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

湊　真一[†]　　有村　博紀[†]

Efficient Method of Transaction Database Analysis Using Zero-Suppressed BDDs

Shin-ichi MINATO[†] and Hiroki ARIMURA[†]

あらまし　大規模なトランザクションデータを，計算機上でコンパクトに表現して効率的に処理することは，データマイニングにおける重要な基盤技術の一つである．本論文では，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）をデータマイニングの分野に応用する方法を提案する．BDD の中でも「ゼロサプレス型 BDD」（ZBDD: Zero-suppressed BDD）と呼ばれるデータ構造は，大規模な組合せ集合データを非明示的に列挙し，効率良く演算処理することができる．この ZBDD を用いて，トランザクションデータベースに関する種々の計算処理を行う手法について述べ，実験結果を示す．

キーワード　二分決定グラフ，BDD，データマイニング，知識発見，データベース解析

1. まえがき

高速ネットワークと大容量記憶装置の急速な発展によって，データベースに蓄積された大量のデータから，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 [3] 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 [9], [11], [21].

本論文では，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）[6] をデータマイニングの分野に応用する方法を提案する．BDD の中でも「ゼロサプレス型 BDD」（ZBDD: Zero-suppressed BDD）[16], [17] と呼ばれるデータ構造は，大規模な組合せ集合データを非明示的に列挙し，効率良く演算処理することができる．この ZBDD を用いて，トランザクションデータベースに関する種々の計算処理を行う手法を述べ，実験結果を示す．

過去に，BDD をデータマイニング分野に応用する取組みとしては，汎用 BDD パッケージ BEM-II [15] を用いたルール抽出法 [12] が報告されている．しかし残念ながら実用的な規模の例題に適用できるような性能を得られてはいなかった．これはトランザクションデータベースに一般的に見られる性質が，通常の BDD には適しておらず，ゼロサプレス型の BDD に適合していることに起因する．本論文ではこの点についても述べる．

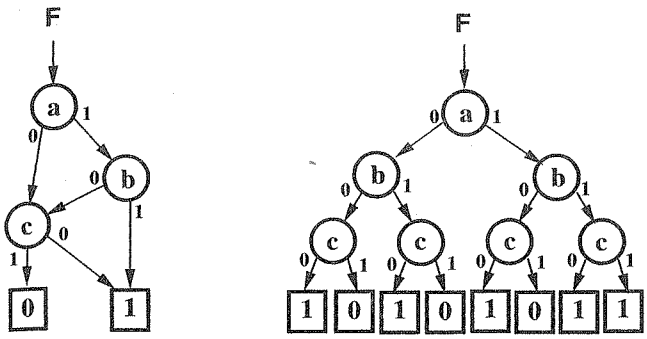
最近，帰納データベース（Inductive Databases）と呼ばれる，データマイニングを統合的な発見プロセスとしてとらえる研究が盛んである [4], [14]．本論文では，ZBDD をトランザクションデータにおける統合的な発見プロセスを実現するための基盤技術として位置づけ，パターンの発見から解析に至る多様な演算を効率良く実現することを目指す．

2. BDD とゼロサプレス型 BDD

2.1 BDD

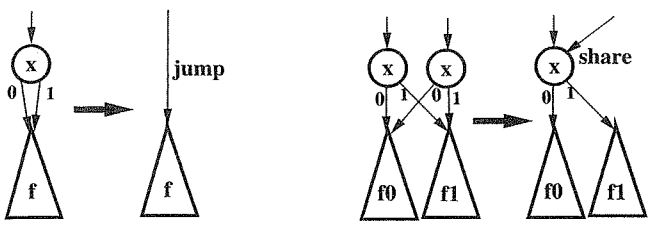
BDD は，図 1 (a) に示すような論理関数のグラフによる表現である．一般に，論理関数のそれぞれの変数について，0，1 の値を代入した結果を，2 分岐の枝（0–枝/1–枝）で場合分けし，最終的に得られる論理関

[†] 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) BDD　(b) 場合分け二分木

図 1　$F = (a \wedge b) \vee \bar{c}$ を表す BDD と場合分け二分木
Fig. 1　BDD and binary tree: $F = (a \wedge b) \vee \bar{c}$.



(a) 冗長節点の削除　(b) 等価な節点の共有

図 2　BDD の簡約化規則
Fig. 2　Reduction rules of ordinary BDDs.

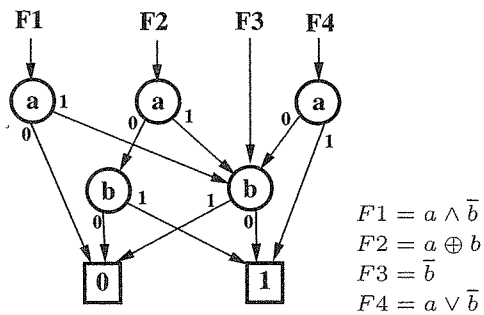


図 3　複数の BDD の共有化
Fig. 3　Shared multiple BDDs.

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 (b) 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，

- 冗長な節点を削除する（図 2 (a)）．
- 等価な節点を共有する（図 2 (b)）．

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，図 1 (a) のような「既約」な形が得られ(注1)，論理関数をコンパクトかつ一意に表せることが知られている [1], [13]．更に，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，互いにグラフを共有することが可能であり（図 3），一つのメモリ空間の中で多数の論理関数データを扱うことができる．

BDD は，パリティ関数や加算器等を含む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（AND，OR 等）の結果を表す BDD を直接生成するアルゴリズム [6] 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という特長をもっている（詳細は文献 [8] を参照）．

このようなデータ構造に基づく BDD 処理系が 1990 年代前半ごろから開発され，主に VLSI CAD の分野を中心として，大規模な論理関数データの処理に広く用いられている．

