データベースシステム 100−15
（1994．10．14）

関係の非明示的表現と演繹データベース質問処理への応用

井上 裕策　　岩井原 瑞穂

九州大学 大学院 総合理工学研究科 情報システム学専攻
〒816 春日市春日公園 6-1
Email:{yinoue,iwaihara}@is.kyushu-u.ac.jp

関係の非明示的表現とは，同じ集合となっている複数の部分関係を共有することにより冗長性を減らす表現法である．本稿では，二分決定グラフによる関係の非明示的表現法として，対数符号化と線形符号化の2種類を示す．この表現法を演繹データベース質問処理に応用することにより，記憶効率・時間効率の良い質問処理が可能になると考えられる．本稿では，これらを用いたボトムアップ評価法と，既存の処理法であるハッシュ結合法との比較を，実験により行った．その結果，本稿で述べる処理法が，線形グラフと密なランダムグラフの推移的閉包を計算する質問に対し，処理速度において優れているという結果を得た．

Implicit Representation of Relations and Its Application to Query Processing in Deductive Databases

Yusaku INOUE　　Mizuho IWAIHARA

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6-1 Kasuga-koen, Kasuga-shi, Fukuoka 816 Japan
Email:{yinoue,iwaihara}@is.kyushu-u.ac.jp

This paper proposes a method of representing relations implicitly using binary decision diagrams (BDD), which are data structures to represent logic functions compactly, and is widely used in computer-aided design (CAD) area. We consider utilization of BDD for query processing in deductive databases.

In this paper, we show two methods, called linear encoding and logarithmic encoding, to represent relations implicitly using BDDs. We compared the performances of these encodings with the traditional evaluation based on hash joins, and the proposed methods are faster than the hash-join-based methods over transitive closure queries on linear graphs and dense random graphs.

1　はじめに

演繹データベースの研究は，1970 年代後半に，主に関係データベースの機能を拡張したものとして始まった．現在では，主に

- データベースの高機能化
- 知識ベースシステムにおけるデータベース処理／管理

の観点から研究されている [1].

演繹データベースの質問処理における研究課題の一つに，処理の効率化がある．例えば，ボトムアップ評価法は，ルール集合を関係代数式の集合に変換し，これを繰り返し適用することにより最小不動点を求めるが，このとき負荷の高い結合が繰り返し用いられ，これが記憶量・処理速度両面での処理コストを大きくしている．

そこで本稿では，論理関数の表現法として提案されている二分決定グラフ (Binary Decision Diagram, 以下 BDD) を用いた演繹データベースの質問処理法について考察する．BDD には，多くの実用的論理関数の等価性および充足可能性判定を効率的に行うことができるという長所があるため，設計検証，論理合成など CAD の分野で広く応用されている [2].

本稿では，BDD が論理関数をコンパクトに表現し，しかも BDD 上での論理演算が効率的に実行できることに着目し，BDD を主記憶上の関係の索引として用いることを検討する．従来の索引法 (ハッシュ法，B 木等) は，関係の要素である組を索引の末端に 1 つづつ列挙する**明示的表現**であるといえる．それに対し，BDD を索引として用いる場合，同じ部分関係がポインタによる共有でまとめられ，重複性が削減される．このように部分関係の共有により重複性を削減した表現を**非明示的表現**と呼ぶことにする．

BDD を用いた関係の非明示的表現では，2 つの関係に対する関係演算がある場合，演算の入力の関係と結果の関係を指すそれぞれのポインタを組として記憶し，以前の演算結果をキャッシュしておくことにより，同じ演算の繰り返しを防ぐことができる．これは演繹データベースにおけるボトムアップ評価において繰り返し行われる結合などに対し，特に有効である．

本稿では，与えられた関係を非明示的表現に変換する方法として，対数符号化と線形符号化の 2 つを示す．さらに，BDD によって非明示的に関係を表現する場合の処理と，既存の処理系 (CORAL)，およびよく採用されるハッシュ結合を用いる方法との性能比較を，線形グラフおよびランダムグラフに対する推移的閉包ルールを例に取って主記憶ベースで行った．その結果，特に密なグラフに対して，本稿で述べる非明示的表現による方法が，ハッシュ結合に基づく処理法に比べ処理速度において優れているという結果を得た．

