AI 86−21

推論機能と関係データベースの融合方式

大森　匡・吉田　敦
田中英彦
（東大）

1986年7月25日

社団法人　電子通信学会

推論機能と関係データベースの融合方式

An approach to a relational database system integrated with the inference power

大森 匡　吉田 敦　田中 英彦

Tadashi OHMORI　Atsushi YOSHIDA　Hidehiko TANAKA

東京大学工学部

University of Tokyo

1. はじめに

Prologに代表される論理型言語はHorn節からなるデータベース上の検索と見なせる機能を核として持つ。この点に着目し，Prologを関係データベースシステムのフロントエンドとして用いることで大規模知識処理システムを構築することが考えられる。これは演繹データベースシステムと呼ばれ，知識ベースシステムなるものの有力な一形態とされている。

しかし，これをアーキテクチャとして支援する場合，単純にPrologマシンと関係データベースマシンとをチャネルで結合する方式では，大規模システムとして限界があり，両者の機能を一体化する方式が注目されている。

本論文では後者の立場から，まず演繹データベースマシン上のトランザクションによる計算モデルを明確にする。このトランザクションを支援する際には，単一化による推論機能と，関係代数による検索機能とが必要となる。そこで我々は，単一化を伴った自然結合演算を考え，これを支援することでこれら二つの機能を一つの基本プロセッサで実行することを提案する。また，ハッシュとソートによってこれをデータベースマシン上で実行するアルゴリズムについても述べ，他方式との比較・検討を行う。

2. 演繹データベースシステムの現状

2.1 結合方式と融合方式

Prologにおけるfact節は，述語ごとに関係データベースのrelationとみなすことができ，rule節は再帰定義を含んだview定義とみなすことができる。こうすることによって，functorによるcompound termの問題はあるものの，Prologの実行過程は関係データベースのqueryと見なせる。[1]

データベースシステムの分野では前者をbase relation，後者をcomputed relation，またはderived relationという。[2] 大量のファクト型知識をbase relationとして関係データベースシステムに格納し，与えられた質問をルール型知識を用いてbase relationへのqueryに変換することで，大規模なルール型知識，ファクト型知識を関係データベースを用いて扱うことができる。これは演繹データベースシステム（Deductive DataBase System 以下DDB）とよばれ，70年代後半から研究がおこなわれてきた。[3]

DDBのシステム構成方式は大別して結合方式（coupling）／融合方式（integrating）の二つに分かれる（Fig.1）。[4]

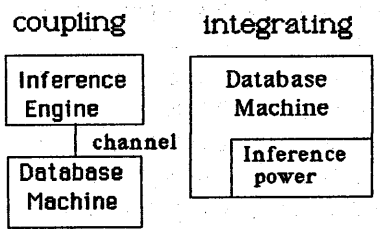


Fig.1 two configulations for DDB.

前者は推論マシン（IE）とデータベースマシン（DB）とをチャネルで結合し，IE側で推論を，DB側で検索を行う。基本的には，ルール型知識はIEのみにおかれ，ファクト型知識はDBに置かれる。対応する評価戦略としてはIEのルール型知識を用いて与えられた質問をファクト型知識のみへの質問に変換した後にDBに問い合わせる

compiled approach がある。[3][5]
後者はDB自身に推論機能を持たせる方式であり，ルール型知識，ファクト型知識ともDBに置かれる。　　対応する戦略としてはPrologインタプリタと同じ方式で動作するinterpretive approach がある。[3][6]

結合方式は現在の技術で容易に実現可能であり，既に多くの実験システムが報告されている。一方，融合方式は，簡単な実験システムは報告されているが[6]，完全な融合形とよべるものはない。ただし，両者のhybrid方式のシステムは報告されており，良好なシミュレーション結果が得られている。[7]

2.2 問題点

現在主流を占めている結合方式の問題点としては次の二点が挙げられる。

① IE・DB間のデータ転送が全体の性能低下をまねく。特に再帰的定義をされた computed relationへのquery 処理の場合，致命的である。

② compiled approach では問題によっては容易に探索空間が拡がる。

compiled approach はファクトへの質問の評価を遅らせることでDBへの問い合わせ回数をへらし，①の欠点を補うための方法ではあるが，複雑なルールを用いた質問処理では容易に探索空間の爆発が起こりうる。

