HOKKAIDO UNIVERSITY

Title	頻出パターンマイニングのためのゼロサプレス型BDDの変数順序付け方法とその評価 ： データ工学論文特集
Author(s)	岩崎, 玄弥; 湊, 真一; ツォイクマン, トーマス
Citation	電子情報通信学会論文誌. D, 情報・システム, J91-D(3), 608-618
Issue Date	2008-03-01
Doc URL	https://hdl.handle.net/2115/47393
Rights	copyright©2008 IEICE
Type	journal article
File Information	23_ieice_2008.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

頻出パターンマイニングのためのゼロサプレス型BDDの変数順序付け方法とその評価

岩崎　玄弥[†a)]　湊　真一[†b)]　トーマス　ツォイクマン[†c)]

A Method of ZBDD Variable Ordering for Frequent Pattern Mining

Haruya IWASAKI[†a)], Shin-ichi MINATO[†b)], and Thomas ZEUGMANN[†c)]

あらまし　近年，ゼロサプレス型二分決定グラフ（ZBDD: Zero-suppressed Binary Decision Diagrams）を用いたデータベースの効果的な解析手法が提案されている．二分決定グラフ（BDD）は大規模論理関数データの表現方法として広く用いられている．今回はトランザクションデータベースに関して，大規模なアイテムの組合せ集合を処理するのに適したZBDDを用いる．ZBDDのデータ構造は変数の順序に大きく影響を受ける．我々は，ZBDDを生成する前処理としてアイテム変数の順序付けを行うことで，生成されるZBDDのサイズを小さくする研究を行ってきた．本論文では，VLSI設計の分野で開発された「動的重み付け法」をトランザクションデータベースの変数順序付けに応用する手法を提案する．動的重み付け法は，トランザクションデータベースの構造情報を用いることで，より良い順序付けを行う手法である．我々は，様々なトランザクションデータベースに対してこの手法を適用してZBDDを生成する実験を行うことで，この手法の有効性を確認することができた．更に，動的重み付けを行うプログラムを改良し高速化を行った．

キーワード　データマイニング，ZBDD，頻出パターン，二分決定グラフ，変数順序付け

1. まえがき

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．頻出パターンマイニング（Frequent Pattern Mining）は，最も基本的なデータマイニング問題であり，Agrawalら[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[4],[13]．

我々はこれまでに，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）[8]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頻出パターンを効率良く生成する手法[10],[11]に関する研究を進めている．ZBDDは大規模な組合せ集合データを非明示的に列挙し，頻出パターンの発見から解析に至る多様な演算を効率良く実行することができると期待されている．

ZBDDのグラフの大きさは，同じ例題に対しても変数の順序によって大きく影響を受けることが知られている．一般的なBDDの変数順序付け方法については過去に多くの研究[3],[5],[7]があるが，ZBDDを用いたデータベース解析処理におけるアイテム変数の順序付けに関しては，まだ良い方法は知られていない．本論文では，データベースの特徴に基づいて順序付けを行う新しい発見的手法を提案し，その効果を示す．以下の本文では，**2.**でZBDDを用いたデータベース表現について述べ，**3.**では，データベース表現におけるZBDDの変数順序付けについて述べる．そして，**4.**で本論文で提案する手法に関して行った実験の結果と考察を述べ，最後に**5.**で簡単なまとめを述べる．

2. ZBDDによるデータベース表現

ここでは以下に示すようなデータベースを考える．まず，Mを空でない集合とする．Mの要素をアイテム

† 北海道大学大学院情報科学研究科，札幌市
Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060–0814 Japan
a) E-mail: iwsk@ist.hokudai.ac.jp
b) E-mail: minato@ist.hokudai.ac.jp
c) E-mail: thomas@ist.hokudai.ac.jp

 © (社)電子情報通信学会 2008

ID	Tuple
1	abc
2	ab
3	abc
4	bc
5	ab
6	abc
7	c
8	abc
9	abc
10	ab
11	bc

(Databese)

Tuple	Freq.
abc	5
ab	3
bc	2
c	1

(Tuple histogram)

図 1　データベース例題とタプル頻度表
Fig. 1　Database and tuple histogram.

Tuple	Freq.	(Included patterns)
abc	5	$\rightarrow \{abc, ab, ac, bc, a, b, c\}$
ab	3	$\rightarrow \{ab, a, b\}$
bc	2	$\rightarrow \{bc, b, c\}$
c	1	$\rightarrow \{c\}$

(Tuple histogram)

Pattern	Freq.
abc	5
ab	8
bc	7
ac	5
a	8
b	10
c	8

(Pattern histogram)

図 2　タプル頻度表とパターン頻度表
Fig. 2　Tuple and pattern histogram.

