データベース解析のためのゼロサプレス型二分決定グラフの簡単化に関する考察

Considerations of Optimizing Zero-Suppressed Binary Decision Diagrams for Database Analysis

岩崎 玄弥 † 湊 真一 † ツォイクマン トーマス †

Haruya Iwasaki Shin-ichi Minato Thomas Zeugmann

Abstract: Recently, an efficient method of database analysis using Zero-suppressed Binary Decision Diagrams (ZBDDs) has been proposed. BDDs are graph-based representation of Boolean functions, now widely used in system design and verification area. Here we focus on ZBDDs, a special type of BDDs, which are suitable for handling large-scale combinatorial itemsets in transaction databases. The ZBDD size greatly depends on variable ordering. In this paper, we investigate the property of ZBDD variable ordering for the histogram of item sets in the database, and consider good ordering methods.

1. はじめに

近年，大規模記憶装置の発展などによって，大規模なデータベースの中から有用な規則を発見するデータマイニングの研究が盛んになっている．頻出パタンマイニング (Frequent Pattern Mining) は，最も基本的なデータマイニング問題であり，Agrawal 等 [1] による Apriori アルゴリズムの研究に始まり，現在までに様々なアルゴリズムが提案されている [7, 3].

本稿では，VLSI CAD の分野で大規模論理関数データの表現法として広く用いられている二分決定グラフ (BDD: Binary Decision Diagrams)[2]，その中でも「ゼロサプレス型 BDD」(ZBDD: Zero-suppressed BDD)[4] と呼ばれるデータ構造を用いて，トランザクションデータベースにおけるアイテムセットの頻度表を効率よく生成する手法 [6] に関して，実験と考察を行った．ZBDD は大規模な組合せ集合データを非明示的に列挙し，効率よく演算処理を行うことができる．また，数式により組合せ集合の演算を記述し，これを ZBDD を用いて計算するプログラム「VSOP」[5] を用いた．

これまで，ZBDD におけるアイテムの順序付けが，生成される ZBDD の大きさに顕著な影響を与えることは知られていたが，実際にどのような場合に顕著な影響があるのかははっきりとは調べられていなかった．そこで，ZBDD のデータベースの頻度表表現に特有の順序付けの性質について調べた．また，データベース中にあまり出てこない非頻出アイテムの削除の影響についても実験を行った．さらに，データベースの特徴に基づくアイテム変数の順序付け法の考察についても述べる．

†北海道大学 大学院 情報科学研究科

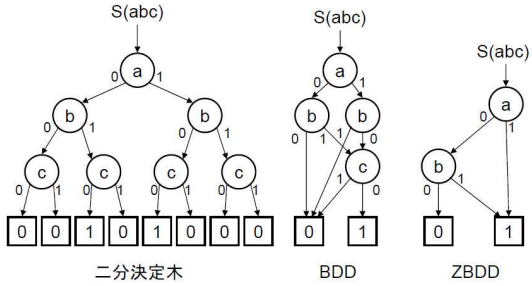


図 1: 二分決定木と BDD,ZBDD

2. ZBDD によるデータベース表現

2.1 BDD

BDD[2] は，図 1 に示すような論理関数のグラフによる表現である．一般に，論理関数のそれぞれの変数に 0,1 の値を代入した結果を，二分岐の枝（0-枝/1-枝）で場合分けし得られる論理関数の値を，2 値の定数節点（0-終端節点/1-終端節点）で表現すると，図 1 のような二分木状のグラフになる．このとき，場合分けする変数の順序を固定し，冗長な節点の削除と等価な節点を共有するという 2 つの縮約規則を可能な限り適用することにより，「既約」な形が一意に得られることが知られている．複数の論理関数を表す BDD の間においても，変数順序を固定すればグラフを共有することができる．

BDD は，多くの実用的な論理関数を比較的少ない記憶量で一意に表現することができる．また，2 つの BDD を入力とし，それらの二項論理演算の結果を表す BDD を直接生成するアルゴリズムが考案されている．このアルゴリズムはハッシュテーブルを巧みに用いることで，データが計算機の主記憶の範囲に収まる限りは，その記憶量にほぼ比例する時間内で論理演算を効率よく実行できる．