総じて結合方式は，単純で少数のルール型知識と大量のファクト型知識とからなる場合にのみ有効であり，ルール型知識が大量にある環境では問題が多い。他方，融合方式は実行モデル自体が不明確であるが，一般的に言って推論機能とデータベース機能を一体化したために，上の①は回避でき，また，②は評価戦略としてinterpretive approach を加味することにより，より柔軟な処理が可能である。

これとは別に，現在のDDB研究自体にかけている視点として次の点を挙げることができる。

③ルール型知識の管理能力に限界がある。

現在のDDB は，結合／融合のいずれかにかかわらずルール型知識の取扱がview定義に限定されており，データベースの変更等を伴う大量で複雑なルール型知識のデータベースとしては不十分である。

ルール型知識をcomputed relation とみなせるものだけに限るなら，再帰的定義とfunctorによるcompound term の処理を除くと関係代数だけで実行可能である。さらに，再帰的に定義されたcomputed relation へのquery を関係代数のプログラムに変換する操作は最小不動点操作（Least Fixed Point Operation ）と呼ばれ，多くの研究者によってその部分的な質問クラスに対し実行アルゴリズムが得られている。[8][9]また，functor 処理は単純にcompound term にあわせてparsing,synthesizingを行うオペレータを導入すれば極めて容易に実現できる。[10]

こうするとcomputed relation はデータベースマシンだけで実行可能であり，効率の問題を除けば推論エンジンを用いる必然性はない。

しかし，これだけではルール型知識の取扱に限界があり，知識処理におけるデータベースシステムとしては不十分である。

知識ベースシステム（以下KB）を使用する実行主体はルール型知識，ファクト型知識を検索するとき次のような機能を必要とする。

- 検索した知識を用いてさらに推論，検索を行う。
- 新たに知識の検索，推論を行う実行主体を起動する。
- ファクト型知識へのread／write
- ルール型知識へのread／write
- 他の知識の実行主体との通信

この内，従来のDDB はルール型知識をview定義，即ちファクト型知識と同じとみなし，単一化・関係代数によるread機能を提供しているにすぎない。こうすると，KB上の実行主体にはKBだけでは上述した機能は提供されず，推論エンジン等の力が必要となる。

3.推論機能と関係データベースの融合

3.1 演繹データベースシステムにおけるトランザクション処理

前章に述べた様に，現在のDDB はルール型知識をcomputed relation とみなしている。　これに対し，ルール型知識をDDB におけるトランザクション（＝推論・検索を行う実行主体）の定義とみなし，このトランザクションを支援するシステムとしてDDB を考えることで，先述の条件を満たすことができる。即ち，

DDB-transaction ＝

- 検索した知識による推論
- DDB-transaction set の起動
- ruleへのread／write
- computed／base relationへのread／write
- 他のDDB-transaction との通信

となる（Fig.2）。
このとき，各DDB-transaction に必要な機能として単一化・縮退による推論機能と関係代数による検索機能が必要となる。

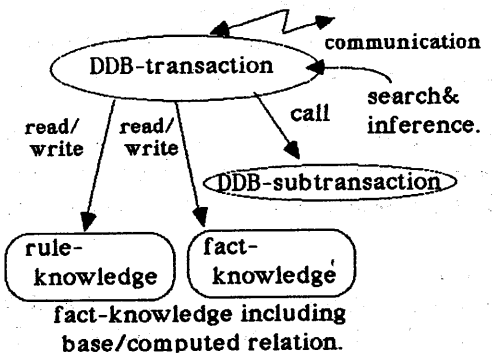


Fig.2 a DDB-transaction model.