本稿では，まず2章で，BDD について簡潔な説明を与える．続く 3章では，BDD を用いて関係を非明示的に表現する方法について述べ，それらの関係に対する関係演算を実現する方法を述べる．4章では，本稿で述べる関係の非明示的表現に対する性能評価を実験によって行い，結果および考察をまとめる．最後に 5章では本稿のまとめを述べる．

2　二分決定グラフ

2.1　二分決定グラフとは

BDD とは，図 1のように，論理関数を非巡回有向グラフにより表現したものである．図 1(a)(b) の BDD は，いずれも論理関数

$$F = (x_2 \wedge x_1) \vee \overline{x_0}$$

を表したものである．

節点の集合は，**変数節点**の集合と**定数節点**の集合とに分けられる．図 1のように，変数節点は円囲みで，定数節点は四角囲みで表すことが多い．各変数節点は，その節点の表す変数と，2 本の有向枝をもつ．変数の値が 0,1 のときにたどる枝を，それぞれ **0 枝**，**1 枝**と呼ぶ．本稿では 1 枝は実線，0 枝は破線で示す．また，定数 0,1 を表す定数節点を，それぞれ 0-定数節点，1-定数節点と呼ぶ．

BDD は普通，図 1のように，論理変数に対して全順序が存在し，根から定数節点に至るすべてのパスについて，変数の順序がその全順序関係に従うような形で表す．このような BDD を**順序付き BDD**(Ordered BDD,OBDD) と呼び [2]，このときの全順序を**変数順**と呼ぶ．

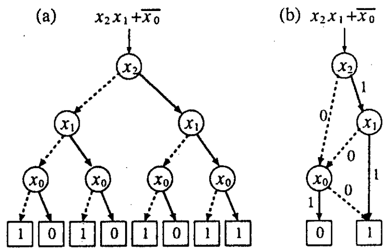

図 1: (a)BDD の例　(b)(a) に対する ROBDD

2.2　既約化ルール

OBDD は普通，図 1(a) のような形よりも，以下の既約化ルールによって既約化した形で用いることが多い：

① 併合ルール：同じ部分 BDD が 2 つ以上存在するならば，それらの BDD は 1 つにまとめる．
② 削除ルール：ある節点からの 0 枝と 1 枝とが同じ節点を指しているならば，その節点を削除する．

OBDD に対して，この既約化ルールを，もはや適用できなくなるまで繰り返し適用して得られる BDD を**既約な OBDD**(Reduced OBDD,ROBDD) と呼ぶ．図 1(b) は，図 1(a) の OBDD に対する OBDD を表したものである．

この ROBDD は，論理関数を一意に表すことができる (表現の一意性) という特徴をもつ [2]．それにより，2 つの論理関数の等価性は節点の比較だけで行うことができる．

本稿では以後，BDD とはすべて ROBDD を指すものとする．

2.3　計算機上での二分決定グラフの処理

BDD を計算機上で表現する場合，各変数節点は

① 変数番号 (論理変数名)
② 0 枝が指す節点 (の番号)
③ 1 枝が指す節点 (の番号)
④ 参照カウンタ (その節点を指している有向枝またはポインタの本数)

の 4 つのメンバからなる構造体として表される．上記の構造体を格納する記憶領域はハッシュテーブル (節点テーブル) として確保される．図 1の BDD に対する節点テーブルは，図 2のように表される．

処理中に新たな節点を生成する前には，必ずこの節点テーブルを参照し，等価な (上記のメンバ①②③の値がそれぞれ同じ) 節点が 2 つ重複して生成されない (併合ルールを適用する) ようにしている．

	① v	② f_0	③ f_1	④ refc	
0					← 0 定数節点
1					← 1 定数節点
2	x_0	1	0	2	
3	x_1	2	1	1	
4	x_2	2	3	1	