2.2　ゼロサプレス型 BDD による組合せ集合の操作

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．n 個のアイテムから任意個を選ぶ組合せは 2^n 通り存在するので，組合せ集合としては 2^{2^n} 通りの取り方がある．例えば，a, b, c, d, e という五つのアイテムに関して，$\{ab, e\}$, $\{abc, cde, bd, acde, e\}$, $\{1, cd\}$, $\emptyset$ は，いずれも組合せ集合の一例である（本論文では “1” は空の組合せ要素を表し，“$\emptyset$” は空の集合を表すこととする）．組合せ集合は，組合せ問題の解集合を表現する基本的・汎用的なデータ構造であり，実問題においては，スイッチの ON/OFF の組合せ，故障の組合せ，ネットワーク経路の組合せ，など，様々な局面で現れる．

ある一つの組合せ集合は，一つの論理関数（特性関数と呼ばれる）に対応づけることができる．例えば，図 4 は $ab + \bar{a}c$ という論理関数を表す真理値表であるが，見方を変えると，$\{ab, abc, bc, c\}$ という組合せ集合も表現している．特性関数を BDD で表すことにより，組合せ集合を非明示的に表現することができる．このとき，論理関数の AND/OR 演算は，集合の交わり（intersection）/結び（union）の演算にそのまま対応するので，多数の要素を含む集合同士の演算を，BDD 処理系でまとめて実行することが可能となる．たとえ要素数が非常に多くても，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

(注1)：一般に BDD という場合には，変数の順序が固定されないものや，既約でないグラフも対象に含まれるが，本論文では上記の簡約化処理を施したものだけを扱う．

a	b	c	F	
0	0	0	0	
0	0	1	1	$\rightarrow c$
0	1	0	0	
0	1	1	1	$\rightarrow bc$
1	0	0	0	
1	0	1	0	
1	1	0	1	$\rightarrow ab$
1	1	1	1	$\rightarrow abc$

論理関数として見た場合:
$F = a\,b + \overline{a}\,c$

組合せ集合として見た場合:
$F = \{ab, abc, bc, c\}$

図 4 組合せ集合と論理関数の対応
Fig. 4 Correspondence between Boolean function and set of combinations.

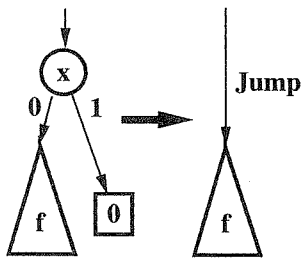


図 5 ゼロサプレス型 BDD の簡約化規則
Fig. 5 ZBDD reduction rule.

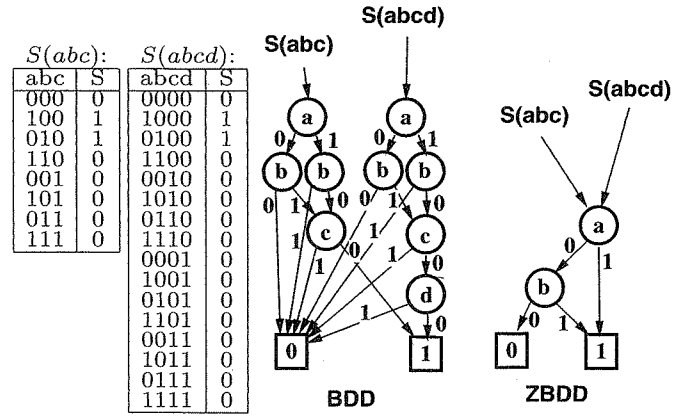


図 6 ゼロサプレス型 BDD の効果
Fig. 6 Example of ZBDD effect.

れて，記憶量や計算時間が大幅に（ときには指数関数的に）削減される場合がある．更に，組合せ集合に特化した「ゼロサプレス型 BDD」(ZBDD) [16] 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 と異なる．通常の BDD では，図 2 (a) のように，0–枝と 1–枝が同じ節点を指しているときに，この節点を冗長として取り除くのに対し，ZBDD ではそのような節点を取り除くことはせず，その代わりに，図 5 のように 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．

ZBDD では，0–枝はその節点のアイテム x を含まない部分集合（コファクタ）を，1–枝はアイテム x を含む部分集合（コファクタ）を表している．もしも 1–枝が 0–終端節点を直接指しているとすると，その組合せ集合には，アイテム x は一度も出現しないということになる．そのような関係ないアイテムに関する節点は冗長とみなして削除し，0–枝側に直結させるというのが，ZBDD の考え方である．この簡約化規則により，組合せ集合に影響を与えない（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，通常の 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．

以下に ZBDD の特長を列挙する．

- ZBDD を用いると，組合せ集合に無関係な（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取り除かれる．通常の BDD ではそのような節点は残ってしまい，折角の BDD の共有化のメリットが損なわれる場合がある．図 6 に例を示す．組合せ集合 $S(abc)$ と $S(abcd)$ は，想定するアイテム変数のドメインの違いを除けば等価な組合せ集合である．このとき，アイテム c, d は，この組合せ集合には関与していないにもかかわらず，通常の BDD 表現では c, d に関する節点が残ってしまう．一方，ZBDD では，c, d に関する節点は自動的に取り除かれ，$S(abc)$ と $S(abcd)$ は同じ形の ZBDD となる．
- 特に疎な組合せからなる集合に対して ZBDD は顕著な効果がある．例えば 1,000 個のアイテムから数個を選ぶような組合せでは，ほとんどの入力変数は 0 である．このような場合にゼロサプレス簡約化規則が非常に効果的である．
- グラフの最上位節点から，1 のラベルをもつ終端節点へ至る経路の数が，集合の要素数と完全に一致する（通常の BDD では必ずしも一致しない）．
- ZBDD は，もしも等価な節点が存在せず共有が全く行われなかった場合，線形リスト表現でアイテム組合せを羅列したのとほぼ同じデータ構造となる（図 7 に例を示す．最上位節点から 1 の終端節点に至る経路が三つあり，それぞれが集合の組合せ要素を線形リストで羅列表現したものになっている）．つまり，ZBDD の記憶量は最悪の場合でも，データを明示的に羅列した場合に要する記憶量（の定数倍）を超える

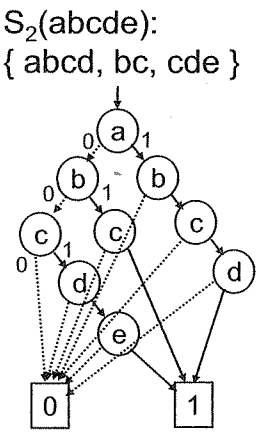


図 7　節点共有ができない例
Fig. 7　Explicit representation by ZBDD.