3.2 単一化を伴う関係代数

DDB として前節に述べたようなトランザクションを支援するアーキテクチャを考える。

このとき，ユーザから与えられたタスクは，一つの共通したワーキングスペース上で協調して作業を行うトランザクションの集合としてDDB 上で動作する（Fig.3）。

ワーキングスペースはCAD データベースなどで用いられる概念である。[11] 一つのタスク（＝トランザクション集合）専用のローカルデータベースの役割を果たし，タスク内のデータのやりとり等に用いられる。

各トランザクションは常に推論・検索を行うため，これらの機能を結合方式のように別々のプロセッサで実行するよりも，融合方式のように一つの基本プロセッサ上で実行するほうがより自然であり，先述した結合方式の欠点も回避できる。

そのために，Fig.4 のように，単一化機能を関係代数の自然結合演算に組み込み，推論・検索の異なる機能をこの一つのオペレーションによって実行する。

単一化を伴う結合演算を用いてPrologマシンを構成する考えはICOTからも提案されている。[12] また，relation 中に変数を含んだタプルを許して結合演算の際に値を代入するという概念は不完全情報の取扱の手段として数多く提案されている。[2]

しかし，我々はFig.3 のような知識処理モデルを直接データベースマシン上で支援する。

そのために，推論・検索の実行主体としてDDB-transaction を考え，単一化を伴う関係代数演算を実行するような基本プロセッサでこれを支援する。この基本プロセッサから成るデータベースマシンを構成することで，従来のDDBに比べより自然にルール型知識，ファクト型知識を管理できる。

また，トランザクションという単位で支援することで，ルール型知識，ファクト型知識のread／write をデータベースの同時並行制御の概念から支援することも可能となる。[13][14]

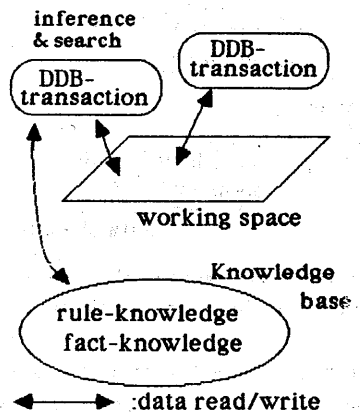


Fig.3 a DDB-transaction working model on Deductive Database System.

4.単一化を伴う自然結合の実行アルゴリズム [15]

4.1 データストリーム指向データベースマシン上の実行環境

以下では，原子論理式を属性値にとった関係A，B間の単一化を伴う自然結合（Fig.4）を例にとり，データストリーム指向の関係データベースマシンを用いてこれを実行するアルゴリズムについて述べ，他方式との比較を行う。

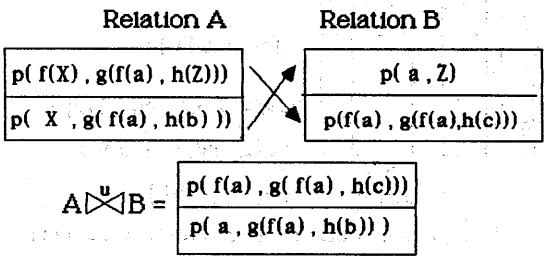


Fig.4 an example of join with unification

但し，各関係はハッシィング等によって述語・アリティ（引数の数）の等しいタプル集合に予め分割するとし，A，B共，述語・アリティ一定のタプル集合と仮定する。

また，実行環境としては，ハードウェアソータを用いてO（n）でマージソートを行うデータストリーム指向の関係データベースマシンを想定する（Fig.5）。[16]

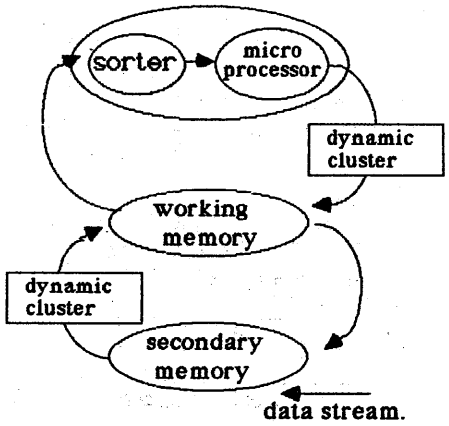


Fig.5 an abstract model of a data-stream oriented database machine.