図 2: 節点テーブル

また，$f \wedge g, f \vee g, f \oplus g$などの演算の際に，オペランドの組 (f, g) と演算結果 h を記録する演算結果テーブル (図 3) を設け，同じ演算を繰り返し評価しないようにする技法も採られている．この演算結果テーブルもハッシュテーブルであるが，過去のすべての演算を記録するのではなく，最近の演算だけがキャッシュされる [3]．

	f	g	op	h
0	$x_1 \vee x_0$	x_1	$\wedge$	x_1
1	$x_1 \wedge x_0$	$\overline{x_2}$	$\vee$	$\overline{x_2} \vee x_1 x_0$
2	$x_2 \vee \overline{x_0}$	$x_1 x_0$	$\oplus$	$x_2 \vee x_1 \vee \overline{x_0}$
3	x_2	x_0	$\vee$	$x_2 \vee x_0$
4				
⋮	⋮	⋮	⋮	⋮

図 3: 演算結果テーブル

3 関係の非明示的表現

BDD を用いた関係の索引として用いる場合，関係を何らかの形で論理関数に変換し，それを BDD として表現することになる．論理関数への変換方法をここでは符号化と呼び，以下，対数符号化および線形符号化という 2 つの符号化法を示す．

3.1 対数符号化

対数符号化とは，関係を**特徴関数** [2] に対する BDD として表現する方法である．対数符号化では，n 個の属性値は $\log n$ ビットで表現される．

関係 Rに対する特徴関数は，次式で定義される．

$$f_R(\vec{t_j}) = \begin{cases} 1 & : t_j \in R \\ 0 & : t_j \notin R \end{cases} \tag{1}$$

ここに，$\vec{t_j}$は，組 t_jに対するビットベクトルである．

特徴関数 F_Rは以下のようにして構成する．

① 各属性値にビットベクトルを割り当てる．

② Rの各組 t_jに対するビットベクトル t_jを充足する積項 f_{Rj} を構成する．

③ 各積項 f_{Rj}の論理和が Rの特徴関数 F_Rである．

例題として，図 4の関係 $R(A, B)$ を考えよう．各属性値に対し，

MIKAMI → 10011 G6 → 010

R

A	B
SAKAI	G8
SASAKI	G6
MIKAMI	H6
MISAKI	G6

図 4: 例題：関係 R

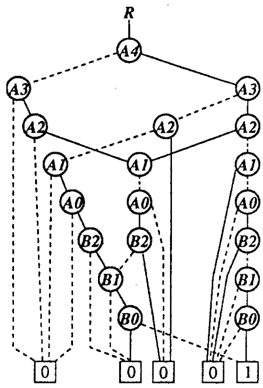

図 5: Rに対する対数符号化

MISAKI → 01101 G8 → 011

SAKAI → 11001 H6 → 110

SASAKI → 11101 (2)

のようにビットベクトルを与えるならば，Rに対する対数符号化は，図 5のようになる．

対数符号化の場合，和集合演算，共通集合演算は，それぞれ特徴関数どうしの論理和，論理積として実現される．BDD に対する AND 演算および OR 演算のアルゴリズムは，[3] に示されている．

3.2 線形符号化

もう一つの表現法として，各属性値を論理変数とした BDD として表現する方法が考えられる．即ち，図 4の関係 Rを，図 6あるいは図 7のように表現する方法である．この方法では，n 個の値は n ビットで表されている．このような非明示的表現を，**線形符号化**と呼ぶことにする．

図 6のような表現法には，大きな問題点がある．例えば，長さ n の英大文字列の集合を定義域にもつ属性に対する論理変数は，最大 26^n個必要となり，そのような多数の変数を扱う BDD の処理系の実現は困難である．

そこで，各属性値の文字列を何文字かごとに分解し，その各部分文字列を論理変数として表現する方法を考える．これを属性の**多段化**と呼ぶ．この手法を用いれば，図 4の関係 Rは，図 7のように表現できる．

多段化した場合は，しない場合と比べ，必要となる論理変数が少ない．例えば，長さ n の英大文字列の集合を定義域にもつ属性に対する論理変数は，各属性値の文字列を c 文字ごとに分解するならば，高々$26^c \cdot n/c$ 個しか必要とならない $(1 \leq c < n)$．