2.2 ZBDD

BDD は元々は論理関数を表現するために考案されたものだが，これを用いて組合せ集合データを表現・操作することもできる．組合せ集合とは，「n 個のアイテムから任意個を選ぶ組合せ」を要素とする集合である．

これを BDD で表現するとき，類似する組合せが多ければ，部分的に共通する組合せがグラフ上で共有されて，記憶量や計算時間が大幅に削減される場合がある．さらに，組合せ集合に特化した ZBDD[4] を用いると，より簡潔な表現が得られ，一層効率よく扱うことができる．

ZBDD では，冗長な節点を削除する簡約化規則が通常の BDD とは異なる．ZBDD では 1-枝が 0-終端節点を直接指している節点を取り除く，という規則になっている．これにより，ZBDD では図 1 のように，組合せ集合に一度も選ばれることのないアイテムに関する節点が自動的に削除されることになり，BDD よりも効率よく組合せ集合を表現・操作することができる．

2.3 VSOP

VSOP(Valued-Sum-Of-Products)[5] は，数式により組合せ集合の演算を記述し，これを ZBDD を用いて計算するプログラムである．VSOP は非常に多数のアイテム変数の組合せを要素とする積和集合を扱うことができ，また，単なる集合演算だけでなく，集合の各要素に整数値（係数あるいは重み）を定義し，加減乗除や大小比較などの算術演算を含む数式を処理できることを特徴とする．

ここで，ZBDD を用いて重み付き積和集合を非明示的に表現する方法について述べる．ZBDD は組合せ集合を表すことしかできないため，そのままでは重みを表現できない．これの解決法として，本稿では整数値を 2 進数に分解する方法を用いる．n 個の ZBDD$\{F_0, F_1, \ldots, F_{n-1}\}$ をベクトル状に並べると，最大 $(2^n - 1)$ までの重みを表現することができる．すなわち，整数値の重みを 2 進数で符号化し，最下位ビットが 1 になる組合せ集合を F_0，次のビットが 1 になるような組合せ集合を F_1，という形で F_{n-1} まで並べることにより，図 2 のように，各項の重みを非明示的に表すことができる．

2.4 ZBDD によるデータベース表現

VSOP を用いることで，トランザクションデータベースにおいて頻度表の生成を効率よく行うことができる．本稿では，タプル頻度表とパタン頻度表について実験，考察を行う．

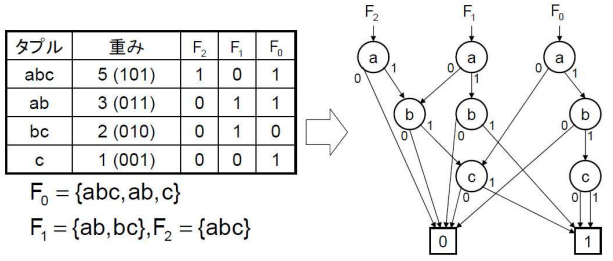


図 2: $(5abc + 3ab + 2bc + c)$ の 2 進ベクトルによる表現

タプル頻度表とは，トランザクションデータ中に同じアイテムの組合せ（タプル）が何回出現したかを数えて表にしたものであり，パタン頻度表とは，各タプルの中に出現するアイテムの部分集合（パタン）の出現回数を数えて表にしたものである．通常の組合せ集合では，組合せパタンの有無のみに着目し重複は考慮しないが，実際のデータベースでは同じタプルやパタンが複数回出現することがしばしばあり，その出現回数を数えることが必要な場合がある．このような場合に重み付き積和集合を用いると頻度表を効率的に生成することができる．また，一般に k 個のアイテムからなるタプルは 2^k 個のパタンを含んでいるので，パタン頻度表はタプル頻度表よりもはるかに大きなものになり，小規模な場合を除いて完全なパタン頻度表を生成することは困難である．しかし，ZBDD を用いる場合，多数の類似するパタンをグラフで共有してコンパクトに表現できるので，ある程度の規模までパタン頻度表を現実的に生成できる可能性がある．