表 1　ZBDD 処理系の基本演算
Table 1　Primitive ZBDD operations.

"∅"	空集合（0-定数節点を返す）.
"1"	0 個のアイテムからなる組合せの集合（1-定数節点を返す）.
P.top	組合せ集合 P の最上位の節点のアイテム番号を返す.
P.offset(v)	P の中でアイテム v を含まない組合せを集めた部分集合を取り出す.
P.onset(v)	P の中でアイテム v を含む組合せを集めた部分集合を取り出し，各組合せから v を取り除いた組合せ集合を返す.
P.change(v)	P の各組合せについて，アイテム v の有無を反転させる.
$P \cup Q$	P と Q の結び（union）.
$P \cap Q$	P と Q の交わり（intersection）.
$P - Q$	差集合（P にあって Q にないもの）.
P.count	P の要素数を数える.

ことはない（通常の BDD では，疎な組合せ集合を扱う場合にオーバヘッドが大きいが，ZBDD ではオーバヘッドがほとんどない）.

ZBDD 処理系では，表 1 に示すように，基本的な演算として，"0"，"1" の定数集合を返す演算，与えられた ZBDD の特定のアイテム変数に 0，1 を代入・操作する offset，onset，change の各演算，与えられた二つの ZBDD に関する組合せ集合の結び，交わり，差集合を表す ZBDD を新しく生成する二項集合演算，及び要素数を数える演算などが用意されている．ZBDD を構築する場合，まず単一のアイテム変数からなる ZBDD を作っておき，それらの二項集合演算を繰り返し実行することにより，より複雑な組合せ集合を表す ZBDD を構築していく．二項集合演算は，BDD の論理演算アルゴリズムと同様の原理に基づいており，最上位の変数にそれぞれ 0，1 を代入して二分木状に展開して計算を行う再帰的なアルゴリズムである．単純に二分木展開を続けると変数の個数に対して指数関数的な時間がかかってしまうが，過去の演算結果を記憶・参照するハッシュテーブルを用いて冗長な計算を省く技法により，演算に関与する ZBDD の節点数にほぼ比例する計算時間と記憶量で，効率良く二項演算を実行できることが知られている．ZBDD の処理系に関する詳細は文献 [16], [17] 等に記載されている.

3. トランザクションデータベース解析への応用

前章で紹介した ZBDD 処理系を用いて，大規模なトランザクションデータを非明示的に表現し，効率良く解析処理を行う方法について述べる．ここでは，複数のアイテムの任意の組合せ（トランザクションまたはタプルと呼ぶ）を 1 個のレコードとする 2 値データベースを対象とする．このようなデータに関する基本的な問題として，頻出パターン集合発見問題 [3] や極大頻出パターン発見問題 [7] 等のデータマイニングの問題が従来より研究されている．近年，FP-Tree [11] のようなグラフ構造を用いて主記憶上にパターン集合データをコンパクトに表現し，高速に計算を行う手法が注目されている．主記憶上で大規模な組合せ集合データを扱うという点では，ZBDD を用いる手法も同様のアプローチであり，効果的に適用できる可能性がある.

3.1　データマイニングで扱われるデータの特性

頻出パターン集合発見問題で用いられている代表的な例題として，国際会議 FIMI-2003 のベンチマークデータ [10] から抜粋したものを表 2 に示す．$\#I$ 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.

これを見ると，アイテムの平均出現頻度が非常に小さい例題が多いことが分かる．実問題を考えても，商店の陳列商品数に比べて 1 人の客の購入品数はずっと少ないことが普通であるし，ある Web ページがリンクしているページの数はインターネット全体のページ数に比べてはるかに少ないのが自然である．つまり，データマイニングの問題では非常に疎な組合せからなる集合(注2)を扱う例が多いということであり，その

(注2)：「疎な集合」と「疎な組合せからなる集合」は意味が異なるので注意されたい．ここでは集合の要素数ではなく，各要素におけるアイテム出現頻度を問題としている.

表 2　代表的なデータマイニング例題の特徴
Table 2　Statistics of typical benchmark data.

Data name	$\#I$	$\#T$	total$\|T\|$	avg$\|T\|$	avg$\|T\|/\#I$
T10I4D100K	870	100,000	1,010,228	10.1	1.16%
T40I10D100K	942	100,000	3,960,507	39.6	4.20%
chess	75	3,196	118,252	37.0	49.30%
connect	129	67,557	2,904,951	43.0	33.33%
mushroom	119	8,124	186,852	23.0	19.32%
pumsb	2,113	49,046	3,629,404	74.0	3.50%
pumsb_star	2,088	49,046	2,475,947	50.5	2.42%
BMS-POS	1,657	515,597	3,367,020	6.5	0.39%
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%
accidents	468	340,183	11,500,870	33.8	7.22%

ような場合には ZBDD の簡約化規則が極めて効果的に働く．もしも平均出現頻度が 1%であれば，通常の BDD に比べて ZBDD が約 100 倍コンパクトに表現できる可能性がある．言い換えると，データマイニングの問題では，通常の BDD ではなく ZBDD を使わなければ有効な結果が得られない可能性が高い．