線形符号化の場合，0 枝は同じ属性 (多段化された場合は同じ段) の他の値の節点か，あるいは 0-定数節点を指す．一方，1 枝

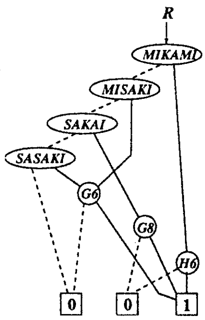

図 6: 関係 Rに対する線形符号化 (1)

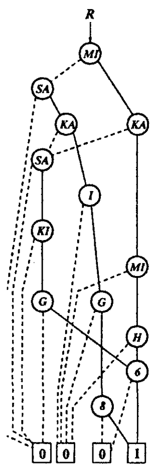

図 7: 関係 Rに対する線形符号化 (2)

は次の属性 (または次の段) の節点か 1-定数節点を指す．このため，2.2節で述べた規約化ルールのうち，削除ルールは適用されず，併合ルールのみが適用されることになる．

3.3 非明示的表現による索引の特徴

対数符号化および線形符号化による両者の索引に共通する特徴を以下にまとめる．

- TRIE 型の索引のように，BDD の根節点から定数節点までのパスが 1 つの組を表す．ただし TRIE は木構造で，BDD のような共有は行わない．
- BDD の根節点からのパスで，根節点に近い部分に位置する属性をキー属性とすることができる．
- BDD の下位に同じ部分関係が複数回現われるならば，併合ルールによりひとつにまとめられるため，記憶効率が改善される．
- BDD に対する集合演算および関係演算は，結果を演算結果テーブルにキャッシュすることにより，同じ演算が繰り返し評価されるのを防ぐことができる．これは，演繹データベースにおけるボトムアップ評価のように再帰的ルールから導かれる関係演算式を繰り返し適用する場合に特に有効である．キャッシュはBDD の節点単位で行われるので，関係の一部分に対しても演算の繰り返し評価を防げる．
- 前節では正規形の関係についての符号化法を述べたが，これらは非正規形の関係に容易に拡張可能であり，集合オブジェクトに対する索引としても用いることができる．

3.4 線形符号化された関係の上での演算の実現

3.4.1 和集合演算

線形符号化における和集合演算 $R_3 = R_1 \cup R_2$のアルゴリズムを以下に示す．ここに，F, G, Hは，それぞれ R_1, R_2, R_3を線形符号化した BDD である．また，x,yは，それぞれ F, G の最上位変数である．なお，共通集合，差集合の演算も同様なアルゴリズムとなる．

和集合演算アルゴリズム $Union(F, G)$

① $F = 0$ または $F = G$ ならば　$H := G;$
② $F = 1$ かつ $G = 1$ ならば　$H := 1;$
③ $G = 0$ ならば　$H := F;$
④ x と yが同じレベルならば
　$H|_{x=1} := Union(F|_{x=1}, G|_{x=1});$
　$H|_{x=0} := Union(F|_{x=0}, G|_{x=0});$
⑤ x が yよりも下位のレベルであるならば
　$H|_{y=1} := G|_{y=1};$
　$H|_{y=0} := Union(F, G|_{y=0});$
⑥ x が yよりも上位のレベルであるならば
　$H := Union(G, F);$

アルゴリズム中の $F|_{x=1}(F|_{x=0})$ は，関数 Fの変数 x の値を 1(0) に固定して得られる関数であり，Fの節点から 1 枝 (0 枝) をたどる操作に対応している．

3.4.2 自然結合

線形符号化された $R_1(A, B)$ および $R_2(B, C)$ に対して，

$$
\begin{aligned}
R_3(A,B,C) &= R_1(A,B) \bowtie R_2(B,C) \\
R_4(A,C) &= R_1(A,B) \underline{\bowtie} R_2(B,C)
\end{aligned}
$$

の実現の例題を図 8に示す．ここに $R \bowtie S$は 2 つの関係 R, Sに対する自然結合で，$R \underline{\bowtie} S$とは，$R \bowtie S$から結合属性 Bを射影で取り除いて得られる関係である．