3. データベース表現における ZBDD の簡単化

前述のように，ZBDD を用いることで頻度表を効率よく生成できる可能性があるが，データベースが大規模になると，頻度表を生成することが困難になる．これを改善する手法として，本稿では，「非頻出アイテムの削除」と「アイテム変数の順序付け」ということを考える．

3.1 非頻出アイテムの削除

実際のデータベースにおいては，アイテムの出現頻度に大きな偏りがある場合が多く見られる．このとき，出現頻度の低いアイテムを取り除くことによって，ZBDD のデータ構造を小さくしようと

いうのが「非頻出アイテムの削除」である．その方法として，適当なしきい値 α を指定して，それより多く出現するアイテムだけを用いて頻度表を生成する，ということを行う．本稿では，しきい値をある値に定めたときに，どの程度のデータの縮約効果があるのか，また，演算処理にかかる時間がどう変化するのかについて調べた．結果は 4 章に示す．

3.2 アイテム変数の順序付けの影響

本稿で使用した VSOP プログラムでは，データベースに含まれるをアイテムをアイテム変数として最初に宣言する．宣言されなかった変数は，その変数が算術式中で使用されたときに，その場で新たに宣言したものとして，最上位に追加される．アイテム変数の順序付けは，生成される ZBDD のサイズに影響を与える．

ところで，ZBDD の性質上，最悪な順序付けを用いても節点数はパタンを羅列したときの総文字数を超えることがない [6] ため，総文字数が少なければどのような順序付けでも，ZBDD のサイズが膨らむことはない．このことから，アイテム変数の順序付けにより指数関数的な効果が表れるためには，最悪な場合が指数関数的な節点数になる必要があるので，データ中にはパタンを羅列したときの総文字数が非常に多くなければならないということがいえる．一般的に，パタンはアイテム数の指数個あるので，アイテム変数の順序付けにより，大きな ZBDD の簡単化効果が得られると期待できる．

3.3 データベースの特徴を用いた発見的手法による順序付け法

前述のことを踏まえて，アイテム変数の順序付けに関して，データベースを読み込んでアイテムが初めて現れた順や，アイテム変数の出現頻度の降順，または昇順に並べた場合において実験した．結果は 4 章に示す．ここで出現頻度に関する順序付けを行うのは，出現頻度が高いアイテム変数は多くのアイテムと関係している可能性が高いからである．

また，LSI 論理設計分野の BDD の変数順序付け法を応用した手法についても後述する．

4. 実験と考察

本実験において使用した PC は Pentium4, 3.0GHz，SuSE Linux 9.3，主記憶 512Mbyte で ZBDD の最大節点数は 1,000 万個とした．

4.1 非頻出アイテムの削除

まず，ZBDD の簡単化のための実験として，代表的なトランザクションデータベースについて，非頻出アイテムの削除を行って，タプル頻度表とパタン頻度表を生成する実験を行った．それぞれの結果を表 1,2 に示す．今回，タプル頻度表に関しては 6 つのデータベースを，パタン頻度表に関しては ZBDD のサイズが膨大なものになってしまうので，3 つのデータベースを用いた．ここで α はしきい値，$\#R$ はデータ中のアイテムの削除率，size は節点数，count は式に現れる組合せの個数，time(s) は VSOP の処理時間（秒）である．

この結果を見ると，タプル頻度表，パタン頻度表の両方の場合とも，データベースにより差はあれど，大きな縮約効果が見て取れる．

処理時間に関して見ると，タプル頻度表については多くの場合，しきい値を上げるにつれて処理時間が増加している．これは，非頻出アイテムを削除していくにつれて同じものを表すタプルが増えていき，VSOP で計算する際の重みの数値を表す 2 進数の桁数が増えていったことによるものと考えられる．これと同じことがパタン頻度表についても言えると思われるのだが，パタン頻度表では処理時間が減少していく傾向にある．これは，桁数増大の影響以上に非頻出アイテムの削除によってパタンの数が激減しているためと思われる．