と呼ぶ．ここでは図 1，図 2 のように，$M = \{a, b, c\}$ とする．この集合から得られるすべての組合せの集合は M のべき集合 $\wp(M)$ である．部分集合 $\mathcal{C} \subseteq \wp(M)$ は組合せ集合と呼ばれる．この組合せ集合の要素は，例えば $\{a, c\}$ のようにアイテムの集合である．記述を簡略化するために，$\{a, c\}$ を ac と書き，組合せ集合の要素をタプルと呼ぶ．トランザクションデータベースはタプルのリストである．

トランザクションデータベースから有用な規則を見つけるために，2 種類の頻度表を考える．一つ目はタプル頻度表である．タプル頻度表は二つの行からなっており，最初の行にはデータベース中に存在するすべてのタプルが入り，もう一つの行には各々のタプルの出現頻度が入っている．図 1 はタプル頻度表の例であり，出現頻度の高い順に並べられている．

二つ目の頻度表はパターン頻度表と呼ばれている．パターン頻度表の最初の行には，タプルとそれに含まれるすべての部分集合が入っており，もう一つの行にはアイテム集合のデータベース中の出現頻度が入っている．例えば，図 2 ではパターン ab と a，c が 8 回出現していて，パターン b が 10 回出現していることが分かる．

タプル頻度表からは頻出タプルを，パターン頻度表からは頻出パターンを用意に見つけることができるので，頻度表は非常に重要である．我々はパターン頻度表をタプル頻度表を処理することで構成する．詳細は後ほど記述する．頻出パターンマイニング問題とは，あるしきい値以上出現する頻出パターンをすべて見つけるという問題である．例えば，図 1 の頻度表としきい値として 10 が与えられた場合，データマイニングを行った結果はパターン b を返すことになる．またしきい値を 8 とすると，パターン ab，a，b，c を返す．このような問題を扱う場合，現実的にはデータベースのサイズは非常に大きいものである場合が多いので，最も重要な問題はこの処理をどのようにして効率良く行うかである．

我々は，この問題を解決するために VSOP (Valued-Sum-Of-Products calculator) [9] と呼ばれるプログラムを用いる．VSOP では，入力としてデータベースとしきい値を受け取ると，タプル頻度表を表す ZBDD を内部で構築する．そして，我々の研究グループで開発した ZBDD-growth 法 [11] を用いてパターン頻度表を生成する．ZBDD-growth 法では，与えられたしきい値以上のパターンすべてを表す ZBDD をタプル頻度表から構築することができる．この二つの過程では同じ変数の順序付けが用いられており，この順序付けが生成される ZBDD のサイズに非常に大きな影響を与える（**4.2** を参照）．

2.1　BDD と ZBDD

本研究で用いる二分決定グラフ，BDD と ZBDD について述べる．BDD [2] は，図 3 に示すような論理関数のグラフによる表現である．図 3 は，$S(a, b, c) = a\bar{b}c \vee \bar{a}b\bar{c}$ を表す BDD の例である．図 3 では a，b，c という順序付けを用いたが，順序付けが与えられれば，Bryant のアルゴリズムを用いてどのような論理関数に対しても BDD を構築することができる．現実の問題では非常に多くの論理関数を扱うことになるが，このアルゴリズムは非常に効果的であり，生成された BDD は二分決定木に比べて効率の良い表現となっている．

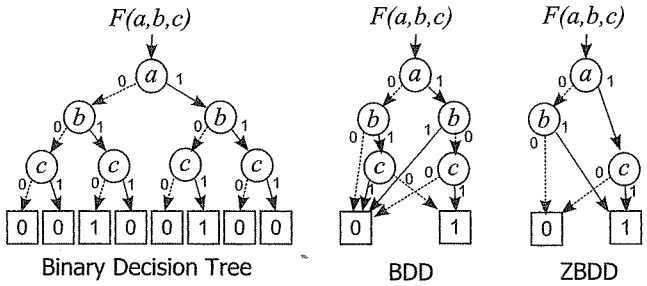


図 3　二分決定木と BDD，ZBDD
Fig. 3　Binary decision tree, BDDs and ZBDDs.

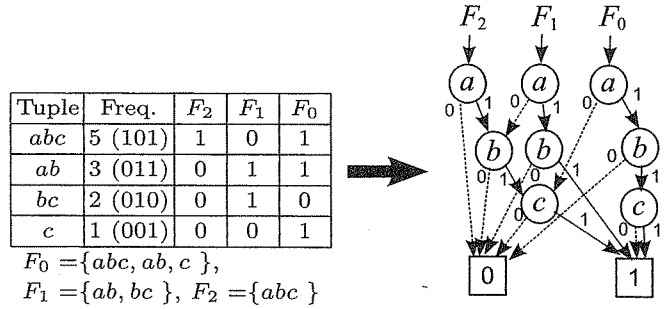


図 4　ZBDD の 2 進ベクトルによるタプル出現頻度の表現
Fig. 4　Representation of a tuple histogram.

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 3 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二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，二つの 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良く実行できる．

BDD はもともと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 n 変数の論理空間で組合せ集合を表すこともできる．ここで n は集合 M の要素数であり，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．このように，BDD を用いることで組合せ集合を現すことができるが，ZBDD を用いることでより効率良く組合せ集合を表すことができる．

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．更に，組合せ集合に特化した ZBDD [8] を用いると，より簡潔な表現が得られ，いっそう効率良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 3 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 よりも効率良く組合せ集合を表現・操作することができる．

2.2　ZBDD を用いた頻度表の表現

ここで，ZBDD を用いてどのように頻度表を表現するのかを説明する．ZBDD は組合せ集合を表すことしかできないため，そのままでは組合せの出現回数を表すことができない．これの解決法として，本論文では整数値を 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，出現回数を 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 4 のように，各項の重みを非明示的に表すことができる．図 4 では，アイテムの集合が $F_0 = \{abc, ab, c\}$，$F_1 = \{ab, bc\}$，$F_2 = \{abc\}$ のように符号化されており，それぞれのけたは単純な ZBDD によって表されている．また，この三つの ZBDD は互いの部分グラフを共有している．