3.5 対数符号化と線形符号化との比較

演繹データベース質問処理への応用の観点から，対数符号化と線形符号化との比較を，以下にまとめる．

対数符号化は，関係を特徴関数 (に対する BDD) として表現するもので，BDD による集合の表現法としてよく用いられている方法をそのまま適用したものである．特徴は，関係 $R(A_1, A_2, \ldots, A_n)$ において，

$$R \simeq dom(A_1) \times dom(A_2) \times \cdots \times dom(A_n) \qquad (3)$$

であるような場合，削除ルールが多く適用されるため，BDD 節点が少なくなることである．

一方，線形符号化の特徴は，以下の通りである．

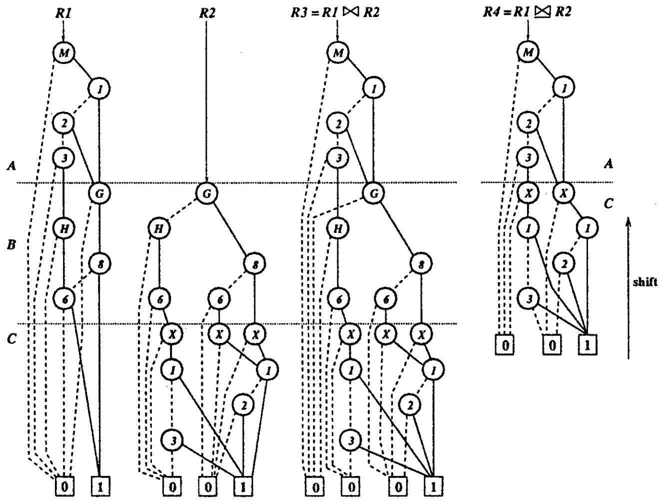

図 8: 例題：自然結合の実現

- 1 つの組に対する BDD 節点数が，対数符号化に比べて少ない．例えば，10000 個の属性値に対し，対数符号化では $\lceil \log_2 10^4 \rceil \simeq 14$ 個の変数が必要であるが，線形符号化は，10 個ずつの変数で多段化した場合 4 個でよい．
- 関係演算では，削除ルールは適用されないため，併合ルールが適用されない限り BDD 節点数は減少しない．したがって，式 (3) のような場合には，対数符号化に比べて不利となる．
- 逆に，式 (3) の右辺より Rがずっと小さい場合は，対数符号化では削除ルールを適用する機会が少なく，しかも 1 組あたりの変数が多くなるので，線形符号化のほうが有利になる．

2 つの符号化のいずれにおいても，基本的な集合演算 (和，差，共通) の計算時間の上限は，オペランドの 2 つの BDD について，共有されている部分を展開して得られる 2 つの木の節点数の和で抑えられる．実際は，共有によるキャッシュにより，これより高速化される．

3.6 BDD の変数順

BDD を構築する場合，まず BDD の各変数を根節点から並べる順序を定めなければならない．変数順序が質問処理の効率に影響を及ぼすため，以下に示す要素を考慮して変数順序を決定する必要がある．なお以下の議論は対数符号化と線形符号化に共通である．

3.6.1 属性分離

1 つの属性値は複数の変数で表わされているが，これらの変数は近接して並べる．例えば，関係 $R(X, Y, Z)$ について，$\{x_1, x_2\}$, $\{y_1, y_2\}$, $\{z_1, z_2\}$ を，それぞれ属性 X, Y, Z を表わす変数集合とすれば，変数順序として，$x_1, x_2, y_1, y_2, z_1, z_2$ と並べるようにする．図 5から図 7までの BDD はいずれもこの属性分離の変数順序である．このようにすると，関係演算の実現が容易になる．

3.6.2 結合

BDD の上位に置かれた属性は，索引のキー属性となる．このため，結合を行う場合は，結合属性を変数順の上位に置く必要がある．例えば，結合 $R_1(X, Y) \bowtie R_2(Y, Z)$ を計算する場合は，X, Y, Z の変数順にすることにより，R_2の BDD の Yの部分を索引として用いることができる．また，結合により BDD を探索するときは上位に置かれた関係から探索し，得られた値で下位に置かれた関係の索引を探索することになる．そのため，通常の結合順序の決定と同じように，組数の小さな関係を変数順の上位に配置することにより効率化できる．上の例では，R_1が選択により小さくなっている場合や，Semi-Naive 法の差分に相当する関係のとき，R_1を R_2より上に配置する．

