわれた特性数の個数。(オブジェクト数 - 衝突数) に等しい。

最大マッピング数 -- 1特性数あたりに割り当てられた複合オブジェクトの数で最大のもの。

ここでは、特性数総数がオブジェクト数に比べ十分に多い場合を想定しているので、理想的な特性数関数は衝突なしマッピングであり、特性数割り当て数 = オブジェクト数となる。

3．2 集合の特性数関数

集合 S={e_1, e_2,..., e_n} の特性数関数 Fs(S) について検討する。まず、基本的な演算（和、積、2乗和）の性質を調べる。排他的論理和は和と似た性質を持つと思われるので、解析を省く。排他的論理和および2乗排他的論理和の評価は、5．2の比較平均の検証で行なう。

基本的な演算の組み合わせ方については、各要素を特性数にまとめあげる関数（重ね合わせ関数）、各要素に前処理をほどこす関数（前処理関数）、さらに判別能力をあげるために2つ以上の特性数を組み合わせるための関数（組み合わせ関数）の3つの視点から検討する。

(1) 重ね合わせ関数：以下の基本的な演算の性質を解析し評価を行なう。

[1] 和 Σe_i
[2] 積 Πe_i
[3] 2乗和 Σe_i^2

(2) 前処理関数：重ね合わせによる情報損失を緩和するために、2種類の前処理を検討する。

(2-1) シフト ： 判別に有効に働くビットが偏って重ね合わされることを補正する。値によってシフトの大きさを変える。

shif(e_i) = e_i 〈〈〈 (e_i mod N)

N は特性数のビット数, 〈〈〈 は巡回シフト

(2-2) 圧縮 ： オーバーフローによる情報損失を軽減するため半分のビット数に圧縮する。上位ビットと下位ビットを重ねる。

comp(e_i) = ((e_i 〈〈 N/2) + e_i) 〉〉 N/2

N は特性数のビット数, 〈〈 は非巡回シフト

和に対してシフト、積と2乗和に対してシフトと圧縮を適用し、以下の7つの関数に対して評価を行なう。

[4] 和(ｼﾌﾄ) Σshif(e_i)
[5] 積(ｼﾌﾄ) Πshif(e_i)
[6] 積(圧縮) Πcomp(e_i)
[7] 積(ｼﾌﾄ圧縮) Πcomp(shif(e_i))
[8] 2乗和(ｼﾌﾄ) Σshif(e_i)^2
[9] 2乗和(圧縮) Σcomp(e_i)^2
[10] 2乗和(ｼﾌﾄ圧縮) Σcomp(shif(e_i))^2

(3) 組み合わせ関数：基本的な演算の解析結果から組み合わせ関数および関数の組み合わせ方を提案し、評価を行なう。

3．3　タプルの特性数関数

タプルにおける要素と属性の組 $[a_i:x_i]$ の特性を表現する関数 $Fa(a_i,x_i)$ について検討する。

従来、ハッシュ関数などで要素列の順序を反映する手法として、i 番目の属性を i だけ巡回シフトするものがある。[8][9]

$Fa'(a_i,x_i) = x_i \lll i$,　$\lll$ は巡回シフト

x_i を i だけシフトすることは、x_i と 2^i との積を取ることに等しい。$Fa'(a_i,x_i)$ では巡回シフトを行なうが、情報損失がない場合を考えれば、衝突の起こりやすさは同じである。

$Fa'(a_i,x_i) = x_i * 2^i$

そこで、属性 a_i ごとに異なる素数 $p(a_i)$ を定め、素数との積によって属性情報を反映させる手法を提案する。2^i との積を取るよりも素数との積を取るほうが衝突は起こりにくくなる。