過去に，BDD をデータマイニング分野に応用する取組みとしては，汎用 BDD パッケージ BEM-II [15] を用いたルール抽出法 [12] が報告されている．この方法は，指定した回数以上の頻出アイテムの組合せを求める手続きを，論理変数及び論理演算の式で記述し，これを BEM-II を用いて処理することにより，計算結果の BDD を構築するというものである．しかし残念ながら実用的な規模の例題に適用できるような性能を得られてはいなかった．その主な理由は上述のとおり，扱うデータベースが，疎な組合せからなる集合であることが多いためと考えられる．本論文の **4.** では，BDD と ZBDD の比較実験についても示す．

3.2　ZBDD を用いたタプル頻度表の生成

タプル頻度表とは，与えられたトランザクションデータの中に，同じタプル（アイテムの組合せ）が何回出現したかを，タプルごとに数えて表にしたものである．通常の組合せ集合では，組合せパター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．例えば，表 2 の BMS-WebView-1 データに含まれる 59,602 個のレコード中に，最も頻度が高いタプルは 1,533 回出現しており，上位 10 個のタプルで総計 8,404 回（タプル全体の 14%）出現している．図 8 に，簡単なデータベースの例とそのタプル頻度表との関係を示す．

ZBDD を用いてタプル頻度表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（すなわちタプルの有無）を表すことしかできないため，単純な ZBDD では出現回数を記憶できない．そこで図 9 のように，n 個の ZBDD $\{F_0, F_1, \ldots, F_{n-1}\}$ をベクトル状に並べ，最大 $2^n - 1$ までの出現回数を表現することとする．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（=奇数回出現する）タプルの集合を F_0，次のビットが 1 になるような出現回数のタプルの集合を F_1，という形で，F_{n-1} まで並べることにより，各タプルの出現回数を非明示的に表すことができる．例えば，図 9 では，$F_0 = \{abc, ab, c\}$，$F_1 = \{ab, bc\}$，$F_2 = \{abc\}$ と分解されている．

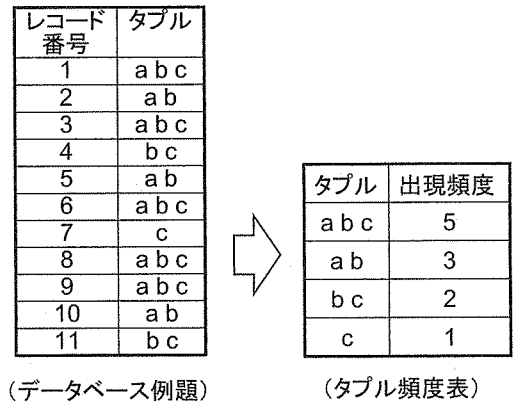


図 8　データベース例題とタプル頻度表
Fig. 8　Database example and tuple-histogram.

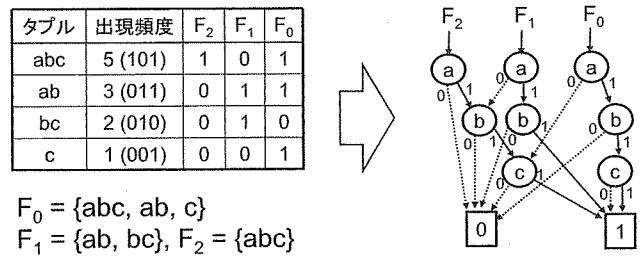


図 9　ZBDD の 2 進ベクトルによるタプル出現頻度の表現
Fig. 9　ZBDD vector for tuple-histogram.

トランザクションデータのファイルを読み込んでZBDDのベクトルを作るには，データベースからレコードを1個ずつ読み込み，そのタプル情報を頻度表に累算していけばよい．具体的には，読み込んだタプルのアイテム組合せのみを表す組合せ集合 T を表すZBDDを作る．これは，まず“1”（アイテム0個のタプルを表す集合）を作り，これにChange演算を繰り返し適用して必要なアイテムを付加することで作成できる．すなわち，タプルに出現するアイテム数に比例する時間で作成可能である．次に，F_0 と T を比較し，共通部分がなければ T を F_0 に加えて終わり，もしも共通部分があれば F_0 から T を引き去り，次の F_1 にけた上げさせる．このようにして，けた上がりがなくなるまで上位のけたに波及していく．この頻度表 F へのタプル T の累算を F.add(T) で表す．

F.add(T) の計算時間を分析すると，各けたごとの処理は定数回の二項集合演算で実現できるので，関与するZBDDの節点数にほぼ比例する．そしてけた上げ処理を繰り返す回数は，出現頻度の最大値のlogで抑えられる．また，使用記憶量は，関与するZBDDの節点数に比例する．

以上のタプル累算処理を，すべてのデータレコードについて繰り返し実行すれば，タプル頻度表が完成する．データベース D からタプル頻度表 F_T を作る手続きは次のようになる．

$F_T = \mathbf{0}$
forall $T \in D$ **do**
  $F_T = F_T$.add(T)
return F_T

---

2. で述べたように，ZBDDの節点数は，最悪の場合でも組合せ集合データを明示的に列挙したときの記憶量を超えない．したがって，ZBDDのベクトルでタプル頻度表を表した場合，データベースの総文字数と2進符号けた数の積に比例する上界が存在するため，ZBDDの大きさが指数爆発的に増大することはない．そして，部分的に共通のパターンをもつタプルが多ければ，ZBDDのグラフが共有され，コンパクトな表現が得られる．

いったん，ZBDDでタプル頻度表を作ってしまえば，「出現回数 α 以上のタプル集合を取り出せ」という問題でも，α を2進数で表し，これとの大小比較をビットごとのZBDDの集合演算で記述すれば，グラフのサイズに比例する時間で容易に計算できる．

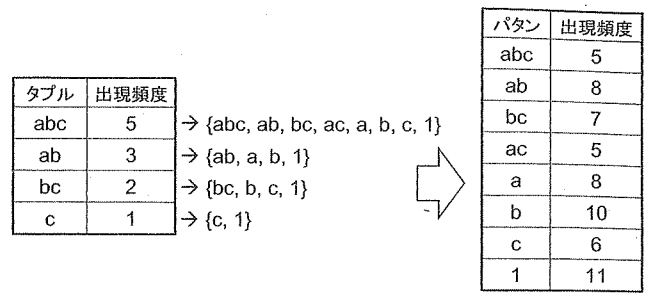


図10　タプル頻度表とパターン頻度表
Fig. 10　Tuple-histogram and pattern-histogram.