即ちハードウェアソータ＋マイクロプロセッサ（または単一化プロセッサ）を，単一化を伴う関係代数を処理する基本プロセッサと見なし，これでデータベースマシンを構成する。
このとき，通常の自然結合A⋈Bは次の手順で実行される（Fig.6）。
step1．　二次記憶から一次記憶に関係A，Bのデータストリームを流す途中で　これらをハッシュサイズNでバケット$\{ai\},\{bj\}$（i＝1～T，Tはバケット数）に分割する。
step2．　対応するバケット同士のストリームai,bi（i＝1～T）を一次記憶からハードウェアソータに送りマージソートを行う。

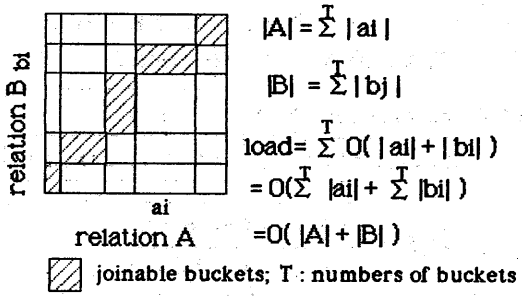


Fig.6 a join-algorithm by hash& sort.

こうすると
　処理量　＝　K（｜A｜＋｜B｜）
（Kはソートの性能係数，｜A｜，｜B｜は関係A，Bの濃度）

ソートの性能がO（n）のため，ハッシィングはハードウェアソータのソート容量を調整し，ソータの稼働率を上げるために行われている。
　単一化結合の場合は自然結合の場合と異なり，次の2点が問題となる。
①構造体を含む可変長レコード間のマッチング
②変数を含むレコード間のマッチング

先に，述語・アリティ一定としたのはこの差を強調するためであり，述語が異なるなら述語シンボルも含めてハッシュ，ソートを行えば良い。

4.2 ハッシィング
与えられた述語のアリティに応じて木構造（ハッシュ木とよぶ）をあらかじめ設定し，原子論理式をこれにあわせて展開する（Fig.7）。
これを構成するノードはタプルの分布に応じて木構造中から任意に指定できるとする。

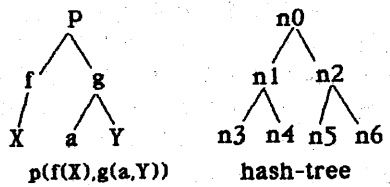


Fig.7 an example of parsed atomic formulae p（f（X），g（a,Y））corresponding to hash-tree of arity ＝2

　そして関係A，Bの各タプルtについて，ハッシュ木上の各ノード　n0，n1．．．に対応したtのシンボルa0，a1．．．を取り，ベクトル形式のハッシュ値（ハッシュベクトルと呼ぶ）〔h(a0)，h(a1)，...〕をタプルtに与えた後に，A，B各々のタプルをこのハッシュベクトルに応じて，バケット集合$\{ai\}$，$\{bj\}$にクラスタリングする。
但し，h（a）＝
$$\begin{cases} 0 & (a\text{が変数}) \\ int(a) \bmod N + 1 & (\text{その他}) \end{cases}$$
とし，　また，niに対応するシンボルがtに存在しないときはh（a）＝0　とする。
　例えば，Fig.7のハッシュ木上でタプルtがp（f（X），g（a,Y））ならtにはハッシュベクトル〔h(p)，h(f)，h(g)，0，0，h（a），0）が与えられる。
　従ってバケットaiのタプルとbjのタプルとが単一化可能であれば，各バケットに対応するハッシュベクトルをH（ai），H（bj）として，H（ai）とH（bj）の対応する要

