飽和集合上の極小生成子の支持度計算を行わない高速抽出 ー負の相関ルール抽出の効率化にむけてー

A Fast Extraction Method of Minimal Generators from Closed Itemsets Without Support Calculation - Toward Improving Negative Association Rule Mining-

谷島 健斗 [1*]　岩沼 宏治 [2]　山本 泰生 [2,3]
Kento Yajima[1]　Koji Iwanuma[2]　Yoshitaka Yamamoto[2,3]

[1] 山梨大学大学院医工農学総合教育部工学専攻コンピュータ理工学コース
[1] Computer Science and Engineering Course, Integrated Graduate School of Medicine, Engineering and Agricultural Sciences, University of Yamanashi
[2] 山梨大学大学院総合研究部
[2] Interdisciplinary Graduate School, University of Yamanashi
[3] 科学技術振興機構 さきがけ
[3] JST PRESTO

Abstract: A fast extraction of the minimal generator plays a very important role for compressing the huge set of valid negative association rules. We propose a novel algorithm that can efficiently extract all minimal generators from closed itemsets without support calculations. We also show some experimental results for evaluating our proposed methods.

1 はじめに

負の相関ルールマイニングの効率的な実装のためには, 極小生成子を高速に生成することが必要である. 本論文では, 支持度計算を行わずに極小生成子を飽和アイテム集合から高速に抽出する手法について考察する.

相関ルールとは, トランザクションデータベース中に頻繁に共起するアイテム集合の関係を記述したものである. X と Y をアイテム集合とするとき, $X \Rightarrow Y$ のように表し, 正の相関ルール [1] と呼ぶ. これに対して, 本論文では, X と Y がほとんど同時に出現しない現象を表現する $X \Rightarrow \neg Y$ や $\neg X \Rightarrow Y$ なる形の負の相関ルールを考察する. 負の相関ルールは正のルールでは表現が困難な共起関係を記述でき, 有用である [2, 3, 5]. ただし, 負の相関ルールを抽出するためには, 非頻出なアイテム集合を扱う必要がある. そのため, 正の相関ルールの場合に比べて探索空間が格段に大きく, また抽出されるルールの数も非常に多くなる. そのため岩沼ら [6] は, 極小生成子を用いた負ルール集合の可逆圧縮手法を提案した. さらに我々[9] は, 頻出アイテム集合ではなく, 極小生成子を用いて負の相関ルールを圧縮したままの形で抽出を行う効率的な計算手法を提案した. また [10] では, 極小生成子を飽和アイテム集合から効率的に抽出する下降型と上昇型の探索アルゴリズムを考察した. それらの計算上のボトルネックは, アイテム集合の支持度計算にあることが判明している. そこで本研究では, 支持度計算を行わずに極小生成子を高速抽出する新しい手法を提案する.

*連絡先： 山梨大学大学院工学専攻 (修士課程)
コンピュータ理工学コース
山梨県甲府市武田 4-3-11
E-mail:g17tk024@yamanashi.ac.jp

2 準備

2.1 正負の相関ルール

$I = \{x_1, x_2, ..., x_n\}$ を**アイテムの全体集合**とする時，**トランザクション** t をアイテム集合 $t \subseteq I$ と定める．**トランザクションデータベース** D をトランザクションの多重集合とする．X をアイテム集合とするとき，$X \subseteq t$ となる D 中のトランザクション t を X の**出現**と呼び，その多重集合を $D(X)$ と略記する．多重集合 α の**大きさ**を $|\alpha|$ と表記するとき，$|D(X)|$ を X の D 中の**支持度**と呼ぶ．X の D 中の**相対支持度** $\mathbf{sup}(X)$ を $\sup(X) = \frac{|D(X)|}{|D|}$ と定義する．**最小支持度** ms と**最小確信度** mc とはユーザが相対支持度と確信度に関して与え

る閾値 (0 から 1 の間の実数値) である．$\sup(X) \geq ms$ を満たす X を**頻出アイテム集合**と呼ぶ．**正の相関ルール** (以下，“正ルール”と略記する) を，$X \cap Y = \emptyset$ であるアイテム集合 X，Y からなる表現 $X \Rightarrow Y$ と定める．X と Y をそれぞれルールの**前件**，**後件**と呼ぶ．

本論文では，**負の相関ルール** (以下では “負ルール” と略記) を考察する．X と Y を $X \cap Y = \emptyset$ なるアイテム集合とするとき，負ルールとは以下のいずれかの表現である．