$Fa(a_i,x_i) = x_i * p(a_i)$,　$p(ai)$ は素数

4．特性数関数の評価法

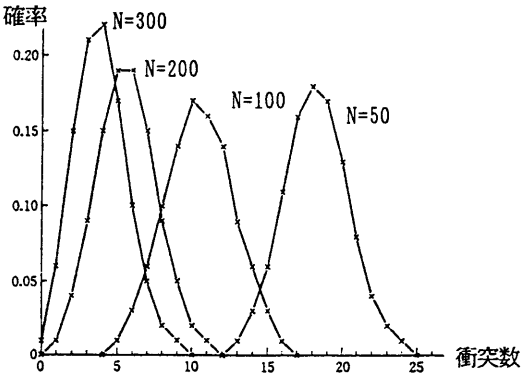
4．1　識別率と等確率マッピング

特性数関数の衝突の起こりやすさの評価尺度として、**識別率**を導入する。

識別率 = 特性数割り当て数／ｵﾌﾞｼﾞｪｸﾄ数

識別率は 0〜1 の数値を取り、1 ならば衝突なしマッピングであることを示す。また、すべての複合オブジェクトが独立に等しい確率で特性数に割り当てられることを**等確率マッピング**と呼ぶ。つまり、特性数総数 N に対してオブジェクト数 C が十分大きい時すべての特性数に C/N 個の複合オブジェクトが割り当てられる。等確率マッピングは衝突なしマッピングではないが、作為的に１つの特性数に複合オブジェクトが集中してしまう状態を避けることができる。等確率マッピングを特性数関数の評価基準とする。

以下で、等確率マッピングにおける衝突数の確率について述べる。特性数総数 N、オブジェクト数 C で等確率マッピングである時の衝突数を W とする。特性数割り当て数 M(=C-W) の確率分布は一般に占有分布と呼ばれ、



[図3]　等確率マッピングの衝突数の確率分布

$$Pr[M=k] = \left\{ {C \atop k} \right\} \frac{N(k)}{N^C} = \left\{ {C \atop K} \right\} \binom{N}{k} \frac{k!}{N^C}$$

$$(k = 1,2,\ldots,\min(C,N))$$

で表わせる。$\left\{ {C \atop k} \right\}$ は第２種Stirling数列と呼ばれ、

$$\left\{ {C \atop 1} \right\} = \left\{ {C \atop C} \right\} = 1; \quad \left\{ {C \atop k} \right\} = 0,\ k \notin [1,C]$$

$$\left\{ {C+1 \atop k} \right\} = k \left\{ {C \atop k} \right\} + \left\{ {C \atop k-1} \right\} \quad (1 \leqq k \leqq C)$$

で定まる三角数列である。$N(k)$ は N 個の特性数のうち k 個に割り当てが起きる時の順列の数 $N(N-1)..(N-k-1)$ であり、N^C は C 個の複合オブジェクトを N 個の特性数に割り当てた時の場合の数である。衝突数の確率分布

$Pr[W=k] = Pr[M=C-k]$

の例を[図３]に示す。衝突数 W の平均値は、

$W = N(1-1/N)^C - (N-C)$

$\fallingdotseq Ne^{-C/N} - (N-C)$

で近似できる。[図３]から、等確率マッピングの場合、

オブジェクト数(C) = 特性数総数(N) = 50

であれば衝突数(W) = 18 である確率がもっとも高いことがわかる。[8] 上記の式から算出される等確率マッピングでの識別率を[表１]に示す。

特性数総数 : ｵﾌﾞｼﾞｪｸﾄ数	識別率
1 : 1	0.632
10 : 1	0.952
100 : 1	0.995

［表1］ 等確率マッピングでの識別率

4．2 比較平均と衝突なしマッピング

識別率によって評価した関数全体の衝突数が同じであっても、1特性数あたりの衝突数の分布によって、複合オブジェクトの一致判定の回数は異なる。1つの複合オブジェクトを符号化する時に必要な一致判定の平均回数を**比較平均**と呼び、評価尺度とする。