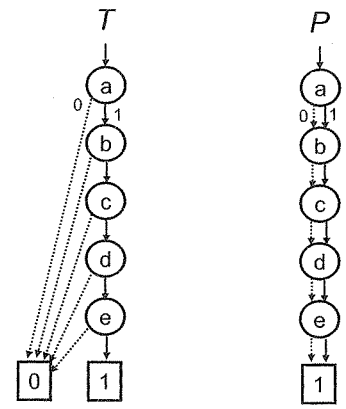


図11　タプルとそれに含まれるパターン集合を表すZBDD
Fig. 11　ZBDDs for a tuple and all sub-patterns.

3.3　パターン頻度表の生成

ここで「パターン」とは，トランザクションデータの各タプルの中に出現するアイテムの部分集合のことを呼ぶ．パターン頻度表は，トランザクションデータに含まれるすべてのパターンについて，その出現回数を数え上げて表にしたものである．タプル頻度表とパターン頻度表の対応例を図10に示す．

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データ量となり，小規模な例題を除けば，完全なパターン頻度表を作ることは困難である．そこで実用的な問題として，適当なしきい値 α より多く出現する頻出パターンの集合を，現実的な計算時間と記憶量で求めるアルゴリズムが盛んに研究されている[9]．

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ため，従来不可能だと思われていた完全なパターン頻度表を，ある程度の規模までは現実的に生成できる可能性がある．例えば，図11は，五つのアイテムからなるタプル $T = abcde$ と，それに含まれるすべてのパターン集合 $P = \{1, a, b, c, d, e, ab, ac, bc, cd, abc, \ldots, abcde\}$ を，

それぞれ ZBDD で表現したものである．これを見ると，2^k 個のパターン集合をわずか k 個の節点で非明示的に表現できることが分かる．つまり，各タプルに含まれるパターンを列挙する ZBDD は，タプルに出現するアイテム数に比例する時間で作成可能である．そしてタプル頻度表を作るのと全く同様に，2 進数で符号化した ZBDD のベクトルに累算していけば，パターン頻度表を生成できる．

以上をまとめると，データベース D からパターン頻度表 F_P を作る手続きは次のようになる．

$F_P = \mathbf{0}$
forall $T \in D$ **do**
  $P = T$
  forall $v \in T$ **do**
    $P = P \cup P.\mathrm{onset}(v)$
  $F_P = F_P.\mathrm{add}(P)$
return F_P

パターン頻度表を生成するための計算量は，タプル頻度表生成の場合と同様に，ZBDD の節点数に依存する．ただし，パターン頻度表の場合，データを明示的に列挙したときの記憶量が，アイテム数に対して指数関数であるので，ZBDD の節点数を抑える上界が非常に大きい．そのため，パターン集合を累算していく過程で，タプル頻度表の場合よりも急速に ZBDD 節点数が増大する．したがって，ある程度の規模の問題までしか，この方法は適用できない．しかし，もし完全なパターン頻度表を作成できてしまえば非常に強力であり，その後は様々なデータマイニングの解析処理を，ZBDD 処理系の演算により効率良く行うことが可能となる．どの程度の規模まで実際にパターン頻度表を生成できるのかは，非常に興味深い問題である．後の章で実験結果を示す．

なお，上記で述べたように，データベースのレコードごとにパターン集合を作って累算していく方法のほかに，いったん，タプル頻度表 F_T を完成させてから，タプル頻度表全体に ZBDD の演算処理を適用して，パターン頻度表 F_P を作り出す方法もある．

$F_P = F_T$
forall $v \in F_T$ **do:**
  $F_P = F_P.\mathrm{add}(F_P.\mathrm{onset}(v))$
return F_P

どちらが速く計算できるかは今後の検討課題だが，いずれの方法でも最終的に得られる ZBDD の形や節点数は同じである．

3.4 タプル/パターン頻度表の活用

ZBDD によりタプル頻度表やパターン頻度表を生成すると，様々な計算処理が効率良く行える．その例をいくつか示す．タプル頻度表またはパターン頻度表を $F : \{F_0, F_1, \ldots, F_{n-1}\}$ とおく．

- 任意のアイテムあるいはパターン P を含むような，タプル/パターン集合 S を抽出することができる．

$S = \bigcup F_k$
forall $v \in P$ **do:**
  $S = S.\mathrm{onset}(v).\mathrm{change}(v)$
return S

逆に条件を満たさないタプル/パターン集合を抽出することも可能である．集合演算 $\bigcup F_k - S$ を求めればよい．

また，抽出したタプル/パターンが何通りあるかを高速に数える手続き $S.\mathrm{count}$ が用意されている．計算時間は，抽出した要素数ではなく ZBDD の節点数にほぼ比例する．

- 様々なしきい値 α を与えて，それを超える出現頻度のタプル/パターン集合を抽出することができる．計算時間は ZBDD の節点数にほぼ比例する．これを何回か繰り返せば，上位 m 個までの頻出集合を抽出するためのしきい値 α を求めることができる．従来の方法では，α を変えるたびに計算をし直す必要があったが，ZBDD をいったん作ってしまえば，異なる α に対して何度でも頻出集合を効率良く抽出することができる．
- 確率統計処理や機械学習の分野でも用いられる支持度（support）や確信度（confidence）等の指標が効率良く計算できる．

ZBDD を用いた手法の特長は，主記憶上に強力なデータベースをいったん構築し，それに対して様々なクエリーを対話的に適用できることである．しかもクエリーは，集合演算の組合せにより数学的な式で記述できるところが興味深い点といえる．

4. 実験結果

4.1 タプル頻度表の生成

本手法の実用性を評価するための実験として，**3.** で示したのと同じ代表的な例題について，タプル頻度表を生成する実験を行った．使用した PC は Pentium4, 800 MHz, SuSE Linux 9, 主記憶 512 MByte で，

表 3　タプル頻度表の生成
Table 3　Generation of tuple-histograms.

