データベースシステム 116－17
（1998. 7. 8）

概念階層を考慮した相関ルールマイニングの関係データベース管理システム上での実現

佐伯 敏章[1] 新谷 隆彦[2] 茂木 和彦[3] 田村 孝之[4] 喜連川 優[5]
東京大学生産技術研究所

概要

従来の相関ルール (association rule) 抽出は，専用マイニングエンジンを用いるのが普通であったが，コスト・移植性の面で問題があった．これに対し，SETM[4] というSQL(Structured Query Language) で記述可能なアルゴリズムが提案されたが，アイテムの階層関係を考慮しておらず，抽出されるルールの質が十分ではなかった．

本論文では，アイテムの階層関係を考慮した相関ルールの抽出を，関係データベース管理システム (RDBMS: Relational DataBase Management System) 上のSQLを用いて直接実現する方法について検討する．

キーワード：相関ルール，概念階層，SQL

Mining association rules with classification hierarchies on a relational database management system

Toshiaki Saeki[1] Takahiko Shintani[2] Kazuhiko Mogi[3] Takayuki Tamura[4] Masaru Kitsuregawa[5]
Institute of Industrial Science, University of Tokyo

Abstract

It has been the common practice to use a specialized mining engine to extract association rules, but the engine comes with the problem of its high development cost and its portability. In response to this problem, SETM[4], an algorithm programmable in SQL(Structured Query Language), is developed. However, this algorithm does not take into consideration the hierarchical relations, and the quality of the resulting rules are said to be unsatisfactory.

In this paper, we propose an algorithm that takes account of the hierarchical relations, and evaluate its implementation method using the SQL on RDBMS(Relational DataBase Management System).

Key words: association rule, classification hierarchy, SQL

[1] *saeki@flab.fujitsu.co.jp* 現在，株式会社富士通研究所に所属 Now belonging to Fujitsu Laboratories ltd.
[2] *shintani@tkl.iis.u-tokyo.ac.jp*
[3] *mogi@sdl.hitachi.co.jp* 現在，株式会社日立製作所に所属 Now belonging to Hitachi, Ltd.
[4] *ttamura@icc.melco.co.jp* 現在，三菱電機株式会社に所属 Now belonging to Mitsubishi Electric Corporation.
[5] *kitsure@tkl.iis.u-tokyo.ac.jp*

1 はじめに

コンピュータ技術の著しい発展に伴い，さまざまな分野で膨大な量のデータがデータベースに格納されるようになった．しかし，データを蓄積しただけでは，有用な知識を持ったことにはならない．だが，データの傾向分析は，サイズが大きくなればなるほど実時間内に処理することが難しくなる．そのため，膨大な量のデータを保持していても，それを処理できず知識として活用できないという状態がしばしば生じていた．

そこで，データマイニング (data mining) の研究が注目を集め始めた．これは，効率の良いアルゴリズムで膨大なデータを実時間内に処理し，データの中に埋もれていた知識を獲得する研究である．これを実現するために，アルゴリズムや専用マイニングエンジンなどの，種々の研究開発が行われている．しかし専用エンジンは，ソフトウェア開発や並列化による性能向上のためのコストが非常に大きく，また異機種への移植性も低い．

現在，大規模データベースの管理は関係データベース管理システム (RDBMS: Relational Database Management System) で行われるのが一般的である．また，普通 RDBMS にはデータベース操作言語として SQL(Structured Query Language) が実装されている．この点に着目し，データマイニング研究の一分野である相関ルールマイニングに関して，SETM[4] という，SQL によって表現可能なアルゴリズムが提案された．

本論文では，この SETM を発展させ，概念階層を考慮することにより抽出されるルールの質の向上を図った上で，SQL によって商用 RDBMS 上で直接実現することを試みる．

データベース管理システムから直接データマイニングが実現できれば，様々な利点が生まれる．まず，SQL は汎用言語であるから，高い移植性が得られる．次に，RDBMS の効率的な実装技術や，自動並列化機能を利用して，計算機資源を手軽に有効利用することができる．これらによって，データマイニングの利用が大きく広がることが期待される．