オブジェクト数を C、特性数割り当て数を M、1特性数あたりの複合オブジェクトのマッピング数を $T_i\ (1 \leqq i \leqq M)$ とすると、1特性数あたりの一致判定の回数 $P_i\ (1 \leqq i \leqq M)$ は、

$$P_i = \sum_{x=1}^{Ti-1} x = \frac{T_i(T_i+1)}{2}$$

である。したがって、比較平均 P は

$$P = \frac{\sum_{i=1}^{M} P_i}{C} = \frac{\sum_{i=1}^{M} \frac{T_i(T_i+1)}{2}}{C}$$

となる。比較平均 P は、0 以上の数値を取り、0 ならば衝突なしマッピングであることを示す。特性数総数がオブジェクト数に比べ十分に多い場合が検討対象であることから、**衝突なしマッピング**を評価基準とする。

5．特性数関数の検証

5．1 集合の特性数関数---識別率からの検討

入れ子型データベースの符号化機構に適用した時に十分な判別能力を持つことを目的として、集合の特性数関数に対して識別率を用いた検証を行なった。

特性数総数 65,536(16 bit) = オブジェクト数として等確率マッピング相当の識別率(0.632)が出せるかどうかを評価した。要素数が少ない場合（要素数 2、5、10）を中心に検証を行ない、衝突の起こり方を解析した。実際の適用では要素数はせいぜい 100 までと思われるため、要素数 100 で等確率マッピングを実現することをめざした。また、要素範囲が狭いほど判別が困難であると考え、できるだけ狭い範囲の値で必要な数の集合を生成するように要素を定めた。要素範囲は、以下の2種類を用いた。

1) 最も判別が困難と思われる 0近傍
2) 0近傍との比較のため、ある程度大きい正数

各演算に関して、以下が明らかになった。

(1) 和：要素範囲に関わらず一定の識別率を示すが、識別率は 0.011 以下と低い。これは、

$x+y = (x-n)+(y+n)$

が成立する場合に衝突する和の性質による。衝突は要素数が多い場合に起こりやすく、要素範囲をどこに定めるかには関係しない。和で起こる衝突は特定のビットが偏って重ねられてしまうためであり、前処理にシフトを用いることにより改善されるが十分ではない。

(2) 積：要素数が少なく"ある程度大きい正数"の場合に比較的高い識別率を出す。これは、

$x*y = (x/n) * (y*n)$　(x/n) は整数

が成立する場合に衝突が起こるので、要素範囲が狭く"ある程度大きい正数"ではこれが成立しにくい。このように考えると衝突が起こる可能性は和の場合よりも少ない。ただ、要素に 0 が含まれる場合に弱いこと、要素数が多くなるとオーバーフローのために識別率が 0.000 になってしまうことが致命的であり、前処理によっても改善されない。

(3) 2乗和：和と積の中間の性質を持つ。和ほどビットの偏りが起きず、積のように深刻なオーバーフローもない。衝突は、

$x^2+y^2 = (x^2-n) + (y^2+n)$

sqrt(x²-n) と sqrt(y²+n) は整数

が成立する場合に起こり、積と同様に要素範囲が狭く"ある程度大きい正数"で成立しにくくなる。前処理にシフトと圧縮を用いることによって、和によるビットの偏りと積によるオーバーフローを緩和できるため、「2乗和(ｼﾌﾄ圧縮)」は平均的にみて最もよい結果を出す。

「2乗和(ｼﾌﾄ圧縮)」でも識別率が十分でなか

ったので、関数の組み合わせを検討した。組み合わせ関数にはオーバーフローのほとんどない和を利用し、要素数が多い場合を考えて、

〈1〉 和(ｼﾌﾄ)＋２乗和(ｼﾌﾄ圧縮)

$$\Sigma shif(e_i) + \Sigma comp(shif(e_i))^2$$

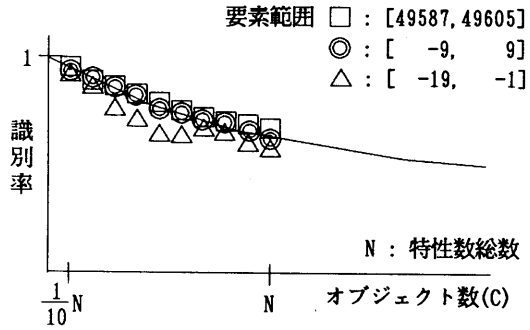
を採用した。これによって全体的に識別率が改善された。さらに、関数の組み合わせでは互いに性質の異なる関数を組み合わせたほうが識別率が高くなると思われる。そこで、