Data name	$\#I$	$\#T$	total$\|T\|$	本手法		BEM-II による方法 [12]	
				ZBDD 節点数	Time(s)	BDD 節点数	Time(s)
T10I4D100K	870	100,000	1,010,228	552,429	43.2	(>10,000,000)	–
T40I10D100K	942	100,000	3,960,507	3,396,395	895.0	(>10,000,000)	–
chess	75	3,196	118,252	40,028	1.4	66,844	6.2
connect	129	67,557	2,904,951	309,075	58.1	647,387	268.9
mushroom	119	8,124	186,852	8,006	1.5	25,299	9.8
pumsb	2,113	49,046	3,629,404	1,750,883	188.5	(>10,000,000)	–
pumsb_star	2,088	49,046	2,475,947	1,324,502	123.6	(>10,000,000)	–
BMS-POS	1,657	515,597	3,367,020	1,350,970	895.0	(>10,000,000)	–
BMS-WebView-1	497	59,602	149,639	46,148	18.3	1,011,675	91.2
BMS-WebView-2	3,340	77,512	358,278	198,471	138.0	(>10,000,000)	–
accidents	468	340,183	11,500,870	3,877,333	107.0	(>10,000,000)	–

表 4　“mushroom” のパターン頻度表の生成
Table 4　Pattern-histogram for “mushroom.”

ZBDD 節点	Time(s)	パターン総数
513,762	242.0	5,574,930,438

表 5　FP-Tree を用いた “mushroom” の頻出パターン生成
Table 5　FP-Tree-based method for “mushroom.”

最小出現回数 α	Time(s)	頻出パターン総数
81	22.60	91,273,269
40	67.96	295,117,613
16	244.06	1,176,182,553
8	494.05	1,983,493,667
4	891.31	(> 2 G)
1	1,322.48	(> 2 G)

ZBDD の最大節点数は 1,000 万個とした．表 3 に実験結果を示す．$\#I$ はアイテム数，$\#T$ はタプル数（レコード数），total$|T|$ はタプル長（タプルに含まれるアイテム数）の総和を表している．これらの例題に対して，本手法（ZBDD を使用）と，BEM-II による方法 [12]（通常の BDD を使用）の両方を実行し，結果を比較した．

まず本手法の結果を見ると，すべての例題について，現実的な時間でタプル頻度表を生成することができた．それぞれの ZBDD 節点数を見ると，総タプル長の総和とほぼ同数，多くの場合は数分の 1 以下に抑えられていることが分かる．

一方，通常の BDD を使用した場合，多くの例題で数倍から数十倍以上の記憶量や計算時間を要しており，メモリ容量不足で途中終了する例が続出した．特に，アイテム数の多い例題では ZBDD との差が顕著である．この実験結果からも，データマイニング分野における ZBDD の優位性が示されている．

4.2　パターン頻度表の生成

パターン頻度表については，今回用いた例題の中では mushroom についてのみ，与えられた主記憶の範囲で生成することができた（表 4）．mushroom はタプル頻度表を生成したときの ZBDD 節点数が最も小さかった例題である．タプル頻度表の ZBDD 節点数とパターン頻度表の ZBDD 節点数には正の相関があると考えられる．mushroom のパターン頻度表を表現するために，タプル頻度表に比べて数十倍の ZBDD 節点数を要している．逆にいうと，タプル頻度表が主記憶容量の数十分の 1 で表現できるようであれば，完全なパターン頻度表を生成できる可能性があるということになる．この ZBDD が表しているパターン数は，実に 5,574,930,438 に上る（32 ビット整数のレンジを超えている）．ZBDD の節点数はその約 10,000 分の 1 であり，著しくデータ量が圧縮されていることが分かる．

本手法の性能を従来手法と比較するため，FP-Tree [11] を用いたプログラムで，同じ mushroom に対して，出現回数が α 以上の頻出パターン集合を求めた場合の計算時間を表 5 に示す．α を小さくしていくと，頻出パターン集合が増大していき，数百～1,000 秒以上の計算時間を要することが分かる．これに対して，ZBDD によるパターン頻度表は 242 秒で生成でき，一度生成してしまえば任意の α について頻出パターン集合を何度でも高速に抽出できる．このように，特に多数のパターン集合を保持する場合において，本手法が非常に有効であることが分かる．

なお，単に頻出パターン集合を求めるだけであれば，しきい値 α を比較的大きな値に設定して，抽出パターン数を現実的な規模に絞り込む使用法が一般的であり，そのような場合には，従来の FP-tree 法の方が効率が良いと考えられる [2]．しかし，FP-tree 法は頻出

表 6　部分サンプリングデータに対するパターン頻度表生成
Table 6　Pattern-histograms for sampling data.