素が互いに零でないときはこれらは一致しなければならない（Fig.8）。

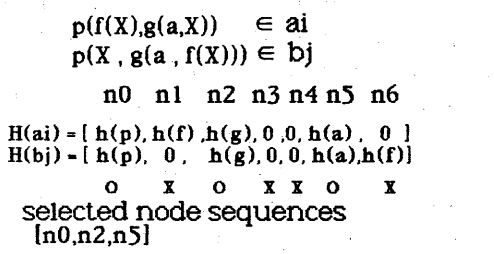


Fig.8 an example for unifiable hash-bucket

これをハッシュベクトルの単一化とみなすと，全体の単一化アルゴリズムはFig.9のようになる。

関係Aをクラスタリングしたバケット集合を $\{ai\}$ $(i = 1,..)$，関係Bのそれを $\{bj\}$ $(j = 1,...)$ とすると

$$\text{処理量} = K \sum_{(i,j)\in C}^{U} |ai| \cdot |bj| \quad \text{式（1）}$$

（但しKは定数，Cは単一化可能なハッシュベクトルH（ai）とH（bj）との組合せ集合，Uは単一化可能なバケット組合せ数）

この方法では，木構造を設定することで可変長レコードを固定長レコードに変換している。そのため，予め設定した木構造からはずれた部分で単一化不可能なタプルとの単一化試行は避けられない。

また，ハッシュ木はタプルの分布に応じて分散の大きいノードのみから構成することが望ましい。

```
Begin
   partition relation A&B into
   hash-buckets {ai} & {bj}

   for all  ai , bj
      if H(ai) is  unifiable with H(bj)
      then
(*) ->   unification-try  for  each
         combination of members
         in bucket ai,bj.
      endif.
   endfor
End.
```

Fig.9 an algorithm for join with unification by hashing.

4.3 ソーティング

単一化可能なバケットａｉ，ｂｊに対して，ハッシュベクトルH（ai）＝〔ｎｋ,..〕，H（ｂｊ）＝〔ｍｋ,...〕（ｋ＝1,..）とする。今，ｎｋ＝ｍｋ≠0となるノード列をキーとしてバケットａｉ，ｂｊのタプルをマージソートすると，ソート列中のキー同値類は ハッシュ木中での単一化可能なタプル集合である。この場合，Fig.9での（＊）の部分は下のようにかわる。

```
Begin
   mergesort all members in
   buckets ai & bj .
(*) -->
   unification-try for each
   combination of members
   in all key equivalent
   classes in  sorted stream.
End
```

従って本方式の処理量は，

$$\text{処理量} = K \sum_{(i,j)\in C}^{U} (|ai| + |bj|) \quad \text{式（2）}$$

（定数は式（1）に同じ）

Ｈ（ａｉ）＝〔 1，2，3，0，5，0，3 〕

Ｈ（ｂｊ）＝〔 1，2，3，4，0，5，0 〕

とすると，ハッシュ木中のノード列〔ｎ０，ｎ１，ｎ２〕をキーとしてバケットａｉ，ｂｊのタプルをソートすることになる。

この方法は，各バケット毎にハッシュ木上の変数の位置が予めわかっているため，『バケットａｉ，ｂｊ間の単一化可能なタプルはハッシュ木上で互いに変数でないノードのシンボルが等しい』という性質を利用して絞り込んでいるもので，式（1）に比してかなりの処理量の削減が期待できる。しかし，Fig.9のハッシィングのみ用いた場合と同じくハッシュ木上にないノードでシンボルが等しくないタプル間の単一化試行は避けられない。さらに，変数は全て0とし，例えばｐ（Ｘ，Ｘ），ｐ（Ｘ，Ｙ）はともに同じバケットにはいるので，ｐ（Ｘ，Ｘ）とｐ（ａ，ｂ）との単一化試行も避けられない。

4.4 評価式の導出

ハッシュとソートを用いて上述したアルゴリズムを実行する場合，

$$\text{処理量} = K \cdot \sum_{(i,j)\in C}^{U} (|ai| + |bj|)$$

＝K・U／T・T・（｜a｜＋｜b｜）

＝K・U／T・（｜A｜＋｜B｜）式（3）

（但し，K：ソータの性能係数
U：単一化可能なバケット組合せ数
T：各関係が分解されたバケット数
｜a｜，｜b｜：バケットai,bj の平均濃度
｜A｜，｜B｜：関係A，Bの濃度
）