2 相関ルールの抽出

データマイニングの研究は，様々な分野に応用されている．例えばコンビニエンスストアでは，POS システムにより販売時刻・客層・売上品目などが常に記録されている．この記録から作られるデータベースにデータマイニングの技術を応用することにより，客層の分布・購入パターン・広告の効果などが分析できる．新しく購入パターンが発見されれば，パターンに合わせて商品を配置することにより，売上に大きく貢献できるだろう．

相関ルール (association rule) は，データマイニングで得られる代表的な情報の一つである．簡単な例としては，「パンとバターを購入した客は，高い確率で牛乳も購入する」とか，「子供連れの客は，野球選手のカードが付いた菓子を買うことが多い」という傾向である．

しかし，相関ルールの抽出に限らず，データマイニングは巨大なデータが処理対象であるため，実用的な時間での処理は難しく，効率の良いアルゴリズムの研究が進められている．

2.1 相関ルールの定義

相関ルールを抽出する時の最小の要素をアイテムと呼ぶ．上の例では，「パン」「バター」「牛乳」「子供連れの客」「(野球選手のカードが付いた) 菓子」である．

ここで，アイテムの集合 I とトランザクションデータベース $\mathcal{D}$ を次式の通り定義する．

$$I = \{i_1, i_2, \ldots, i_m\} \quad (1)$$

$$\mathcal{D} = \{T_1, T_2, \ldots, T_n\} \quad (T_i \subseteq I) \quad (2)$$

k 個のアイテムの組合せを，長さ k のアイテム集合 (itemset) と呼ぶ．トランザクションデータベースの各要素であるトランザクション T_i は，アイテム集合である．

相関ルールは，次式で表現される．

$$X \Rightarrow Y \quad (X, Y \subset I, X \cap Y = \phi) \quad (3)$$

相関ルールは支持度 (support)，確信度 (confidence) の 2 つのパラメータを持つ．この値により，処理時間とルールの有意性が決定される．

相関ルール $X \Rightarrow Y$ の支持度 $support(X \Rightarrow Y)$ とは，$\mathcal{D}$ 全体に対し X と Y を共に含むトランザクションの割合であり，次式で求められる．

$$support(X \Rightarrow Y) = \frac{n(\{T_i \mid T_i \supseteq X \cup Y\})}{n(\mathcal{D})} \quad (4)$$

ID	item
2000	1,2,3
1000	1,3
4000	1,4
5000	2,5,6

(a)transaction DB

⇒

large itemset	support
{1}	75%
{2}	50%
{3}	50%
{1,3}	50%

(b)large itemset

表 1: 相関ルール抽出の例 (最小支持度 50%)

また, 確信度 $confidence(X \Rightarrow Y)$ は, $\mathcal{D}$ の中で X を含むトランザクションのうち, X と Y を共に含むトランザクションの割合であり, 次式で求められる.

$$confidence(X \Rightarrow Y) = \frac{n(\{T_i \mid T_i \supseteq X \cup Y\})}{n(\{T_i \mid T_i \supseteq X\})} \quad (5)$$

相関ルールの抽出問題とは, ユーザによって指定された最小支持度 (minimum support) と最小確信度 (minimum confidence) を満足する全てのルールを見出すことである.

2.2 相関ルールの抽出処理

この問題は次の 2 段階の処理で求める.

1. ラージアイテム集合 (large itemset: 最小支持度を満足するアイテム集合) をすべて見つけ出す.
2. ラージアイテム集合から, 最小確信度を満足する相関ルールを導出する.

例えば, 表 1 に示されるトランザクションデータベースに対して, 最小支持度 50%, 最小確信度 50%を指定すると, 表 1(b) に示されるラージアイテム集合が求められることになり, それらを基にして次の相関ルール

- $1 \Rightarrow 3$ (支持度 50%, 確信度 66.6%)
- $3 \Rightarrow 1$ (支持度 50%, 確信度 100%)

が抽出される.