Data name	$\#T_{all}$	$\#T_{done}$	ratio%	本手法		FP-tree: Time(s)	
				ZBDD 節点数	Time(s)	$\alpha = 1$	$\alpha = 50$
T10I4D100K	100,000	43,395	43.40	9,968,062	1,711	103.5	2.2
T40I10D100K	100,000	740	0.74	9,863,736	123	(>10,000)	0.1
chess	3,196	703	22.00	8,242,212	919	(>10,000)	(>10,000)
connect	67,557	2,095	3.10	9,146,429	2,502	(>10,000)	(>10,000)
mushroom	8,124	8,124	100.00	513,762	242	1,322.5	44.2
pumsb	49,046	61	0.12	7,018,198	89	(>10,000)	1.2
pumsb_star	49,046	288	0.59	8,343,418	375	(>10,000)	1,869.9
BMS-POS	515,597	9,837	1.90	9,679,161	266	1,675.4	0.4
BMS-WebView-1	59,602	17,757	29.79	9,372,436	134	(>10,000)	0.2
BMS-WebView-2	77,512	39,045	50.37	9,515,386	1,113	(>10,000)	0.9
accidents	340,183	133	0.04	8,394,811	223	(>10,000)	0.2

パターン抽出に特化されているのに対し，本手法はすべての組合せパターンの出現頻度を頻度表として保持することにより，出現頻度に限らず，様々な条件を与えて部分集合を繰り返し抽出し，インタラクティブにデータベース解析を進めることができる．このような使用法が可能であることが本手法の特長である．

4.3　タプル/パターン頻度表の活用

いったん，頻度表を生成した後に，これに種々の操作を加える実験を行った．まず，上記の例題 mushroom に 1 個以上含まれているすべてのパターン集合 S を生成した（これに要する計算時間はパターン頻度表を作成するのとほぼ同じである）．その後，あるアイテム v に関して，それを含むパターンを取り出す演算 $S.\mathrm{onset}(v)$ を実行した．同じ S に対して，それに含まれる 119 個のアイテムすべてについてそれぞれ $S.\mathrm{onset}(v)$ 実行したところ，1 回当りの平均計算時間は，わずか 0.10 秒であった．その他の ZBDD 基本演算（例えば offset，change，union，intersection 等）についても，ほぼ同じ計算時間となる．

このように，ZBDD を用いた非明示的なパターン集合表現は，膨大な数のパターンを圧縮してメモリ上に表現する非常に強力な方法であり，しかも圧縮されたデータを解凍することなく，圧縮時のデータ量にほぼ比例する時間で効率良く演算処理できることが大きな特長である．更に ZBDD 表現は，頻出パターン抽出のような特定の演算処理に特化されたものではなく，組合せ集合の演算に基づく数式で表現できる処理であれば何でも実行できるので，汎用性の面でも非常に優れている．

4.4　サンプリングデータに対するパターン頻度表の生成

例題 mushroom だけでなく，より大規模な例題に対してパターン頻度表を生成する実験の結果を表 6 に示す．mushroom を除き，与えられたメモリ容量の範囲では，データベースからすべてのタプルを読み込み，完全なパターン頻度表を生成することはできないが，データベースの一部を取り出したサンプリングデータについて，パターン頻度表を作成することは可能である．我々の実験では，データベースの先頭から順にレコードを一つずつ読み込んでパターン頻度表の積算を実行し，メモリあふれを起こす直前までのレコード数を調べた．$\#T_{all}$ は全レコード数，$\#T_{done}$ は処理できたレコード数，ratio はその割合を示している．本実験の環境（ZBDD 節点数の上限 10,000,000 個）では，中規模の例題であれば，20%以上のレコード数の処理ができた．このようにサンプル数を調節することで，解析の規模と精度のトレードオフを図ることが可能である．

一方，参考のため，同じ例題，同じレコードを取り出して，FP-tree 法を用いた頻出パターン抽出を行い，処理時間を比較した．最小出現頻度 α が 1 の場合と 50 の場合の 2 通りの結果を表 6 に示している．$\alpha = 1$ の場合は，抽出パターン数が膨大であるため，ほとんどの例題において現実的な時間内で計算が終了しない（本実験では約 3 時間で打切りとした）．$\alpha = 50$ とすれば，多くの例題で FP-Tree 法の方が短時間に終了するが，それでもまだ時間がかかる例もある．α に関する挙動は例題の性質に大きく依存する．いずれにしても，すべての α に関する頻度表を一度に構築可能な本手法の有効性が示されている．

なお，この実験ではデータベースの先頭から順に $\#T_{done}$ 個のレコードを取り出したが，実際の応用では乱数または何らかの基準を用いてサンプリングを行うこともできる．このような部分的データに対するパ

ターン頻度表でも，統計的なサンプリング処理結果としての意味は十分にあると考えられる．

5. 関連研究

以下に，従来知られているデータベース解析手法の主なものについて，本手法との関係を述べる．

- FP-tree 表現に基づく方法 [11]

従来手法の中では最も性能が良い頻出パターン抽出法の一つである．木構造のグラフ表現を用いて，圧縮したパターン集合表現をメモリ上に構築するという点では，ZBDD 手法と類似している．ただし，FP-tree は木構造の根に近い部分のパターンは共有できるが，ZBDD のように葉に近い方のサブグラフを共有する機構はもっていないので，基本的なパターン圧縮能力は ZBDD よりも劣る．一方，FP-tree は用途を頻出パターン抽出に特化し，グラフの節点にパターン出現度数を記録することによりアルゴリズムを高速化している．したがって，比較的少ない個数の頻出パターン抽出のみを行う場合は，FP-tree の方が有利である．

- ハッシュによるアイテムカウンティング [18]

この手法は主にサイズ 2 のアイテム集合発見の高速化に有効なのに対して，ZBDD による手法はすべてのサイズの集合の効率化に有効である．また，この手法は近似的なハッシュテーブルを用いているのに対して，ZBDD は完全な（ロスのない）ハッシュテーブルを作る点が異なる．

- 更新を意識した動的カウント [5]

これは，本質的に漸増的なストリーム的処理の繰返しにより，高速化を図る手法であり，ZBDD 手法とは異なる特性と応用をもつ．特に，データ分布によっては必ずしも最悪時の高速性の保障がない．これに対して，ZBDD による手法は，ほとんどの演算に対して，最悪時の演算性能が，ZBDD 表現のほぼ線形時間となる性能保証がある点で異なっている．