となり濃度U／T・｜A｜とU／T・｜B｜間の自然結合と処理量は等しくなる（Fig.10）。

U／Tは，ハッシュ木の形，とくに木の深さと，1ノードあたりのハッシィングサイズNとで決定される。

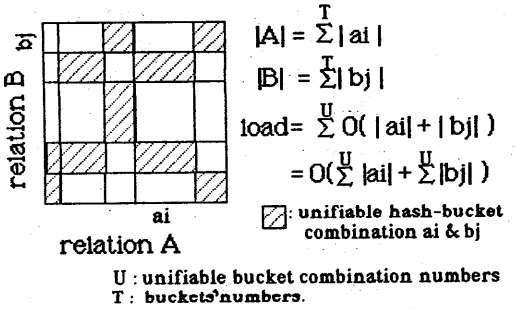


Fig.10 an algorithm for join with unification by hash& sort.

4.5 ハードウェアソータ上の実行に関する検討

4.1 に述べた様に，ハードウェアソータを用いる場合，その最大ソート容量を一杯に使用することが重要である。また，ソートしたい量がこの容量を越えるときは，これをハッシィングによって分割し，いくつかのバケットを合わせて一度にソートする量を調整しなければならない。

単一化結合では，ソート量調整の方法として次の三点が考えられる。（簡単のために1ノードあたりのハッシィングサイズN＝1として説明する。N＝1のため，任意のバケット組合せai，bjは単一化可能となる。）

① 各組合せai,bj のソート量が大きすぎる時は，ソートのキーとなるノード列nが決まった時点でnにたいして大きなハッシュサイズでハッシィングを行った後にマージソートする。

② キーとなるノード列が等しい，相異なるバケット組合せ（a1, b1），（a2, b2）は，組合せの識別子をキーにつけることで1つのストリーム（a1＋a2 ，b1＋b2）として扱える。

例えばFig.11のハッシュ木でノード列〔n0,n1〕をキーとするバケット組合せ
（a1 ，b1）＝（〔1,1,1〕，〔1,1,0〕），
（a2 ，b2）＝（〔1,1,0〕，〔1,1,1〕）と
（a3 ，b3）＝（〔1,1,0〕，〔1,1,0〕）は組合せ識別子00,01,10を各タプルのキー〔n0,n1〕の先頭につけることで〔n0,n1,n2〕をキーとする誤った組合せ（a1 ，b2 ）を避けることができ，一つの組合せ（a1＋a2＋a3，b1＋b2＋b3 ）としてソートできる。 これによって容量の小さな組合せは一つにまとめることができる。

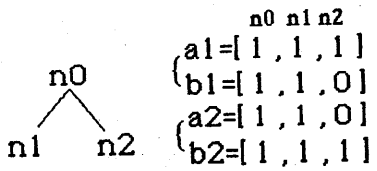


Fig.11 an example of unifiable buckets' combination with key=〔n0,n1〕

さらに，

③ キーとなるノード列nがハッシュ木の全ノード列でないときは，省かれたノード列でのハッシュ値が異なるバケットは一つにまとめることができる。

例えば，上のFig.11で〔n0,n1〕をキーとする組合せ
（a2 ，b1）＝（〔1,1,0〕，〔1,1,0〕），
（a2 ，b2）＝（〔1,1,0〕，〔1,1,1〕）は一つの組合せ（a2 ，b1＋b2 ）にすることができる。これは②と異なり組合せ識別子は不要で，実際に発生するストリームを減らすことができる。

ストリームの発生順序は，ハッシュ木上でキーとなるノード列n＝〔1,0,,,,,,〕を2進数とみなして，nの値の大きい順とし，それに上述した②，③の方法を用いて処理すれば良い。①は，N＞1のときは既にクラスタリングされているので行う必要はない。

またハッシュ木上でキーとなるノード列が述語のみとなる場合には仮定より述語は一定なのでソートは無効であるので省く。

4.6 問題点

上述したアルゴリズムは『単一化可能なタプル同士は対応する定数シンボルが等しい。』という性質を用いて，ハッシュとソートによってこのノード列が等しい部分を取り出す。故に，