〈1'〉 和(ｼﾌﾄ)＋２乗和(圧縮)

$$\Sigma shif(e_i) + \Sigma comp(e_i)^2$$

についても検証を行ない、ほぼ等確率マッピング相当の識別率(0.615～0.637)を得た。

[付図１]に、和、積、２乗和およびそれらに前処理を加えた 10 個の関数と関数の組み合わせ〈1〉,〈1'〉についての検証結果を示す。また、要素数 10 の場合についてオブジェクト数を変えて識別率を検証し、等確率マッピングと比較したグラフを[図４]に示す。



[図４] 等確率マッピングとの識別率の比較

５．２ 集合の特性数関数

---比較平均からの検討

ここでは、１特性数あたりの衝突数の分布を調べ、複合オブジェクトの符号化に必要な一致判定の平均回数（比較平均）を用いて特性数関数を評価した。積は要素数が多くなると識別率が極端に下がるため検討対象からはずし、ここでは主に和と排他的論理和の比較を行なった。排他的論理和および２乗排他的論理和を基本的な関数に加え、１２の関数について評価した。つまり、前述の関数 [1],[3],[4],[8]～[10] および以下の [11]～[16] である。

[11] 排他的論理和 Λe_i

[12] 排他的論理和(ｼﾌﾄ) $\Lambda shif(e_i)$

[13] ２乗排他的論理和 Λe_i^2

[14] ２乗排他的論理和(ｼﾌﾄ) $\Lambda shif(e_i)^2$

[15] ２乗排他的論理和(圧縮) $\Lambda comp(e_i)^2$

[16] ２乗排他的論理和(ｼﾌﾄ圧縮) $\Lambda comp(shif(e_i))^2$

要素範囲を [0,255] (8 bit) とし要素数を 3 として、要素のすべての組み合わせからなる集合 2,763,520 個を 32 bit (4,294,967,296個) の特性数に割り当てて、比較平均、（１特性数あたりの）最大マッピング数、識別率の３つについて評価を行なった。要素数を 3 としたのは、要素範囲を 8 bit としてすべての組み合わせで集合を生成する場合、要素数が 3 を超えると集合数が多すぎて検証不可能なためである。また、特性数総数 〉〉 オブジェクト数とするために、特性数は 32 bit とした。

各関数の比較平均のオーダを [表２] に示す。最大マッピング数および識別率もほぼ比較平均に比例している。

関数の組み合わせでは、以下の４つの関数について検証を行なった。

〈1〉 和(ｼﾌﾄ)＋２乗和(ｼﾌﾄ圧縮)

$$\Sigma shif(e_i) + \Sigma comp(shif(e_i))^2$$

〈2〉 和(ｼﾌﾄ)＋２乗排他的論理和(ｼﾌﾄ圧縮)

$$\Sigma shif(e_i) + \Lambda comp(shif(e_i))^2$$

関数	比較平均
和 排他的論理和 ２乗和(ｼﾌﾄ) ２乗排他的論理和(ｼﾌﾄ)	2447 ～ 5396
２乗和 ２乗和(圧縮) ２乗排他的論理和 ２乗排他的論理和(圧縮)	12 ～ 45
和(ｼﾌﾄ) 排他的論理和(ｼﾌﾄ)	1.7 ～ 2.9
２乗和(ｼﾌﾄ圧縮) ２乗排他的論理和(ｼﾌﾄ圧縮)	0.2 ～ 0.3