- $X \Rightarrow \neg Y$　　(右否定形もしくは後件負形)
- $\neg X \Rightarrow Y$　　(左否定形もしくは前件負形)
- $\neg X \Rightarrow \neg Y$　　(両否定形)

両否定形 $\neg X \Rightarrow \neg Y$ は，一般に非常に数が多く，効率的な抽出は困難であり，本論文では両否定形を扱わない．上記の $\neg X$ はアイテム集合の否定表現であり，**負アイテム集合**と呼ぶ．$\neg X$ は，X の各アイテムが同時に出現することはほとんどないことを示すものと考える．以下では C_X はアイテム集合 X または負アイテム集合 $\neg X$ のどちらかを表すものとする．負アイテム集合および負ルールの**相対支持度** sup と**確信度** conf を以下のように定める [2].

$$\sup(\neg X) = 1 - \sup(X) \tag{1}$$

$$\sup(X \Rightarrow \neg Y) = \sup(X) - \sup(X \cup Y) \tag{2}$$

$$\sup(\neg X \Rightarrow Y) = \sup(Y) - \sup(X \cup Y) \tag{3}$$

$$\mathrm{conf}(C_X \Rightarrow C_Y) = \frac{\sup(C_X \Rightarrow C_Y)}{\sup(C_X)} \tag{4}$$

このとき，以下の 4 つの条件を満たすルール $C_X \Rightarrow C_Y$ を**妥当 (valid) な負ルール**と呼ぶ [2, 3].

1. $X \cap Y = \emptyset$　　(独立性)
2. $\sup(X) \geq ms$ かつ $\sup(Y) \geq ms$
   (前件と後件の頻出性)
3. $\sup(C_X \Rightarrow C_Y) \geq ms$　　(ルールの頻出性)
4. $\mathrm{conf}(C_X \Rightarrow C_Y) \geq mc$　　(ルールの確信度)

2.2　極小生成子を用いた負ルール集合の圧縮と抽出

アイテム集合 X に対して，$X \subset X'$, $X \neq X'$ かつ $\sup(X)$=$\sup(X')$ を満たす X' が存在しないならば，X を**飽和アイテム集合** (*closed itemset*)[4] と呼ぶ．飽和アイテム集合の最も重要な性質の一つとして，以下がある．

定理 1 (飽和集合の積演算の閉包性 [4]) X と Y が飽和アイテム集合ならば，その積 $X \cap Y$ も飽和アイテム集合．

また X が X' に対して，$X \subseteq X'$ かつ $\sup(X') = \sup(X)$ を満たすならば，X を X' の**生成子** (*generator*) と呼ぶ．生成子は一般には複数存在するので，その中でより小さな生成子が存在しないものを**極小生成子** (*minimal generator*) [4] と呼ぶ．

頻出アイテム集合や正のルールの圧縮には飽和アイテム集合がよく用いられているが，負ルールの圧縮に用いた場合，本来抽出すべき負ルールが表現できなくなる現象が生じる [6]．そこで，飽和アイテム集合ではなく極小生成子を用いることで妥当な負ルールの集合の可逆圧縮ができる [6] ことが分かっている．

極小生成子の集合から接尾 (もしくは接頭) 辞木が必ず作れることを保証できる [9]．そこで，先行研究 [9] のアルゴリズムでは，極小生成子から**接尾辞木** [5] を作成し，左優先深さ優先探索を行いながら，妥当な負ルールの全てを抽出することが可能である．先行研究 [9] の負ルール抽出アルゴリズムの大枠を **Algorithm 1** に示す．擬似コード中の Check_Rule(X, Y) で $X \Rightarrow \neg Y$ と $\neg Y \Rightarrow X$ が妥当なルールであるかチェックしている．また，同時に接尾辞木の性質などを用いて適宜枝刈りを行い，探索の効率化を行っている．詳細は，文献 [9] を参照いただきたい．

Algorithm 1 極小生成子上の負の相関ルール抽出法

トランザクションデータベースから極小生成子（*MGS*）の集合を抽出し，N を抽出した極小生成子の総数とする;
1: *MGS* の集合から接尾辞木を生成;
2: 接尾辞木上で左優先深さ優先探索で極小生成子を MGS_1, ... ,MGS_N の順に並べる;
3: **for** $i = 1$ to N **do**
4:　　$X \leftarrow MGS_i$
5:　　**for** $j = 1$ to N **do**
6:　　　　$Y \leftarrow MGS_j$
7:　　　　Check_Rule(X, Y)
8:　　**end for**
9: **end for**

2.3　支持度計算を用いる飽和集合からの極小生成子の抽出

以上から，負ルールの効率的な抽出には，極小生成子の高速な生成が非常に重要となる．先行研究 [10] では，上昇型と下降型の 2 つの極小生成子抽出アルゴリズムを提案している．極小生成子は，あとの復元を考えて，飽和アイテム集合から生成を行う．その計算は，図 2 に示すような，飽和アイテム集合の部分集合からなる束空間の探索によって行う．

命題 1 [8]　ある飽和アイテム集合 X に対して，Y がその極小生成子である場合，Y を真に含む X の部分集合

X' は X の極小生成子になり得ない.