ハッシュ木を構成するノードは
① ソート時にキーになる確率が高く，
② このノードによる絞り込み効果が大きい。
ことが重要である。

一方，ハッシュ木を十分大きくとると，単一化可能なバケット組合せ数が増大してしまう。例えば，Fig.12のようにアリティ2で深さleafのハッシュ木を用いた場合，$(const)^{2^{leaf-1}}$でU／Tが増加し，ソート時の負荷が大きくなる。

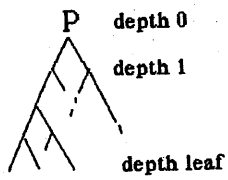


Fig.12 hash-tree of arity= 2 & depth = leaf

従って，ソート負荷とこれによる絞り込み効果との間のトレードオフを関係A,B双方のパタン分布から動的に最適化するようなハッシュ木の設定法が重要である。

5.単一化を伴う自然結合の実行アルゴリズムの評価・検討

5.1 評価方法

ここでは，単一化を伴う自然結合のアルゴリズムを，上述したハッシュとソートを用いる方式（以下，hash& sort法）と，ICOTから提案されている方式（以下ICOT法）とで簡単な評価を行う。

ICOT法は，Fig.13のように，タプルをbreadth firstに展開し，変数＞定数の全順序をつけてソートする方式である。[12] クラスタリングの方法は発表されていないため，ICOT法でのデータはクラスタリングは行っていない。

また，hash& sort法では，1ノードあたりのハッシュサイズは1とし，ソート時には4.5節の方式を用いた。

サンプルはノード毎に変数の出現確率を変えて作成した（Fig.14）。

今，関係A,B間の単一化を伴う結合を行うとして，評価指数には次の三点をとる。
・単一化負荷＝単一化試行数／｜A｜・｜B｜
・成功率＝ 成功数／単一化試行数
・ソート負荷
＝ソートしたタプル数／（｜A｜+｜B｜）

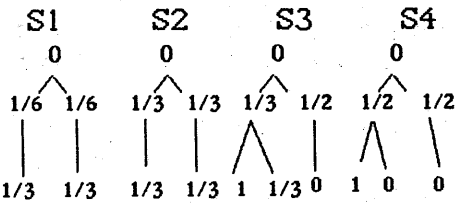
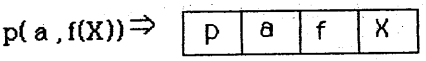


Fig.14 data-samples with treeform. fractional numbers are appearence-probabilities for variable symbols.



total-order for sorting

f(S) < g(T) iff f < g
$f(S^n) < f(T^m)$ iff n < m
$f(S^n) < f(T^n)$ iff $(s_1,...s_i) \approx (t_1,...t_i)$ and $s_{i+1} < t_{i+1}$
f(S) < X iff X is variable.
$f(S^n) \approx f(T^n)$ iff $S^n \approx T^n$

X≈Y iff X and Y have not appeared yet or X and Y have appeared in corresponding positions.

X < Y iff X has already appeared but Y has not, or X and Y have appeared but X has appeared before Y.

Fig.13 an algorithm for join with unification by ICOT .[12]

5.2 結果・検討

実験結果をTable.1に示す。これより，以下の三点がいえる。

① S1,S2のhash& sort法でのソート負荷をみると｜A｜・｜B｜が一桁違っても共に5倍程度と大差ない。これは，構造情報に基づいてクラスタリングしているためで，同じクラスタ内にタプルが集中しても O（n）のソートによってこの負荷を吸収できるからである。

即ち，hash & sort法では，ソート負荷はその木構造に大きく依存し，ソートする関係A,Bの大きさにはあまり影響されないことがわかる。

② hash& sort法がノード列中のどこで変数の出現確率が高くても総当たり時に比べ常に高い絞り込み効果を達成できるのに対し，ICOT法は優先順位の高いノードの変数確率に大きく影響される（S1-S4の単一化負荷の欄）。

Table 1. results of join with unification.
value = <ICOT> / <hash&sort>
( ) is default success ratio.