3.6.3 変数のシフト

射影した関係を他の関係に代入するときなど，変数番号のつけかえが必要になることがある．例えば，図 8に示す演算 $\bowtie$ の場合，属性 Cの部分を上にシフトする．変数番号のシフトは，移動される BDD の節点数に比例する時間を要する．このため，ある演算の前にシフトするか後にシフトするかは，演算前および演算後の関係の大きさを予測して決める必要がある．

以上，属性分離，結合属性の位置，結合される関係の大きさ，変数をシフトする関係の大きさなどをもとに，与えられた関係代数式あるいはボトムアップ評価するプログラムから変数順序を決定する戦略を作ることができる．ただし非明示的表現の場合，表現の大きさを予測することが困難であるという問題がある．

4 実験および考察

本章では，3章で述べた方法による質問処理法と，他の処理法との比較を，実験によって行う．ここでは，演繹データベースシステム CORAL，理想的なハッシュ結合，およびビットマップによる結合を用いる方法である．例題としては推移的閉包質問を取り上げる．

4.1 CORAL における処理法

CORALはトップダウン・ボトムアップ両方の質問評価法を支援している [4][5]．ボトムアップ方式による不動点演算では，Semi-Naive 評価法 [6] を用いている．また，最適化のためのルールの変換については，プログラマが注釈 (annotations) によって指定することができる．

結合に関しては，各述語に対する索引を用いた入れ子ループ方式を用いている．この索引は，質問最適化システム (query optimizer) によって自動的に生成される．また，生成すべき索引を，プログラマが注釈で指定することもできる．索引には，メモリ内の関係に対する索引と，永続の関係に対する索引とがある．前者の索引はハッシュベースの索引で，後者の索引は B 木索引である．

4.2 ハッシュ結合法

ハッシュ結合法は，結合の実現法として，主記憶ベース／二次記憶ベースの両者で多く用いられている方法である．図 9は，ここでのハッシュ結合法に用いるハッシュテーブルを示したもので，同じキーをもつ組の集合は，図 9のように連結リストの形で表されている．今回の実験では，各バケットには 1 つのキー値のみが対応するようにしてあり，キー値の衝突が起きないため，ハッシュ結合法としては理想的な状態といえる．

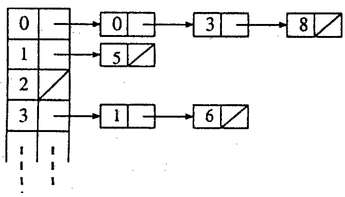

図 9: ハッシュテーブル

大きな関係を扱う時など，連結リストが長くなる場合は，連結リストへの挿入，マージなどのオーバヘッドが大きくなる．

4.3 ビットマップ結合法

ビットマップ結合法は，n 個の属性値からなる 2 項関係を $n \times n$ ビットのビットマップ (組があるときは 1，ないときは 0) で表し，この上で関係演算を行う方法である．関係が密なとき (1 が多いとき) は記憶効率がよいが，関係が疎のときは，$O(n^2)$ の記憶量ではオーバヘッドになる．

4.4 実験および結果

推移的閉包ルールは，

```
anc(X,Y) :- parent(X,Y).
anc(X,Z) :- anc(X,Y), anc(Y,Z).
```

と表すことができる．これに対して，質問

```
:-anc(X,Y).
```

が与えられた場合の計算時間を，CORAL，ハッシュ結合法，ビットマップ結合法，対数符号化法，線形符号化法で比較した．関係のインスタンスとしては以下の 3 つを用いた．

① 線形グラフ

② 節点数 N，有向枝数 $4N$の密なランダムグラフ

③ 節点数 N，有向枝数 Nの疎なランダムグラフ

密なランダムグラフは，その推移的閉包が完全グラフに近くなる．一方，疎なランダムグラフは，節点数 n について $50n$〜$200n$ 程度の有向辺数の推移的閉包となる．対数符号化および線形符号化については，符号化部，質問処理部を C 言語で記述し，BDD 処理部分については，京都大学で公開されている SBDD 演算パッケージを用いた．計算時間には，与えられた関係を符号化し，BDD を構築する時間も含めている．