命題 2 [8] 飽和アイテム集合 X の極小生成子 Y の真の部分集合 Y' は X の極小生成子となり得ない.

上昇型抽出は, 命題 1 に基づいた探索空間の枝刈りを行う. 極小生成子の大きさが小さい場合には, 上昇型手法が極小生成子を高速に抽出することができる. 下降型は, 飽和アイテム集合の要素を 1 つずつ順に削除しながら部分集合を探索するため, 命題 2 に基づいた探索空間の枝刈りを行う. 飽和アイテム集合と極小生成子の大きさがあまり変わらないとき, 下降型手法は極小生成子を高速に抽出することができる. 図 2 に表 1 の飽和アイテム集合 $ABCD$ に対する束構造, 図 3 に上昇型による探索空間, 図 4 に下降型による探索空間を示す. 図 2, 3 と 4 における網掛け背景は飽和アイテム集合を示し, 破線は極小生成子を示している. 擬似コード等の詳細は, 文献 [10] を参照いただきたい. この上昇型, 下降型抽出法では, 各部分集合が (一番上の) 飽和アイテム集合の生成子であるか否かの判定を支持度を毎回計算して行っている. 特に下降型においては, 検査する部分集合が大きい場合が多いので, この支持度計算は大変重く, 計算上のボトルネックになっている. そこで, 本論文では, 支持度計算を全く行わずに生成子か否か判定する手法を提案する.

表 1: データベース

TID	アイテム集合
1	ABCD
2	ABCD
3	AB
4	A
5	B
6	D

飽和集合	-	極小生成子
A	-	A
B	-	B
D	-	D
AB	-	AB
ABCD	-	C, AD, BD

図 1: 飽和集合と対応する極小生成子

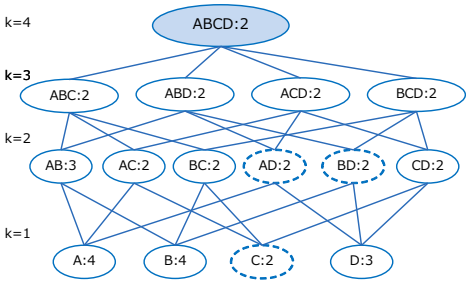

図 2: 束構造

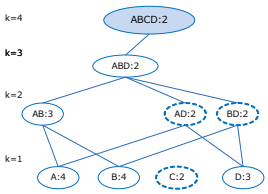

図 3: 上昇型の探索空間

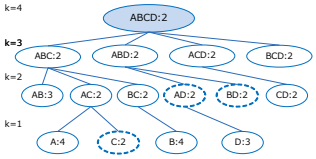

図 4: 下降型の探索空間

3 支持度計算なし極小生成子抽出アルゴリズム

本章では, 飽和アイテム集合から支持度計算を行わずに極小生成子を効率的に抽出する手法について考察する. 以下の定理 2 が探索の基本となる.

定理 2 (生成子判定原理) 飽和アイテム集合 X とその任意の部分集合 Y に関して以下が同値である.

(1) $\sup(X) = \sup(Y)$, 即ち Y は X の生成子である

(2) $Y \subseteq Y' \subsetneqq X$ なる飽和アイテム集合 Y' が存在しない

証明: 上記の (1) から (2) は, 飽和集合の定義と支持度の逆単調性から明らかなので, 証明は省略する. ここでは (2) から (1) の証明を示す. まず (2), 即ち $Y \subseteq Y' \subsetneqq X$ なる飽和集合 Y' は存在しないと仮定する. アイテム集合 Y の飽和集合は必ず存在して一意に定まるので, それを Z とする. もし $Z \neq X$ であれば, 仮定より $Z \not\subset X$ となる. この場合, 集合積 $Z \cap X$ を Z_X と表せば, 定理 1 より Z_X は必ず飽和集合となる. また, 明らかに $Y \subseteq Z_X$ かつ $Z_X \subsetneqq X$ となるので, 仮定に矛盾する. よって $Z = X$ である. この場合, 明らかに $\sup(X) = \sup(Z) = \sup(Y)$ となるので, (1) が成り立つ. 以上から, (2) から (1) が証明できた. ■

飽和アイテム集合をその大きさで昇順にソートし, 順に極小生成子を探索することで, 要素数 k の飽和アイテム集合 X を探索する際に, X の真部分集合となる飽和アイテム集合 Y の極小生成子の探索が必ず終了していることが保証される. この場合, 定理 2 を用いて, 支持度計算を行わずに, X の部分集合 Z が生成子であるか否か, 即ち, X と Z の支持度が同じであるか否かの判定ができることになる.