sample	load of unification	success ratio	load of sorting	|A|•|B|
S1	0.3 / 0.12	0.3 / 0.64 (0.08)	1.0 / 5.6	10^4
S2	0.63/0.42	0.61/0.82 (0.37)	1.0 / 5.3	10^3
S3	0.74 /0.35	0.34/0.72 (0.25)	1.0 / 3.0	10^3
S4	1.0 / 0.27	0.22/0.81 (0.22)	1.0/1.9	10^3

これはhash& sort法が予め変数／定数の構造情報に応じてクラスタリングすることで『大量にタプルを発生させる原因となる，　定数になりやすいノード』を選び，そこをキーにしてソートするのに対し，ICOT法はbreadth-firstという固定順序のためにこれができないためである。
③ICOT法は，単純なだけソート負荷が小さい。

①については，ソート負荷はハッシュ木の木構造と一ノードあたりのハッシュサイズで支配されることがわかっている。
hash& sort法とICOT法との最大の差は，『大量のタプルを発生させる原因となる定数ノードを選び，これを使って絞り込むことができるか』という点にある。変数になる確率が高いノードは，たとえ p(X,X) のように変数が共通して出現する場合でも大量のタプルを発生させることは少ない。これはPrologプログラムのrule節をみれば明らかである。一方で，一つのプロシージャを構成するrule節はHead部の構造，定数によって場合わけをおこなっており，hash & sort法のような構造情報によるクラスタリングが有効である。

さらに，非正規形relationにみられるようにタプルの属性値としてcompound termを許す一般的な場合でも，hash& sort法なら予めクラスタリングを行うことで絞り込み効果をあげることが期待できる。[17]
もう一つの差はソート負荷である。hash& sort法はICOT法にたいし常に2～6倍のソート負荷が必要となる。これについては，ソートと単一化の単位処理時間の比の問題であり，今後，より詳しい検討が必要である。

6.おわりに

本論文では，演繹データベース上のトランザクションによる知識処理モデルを明確にし，単一化を伴う関係代数によってこれを支援する考えを述べた。そして，データベースマシン上で単一化を伴う結合演算を実行するアルゴリズムについて述べ，他の方式との比較を行った。

その結果，ここで提案したhash& sort法がICOTから提案された方式に比べソート負荷は大きくなるが，常により小さな単一化試行数を達成できることが分かった。

今後の課題として，

- タプル分布情報によるハッシュ木の動的な最適化
- DDB-transaction モデルに基づく同時並行制御のシミュレーション

が重要である。

〔参考文献〕

1.Logic and Databases : A Deductive Approach
Gallaire Computing Surveys vol.16,No.2 '84
2.THE THEORY OF RELATINAL DATABASES
Maier Computer Science Press '83
3.Logic and Database
Gallaire et al. Plenum Press '78
4.Proceedigs of First International Workshop on Expert Database Systems '84
5.A PROLOG DATBASE SYSTEMS
Li Reseach Studies Press '84
6.An experimental Relational Database system based on logic Minker in 3.
7.On the Evaluation Strategy of EDUCE
Bocca SIGMOD 86
8.Universality of data retrieval languages
Aho,Ullman POPL 79
9.A TIME BOUND ON THE MATERIALIZATION OF SOME RECUSIVELY DEFINED VIEWS
Ioannidis VLDB 85
10. The Representation and Deductive Retrieval of Complex Object Zaniolo VLDB85
11. A transaction model supporting complex applications in integrated information systems
Klahold et al. SIGMOD85
12. A Model and an Architecture for a Relational Knowledge Base Yokota et al. ICOT-TR-141
13. Transaction Restarts in Prolog Database Systems
Acharya et al. SIGMOD85
14. NESTD TRANSACTIONS : AN APPROACH TO RELIABLE DISTRIBUTED COMPUTING Moss , '81 MIT Ph.D.
15. 推論機能と関係データベースの融合
—ハッシュとソートによる検索—
大森 他. 情報処理32回全国大会1M-6
16. Relational Algbra Machine GRACE
Kitsuregawa et al. LNCS 147 '82
17. Synthesis of Unnormalized Relation Incorporating More Meaning
Kambayashi et al. Information Sciences '83