二分決定グラフを用いた演繹データベースの結合処理

1F－10

岩井原 瑞穂†, 井上 裕策‡, 安浦 寛人†

† 九州大学 大学院総合理工学研究科 情報システム学専攻
‡ 九州大学 工学部 情報工学科

1 まえがき

演繹データベースのボトムアップ質問評価においては，コストの大きな結合処理が多用されており，その結合処理の高速化のために，ハッシュ結合や並列化など様々な改良が試みられている．またボトムアップ評価時に現われる中間ファクトの記憶コストも無視できない問題である．

一方，論理回路設計および設計検証の分野では，Akers が考案し，Bryant[1] が効率的な演算法を示した，二分決定グラフ (Binary Decision Diagram; BDD) と呼ばれる論理関数の記憶効率の良い表現が注目されている．これにより従来扱うことのできなかったサイズの論理関数の処理が可能になっている．

本稿では，関係を論理関数に変換し，さらに BDD として表現することにより，演繹データベース質問処理中の中間ファクトを記憶効率良く表現し，BDD 上の演算によりボトムアップ質問評価する方法を示す．推移的閉包の実験結果では，従来のビットベクトルを用いたハッシュ結合による方法と比べ，BDD による方法は記憶効率が良く，多くの場合高速であるとの結果を得た．

2 二分決定グラフ

真理値表や積和表現などに代る論理関数の表現法として，BDD がある．BDD は変数節点 (非終端節点；図1中は丸) および定数節点 (終端節点；図1中は四角) およびその間の有向枝からなる非巡回有向グラフである．変数節点には論理変数名がラベル付けされており，変数節点からは 0 枝および 1 枝とよばれる 0 または 1 でラベル付けされた有向枝が張られており，それぞれラベルの変数が 0 または 1 の値を取ったときの行き先を示す．2 つの定数節点はそれぞれ論理値の 0 および 1 を表わす．与えられた入力値から，BDD の表わす論理関数の値を計算するには，最上位の節点から順に現われる変数節点に入力値に従って 0 枝または 1 枝のどちらかを降ってゆき，最終的に到達した定数節点のラベルがその論理関数の値となる．

最上位の節点から表われる変数の順序を一意に指定し，しかも既約化と呼ばれる冗長な節点を除く操作を行なった BDD を既約な順序付き BDD(reduced ordered BDD; ROBDD) と呼ぶ．ROBDD の重要な性質として，論理関数および変数順序を定めれば，ROBDD は一意に定まることであり，つまり論理関数の標準形となっていることである．以下, ROBDD を単に BDD と呼ぶ．加算などの多くの実用的な論理関数が n 個の論理変数に対して，$O(n)$ 個の節点数で表現可能である．また変数順序が BDD の節点数に大きな影響を及ぼし，加算でも変数順序によっては 2"個以上の節点数となる．和差積などのブール演算は，2 つの入力 BDD の節点数の積に比例する時間で可能である．

"Join for Deductive Databases Based on Binary Decision Diagram," M. Iwaihara, Y. Inoue and H. Yasuura, Kyushu University

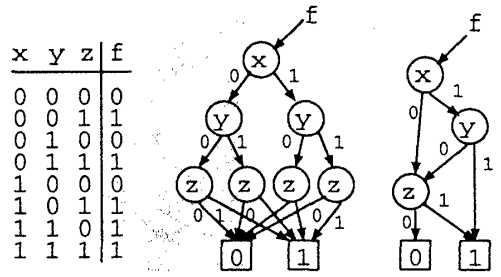

図 1: 論理関数 $f = x \cdot y + z$ の表現

図1に同一の論理関数の (a) 真理値表, (b) 既約でない BDD, (c)ROBDD をそれぞれ示す．

3 二分決定グラフを用いた結合処理

演繹データベース言語 DATALOG においては，質問処理技法としてマジックセット法などによるルール変換を行なうコンパイル時最適化と，ルールをボトムアップ評価する実行時最適化がある．ボトムアップ評価は，ルールを射影・選択・結合などからなる関係代数式に変換し，新らしいファクトが得られなくなるまで繰り返し関係代数式を評価する不動点演算が基本となる．