[表２] 関数の比較平均(オーダ)

〈3〉 排他的論理和(ｼﾌﾄ)＋２乗和(ｼﾌﾄ圧縮)

$\Lambda shif(e_i) + \Sigma comp(shif(e_i))^2$

〈4〉 排他的論理和(ｼﾌﾄ)

＋２乗排他的論理和(ｼﾌﾄ圧縮)

$\Lambda shif(e_i) + \Lambda comp(shif(e_i))^2$

どの関数もほぼ 0.1(0.116～0.124) の比較平均を示し、和と排他的論理和ではあまり差がないが和のほうが若干よいことがわかった。識別率はほぼ 0.9(0.891～0.896) であった。関数の組み合わせでもっとも判別能力が高かったのは、「和(ｼﾌﾄ)＋２乗和(ｼﾌﾄ圧縮)」であり、最大マッピング数は 9、集合の約 80% が衝突なしで割り当てられた。実用に耐えうる数値と判断する。

5．3　入れ子型関係の特性数関数

入れ子型関係の特性数関数を検証することで、タプルにおける要素と属性の組 $[a_i:x_i]$ の特性を表現する関数

$Fa(a_i, x_i) = x_i * p(a_i)$　　$p(a_i)$は素数

を評価する。3．1で述べた検討方針に従うと、タプルの特性数関数は

$Ft1 = \Sigma shif(x_i * p(a_i)) + \Sigma comp(shif(x_i * p(a_i)))^2$

である。入れ子型関係は複数個の特性数を持つことを許すので、集合の特性数関数を以下のように２つの関数に分けて、ビットの重ね合わせによる情報損失を緩和した。

$Fr1 = \Sigma shif(Ft1)$

$Fr2 = \Sigma comp(shif(Ft1))^2$

入れ子型関係 100,000個について識別率を評価した。属性数 5以下、タプル数 100以下とし、

1) 2 属性 5 タプル
2) 5 属性 20 タプル
3) 5 属性 100 タプル

の３種類の入れ子型関係について評価した。要素範囲は、必要な数の入れ子型関係を生成するできるだけ少ない個数の要素を使用し、

1) 0 近傍
2) 小さい正数
3) 小さい負数
4) ある程度大きい正数

の４種類を用いた。入れ子型関係の大きさ(３種類)×要素範囲(４種類)＝１２種類のどの場合についても識別率 1.000 を実現した。

比較のために、従来用いられているシフトを属性情報の反映に利用した特性数関数

	素数との積 [Fr1,Fr2]	シフト [Fr1',Fr2']
2 属性　5 タプル		
要素[1, 6]	1.000	0.712
[-2, 3]	1.000	0.524
[-6,-1]	1.000	0.324
[49587,49592]	1.000	1.000
5 属性　20 タプル		
要素[1, 2]	1.000	0.993
[0, 1]	1.000	0.989
[-2,-1]	1.000	0.326
[49587,49588]	1.000	1.000
5 属性 100 タプル		
要素[1, 3]	1.000	1.000
[-1, 1]	1.000	0.351
[-3,-1]	1.000	0.254
[49587,49589]	1.000	1.000

［表３］ 入れ子型関係の特性数関数の識別率 (1)

	素数との積 [Fr3,Fr4]	シフト [Fr3',Fr4']
2 属性　5 タプル		
要素[1, 6]	1.000	0.054
[-2, 3]	1.000	0.030
[-6,-1]	1.000	0.045
[49587,49592]	1.000	0.054
5 属性　20 タプル		
要素[1, 2]	1.000	1.000
[0, 1]	1.000	1.000
[-2,-1]	1.000	1.000
[49587,49588]	1.000	1.000
5 属性 100 タプル		
要素[1, 3]	1.000	0.139
[-1, 1]	1.000	0.043
[-3,-1]	1.000	0.135
[49587,49589]	1.000	0.139

［表４］ 入れ子型関係の特性数関数の識別率 (2)