相関ルール抽出処理のうち, 第 2 段階は絞り込まれたラージアイテム集合に対して行う処理であり, 処理時間は短い. しかし, 第 1 段階はトランザクションデータベース全体を検索する処理であるため, 非常に長い処理時間を必要とする. そのため, 第 1 段階の処理の効率化が重要である.

```
INSERT INTO R'_k
  SELECT p.trans_id, p.item_1, ..., p.item_{k-1}, q.item
    FROM R_{k-1} p, R_{k-1} q
    WHERE q.trans_id = p.trans_id AND
          p.item_1   = q.item_1   AND
                 :
          p.item_{k-2} = q.item_{k-2} AND
          q.item > p.item_{k-1}

INSERT INTO C_k
  SELECT p.item_1, ..., p.item_k, COUNT(*)
    FROM R'_k p
    GROUP BY p.item_1, ..., p.item_k
    HAVING COUNT(*) ≥: min_support

INSERT INTO R_k
  SELECT p.trans_id, p.item_1, ... p.item_k
    FROM R'_k p, C_k q
    WHERE p.item_1 = q.item_1 AND
                 :
          p.item_k = q.item_k
    ORDER BY p.trans_id, p.item_1, ..., p.item_k
```

図 1: SETM の SQL 表現

2.3 相関ルール抽出アルゴリズム

1994 年, R.Agrawal らによって「アプリオリ (Apriori)」という, 相関ルール抽出の基本的な逐次アルゴリズムが提案された [1]. これを元に様々な相関ルール抽出アルゴリズムが研究された.

また 1995 年, Park, Chen, Yu は DHP (Direct Hashing and Pruning) という手法を提案した [2]. これは, ハッシュを利用することにより候補アイテムの数 (特にパス 2) を減少させ, さらにトランザクションデータベースを縮小させる手法である.

これに対し, 1995 年, Maurice Houtsma, Arun Swami は, 「Set-oriented Mining for Association Rules in Relational Databases」という論文 [4] の中で, SETM という RDBMS 上で表現可能なアルゴリズムを提案した. その SQL 表現を図 1 に示す.

RDBMS 上では, 保存されるデータはリレーションを持った表の形を取らなければならない. SETM では作成される中間データ R_i を $(trans\_id, item_1, \ldots, item_i)$ のリレーションの形でソートして保存する. これにより, 簡潔な SQL 表現を実現した.

我々は, この SETM を元に, 高速表結合アルゴリズムであるハッシュジョインの利用を前提に最適化

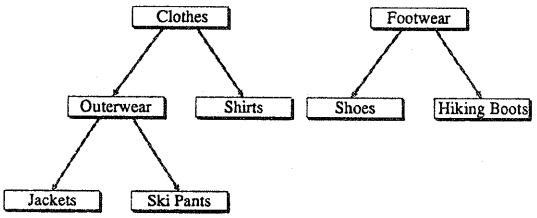

図 2: 階層関係の例

することにより, 商用 RDBMS 上においても実用的な時間での処理が可能であることを示した [3].

3 概念階層を考慮した相関ルール

この節では, 更に一般化した相関ルールを導入する. 巨大なデータベースが与えられた時に, アイテムを階層的に分類し, 階層の様々なレベル間での相関ルールを発見することである [5].

3.1 概念階層

ほとんどの場合, アイテムに対して, 階層的な分類 (taxonomy) が可能である. 例えば, 図 2 のような分類を考えることが出来る. この階層関係は, 次のようなことを示している.

- 「Jackets」は「Outerwear」である
- 「Ski Pants」は「Outerwear」である
- 「Outerwear」は「Clothes」である

例えば, Jackets と Hiking Boots, また Ski Pants と Hiking Boots を同時に買う人が多ければ, 「Outerwear を買う人は Hiking Boots を買う傾向がある」というルールが推測されるだろう. しかしながら, 「Outerwear ⇒ Hiking Boots」というルールの支持度は, 「Jackets ⇒ Hiking Boots」と「Ski Pants ⇒ Hiking Boots」の支持度の合計にはならない. 同様に, 「Outerwear ⇒ Hiking Boots」が正当なルールだったとしても, 「Jackets ⇒ Hiking Boots」と「Clothes ⇒ Hiking Boots」というルールは成立しないことがあるが, 何故ならば前者は最小支持度を満たさない可能性が, 後者は最小確信度を満たさない可能性があるからである.