VSOP プログラムを用いることで，トランザクションデータベースにおいて，タプル頻度表を効率的に生成することができるが，しかしながら，一般に k 個のアイテムからなるタプルは 2^k 個のパターンを含んでいるので，パター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る．しかし，ZBDD を用いる場合，多数の類似するパター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ーン頻度表を現実的に生成できる可能性がある．このタプル頻度表から頻出パターンを表すパターン頻度表を生成する処理は，ZBDD-growth 法を用いて行われる [11]．いったんパターン頻度表の生成に成功すれ

ば，任意の頻度に対応する頻出パターン集合をすべて同時に保持しているのと同等であるため，非常に強力なデータ表現であり，その後は様々なデータベース解析処理を ZBDD 処理系の演算により効率良く解くことが可能となる．

2.3 ZBDD-growth 法による頻出パターン生成

パターン頻度表は生成できれば強力であるが，中規模以上のベンチマーク例題に対しては，ZBDD が巨大になりすぎて，現実的な記憶量では表現することができない．しかし，ある一定の頻度以上の頻出パターン集合だけを抽出するだけであれば，最小頻度のしきい値 α を大きくしていけばパターン数が単調減少するので，適度に大きい α を与えれば ZBDD を構築することが可能となる．そこで我々の研究グループでは，ZBDD のベクトルで表現されたタプル頻度表から，パターン頻度表を経由せずに，直接，頻出パターン集合を表す ZBDD を生成するアルゴリズム「ZBDD-growth 法」を開発した [11]．これにより，中規模以上のベンチマーク例題でも頻出パターン集合を表す ZBDD を生成できるようになったが，それでもやはり ZBDD が小さければ計算時間が少なくて済むため，良い変数順序付けを求めることは依然として重要な研究課題である．

3. データベース表現における ZBDD の変数順序付け

前述のように，ZBDD を用いることで頻度表や頻出パターン集合をコンパクトに表現できる可能性があるが，データベースが大規模になると，ZBDD 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，本論文では，「アイテム変数の順序付け」を考える．既に述べたように，生成される ZBDD のサイズは変数の順序に大きな影響を受ける．よって，我々の目的は ZBDD のサイズができるだけ最小に近くなる順序付けを見つけることである．

ZBDD の性質上，最悪な順序付けを用いても節点数はパターンを羅列したときの総文字数（各タプルに現れるアイテム数の総和）を超えることがない [10] 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現れるためには，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，データ中にはパター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，パター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．

本論文で使用した VSOP プログラムでは，データベースに含まれる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．（以下では，この変数順序を「後出し上位順」と呼ぶ．）例えば，ac, $befg$, ba, h というリストをもつデータベースが与えられた場合，変数の順序付けは，h, g, f, e, b, c, a となる．

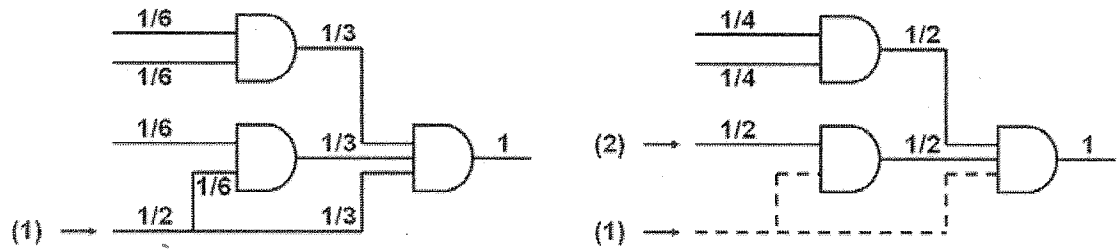
我々がこれまでに提案した ZBDD によるデータマイニングの実験は，ほとんどがこの順序付けを用いている．後出し上位順は，データベースの読込みと同時に行うことができるので，計算コストを無視することができ，経験的には比較的良い順序を得られることが多い．しかし，この順序付けでは，同じ内容のデータベースでも，処理するレコードの順番によって変数の順序が大きく変わるため，結果が不安定で，例題によっては非常に悪い順序付けとなる場合がある．そこで我々は，与えられたデータベース例題の特徴に基づいて，データベース中のアイテムの順序に依存せず，より良い変数順序を安定的に求めるための発見的手法を提案する．この手法は従来の BDD に対して用いられている手法を，データベース解析処理向けに改良したものである．よって，まず従来の BDD に対する手法に関して説明する．

3.1 論理回路における BDD の動的重み付け法

通常の BDD の変数順序付け方法については，これまでに VLSI 論理設計の分野で多くの研究がある．一般的に，最適な順序付けを求めることは NP 完全であることが知られている [12] が，現実的な時間で比較的よい順序を求める発見的手法がいくつか提案 [3], [5], [7] されている．ここでは，論理回路における BDD の「動的重み付け法」[7] を紹介する．この手法は，論理回路における BDD の入力の順序付けに関して効果を上げている．

通常の BDD においては，グラフの大きさに影響する要素として，次の二つの性質が知られている [3]．

(1) 局所計算性のある入力の組は，なるべく近い順序にする．これにより，多くのサブグラフが共有できるという傾向がある．例えば 2 進数の加算器の論理回路では，対応するビット同士の変数を並べて配置する



(a) First (b) Second

図 5 論理回路における動的重み付け法
Fig. 5 The dynamic weight assignment method for logic circuit.

とよいことが知られている.

(2) 出力を制御する力の強い入力は上位に配置する. 例えば, データセレクタの論理回路では, データ入力を上位に並べた場合は制御入力を上位に並べた場合に比べ, BDD のサイズが指数的に増大してしまう. これは, 制御入力の値を先に決定すると, 多くのデータ入力を無視できるという性質があるためと考えられる.