これにより，VSOP によるデータベース解析の際に，非頻出アイテムの削除が非常に大きな影響を与えるということがわかった．特に，パタン頻度表を生成する際に ZBDD がメモリ溢れを起こして，頻度表を生成することができなかった場合でも，非頻出アイテムの削除を行うことで，現実的な範囲で有用なタプル頻度表を生成できる場合がある．実際のデータベース解析において，その程度の頻度を設定するかによって，ZBDD のデータの削減量が左右される．

4.2 人工的な例題におけるアイテム変数の順序付けの影響

次に VSOP のアイテム変数の順序付けの効果に関して，顕著な効果が現れる場合を示す．以下に示すようなデータとアイテム変数の順序付けを用

表 1: 非頻出アイテムの削除を行ったタプル頻度表の生成

BMS-WebView-1					BMS-WebView-2					mushroom				
α	$\#R$	size	count	time(s)	α	$\#R$	size	count	time(s)	α	$\#R$	size	count	time(s)
0	0	46,148	18,473	23.9	0	0	198,471	48,684	185.8	0	0	8,006	8,124	1.1
700	52.0	8,998	5,561	18.0	250	49.4	60,520	24,485	51.7	3,600	48.8	370	141	17.6
2,800	90.4	52	16	26.0	1,300	89.7	2,188	2,265	61.2	7,900	87.2	19	4	30.6
3,659	100	10	1	28.3	3,767	100	10	1	66.7	8,125	100	7	1	35.1

pumsb					pumsb_star					T10I4D100K				
α	$\#R$	size	count	time(s)	α	$\#R$	size	count	time(s)	α	$\#R$	size	count	time(s)
0	0	1,750,883	48,474	162.8	0	0	1,324,502	48,474	118.9	0	0	552,429	89,135	59.0
26,000	47.2	29,324	10,044	331.9	21,000	49.5	27,867	10,856	238.1	1,900	50.1	151,202	78,855	109.5
48,000	92.0	68	14	620.4	36,000	89.6	114	39	414.5	4,400	90.3	2,074	1,908	172.0
48,945	100	7	1	673.7	38,750	100	7	1	457.4	7,829	100	10	1	187.1

表 2: 非頻出アイテムの削除を行ったパタン頻度表の生成

mushroom				
α	$\#R$	size	count	time(s)
0	0	513,762	5,574,930,438	303.5
3,600	48.8	7,915	127,516	43.5
7,900	87.2	19	8	35.1
8,125	100	7	1	35.0

BMS-WebView-1				
α	$\#R$	size	count	time(s)
0	0	Memory Overflow		
700	52.0	1,860,766	1,366,686,814,614,193	124.2
2,800	90.4	54	16	28.8
3,659	100	10	1	28.3

T10I4D100K				
α	$\#R$	size	count	time(s)
0	0	Memory Overflow		
1,900	50.1	2,560,170	15,144,453	748.7
4,400	90.3	2,759	2,845	193.9
7,829	100	10	1	187.0

いる.

$$
\begin{array}{cccccccccc}
a2 & b2 & a3 & b3 & a4 & b4 & \cdots & an & & bn \\
a1 & b1 & a3 & b3 & a4 & b4 & \cdots & an & & bn \\
a1 & b1 & a2 & b2 & a4 & b4 & \cdots & an & & bn \\
 & & \vdots & & & & \ddots & & \vdots & \\
a1 & b1 & a2 & b2 & a3 & b3 & \cdots & a(n-1) & & b(n-1)
\end{array}
$$