TID	購買アイテム
100	Shirt
200	Jacket, Hiking Boots
300	Ski Pants, Hiking Boots
400	Shoes
500	Shoes
600	Jacket

表 2: トランザクションデータベースの例

アイテム集合	支持度
{ Jacket }	33%
{ Outerwear }	50%
{ Clothes }	67%
{ Shoes }	33%
{ Hiking Boots }	33%
{ Footwear }	67%
{ Outerwear, Hiking Boots }	33%
{ Clothes, Hiking Boots }	33%
{ Outerwear, Footwear }	33%
{ Clothes, Footwear }	33%

表 3: 最小支持度 30%の場合のラージアイテム集合

ルール	支持度	確信度
Outerwear ⇒ Hiking Boots	33%	67%
Outerwear ⇒ Footwear	33%	67%
Hiking Boots ⇒ Outerwear	33%	100%
Hiking Boots ⇒ Clothes	33%	100%

表 4: 最小確信度 60%の場合の相関ルールの例

例えば, 図 2 で示される分類階層に対してトランザクションデータベース 表 2, 最小支持度 30%, 最小確信度 60%が与えられたとする. この時, ラージアイテム集合が表 3, 相関ルールが表 4 と求められる. ここで, ルール 「Ski Pants ⇒ Hiking Boots」と「Jackets ⇒ Hiking Boots」は最小支持度を満足していないが, 「Outerwear ⇒ Hiking Boots」は最小支持度を満足していることが分かる. このようにデータの分類階層を考慮することにより, 従来は抽出できなかったルールを抽出することが可能となる.

3.2 定義の拡張

データの分類階層構造 $\mathcal{H}$ を, 循環していないグラフとする. 図 2 に示されるように $\mathcal{H}$ はアイテムによる木構造を持つ. 本論文では, 分類において一つのアイテムを複数の木によって分類しないで, DAG

を用いて一つにまとめた木を利用することにする.

p から c への枝が $\mathcal{H}$ に存在した場合, p を c の親 (parent) と呼び, c を p の子 (child) と呼ぶ. また, $\mathcal{H}$ に $\hat{x}$ から x への経路が存在した場合, $\hat{x}$ を x の先祖 (ancestor) と表現し, 同時に x を $\hat{x}$ の子孫 (desendant) と表現する. ここで, $\mathcal{H}$ は循環していないので, $\hat{x} = x$ となることはない.

$x \in I$ であるアイテム x について, もし x がトランザクション T に含まれているか, T に含まれているアイテムの先祖である場合, T が x を支持 (support) すると表現する. $X \subseteq I$ である X に対して, X に含まれる全てのアイテムが T に支持されている場合, T は X を支持すると表現する.

分類階層を考慮した相関ルールも式 (3) 同様に,

$$X \Rightarrow Y \quad (X, Y \subset I, X \cap Y = \phi) \qquad (6)$$

と表現される. ただし, Y は X の上位アイテムを含まないものとする.

分類階層を考慮しない場合と同様に, 相関ルールは支持度, 確信度の 2 つのパラメータを値を持つ. トランザクション集合 $\mathcal{D}$ のうち, X と Y が支持するトランザクションが c % であった場合, 相関ルール $X \Rightarrow Y$ は $\mathcal{D}$ において c の確信度を持つと定義する. また, トランザクション集合 $\mathcal{D}$ のうち, $X \cup Y$ が支持するトランザクションが s% であった場合, 相関ルール $X \Rightarrow Y$ は $\mathcal{D}$ において s の支持度を持つと定義する.

4 概念階層を考慮した相関ルール抽出の RDBMS 上での表現

概念階層の導入は抽出される相関ルールの質の向上に不可欠であるが, アイテムの階層情報とトランザクションデータを結合し, 数倍に膨れ上がったデータを処理する必要があるため, 処理負荷は更に悪化する. よって, 不要なデータを効率良く削除することが重要である.

これまでに, 概念階層を考慮した相関ルール抽出アルゴリズムがいくつか提案されている [5][8] が, 本節では概念階層を考慮した相関ルール抽出を RDBMS 上で直接実現する手法の検討を行う.

4.1 概念階層データとアルゴリズムの SQL 表現

概念階層を導入するために, アイテムの階層関係を記述した表を参照する. 図 5 の階層木の例から生成される階層関係表の例を表 6 に示す.

概念階層を考慮して相関ルール抽出アルゴリズムを SQL で記述したのが図 4.1.3 である. パス 1 で階層の情報全てを持たせる代わりに, 階層関係をチェックして冗長なトランザクションを削減している. また, 可能ならば高速な表結合アルゴリズムであるハッシュジョインを使うべきである.