上記ボトムアップ評価を BDD を用いて行なう方法を以下に示す．まず，k項の関係 Rについて，その各属性 $X1, \ldots, X_k$ の値を適当な二進数に符号化する．外延データベースに記憶されている各属性の異なる値の数により，必要な二進符号の桁数が求まる．ある値 vの二進符号表現を$\tilde{v}$で表わす．関係 R は以下の特徴関数と呼ばれる論理関数 f_Rで表現できる．

$$f_R(\tilde{v_1}, \ldots, \tilde{v_k}) = 1 \iff (v_1, \ldots, v_k) \in R$$

上記特徴関数を BDD で表現することができ，もとの関係に同じ値の組み合わせが繰り返し現われるなどの冗長性が含まれる場合，BDD の部分グラフの共有が生じ，冗長性が削減されることが期待できる．

2 つの関係 $R_1(X, Y)$ と $R_2(Y, Z)$ について，属性 Yについて (自然) 結合 $R_1 \bowtie R_2$を計算する場合は，BDD では両者の特徴関数の論理積，すなわち $f_{R1}(\tilde{X}, \tilde{Y}) \cdot f_{R2}(\tilde{Y}, \tilde{Z})$ を計算することに等しい．選択については，特徴関数のある論理変数に 1 または 0 を代入して得られる論理関数を計算することに等しい．射影は，除かれる属性に対応する論理変数をドントケアとした論理関数を計算することに相当する．

上記の演算はすべて BDD の基本演算で実現できるが，基本演算をそのまま利用するよりも，関係代数式に応じて直接 BDD をグラフ探索した方が高速になる場合が多い．

4 推移的閉包問題

演繹データベースの古典的例題である推移的閉包の計算を実験の例題として用いた．二項関係 r(X, Y) の推移的閉包は，以下のルールで計算される．

```
a(X, Y) :- r(X, Y).
a(X, Y) :- r(X, Z), a(Z, Y).
```

上記ルールの評価法のひとつとして，左辺を関係代数式に変換して評価し，得られるファクトを再帰述語 a に追加してゆき，それを a のファクトが増えなくまで繰り返す Naive 法がある．また得られたファクトから a の差分を取り，その差分のみで左辺を評価する Semi-Naive 法が知られているが，ここでは前者を反復法，後者を差分反復法と呼ぶことにする．

推移的閉包は以下のルールによっても計算でき，この場合 r(X,Y) のグラフの最大経路長 n に対し，$\log n$ 回の反復で収束するという性質がある．

```
a(X, Y) :- r(X, Y).
a(X, Y) :- a(X, Z), a(Z, Y).
```

上記ルールの Naive 法による評価を反復二乗法， Semi-Naive 法による評価を差分反復二乗法と呼ぶことにする．

5 実験

前節の推移的閉包を計算する 4 つのアルゴリズム (反復法，差分反復法，反復二乗法，差分反復二乗法) について， BDD による方法と，ビットベクトルを用いたハッシュ結合による方法 (ベクトル法) の計算時間および記憶量を比較した．ここでベクトル法とは，各反復の結合演算について，結合属性 (ルール中の Z) を BDD と同じ二進数に符号化し，そのビットベクトルを用いて直接アドレス法によるハッシュ結合を行なう方法である．

関係 r(X,Y) の入力インスタンスとして， (1) 単一閉路グラフ (円形でその推移的閉包は完全グラフ) (2) 完全二分木 (n 節点のとき最大経路長は $O(\log n)$)， (3) ランダムグラフ (枝の確率 3%で 5 つのグラフを生成) を用いた．

BDD は変数順序により節点数が大きく変るため，2 種類の変数順序について調べた．変数順 A は節点番号の MSB が BDD で上位に，LSB が下位に来るように順番にならべたものである． 変数順 B は逆に節点番号の LSB が BDD の上位に，MSB が下位に来るように順に並べたものである．

BDD による方法は，京都大学で公開されている SBDD パッケージ [2] を用いて C 言語で実装した．使用したワークステーションは Sparc Station 10 model 30 である．