表 2 に表 1 のデータベースから抽出される飽和アイテム集合と支持度を示し, 図 5 から図 9 に支持度計算なし手法における各飽和アイテム集合の探索空間を示す. また, 図 10 に飽和アイテム集合 $ABCD$ の探索前のハッシュ表, 図 11 に $ABCD$ の探索後のハッシュ表

表 2: 飽和集合のデータベース

飽和 ID	飽和集合	頻度
1	A	4
2	B	4
3	D	3
4	AB	3
5	ABCD	2

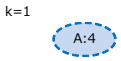

図 5: A の探索空間

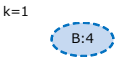

図 6: B の探索空間

を示す．支持度計算なし極小生成子抽出のアルゴリズムは **Algorithm 2** に擬似コードを示す．

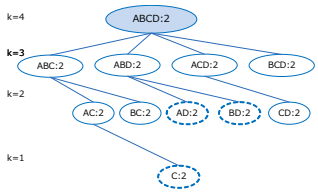

図 9: 飽和集合 $ABCD$ の探索空間

	アイテム集合	対応飽和ID
1	A	1
2		
3		
4		
5		
6	B	2
7		
8		
9		
10		
11		
12		
13		
14		
15		

	アイテム集合	対応飽和ID
16		
17	AB	4
18		
19		
20		
21		
22		
23	D	3
24		
25		
26		
27		
28		
29		
30		

図 10: 探索前のハッシュ表

	アイテム集合	対応飽和ID
1	A	1
2		
3	AC	5
4	ABD	5
5		
6	B	2
7		
8	CD	5
9		
10	ACD	5
11		
12	BD	5
13		
14		
15	BCD	5

	アイテム集合	対応飽和ID
16		
17	AB	4
18		
19	C	5
20	ABC	5
21		
22		
23	D	3
24		
25	BC	5
26		
27	ABCD	5
28		
29	AD	5
30		

図 11: 探索後のハッシュ表

Algorithm 2 では，まず事前に LCM[7] 等を用いて抽出した飽和アイテム集合を大きさで昇順にソートする．表 2 のようにソートする．飽和 ID:1 から 3 は大きさが 1 なので，自身が極小生成子となる (図 5 から 7)．飽和 ID が 4 の AB は小さい集合が飽和集合なので，AB が極小生成子になる (図 8)．飽和 ID:4 まで探索終了時のハッシュ表は図 10 のようになる．この時，登録されているアイテム集合は飽和 ID:5 の生成子にはならない．$ABCD$ の部分集合でハッシュ表に登録されていない集合は図 9 のようになり，C, AD, BD が極小生成子となる．探索終了後のハッシュ表は，図 11 のようになる．