4.1.1 初期設定

まず, 表 $TAXONOMY(anc, des)$ に, 各アイテム $x \in I$ と, それに対する全ての先祖 $\hat{x}$ の組を記述しておき, これによって分類階層を表現しておく.

$$(taxonomy.anc, taxonomy.des) = (x, \hat{x}), \quad \forall x \in I \qquad (7)$$

この表は, 読み出しのみで更新は行わないため, 表のインデックスを作成しておくと速度が向上する.

4.1.2 パス 1 での処理

初めに, 表 $TAXONOMY$ と トランザクションデータベース $SALES$ を結合し, トランザクションの各アイテムの先祖の情報を全て持たせた R'_1 を作成する. ここで検索すべきトランザクションデータベースは数倍にふくれあがる. 次に, R'_1 を検索してアイテムの数を数え, C_1 を作成する. 最後に, R'_1 と C_1 を結合する事によって, 不要なトランザクションを除いたトランザクションデータベース R_1 を新たに作成する (図 4.1.3 パス 1).

4.1.3 パス $k(k > 1)$ での処理

まず初めに, パス $k-1$ で作成された R_{k-1} を自己結合して $R'_k(id, item_1, item_2, \ldots, item_k)$ を作成する. この時, アイテム $item_1, item_2, \ldots, item_k$ が R_{k-1} に辞書順に格納されることを保証することにより, 処理を簡潔なものにしている. また, 新しく作成される行も, 各アイテムは辞書順にソートして格納される.

同時に, 生成されるトランザクションをチェックし, 階層関係表 *TAXONOMY* を参照して先祖と子孫のアイテムを両方とも持つものを除去する. これを実現しているのが図 4.1.3 パス k の 1 つめの SQL 文に含まれる **NOT IN** の文である. ここで, 必要な比較文の数について検討する. 下の 2 式

$$\mathbf{p} = (p.id, p.item_1, p.item_2, \ldots, p.item_k) \quad (8)$$

$$\mathbf{q} = (q.id, q.item_1, q.item_2, \ldots, q.item_k) \quad (9)$$

を結合し,

$$\mathbf{r} = (p.id, p.item_1, p.item_2, \ldots, p.item_k, q.item_k) \quad (10)$$

を作成する場合を考える. 本来は, 生成されるトランザクションに含まれるアイテムの全ての組に関して階層関係をチェックしなければならない. しかし, まずパス $k-1$ で生成されたトランザクションが冗長な組を含まないことに着目すると, $p.item_1$ ... $p.item_k$の間の各組についてはチェックの必要がないことが分かる. 次に,

$$p.item_i = q.item_i \quad (1 \leq i \leq k-1) \quad (11)$$

という結合条件に着目すると, $(p.item_1, q.item_k)$, $(p.item_1, q.item_k)$, ..., $(p.item_{k-1}, q.item_k)$ についてもチェックの必要がないことが分かる. よって, $(p.item_k, q.item_k)$ の 1 組だけをチェックすれば良い.

さらに, リレーションに格納する時にアイテムをソートする順番に関して, 先祖のアイテムが必ず子孫のアイテムの前に来ること, つまり $\hat{x} < x$ であることを保証すれば, 比較文は 1 つですむ.

この後, パス 1 と同様に R'_k を検索してアイテムの数を数え, C_k を作成する. 最後に, R'_k と C_k を結合する事によって, 不要なトランザクションを除いたトランザクションデータベース R_kを新たに作成する (図 4.1.3 パス k).

処理の例を図 5, 表 6, 7, 8, 9 に示す.