- 分割 [19] 及びサンプリング [20]

これらの手法は，ZBDD 手法とは独立の関係にあり，ZBDD 技法と組み合わせて使用できると期待される．今後の研究課題として興味深い．

上記のすべての手法は，頻出アイテム集合だけを発見することに特化した手法である．これに対して，ZBDD 手法はそれだけでなく，パターン発見に続く，パターンの後処理（複雑な問合せや統計処理）も同じ枠組みで行うことができる点で，上記の手法すべてと大きく異なる．

6. むすび

ZBDD 処理系をデータマイニングの分野に応用する手法について述べた．本手法により大規模な組合せ集合データを非明示的に列挙し，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．

本手法を用いても，大規模な例題に対して完全なパターン頻度表を生成することは依然として困難であるが，従来手法と同様に，出現頻度が低いアイテムやパターンを前処理で取り除いたり，サンプリング処理を行ったり，極大パターンのみを扱う，等の工夫を行えば，より大規模なデータにも対応できるようになると予想される．

ZBDD 処理系は，トランザクションデータにおける統合的な発見プロセスを実現するための基盤技術として有望であり，今後の発展が期待される．

謝辞 本研究の一部は文部科学省科学研究費特定領域研究 (2)「情報学」（領域番号 006）「最適パターン発見に基づく大規模半構造データからの知的情報獲得システムの開発」（課題番号 16016266），及び日本学術振興会科学研究費基盤研究 (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）による．

文　献

[1] S.B. Akers, “Binary decision diagrams,” IEEE Trans. Comput., vol.C-27, no.6, pp.509–516, 1978.

[2] C.C. Aggarwal and P.S. Yu, “Online generation of association rules,” Proc. IEEE International Conference on Data Engineering (ICDE’98), pp.402–411, 1998.

[3] R. Agrawal, H. Mannila, R. Srikant, 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.

[4] J.-F. Boulicaut, ed., Proc. 2nd International Workshop on Knowledge Discovery in Inductive Databases (KDID’03), Cavtat-Dubrovnik, 2003.

[5] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur, “Dynamic itemset counting and implication rules for market basket data,” Proc. SIGMOD’97, pp.255–264, 1997.

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

[7] D. Burdick, M. Calimlim, and J. Gehrke, “MAFIA: A maximal frequent itemset algorithm for transactional databases,” Proc. ICDE 2001, pp.443–452, 2001.

[8] 藤田昌宏，佐藤政生(編)，“特集「BDD（二分決定グラフ）」，” 情報処理，vol.34, no.5, pp.584–630, 1993.

[9] B. Goethals "Survey on frequent pattern mining," (Pdf) Manuscript, 2003, http://www.adrem.ua.ac.be/~goethals/publications/survey.ps

[10] B. Goethals and M. Javeed Zaki (eds.), "Frequent itemset mining dataset repository," Frequent Itemset Mining Implementations (FIMI'03), 2003. http://fimi.cs.helsinki.fi/data/

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

[12] L. Jiang, M. Inaba, and H. Imai, "An application of BDD to mining association rules," Database Engineering Workshop (DEWS'97), vol.8, pp.61–66, March 1997.

[13] C.Y. Lee, "Representation of switching circuits by binary-decision programs," Bell Syst. Tech. J., vol.38, pp.985–999, 1959.

[14] H. Mannila and H. Toivonen, "Multiple uses of frequent sets and condensed representations," Proc. KDD, pp.189–194, 1996.

[15] S. Minato, "BEM-II: An arithmetic Boolean expression manipulator using BDDs," IEICE Trans. Fundamentals, vol.E76-A, no.10, pp.1721–1729, Oct. 1993.

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

[17] S. Minato, "Zero-suppressed BDDs and their applications," Int. J. Software Tools for Technology Transfer (STTT), vol.3, no.2, pp.156–170, Springer, May 2001.

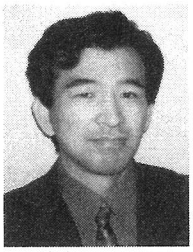
[18] J.-S. Park, M.-S. Chen, and P.S. Yu, "An effective hash based algorithm for mining association rules," Proc. SIGMOD'95, pp.175–186, 1995.

[19] A. Savasere, E. Omiecinski, and S.B. Navathe, "An efficient algorithm for mining association rules in large databases," Proc. VLDB'95, pp.432–444, 1995.

[20] H. Toivonen, "Sampling large databases for association rules," Proc. VLDB'96, pp.134–145, 1996.

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

(平成 17 年 5 月 12 日受付, 8 月 29 日再受付)



湊　真一　(正員)

1988 京大・工・情報卒，1990 同大大学院修士課程了．1995 同博士後期課程（社会人選抜）了．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究員．2004 より北大情報科学研究科助教授．著書 "Binary Decision Diagrams and Applications for VLSI CAD" (Kluwer, 1995)．1992 情報処理学会全国大会奨励賞，同年本会 75 周年記念論文優秀賞，2000 情報処理学会山下記念研究賞．IEEE，情報処理学会各会員．2000～2003 情報処理学会論文誌編集委員．

有村　博紀

1990 九大総合理工学研究科修士課程了．同年，九工大・情報工・助手，同助教授を経て，1996 九大システム情報科学研究科助教授，2004 北大情報科学研究科教授，現在に至る．1996 ヘルシンキ大学訪問研究員．1999～2002 科学技術振興事業団「さきがけ研究 21」研究員．2005 リヨン大学訪問研究員．現在，計算論的学習理論とデータマイニングの研究に従事．1998 情報処理学会全国大会優秀賞，1999 人工知能学会 2000 年度優秀論文賞，PAKDD 2000 Paper with Merit Award，本会 DEWS2002，2003，2004 優秀論文賞，2004 人工知能学会研究会優秀賞，2005 上林記念研究奨励賞等を受賞．ACM，情報処理学会，人工知能学会会員．博士（理学）．