順序付け 1: $a1\ b1\ a2\ b2\ a3\ b3\ \cdots\ an\ bn$

順序付け 2: $a1\ a2\ \cdots\ an\ b1\ b2\ \cdots\ bn$

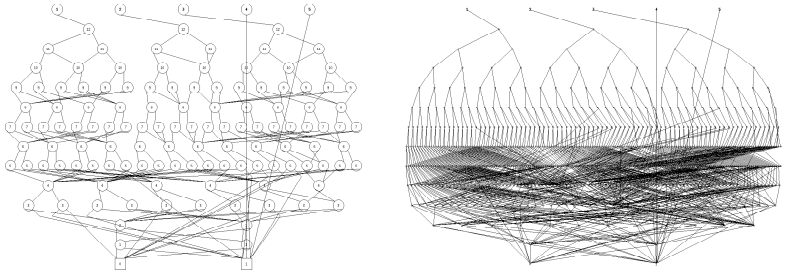
このデータは，各レコードにおいて $ak\ bk$ $(k = 1, ..., n)$ が抜けているものである．また，2 つの順序付けは，1 つ目はデータと同じように ak と bk が隣り合うように，2 つ目は ak と bk が離れるようになっている．今回は $n = 6$ と $n = 10$ の場合の実験結果を表 3，図 3 に示す.

この結果を見ると，特にパタン頻度表に関して，2 つの順序付けによる ZBDD のデータ構造の大きさの差が顕著であることがわかる．1 つ目の順序付けでは n に比例して，2 つ目の順序付けでは n に対して指数関数的に増えていくものと予想できる．つまり，1 つ目の順序付けの方がよいという傾向があるということがわかった．また，タプル頻度表に関しても同様な簡単化効果が認められるが，パタン頻度表とは違い ZBDD のサイズが指数関数的に大きくなることはないので，差がそれほど大きなものにはならなかった．処理時間に関しては，今回用いたデータが小規模であったので大きな差は生まれなかったが，大規模なデータを扱ったときには同様に指数関数的な差が生じるので，膨大な時間がかかってしまう場合がある.

このように，VSOP を用いてデータベースの頻度表を表す ZBDD を生成する際に，アイテム変数の順序付けが，最悪の場合，指数関数的な影響を与えるということが確認できた.

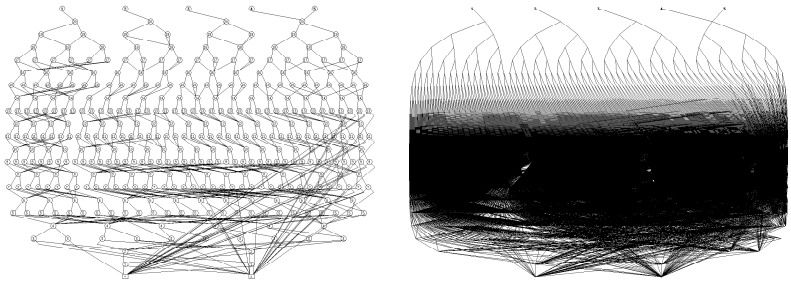
4.3 ベンチマーク例題における順序付けの影響

最後に，実際のトランザクションデータベースにおけるアイテム変数の順序付けの効果を計るために，アイテムの出現頻度の降順と昇順，また，データベースを読み込んでアイテムが初めて現れた順に関して，生成される ZBDD の大きさを比較する



(a) 順序付け 1　　(b) 順序付け 2

$n = 6$ の場合



(a) 順序付け 1　　(b) 順序付け 2

$n = 10$ の場合

図 3: 人工的例題（パタン頻度表）における順序付けの効果

表 3: 人工的な例題における頻度表作成結果

	タプル頻度表		パタン頻度表	
n	順序付け 1	順序付け 2	順序付け 1	順序付け 2
2	4	4	7	7
3	8	10	16	26
4	12	18	30	75
5	16	28	53	215
6	20	40	81	531
7	24	54	123	1,245
8	28	70	165	2,826
9	32	88	213	6,287
10	36	108	261	13,747
11	40	130	317	29,617
12	44	154	373	62,945

実験を行った．このとき，非頻出アイテムの削除も同時に行うことで，アイテムを削除することでアイテム変数の順序付けの効果が変化するか実験した．結果を表 4 に示す．ここで，down は降順，up は昇順，default はデータベースを読み込んでアイテムが初めて現れた順にアイテムを並べた場合である．

この実験結果からもわかるように，それぞれのトランザクションデータベースにおけるアイテム変数の順序付けの効果に関して，一様に縮約効果が得られる順序付けは得られなかった．今回は単純な順序付けでしか実験を行わなかったが，アイテム変数の順序付けの効果はデータベースの状態に大きく左右されることが示された．