これら二つの性質を満たすような順序付けを行えばよいのであるが, 実際には, 二つの性質が同時に現れて, 互いに相反する順序付けを要求する場合があり, これを両立させて最適な順序を求めるのは難しい. また, どんな順序でもそれほど変化の見られない回路もあり, その場合には, 順序付けを工夫してもあまり効果が得られない.

以上の考察に基づいて, 回路の結線情報から変数の順序付けを行う発見的手法がいくつか知られているが, その中の一つとして「動的重み付け法」が提案されている [7]. この方法は, 論理回路の出力の論理関数を BDD で表す場合に, 回路の結線情報から関数の性質を予想し, 最適に近い順序を求めるものである.

動的重み付け法では, まず前述の (2) の性質を満たすため, 制御性の高い入力を見つける. これは, 以下のような傾向をもつ.

- ファンアウト数が多い (出力に至るパスが多い).
- 出力までの段数が少ない.
- 出力へ至るパス上のゲートのファンイン数が少ない.

この傾向を評価するために, 次に示すような簡単な重み付け法を用いる.

(1) 出力線の重みを 1 とする.

(2) 出力から入力に向かって重みを伝達させる. 2 入力以上のゲートでは, 出力線の重みをファンイン数で均等に分けて入力線に伝える.

(3) ファンアウトがある信号線で, 複数のゲートから重みが伝わるときにはそれらを伝える.

以上の手順で重み付けをした例を図 5 (a) に示す. この図では, 出力に与えられた重みが三つのファンインに均等に分けられたことが示されている. この重みが最大となった入力が最も出力を制御しやすいとみなし, その入力に最上位の変数を割り当てる.

次に, 最上位の変数を決定した後, その下のサブグラフを小さくすることを考える. このとき, 既に決定された入力は 0 か 1 に固定されてしまっていると考えれば, その近くの信号線は, 制御性が増大しているはずである. そこで, 既に決定された信号線を切断したと仮定して, 改めて重み付けを行う. このように動的に重み付けを行うことにより, 前節 (1) の局所計算性も反映することができる. その様子を図 5 (b) に示す. 以下, これを動的重み付け法と呼ぶ.

動的重み付け法では, 順序付けに必要な時間は, 入力数 n, 回路規模を m として, $O(nm)$ である. 順序付けによって, 生成される BDD のサイズに指数関数的な違いが見られるので, 順序付けにこの程度の時間がかかっても十分有効である.

3.2 トランザクションデータベースにおける ZBDD の順序付けへの応用

ここまで論理回路における BDD の順序付け法に関して説明してきたが, 本論文では, これをトランザクションデータベースにおける ZBDD の順序付けに応用する.

まずはじめにデータベースの構造を表すグラフを作る. 各々のアイテムを表すノードを生成し, 次に各々のタプルを表すノードを生成する. 最後にデータベース全体を表すノード *All* を作る. 更に, ノード *All* からすべてのタプルに対して枝を引き, 各々のタプルから, そのタプルに含まれないアイテムに対して枝を引く. 論理回路においては, 出力から重みを伝えていきゲートの入力数に応じて重みを分配したが, 本手法

では，

(1) データベース全体の重みを 1 として，これをタプルの数で割った値をそれぞれのタプルの重みとして与える．

(2) 各々のタプルにおいてそのタプルに含まれないアイテムに対して，タプルに与えられた重みをその含まれないアイテムの数で割った値を重みとして伝える．

(3) 複数のタプルから重みが伝わるときにはそれらを加える．の手順で重み付けを行う．

この (2) の手順において，タプルに含まれるアイテムではなく，タプルに含まれないアイテムに重みを伝えて行くところが，従来の論理回路での動的重み付け法とは大きく異なる点である．タプルに含まれるパターンを抽出する場合に，タプルに含まれるアイテムはパターンを形成するアイテムの組合せに含まれるときと含まれないときがあるが，タプルに含まれないアイテムは当然そのタプルが含むパターンの要素とはなり得ず，このことから，タプルに含まれないアイテムは含まれるアイテムより出力に与える影響が大きいと考えられるので，このように重み付けを行う．

このようにして重み付けをした例を図 6 (a) に示す．この重みが最大となったアイテムを ZBDD のサイズに特に影響を与えるものとして，変数の最上位に割り当てる．与えられた重みが同じものが複数あった場合は，最も出現が早かったアイテムを選ぶ．

次に，最上位の変数を決定した後の処理について述べる．順序が決定したアイテムに関する線は BDD の場合と同様に取り除き，もしアイテムの順序付けが決まったことで，タプルからアイテムに伸びる線のすべてが無くなってしまったタプルがあれば，そのタプルは以降の重み付けの際には存在しないものとして扱う．つまり，存在するタプルには，総タプル数から存在しなくなったタプルの数を引いた数で重み 1 を割った値を重みとして与えることになる．これを図 6 (b) に示す．この順序が決定したアイテムに関する線を取り除くという処理によって，取り除かれたアイテムの重みがそのアイテムと関係の深いアイテムに分配されることになるので，局所計算性を反映することができる．