4.4.1 線形グラフ

図 10は，節点数 Nの線形グラフに対する推移的閉包を求める場合の計算時間を，Nを 500 から 3500 まで 100 ごとに変えて調べた結果である．測定は Sparc Station 10 model 30 上で行った．

4.4.2 密なランダムグラフ

図 11は，節点数 N，有向辺数 $4N$であるランダムグラフの場合の計算時間を，Nを 500 から 1500 まで 250 ごとに変えて調べた結果である．サンプルは各 Nに対して 8 個づつ採り，各サンプルに対する時間を縦に並べてある．CORAL およびハッシュ結合法については，計算時間が大きすぎるため，比較対象から外した．測定は Sparc Station 10 model 40 上で行った．

4.4.3 疎なランダムグラフ

図 12は，節点数 N，有向辺数 Nであるランダムグラフの場合の計算時間を，Nを 4000 から 10000 まで 2000 ごとに変えて調べた結果である．サンプルは各 Nに対して 8 個づつ採り，各サンプルに対する時間を縦に並べてある．4.3節で述べたように，ビットマップ結合法は記憶効率が悪くなるため，疎なランダムグラフでは，代わりにハッシュ結合法を用いた．測定は Sparc Station 10 model 40 上で行った．

4.5 考察

以上の実験結果から読み取れる事項を，以下に簡潔に示す．

① 対数符号化，線形符号化ともに，BDD を構築するための時間は，全体の計算時間に対し，ごく小さな割合である．

② CORAL およびハッシュ結合法は，密な関係を計算する時の計算時間の増加が著しい．これは，長くなる連結リストを処理するための時間が大きいためと考えられる．

③ 線形グラフの場合，対数符号化および線形符号化は，BDD 節点の共有が起きやすいため，ビットマップ結合法を用いた方法に比べて高速である．

④ 密なランダムグラフで，節点番号差を制限すると，対数符号化では BDD 節点の削除ルールが多用されるために節点数が小さくなり，これが計算速度を高めているといえる．線形符号化の場合も，対数符号化と同様，節点の共有が起きやすくなっているため，やはりビットマップ結合法に比べて高速になっている．

⑤ しかし，疎なランダムグラフの場合，推移的閉包の有向辺数が比較的大きくなるときを除き，線形符号化，対数符号化は，ハッシュ結合法に比べて低速となる．これは，BDD 節点の共有が起きにくくなっているためであると考えられる．

⑥ 対数符号化は，推移的閉包が密になる場合は有効であるが，疎になる場合は計算時間が線形符号化よりも大きくなる．これは，3.5節で述べたように，対数符号化は疎なときに1組あたりの節点数が線形符号化よりも大きくなるためと考えられる．

⑦ 疎なランダムグラフの場合，大体においてハッシュ結合法が最も高速であったが，いくつかのグラフに対して，ハッシュ結合法の時間が急に増大しているのが読み取れる．これは，推移的閉包の有効辺数が大きくなったときの時間の増加の割合が大きいためであろうと思われる．それに対し，BDDを用いた方法は，そのような不安定さは見られなかった．

5 おわりに

本稿では，関係をBDDを用いて非明示的に表現する方法を示し，それを演繹データベースの質問処理に応用するための方法について議論した．

また，本稿で述べた方法が，既存の処理法よりも処理速度において優れている質問の例を，実験により示した．

ハッシュ結合法は密な関係に対し，ビットマップ結合法は疎な関係に対し，それぞれ著しい性能の低下が見られたが，非明示的表現による2つの方法は，いずれの場合に対しても安定した性能を示した．このことは，本稿の方法による演繹データベース処理系は従来のものより安定性の高いものになると予想できる．

本稿で述べた非明示的表現の方法のうち，対数符号化は，関係が各属性の定義域の直積に近いような場合には優れている．しかし，定義域が文字列である場合などは，符号化の手間が大きく，また，定義域の要素数が非常に大きくなると，削除ルールが適用される機会が少なくなるため，表現効率が悪い．その意味で，線形符号化のほうが対数符号化よりも実用性が高いといえる．