$Ft1' = \Sigma shif(x_i \langle\langle\langle i)$
$+ \Sigma comp(shif(x_i \langle\langle\langle i))^2$
$\langle\langle\langle$ は巡回シフト
$Fr1' = \Sigma shif(Ft1')$
$Fr2' = \Sigma comp(shif(Ft1'))^2$
についても評価を行なった。[Fr1,Fr2] と [Fr1',Fr2'] の識別率の比較を［表３］に示す。

次に、特性数の計算量を減らすために、タプルの特性数関数に利用する関数を「和(ｼﾌﾄ)＋２乗和(ｼﾌﾄ圧縮)」から「和(ｼﾌﾄ)」に変えたものについても評価した。つまり
$Ft2 = \Sigma shif(x_i * p(a_i))$
$Fr3 = \Sigma shif(Ft2)$
$Fr4 = \Sigma comp(shif(Ft2))^2$
である。この場合についても識別率は常に1.000であった。

これについても比較のために、属性情報の反映に従来用いられているシフトを利用した特性数関数
$Ft2' = \Sigma shif(x_i \langle\langle\langle i)$, $\langle\langle\langle$ は巡回シフト
$Fr3' = \Sigma shif(Ft2')$
$Fr4' = \Sigma comp(shif(Ft2'))^2$
の評価を行なった。[Fr3,Fr4] と [Fr3',Fr4'] の識別率の比較を［表４］に示す。

６．おわりに

複合オブジェクトの等値判定において、特性数による不等値オブジェクトの検出の手法を提案した。

集合の特性数関数として
〈1〉 和(ｼﾌﾄ)＋２乗和(ｼﾌﾄ圧縮)
$\Sigma shif(e_i) + \Sigma comp(shif(e_i))^2$
を提案し、識別率と比較平均について有用な結果を得た。タプルの属性情報として素数との積を用いる手法を提案し、関係の特性数関数の評価を通して、従来のシフトによる手法よりも高い識別率を出すことを示した。入れ子型関係の特性数関数として、
$Ft2 = \Sigma shif(x_i * p(a_i))$, $p(a_i)$は素数
$Fr3 = \Sigma shif(Ft2)$
$Fr4 = \Sigma comp(shif(Ft2))^2$
を提案し、識別率について有用な結果を得た。

比較平均の評価において、要素数を増やして一般化を図ろうとしたが、この条件で要素数を増やすと要素のすべての組み合わせによってできる集合の個数が膨大になり検証不可能である。すべての組み合わせではなく、無作為抽出のような方法を取るか、アプリケーション上での検証が必要になろう。

参考文献

[1] Paton,N. & Gray,P.: "Identification of Database Objects by Key", OODBS-88, pp.280-285, 1988.

[2] Bancilhon,F., Briggs,T., Khoshafian,S. & Valduriez,P.: "FAD, A Powerful and Simple Database Language", VLDB-87, pp.97- 105, 1987.

[3] Khoshafian,S. & Copeland,G.: "Object Identity", OOPSLA-86, pp.406-416, 1986.

[4] Eich,M.: "Main Memory Database Research Directions", IWDM-89, pp.251-268, 1989.

[5] Nakano,R. & Kiyama,M.: "MACH:Much Faster Associative Machine", IWDM-87 Proceedings, pp.482-495, 1987.

[6] Deshpande,A. & Gucht,D.V.: "A Storage Structure for Nested Relational Databases", Workshop on Theory and Applications of Nested Relations and Complex Objects, pp.69-83, 1987.

[7] Khoshafian,S. & Frank,D.: "Implementation Techniques for Object Oriented Databases", OODBS-88, pp.60-79, 1988.

[8] 渋谷,山本: "岩波講座情報科学-11 データ管理算法", 岩波書店, 1983.

[9] Knuth,D.: "The Art of Computer Programming, Vol.3 Sorting and Searching", Addison-Wesley Publishing Company, 1973.

〔付図1〕 識別率による関数の評価