以上の操作を繰り返すことでアイテム変数の順序付けを行う．

3.3 変数順序付けプログラムの高速化

更に我々は，変数順序付けに要する計算時間を短縮するため，以下の改良を行った．これまで説明してきた手法では，トランザクションデータに対する順序付け法の説明の (2) に示されているように，各々のタプルに含まれないアイテムに対して重みを分配してきた．しかし，扱うデータベースは疎である場合が多いので，このままでは非常に多くのアイテムに対して重みを分配するという処理が必要になってしまう．これを解消するために，我々はまずすべてのアイテムに *All* に与える重みと同じ 1 を与える．次に，各々のタプルに含まれるアイテムに関して，タプルに与えられた重みをタプルに含まれないアイテムの数で割った重みをそれぞれのアイテムから引くという処理を行う．この処理では，タプルに含まれないアイテムの重みが相対的に増加するので，以前の手法と同じ順序付けが得られる．また，タプルに含まれるアイテムに重みを伝えるという処理に変更することによって，重みを伝えるアイテムの数が格段に減ることが予想される．例えば，アイテムの平均出現個数が全アイテム数に対して 1%とすると約 100 倍高速になる．改良後の計算時間は，アイテム数を n，データベースのサイズを m とすると，

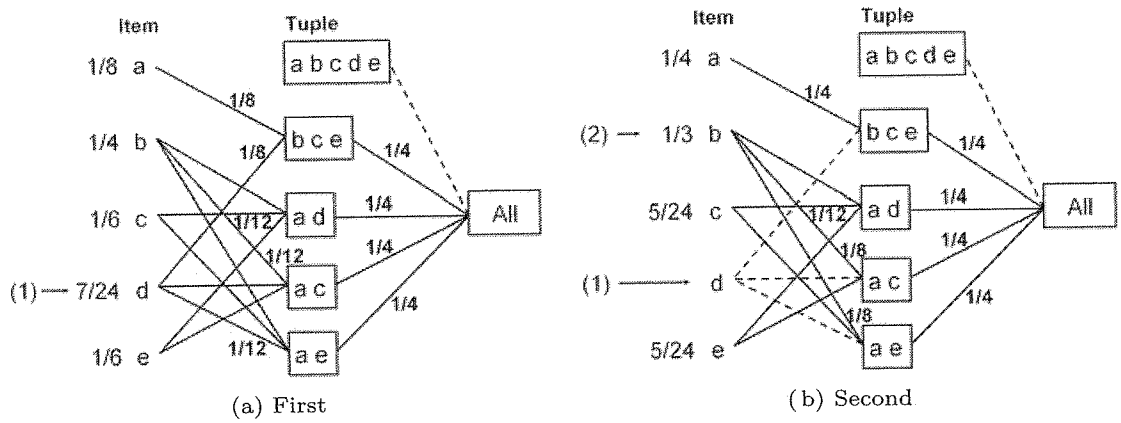


図 6 トランザクションデータベースにおける動的重み付け法
Fig. 6 The DWA method for transaction database.

$O(nm)$ となる．この手法と以前の手法を用いて変数の順序付けを行った処理時間の比較については後述する．

4. 実験と考察

上記のアルゴリズムを C, C++により実装し，実験を行った．本実験において使用した PC は Pentium4, 3.0 GHz, SuSE Linux 9.3, 主記憶 512 MByte で ZBDD の最大節点数は 1,000 万個とした．はじめに，人工的な例題における ZBDD の順序付け法の効果に関して行った実験の結果を示し，次に実際のベンチマーク例題における ZBDD の動的重み付け法に関して実験を行った結果を示す．今回の実験で用いた順序付けは，動的重み付け法，出現頻度の高いアイテムを上位に並べた「高頻度上位順」，逆に出現頻度の低いアイテムを上位に並べた「低頻度上位順」と，アイテムの最初の出現が遅いものから上位に並べた「後出し上位順」の四つである．

4.1 人工的な例題におけるアイテム変数の順序付け法の評価

まず，ZBDD のアイテム変数の順序付け法に関して，人工的な例題における動的重み付け法の効果を調べる．今回は以下に示すような人工的に生成したデータベースを用いて実験を行う．

a_2	a_3	a_4	$\cdots$	a_n	b_2	b_3	b_4	$\cdots$	b_n
a_1	a_3	a_4	$\cdots$	a_n	b_1	b_3	b_4	$\cdots$	b_n
a_1	a_2	a_4	$\cdots$	a_n	b_1	b_2	b_4	$\cdots$	b_n
	$\vdots$				$\vdots$				$\vdots$
a_1	a_2	a_3	$\cdots$	a_{n-1}	b_1	b_2	b_3	$\cdots$	b_{n-1}

このデータは，$a_1\ a_2\ \ldots\ a_n\ b_1\ b_2\ \ldots\ b_n$ という形から a_k と b_k $(k = 1, \ldots, n)$ を取り除いたものを各レコードとしている．また，このデータベースに関して動的重み付け法を用いた場合のアイテム変数の順序付けと，低頻度上位順に並べた順序付けを以下に示す．

- 動的重み付け法：$a_2b_2a_3b_3a_4b_4\cdots a_nb_na_1b_1$
- 低頻度上位順：$a_2a_3a_4\cdots a_nb_2b_3b_4\cdots b_na_1b_1$

今回は低頻度上位順を用いたが，高頻度上位順や後出し上位順でも低頻度上位順とほぼ同じ順序付けとなった．

以上の場合において，タプル頻度表から頻出パターンを抽出する実験を行った結果を表 1 に示す．

この結果を見ると，アイテム変数の順序付けが ZBDD のサイズに大きな影響を及ぼすことが分かる．動的重み付け法の場合はアイテム変数の数に比例して ZBDD のサイズが増加しているのに対して，低頻度上位順では指数的にサイズが増加している．これは，今回の人工的なデータベースではすべてのアイテムの出現回数が同じであるので，出現回数で順序付けをしようとしても結局アイテムの出現順に並んでしまうため，低頻度上位順では効果的な順序付けができなかったためである．これに対して動的重み付け法では，一つのアイテムの順序が決まったときに，その決まったアイテムの重みが関連の深いアイテムに分配されることになるので，ZBDD のサイズに関して非常に縮約効果のある順序付けができたのだと考えられる．このことから，この人工的な例題では，局所計算性が大きく影響したと考えられる．