4.2 実装と評価

以上のアルゴリズムを, 商用 RDBMS に実装して評価を行った. 結果を図 4, 5 に示す. 使用した環境は Oracle version 7.3.2 を組み込んだ SparcStation 20 (SuperSPARC 50MHz, 192MB) で, トランザクション数 3000 件, 平均アイテム数 10 件, 概念階層木の数 50 本のサンプルデータベースを使用した.

```
/* PATH 1 */

INSERT INTO R'1
  SELECT s.id, t.anc
    FROM   SALES s;

INSERT INTO R'1
  SELECT DISTINCT s.id, t.anc
    FROM   SALES s, TAXONOMY t
    WHERE s.item     = t.des;

INSERT INTO C1
  SELECT p.item1, COUNT(*)
    FROM R'1 p
    GROUP BY p.item1
    HAVING COUNT(*) ≥ minimum_support;

INSERT INTO R1
  SELECT p.id, p.item1
    FROM R'1 p, C1 q
    WHERE p.item1 = q.item1;


/* PATH k */

INSERT INTO R'k
  SELECT p.id, p.item1, ..., p.itemk-1, q.itemk-1
    FROM Rk-1 p, Rk-1 q
    WHERE p.id       = q.id       AND
          p.item1    = q.item1    AND
                     ⋮
          p.itemk-2  = q.itemk-2 AND
          p.itemk-1  < q.itemk-1 AND
    (p.itemk-1, q.itemk-1) NOT IN
    (SELECT * FROM TAXONOMY t
      WHERE t.anc    = p.itemk-1 AND
            t.des    = q.itemk-1
    );

INSERT INTO Ck
  SELECT p.item1, ..., p.itemk, COUNT(*)
    FROM R'k p
    GROUP BY p.item1, p.item2, ..., p.itemk
    HAVING COUNT(*) ≥ minimum_support;

INSERT INTO Rk
  SELECT p.id, p.item1, ..., p.itemk
    FROM R'k p, Ck q
    WHERE p.item1 = q.item1 AND
                     ⋮
          p.itemk = q.itemk ;
```

図 3: 階層関係を考慮したアルゴリズム

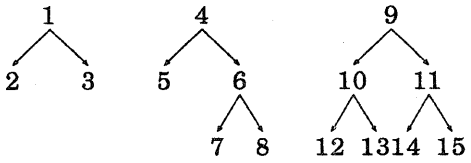

表 5: 階層木の例

transaction database

	Tid	*item*
n	T_1	2, 7, 12
	T_2	3, 8, 14
	T_3	5, 13

relation *Taxonomy*

anc.	*des.*
1	2
1	3
4	5
4	6
4	7
4	8
6	7
6	8
9	10
9	11
9	12
9	13
9	14
9	15
10	12
10	13
11	14
11	15

relation SALES

id	*item*
T_1	2
T_1	7
T_1	12
T_2	3
T_2	8
T_2	14
T_3	5
T_3	13

表 6: 相関ルールの抽出の例 (初期状態)

R'_1

Tid	i_1
T_1	1
T_1	2
T_1	4
T_1	6
T_1	7
T_1	9
T_1	10
T_1	12
T_2	1
T_2	3
T_2	4
T_2	6
T_2	8
T_2	9
T_2	11
T_2	14
T_3	4
T_3	5
T_3	9
T_3	10
T_3	13

C_1

i_1	*cnt*
1	2
2	1
3	1
4	3
5	1
6	2
7	1
8	1
9	3
10	2
11	1
12	1
13	1
14	1

R_1

Tid	i_1
T_1	1
T_1	4
T_1	6
T_1	9
T_1	10
T_2	1
T_2	4
T_2	6
T_2	9
T_3	4
T_3	9
T_3	10

表 7: 相関ルールの抽出の例 (パス 1)

R'_2

Tid	i_1	i_2
T_1	1	4
T_1	1	6
T_1	1	9
T_1	1	10
T_1	4	6
T_1	4	9
T_1	4	10
T_1	6	9
T_1	6	10
T_1	9	10
T_2	1	4
T_2	1	6
T_2	1	9
T_2	4	6
T_2	4	9
T_2	6	9
T_3	4	9
T_3	4	10
T_3	9	10

C_2

i_1	i_2	cnt
1	4	2
1	6	2
1	9	2
1	10	1
4	9	3
4	10	2
6	9	2
6	10	1

R_2

Tid	i_1	i_2
T_1	1	4
T_1	1	6
T_1	1	9
T_1	4	9
T_1	4	10
T_1	6	9
T_1	6	10
T_2	1	4
T_2	1	6
T_2	1	9
T_2	4	9
T_2	6	9
T_3	4	9
T_3	4	10

表 8: 相関ルールの抽出の例 (パス 2)