SBDD パッケージではあらかじめ BDD 節点のハッシュテーブルの大きさを指定できる．そのため，(1) ベクトル法と同程度の記憶量とした場合， (2) (1) の 60%から 8%程度までテーブルを縮小した場合について調べた．BDD 法の欄の下にそのテーブルの大きさ (節点数) が示してある．BDD 法の第 1 行が (1) の場合，第 2 行が (2) の場合である．SBDD パッケージでは，BDD 節点が不足するとガベージコレクションを行なって，参照されてない BDD 節点を回収する．そのため，テーブルの大きさを縮小すると実行速度が低下することが予測される．

また単一閉路の場合の各アルゴリズムについて，各反復ごとにガベージコレクションを行ない，各反復終了時に保持している BDD 節点数を求め，その最大値を取ることにより，BDD 法に必要な最小記憶量を求めた (表 2)．

6 考察

前節の実験結果から次の傾向が読み取れる．単一閉路のように推移的閉包が密になるグラフでは， BDD 法とベクトル法ともに，差分反復二乗法，反復二乗法，差分反復法，反復法の順に高速であるが，二分木や疎なランダムグラフの場合アルゴリズムよる大差は見られなかった．

方法 (テーブル)	変数順	反復法	反復二乗法	差分反復法	差分反復二乗法
BDD (48000)	A	501.30	5.13	33.88	3.13
	B	38.95	4.38	23.05	4.89
BDD (16000)	A	1111.16	5.64	40.00	3.53
	B	45.02	5.64	30.37	6.37
ベクトル法	—	50.99	14.78	30.18	8.07

表 1: 256 節点単一閉路の推移的閉包計算時間 [sec]

	変数順	反復法	反復二乗法	差分反復法	差分反復二乗法
BDD の最小節点数	A	1069	571	1323	952
	B	1394	510	1521	1017

表 2: 256 節点単一閉路の最小必要記憶量 [BDD 節点数]

方法 (テーブル)	変数順	反復法	反復二乗法	差分反復法	差分反復二乗法
BDD (66000)	A	0.69	0.49	0.23	0.25
	B	1.27	0.92	0.93	0.70
BDD (8192)	A	1.44	0.82	0.27	0.26
	B	2.17	1.74	1.25	0.94
ベクトル法	—	4.97	3.29	4.84	3.14

表 3: 512 節点完全二分木の推移的閉包計算時間 [sec]

方法 (テーブル)	変数順	反復法	反復二乗法	差分反復法	差分反復二乗法
BDD (12000)	A	5.63	1.45	4.68	1.45
	B	1.06	1.64	1.69	1.79
BDD (8000)	A	6.26	1.55	4.98	1.57
	B	1.08	1.80	1.74	2.00
ベクトル法	—	0.81	3.78	0.37	1.19

表 4: 128 節点ランダムグラフの推移的閉包計算時間 [sec]

ランダムグラフの場合を除いて BDD 法の方がベクトル法よりも高速であり，また BDD 法でテーブルの大きさの制限により速度が低下した場合でも BDD 法の優位は変らなかった．そのため，記憶量および速度の両面について BDD 法は有効な方法であるといえる．表 2を見ても，BDD が記憶効率良く中間ファクトを表現していることがわかる．

BDD 法の変数順序に関しては，順序 A と順序 B に関して，最大 24.6 倍最小 1.0 倍の速度差となっており，変数順序が速度に及ぼす影響は大きいといえる．

7 まとめ

以上，BDD を用いた推移的閉包の計算実験を通して，その計算速度および記憶量両面での演繹データベース質問処理での有効性を示した．演繹データベースのように，入力データに繰り返し演算を適用する場合は，符号化のためのオーバーヘッドが補償できるため，BDD のような記憶効率のよい符号化は有効である．今回は主記憶ベースで議論したが，BDD を用いた場合ベクトル法のような参照の局所性がなくなるため，二次記憶を前提とした方法が課題となる．

参考文献

[1] Bryant, R. E., "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.

[2] 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, May 1993.