表 1 人工的な例題における VSOP の処理時間と ZBDD の節点数
Table 1 The computation time of VSOP calculator and the ZBDD size for mathematical benchmark examples.

	the DWA Method		Less Freq. Upper Order.	
n	size	time (s)	size	time (s)
2	5	<0.1	5	<0.1
3	10	<0.1	12	<0.1
4	15	<0.1	25	<0.1
5	20	<0.1	50	<0.1
6	25	<0.1	99	<0.1
7	30	<0.1	196	<0.1
8	35	<0.1	389	<0.1
9	40	<0.1	774	<0.1
10	45	<0.1	1,543	<0.1
12	55	<0.1	6,153	<0.1
14	65	<0.1	24,587	<0.1
16	75	<0.1	98,317	<0.1
18	85	<0.1	393,231	0.456
20	95	<0.1	1,572,881	1.954
22	105	<0.1	6,291,475	8.482
24	115	<0.1	Memory overflow	

このように，アイテムの出現頻度だけでは効果的な順序付けができない場合においても，動的重み付け法では効果的な順序付けができることが示された．

4.2 動的重み付け法の効果

次に，実際のベンチマーク例題における ZBDD の動的重み付け法による順序付けの効果に関して行った実験の結果を示す．実験は，動的重み付け法を用いてタプル頻度表を構築する VSOP スクリプトを生成し，そこから ZBDD-growth 法を用いて頻出パターンを抽出するという形で，生成される ZBDD の節点数を調べた．その結果を表 2 に示す．データベース名の隣の () 内の数字は，抽出するパターンの最低頻度を表した

表 2　トランザクションデータベースにおける ZBDD の動的重み付け法の効果
Table 2　The effect of the DWA method for transaction database.

	the DWA Method		More Freq. Upper Order.		Less Freq. Upper Order.		Late Appear. Upper Order.	
	size	time (s)	size	time (s)	size	time (s)	size	time (s)
chess (2,000)	1,422	5.8 s	3,856	2,036.6 s	1,415	5.8 s	2,778	64.8 s
mushroom (1)	16,403	1.0 s	448,734	1.9 s	15,131	1.0 s	40,557	1.1 s
connect (60,000)	348	27.5 s	1,659	5,402.3 s	348	27.5 s	374	62.8 s
BMS-WebView-1 (30)	106,920	98.8 s	389,181	778.2 s	109,989	103.1 s	152,431	153.4 s
BMS-WebView-2 (100)	2,171	352.1 s	3,201	460.0 s	2,298	354.4 s	Memory Overflow	
pumsb (40,000)	808	202.1 s	Memory Overflow		813	202.7 s	Memory Overflow	

表 3　動的重み付け法の順序付けの計算時間
Table 3　The computation time of the DWA method.

Database	(a) (sec)	(b) (sec)
chess	2.8	0.7
mushroom	17.9	1.5
connect	219.6	15.1
BMS-WebView-1	1,255.0	10.6
BMS-WebView-2	(>1 h)	118.5
pumsb	(>1 h)	306.8
pumsb_star	(>1 h)	219.8
retail	(>1 h)	825.5

ものである．つまり，この数字より頻度が高いパターンを抽出するということになる．

この実験結果より，今回の動的重み付け法はアイテム変数を低頻度上位順に並べたものと非常に近い効果を得られることが分かった．BMS-WebView-1 に関しては今回の手法の方が良い結果を得られている．また，高頻度上位順と比べると大きな縮約効果が得られていることが分かる．

また，動的重み付け法にかかる処理時間は表 3 のようになる．この表において，(a) が改良前の動的重み付け法の計算時間であり，(b) が改良を加えた動的重み付け法の計算時間である．

以前の手法と改良を加えた手法を比較すると，格段に処理時間が削減されていることが分かる．この処理時間の削減の度合に違いが見られるのは，データベースがより疎なものほど処理時間の削減が顕著であるからである．

4.3 考　　察

今回の実験で，ZBDD における動的重み付け法の順序付けによる ZBDD の縮約効果が確認できた．出現頻度だけでは効果的な順序付けができない場合にも，動的重み付け法は効果的な順序付けを行えることが分かった．しかし，実際のベンチマーク例題において実験を行った場合では，高頻度上位順と後出し上位順よりは効果的な順序付けができていることが確認できたが，低頻度上位順とは大きな差は見られなかった．これは，人工的な例題では局所計算性が非常に大きく影響したが，実際のベンチマーク例題ではそれほど大きく影響しなかったためと思われる．また，低頻度上位順では出現頻度の低いアイテムから上位に並べられるが，これは動的重み付け法で出現頻度の低いアイテムにより大きな重みが伝えられる傾向があるということと類似している．このことから，実際のベンチマーク例題においては動的重み付け法と低頻度上位順では似たような順序付けになったと考えられる．