今回の実験では，多くの実装されている演繹データベースと同様に，主記憶ベースのボトムアップ評価を対象とした．今後は，二次記憶からのロード，符号化のオーバヘッドを考慮した性能評価を行うことが課題となる．

謝辞

熱心に御討論頂いた九州大学大学院総合理工学研究科の安浦寛人教授，村上和彰助教授をはじめとする安浦研究室の諸氏に深く感謝致します．また，SBDD演算パッケージ(Ver6.0)を御提供頂いたNTTLSI研究所の湊真一氏，および京都大学情報工学教室に感謝します．

参考文献

[1] 横田一正, 宮崎収兄. "新データベース論-関係から演繹・オブジェクト指向へ". 計算機科学／ソフトウェア技術講座, No. 4. 共立出版, 1994.

[2] 石浦菜岐佐. "BDDとは". 情報処理,Vol.34,No.5, pp. 585–592, 1993.

[3] 湊真一. "計算機上でのBDDの処理技法". 情報処理,Vol.34,No.5, pp. 593–599, 1993.

[4] R. Ramakrishnan, D. Srivastava, S. Sudarshan, and P. Seshadri. "Implementation of the CORAL Deductive Database System". In *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data*, pp. 167–176, 1993.

[5] R. Ramakrishnan, P. Seshadri, and D. Srivastava. *"The CORAL User Manual : A Tutorial Introduction to CORAL"*, 1993.

[6] Jeffery D. Ullman. *"Principles of Database and Knowledge Base Systems"*, Vol. I. Computer Science Press, 1988.

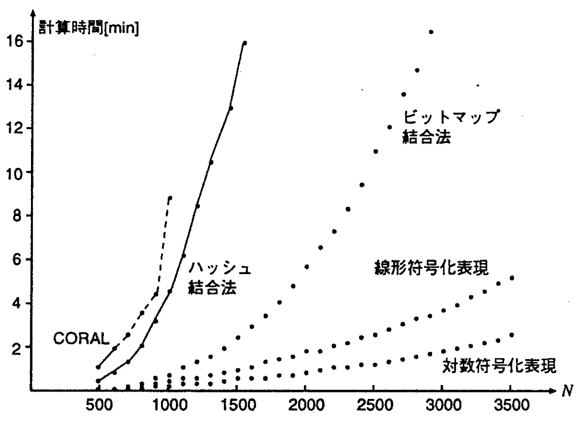

図 10: 節点数 N の線形グラフに対する推移的閉包の計算時間

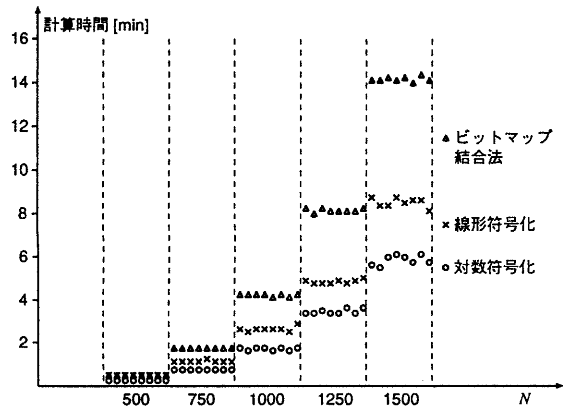

図 11: 節点数 N，有向辺数 $4N$ のランダムグラフに対する推移的閉包の計算時間

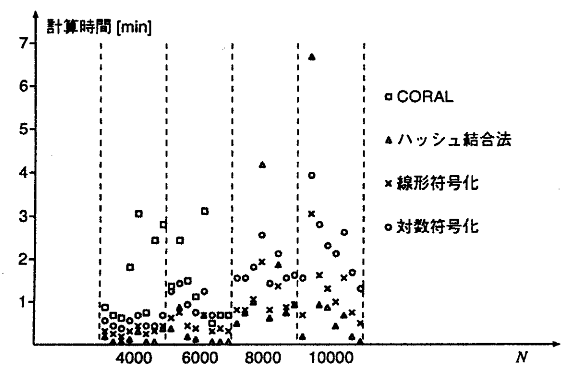

図 12: 節点数 N，有向辺数 N のランダムグラフに対する推移的閉包の計算時間