4.4 考察

以上の実験結果を受けて，よりよい ZBDD の簡単化を実現するために，LSI 論理設計分野の BDD の変数順序づけ法として知られる深さ優先探索 [8] と動的重みづけ法 [9] をこのデータベース処理に応用することを考えている．深さ優先探索は，論理式（または回路）の出力側から深さ優先順にたどっていき，先に到達した入力の変数から上位に並べるというものである．これにより，関連の強いア

表 4: ベンチマーク例題における順序付けの効果

mushroom												
タプル頻度表						パタン頻度表					
1,000			3,000			1,000			3,000		
down	up	default	down	up	default	down	up	default	down	up	default
2,358	1,717	3,765	428	463	695	1,090,600	157,998	376,465	38,346	19,239	34,854

T10I4D100K											
tuple						pattern					
1,800			4,300			1,800			4,300		
down	up	default	down	up	default	down	up	default	down	up	default
152,020	151,548	151,202	2,773	2,697	2,717	3,566,721	2,662,778	3,020,963	3,918	3,890	3,918

イテム変数が近くの順位になりやすくなる．

また，動的重みづけ法は出力側から重みを入力側から伝えていき，重みが最も大きい変数を最上位にする．そして，その変数に関係する線を取り除き，再度重みを計算するということを繰り返すことで，アイテム変数の順序付けを行う方法である．これにより，出力に対して強い制御力があり，その前に選択された変数と関連の強い変数が選ばれることとなる．

これらの手法を，データベースの特徴抽出に応用して ZBDD の順序付けを行う方法について現在研究を進めている．

5. おわりに

本稿では，データベース解析において生成される ZBDD の簡単化に関して種々の実験を行った．その結果，非頻出アイテムの削除の効果やアイテム変数の順序付けの与える影響について考察した．さらに，アイテム変数の順序付けにより指数関数的な効果が生じる典型的な事例を初めて示した．非頻出アイテムの削除は，どのような場合においてもある程度の縮約効果が得られるが，アイテム変数の順序付けに関しては，顕著な効果が得られる場合を示すことはできたものの，実際のデータベースにおいて一様に縮約効果を得られる順序付けを見つけるのは難しいと思われる．しかし，ある種の特別な性質を持ったデータベースの場合を考えれば，よい順序付けを見つけることができるかもしれない．今後も考察を続けていく予定である．

参考文献

[1] R. Agrawal, H. Mannila, R. Strikant, H. Toivonen and A. I.. Verkamo, Fast Discovery of Association Rules, In Advances in Knowledge Discovery and Data Mining, MIT Press, 307-328, 1996.

[2] Bryant, R. E., Graph-based algorithms for Boolean function manipulation, IEEE Trans. Comput., C-35, 8 (1986), 677-691.

[3] J. Han, j. Pei, Y. Yin, R. Mao, Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach, Data Mining and Knowkedge Discovery, 8(1), 53-87, 2004.

[4] Minato, S., Zero-suppressed BDDs for set manipulation in combinatorial problems, In Proc. 30th ACM/IEEE Design Automation Conf. (DAC-93), (1993), 272-277.

[5] 湊真一, VSOP: ゼロサプレス型BDDに基づく「重み付き積和集合」 計算プログラム, IEICE Technical Report COMP2005-13(2005-5)

[6] 湊真一, 有村博紀, ゼロサプレス型二分決定グラフを用いたトランザクションデータベースの効率的解析手法, 電子情報通信学会論文誌, Vol.J89-D, No2, pp. 172-182, 2006.

[7] M. J. Zaki, Scalable Algorithms for Association Mining, IEEE Trans. Knowl. Data Eng. 12(2), 372-390, 2000.

[8] M. Fujita, H. Fujisawa and N. Kawato. Evaluation and improvement of Boolean comparison method based on binary decision diagrams. In Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-88), pp. 2-5, Novenber 1988.

[9] S. Minato, N. Ishiura and S. Yajima. Shared binary decision diagram with attributed edges for efficient Boolean function manipulation. In Proc. of 27th ACM/IEEE Design Automation Conference (DAC'90), pp. 52-57, June 1990.