このように，局所計算性よりもアイテムの出現頻度の方が ZBDD のサイズに大きな影響を与える場合においては，順序付けにかかる計算時間が短い出現頻度に関する順序付け法の方が効果的なこともある．例えば，動的重み付け法の順序付けに BMS-WebView-1 では 10.6 秒かかっているが，単純に頻度を計算するだけなら 0.8 秒で済む．実用的な例題では，低頻度上位順で十分な縮約効果が得られるならば，低頻度上位順で順序付けを行っても，比較的有効であるといえるのかもしれない．これは議論すべき課題の一つであるといえる．

そこで，動的重み付け法が特に有効なデータベースの特徴について考察を行った．実際のベンチマーク例題に関して行った実験では，動的重み付け法は低頻度上位順に比べて順序付けに時間がかかるのに，生成した ZBDD のサイズにはほとんど差が生じない，という例が多く見られる．前に述べたとおり，ZBDD のサイズに影響を与える要素として，出力の制御力と局所計算性があるが，出力の制御力は出現頻度が低いアイテムほど大きくなる．したがって，低頻度上位順は，出力の制御力を素直に反映した順序付けであるといえる．すなわち，動的重み付け法と低頻度上位順の相違

点は，出力制御力だけでなく局所計算性も考慮しているかどうか，ということになる．

表 4 に実験で用いたベンチマーク例題の特徴を示す．$\#I$ はアイテム数，$\#T$ はタプル数，total$|T|$ はタプル長（タプルに含まれるアイテム数）の総和，avg$|T|$ は平均タプル長，avg$|T|/\#I$ はアイテムの平均出現頻度である．実際のベンチマーク例題は，表 4 に示したとおり，疎（アイテムの平均出現頻度が小さい）であることが多い．そのような例題では，どの二つのアイテムの組をとってみても，その二つのアイテムが同時に欠けているタプルが多数存在する可能性が高く，特定のアイテムの組だけに強い局所計算性が生じにくい構造となっている．したがって，動的重み付け法と低頻度上位順の間に大きな差が生まれなかったと考えられる．

一方，アイテムの平均出現頻度が高い例題では，局所計算性の影響が強く出る可能性がある．その一例として，ランダムに生成した高出現頻度データを用いて実験を行った．このデータは，アイテム 30 個を用意し，出現頻度 90%（27 個のアイテム）を一つのタプルとし，これをランダムに 30 タプル生成したものである．このようなデータを異なる乱数列により 10 通り生成し，それぞれに対して，動的重み付け法と低頻度上位順を適用した場合の ZBDD の節点数の平均値，最大値，最小値を比較したところ，上記の表 5 のような結果が得られた．この表を見ると，低頻度上位順よりも動的重み付け法の方が明らかに良い結果が得られている．このように平均出現頻度が高いデータでは，局所計算性の影響が大きくなりやすく，動的重み付け法の方が，単純な低頻度上位順よりも良い順序付けを出す場合がある．

それでは，疎なデータベース例題では動的重み付け法は全く不要かというと，必ずしもそうとは限らない．実際のデータベースでは，アイテムの出現に偏りがあり，全体としては疎であっても，一部に密なアイテムの固まりが存在する例がしばしば見られる．例えば，例題 “BMS-WebView-1” は，ネット販売サイトの 1 トランザクションごとのアクセスログのデータであり，全体としては極めて疎であるが，特定の Web ページ群を表すアイテム群が固まって出現しているタプルが含まれている．このような密な固まりの部分だけでも局所計算性を考慮しないと，それによって ZBDD のサイズが増大してしまう可能性がある．

疎なデータベースの一部に密なアイテムの固まりが存在する例として，上記の方法でランダムに生成された高出現頻度データを，例題 “mushroom” に付け足して一つのデータベースとした例題を作成した．ただし，両者のアイテムは互いに重なりがないように区別している．このデータベースに対して動的重み付け法と低頻度上位順を適用し ZBDD を生成し，異なる乱数列 10 通りについて，節点数の平均値，最大値，最小値を比較した．その結果，もともとの mushroom（表 2）では低頻度上位順の方が良かったのに対し，密なアイテムの固まりを付加した場合には，表 6 のとおり傾向が逆転し，動的重み付けの方が低頻度上位順よりも平均して良い結果が得られることが確認できた．このことから，全体として疎なデータベースの一部に密なアイテムの固まりが存在するようなデータベースの場合にも，動的重み付け法は有効であるといえる．

最後に，全体的な計算時間について考察すると，実験では ZBDD-growth を用いてタプル頻度表から頻出パターンを抽出したが，その際にデータベースの種類やアイテム変数の順序付けによって，ZBDD の生成にかかる時間が著しく変化することが分かった．例えば，chess では動的重み付け法では 5.8 秒で済んだ

表 4　データマイニング例題の特徴
Table 4　Statistics of typical benchmark data.

Database	$\#I$	$\#T$	total$\|T\|$	avg$\|T\|$	avg$\|T\|/\#I$
chess	75	3,196	118,252	37.0	49.30%
mushroom	119	8,124	186,852	23.0	19.32%
connect	129	67,557	2,904,951	43.0	33.33%
BMS-WebView-1	497	59,602	149,639	2.5	0.51%
BMS-WebView-2	3,340	77,512	358,278	4.6	0.14%
pumsb	2,113	49,046	3,629,404	74.0	3.50%

表 5　ランダムに生成した高出現頻度データに対する実験結果
Table 5　Experimental results for random-generated high-frequency data sets.

Method	average	max	min
the DWA Method	4,188	9,102	2,448
Less Freq. Upper Order.	10,086	15,339	3,827

表 6　密なアイテムの固まりを mushroom に付加したデータに対する実験結果
Table 6　Results for “mushroom” with a dense itemset block.