R'_3

Tid	i_1	i_2	i_3
T_1	1	4	6
T_1	1	4	9
T_1	1	6	9
T_1	4	9	10
T_1	6	9	10
T_2	1	4	6
T_2	1	4	9
T_2	1	6	9
T_3	4	9	10

C_3

i_1	i_2	i_3	cnt
1	4	6	2
1	6	10	2

R_3

Tid	i_1	i_2	i_3
T_1	1	4	6
T_1	1	6	9
T_2	1	4	9
T_2	1	6	9

$R'_4 = \phi$

表 9: 相関ルールの抽出の例 (パス 3)

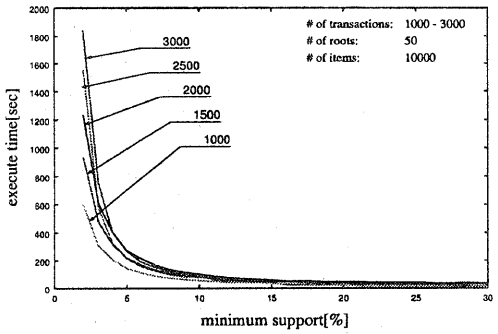

図 4: 最小支持度に対する実行時間

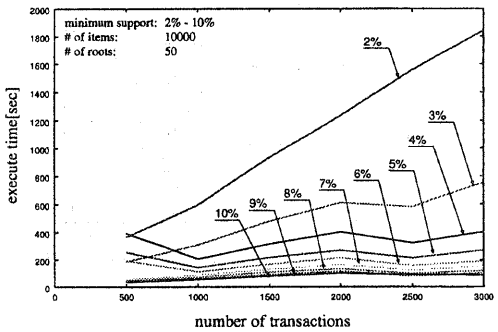

図 5: トランザクション数に対する実行時間

実行時間は, 概念階層を考慮しない場合とほぼ同様の変化を示している. 最小支持度を小さくすると, 急激に実行時間が悪化する. また, 記憶領域の使用量も増加する. トランザクション数に対して実行時間はほぼ比例する. なお, トランザクション数・最小支持度が共に小さい時に実行時間が逆に悪化しているが, これは最小支持度を満たすトランザクションの数が絶対的に少ないためで, 一般の大規模データベースに適用する場合には問題にならないと思われる.

5 結論

商用関係データベース管理システムに対して, 概念階層を考慮した相関ルール抽出アルゴリズムを実装した. 概念階層を考慮しないアルゴリズムに比べて, 処理対象が巨大になるため速度の低下は免れないが, 階層関係を参照することにより冗長なデータを削除し, 速度低下を防ぐ手法を示した.

今後はRDBMSの並列処理オプション, パーティション機能による表の分割管理による性能向上に関して研究を進めて行きたい.

参考文献

[1] Rakesh Agrawal, Ramakrishnan Srikant, "Fast Algorithms for Mining Association Rules", Proceedings of the 20th International Conference on VLDB, pp. 487-499 (1994)

[2] Jong Soo Park, Ming-Syan Chen, Philip S.Yu: "An Effective hash-Based Algorithm for Mining Association Rules", Proceedings of ACM SIGMOD, pp.175-186 (1995)

[3] 佐伯 敏章, 新谷 隆彦, 茂木 和彦, 喜連川 優, "関係データベースシステムを用いた相関関係抽出に関する一考察", 1998年電子情報通信学会総合大会講演論文集 (1998)

[4] Maurice Houtsma, Arun Swami, "Set-Oriented Mining for Association Rules in Relational Databases", Proceedings of the 11th International Conference on Data Engineering, pp.25-33 (1995)

[5] Srikant, R., Agrawal, R.: "Mining Generalized Association Rules", Proc. of VLDB, pp.407–419 (1995).

[6] Srikant, R., Agrawal, R.: "Mining Quantitative Association Rules in Large Relational Tables", Proc. of ACM SIGMOD, pp.1–12 (1996)

[7] Agrawal, R., Imielinski, T., Swami, A.: "Mining Association Rules between Sets of Items in Large Databases", Proc. of ACM SIGMOD, pp.207–216 (1993)

[8] Han, J., Fu, Y.: "Discovery of Multiple-Level Association Rules from Large Databases", Proc. of VLDB, pp.420–431 (1995)