Method	average	max	min
the DWA Method	22,852	26,971	19,121
Less Freq. Upper Order.	25,242	30,495	18,983

が，高頻度上位順では 2,036.6 秒もかかった．しかし，mushroom においてはどの順序付けでも大きな処理時間の差はなかった．これは，VSOP プログラムでは処理を高速に行うためにハッシュテーブルを用いており，そのハッシュテーブルの効果によって処理時間に差が出てしまったと考えられる．

5. むすび

本論文では，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，トランザクションデータベースにおける ZBDD の動的重み付け法による ZBDD の縮約効果が確認できた．人工的な例題においては局所計算性を反映することで，効果的な順序付けを行うことができた．また，実際のデータベースにおいても効果的な順序付けを行えることが確認できたが，出現頻度をもとに行った順序付けと大きな差が生まれない場合もあるということが分かった．そこで，動的重み付け法が効果を発揮するデータベースの特徴について考察し，アイテムの平均出現頻度との関係を示した．

現在の手法では，データベースを n 回スキャンすることになるので $O(nm)$ 時間かかり，大規模なデータベースを扱う際にはまだ時間がかかりすぎる．このことから変数順序付け法に関して，データベースを定数回スキャンして効果のある順序付けを行う手法の検討など，より実用的な変数の順序付け法に関して研究していきたい．

謝辞 本研究の一部は日本学術振興会科研費補助金基盤（B）「二分決定グラフに基づく大規模データベースの効率的解析処理アルゴリズムの研究」（課題番号 17300041，研究代表者湊真一）による．

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, pp.307–328, MIT Press, 1996.

[2] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, 1986.

[3] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and Implementation of Boolean comparison method based on binary decision diagrams," Proc. ACM/IEEE International Conf. on Computer-Aided Design (ICCAD-88), pp.2–5, 1988.

[4] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," Data Mining and Knowkedge Discovery, vol.8, no.1, pp.53–87, 2004.

[5] S. Malik, A.R. Wang, R.K. Brayton, and A.S. Vincentelli, "Logic verification using binary decision diagrams in a logic synthesis environment," Proc. ACM/IEEE Internationl Conf. on Computer-Aided Design (ICCAD-88), pp.6–9, 1988.

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

[7] 湊　真一，石浦菜岐佐，矢島脩三，"論理関数の共有二分決定グラフによる表現とその効率的処理手法，" 情処学論，vol.32, no.1, pp.77–85, Jan. 1991.

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

[9] 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，" 信学技報，COMP2005-13, May 2005.

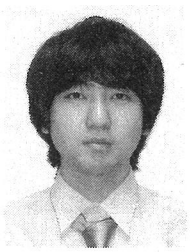
[10] 湊　真一，有村博紀，"ゼロサプレス型二分決定グラフを用いたトランザクションデータベースの効率的解析手法，" 信学論（D），vol.J89-D, no.2, pp.172–182, Feb. 2006.

[11] S. Minato and H. Arimura, "Frequent pattern mining and knowledge indexing based on zero-suppressed BDDs," Proc. 5th International Workshop on Knowledge Discovery in Inductive Databases (KDID-2006), pp.83–94, Sept. 2006.

[12] S. Tani, K. Hamaguchi, and S. Yajima, "The complexity of the optimal variable ordering problems of shared binary decision diagrams," 4th International Symposium on Algorithms and Computation, LNCS-762, pp.389–398, Springer, 1993.

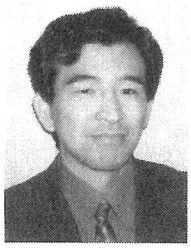
[13] M.J. Zaki, "Scalable algorithms for association mining," IEEE Trans. Knowl. Data Eng., vol.12, no.2, pp.372–390, 2000.

（平成 19 年 5 月 31 日受付，9 月 28 日再受付）



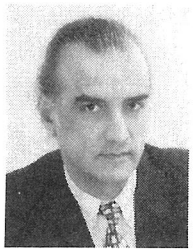
岩崎　玄弥

2006 北大・工・電子卒．現在，同大大学院修士課程に在籍．二分決定グラフを用いたデータベース解析処理の研究に従事．情報処理学会会員．



湊　真一　（正員）

1988 京大・工・情報卒．1990 同大大学院修士課程了．1995 同博士（工学）．1990 NTT 入社．以来，大規模論理データの表現と処理アルゴリズムの研究開発に従事．1997 米国スタンフォード大客員研究員．1999 NTT 未来ねっと研究所主任研員．2004 より北大情報科学研究科助教授（2007 准教授）．著書 “Binary Decision Diagrams and Applications for VLSI CAD”（Kluwer, 1995）．1992 本会 75 周年記念論文優秀賞，2000 情報処理学会山下記念研究賞，2006 人工知能学会研究会優秀賞．IEEE，情報処理学会，人工知能学会各会員．



トーマス　ツォイクマン

1981 ドイツ・フンボルト大大学院修士課程了．1983 同博士（理学）．同年同大助手．1994 九大理学部基礎情報学研究施設助教授．1997 同大大学院システム情報科学研究科教授．2000 ドイツ・リューベック大情報理学専攻教授．2004 より北大情報科学研究科教授．機械学習や知識処理に関する計算理論の研究に一貫して従事．同分野の国際学術誌・著名国際会議での研究発表及び学会運営等，継続的に活動している．1999 より計算論的学習理論国際会議（ALT）運営委員長．