図 7: D の探索空間

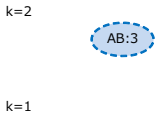

図 8: AB の探索空間

Algorithm 2 支持度計算なし抽出アルゴリズム

Require: 3 項組 $\langle k, X_k, f_k \rangle$ の族 $\mathcal{CS} = \{\langle 1, X_1, f_1 \rangle, \ldots, \langle n, X_n, f_n \rangle\}$．但し，各 k は識別番号，X_k は飽和アイテム集合，f_k は X_k の出現頻度である．
Ensure: $\mathcal{CS}$ の各飽和集合（識別番号 k）に対する極小生成子 M_j との頻度の 3 項組 $\langle M_j, k, f_k \rangle$ の集合 $\mathcal{MG}$

```
1: MG ← ∅                                   ▷ MG は，極小生成子の族を格納
2: S ← ∅                                    ▷ S は，スタック構造で生成子の候補を格納
3: Y ← ∅                                    ▷ Y は，S から取り出すアイテム集合を格納
4: HashTable ← ∅                            ▷ 訪問した頂点を保持するハッシュ表
5: sortCS()
     ▷ CS の各要素 ⟨i, X_i, f_i⟩ を X_i の大きさで昇順にソートする
6: initHashTable()                          ▷ ハッシュ表の初期化
7: for each < i, X_i, f_i > ∈ CS do
8:     key ← calcHashkey(X_i)
9:     if HashTable[key]= NULL then
10:        HashTable[key][0] ← X_i          ▷ X_i を格納
11:        HashTable[key][1] ← i
                ▷ X_i に対応する飽和集合の識別番号を保持
12:    end if
13:    push(S, X_i)                         ▷ X_i を S に追加
14:    while S ≠ ∅ do
15:        Y ← pop(S)                       ▷ S から取り出し Y に登録する
16:        Flag ← true                      ▷ Y の極小生成子フラグの初期化
17:        if Y の要素数が 1 より大きい then
18:            for each Y' s.t. Y' ⊆ Y かつ |Y'| = |Y| − 1 do
19:                key ← calcHashkey(Y')
20:                if HashTable[key]= NULL then
                                ▷ i.e. Y' は X_i の生成子である
21:                    HashTable[key][0]← Y'
                                ▷ 探索したアイテム集合を格納
22:                    HashTable[key][1]← i
                        ▷ Y' に対応する飽和集合の識別番号を保持
23:                    Flag ← false         ▷ 極小生成子フラグを設定
24:                    if |Y'| ≠ 1 then     ▷ Y' の部分集合が存在
25:                        push(S, Y')      ▷ Y' を S に格納する
26:                    else                 ▷ Y' の部分集合が存在しない
27:                        MG ← MG ∪ {⟨Y', i, f_i⟩}
                                ▷ Y' を極小生成子集合に登録
28:                    end if
29:                else
30:                    if HashTable[key][1]= i then
                                    ▷ Y' は X_i の生成子である
                                ▷ i.e. Y は極小生成子にならない
31:                        Flag ← false
                                    ▷ 極小生成子フラグを設定
32:                    else   ▷ Y' が X_i 以外の飽和集合の生成子
33:                        do_nothing
34:                    end if
35:                end if
36:            end for
37:        else
38:            do_nothing
39:        end if
40:        if Flag = ture then              ▷ Y が極小生成子である
41:            MG ← MG ∪ {⟨Y, i, f_i⟩}
42:        end if
43:    end while
44: end for
45: return (MG)
```

18 行目の Y は飽和アイテム集合 X_i の生成子であり, Y' は Y よりも要素が 1 つ少ない部分集合である. そのような Y' がすべて生成子でなければ, アイテム集合 Y を X_i の極小生成子として登録する (41 行目). 擬似コード中の $Flag$ はアイテム集合 Y が X_i の極小生成子か判定するためのフラグであり, $true$ ならば極小生成子であり, $false$ ならば Y が極小生成子ではないことを示す.

擬似コード中の HashTable は探索を行ったアイテム集合を格納するハッシュ表であり, アイテム集合 Y が X_i の生成子か判定するために利用する.

アルゴリズムの構成より, 飽和アイテム集合 X の探索時には, $X' \subset X$ なる飽和アイテム集合 X' はすでに探索が終了していることが保証される. このことからアイテム集合 Y がすでにハッシュ表に登録されていれば, X よりも小さい飽和アイテム集合 X' の部分集合となっていることが確認できるので, Y は X の生成子とはならないことが保証できる. 以上から, ハッシュ表の探索のみで生成子判定を行える. また, ハッシュ表登録時の飽和アイテム集合の識別番号を保持することで, HashTable$[key][1] = i$ の時の探索を 1 回だけに抑えている.

4 実験と考察

実験には, Frequent Itemset Mining Dataset Repository [11] から 4 種のデータセットを使用した. 各データセットの詳細を表 3 と表 4 に示す. #(item) はデータセットに含まれるアイテムの種類数, #(trans) はデータセット中のトランザクションの総数, ave(item) は 1 トランザクション中に出現するアイテムの平均数である. #(FIS) は頻出アイテム集合の総数, #(CIS) は頻出飽和アイテム集合の総数, #(MG) は極小生成子の総数である.

表 3: 実験に使用したデータセットの概要

データセット	#(item)	#(trans)	ave(item)
retail	16,470	88,162	10.3
mushroom	119	8,124	23
connect	129	67,557	43
pumsb	7,117	49,046	74

最小支持度 ms を変化させたときの飽和アイテム集合 (CIS) と抽出した極小生成子（MG）のアイテム数の最大, 最小, 平均の値を表 5 に示す. 表 6 は, 上昇型, 下降型, 支持度計算なしアルゴリズムで飽和アイテム集合から極小生成子を抽出した際の計算時間を示したものである. 表 6 における N/A は, 実験を行った 128GB の計算機でメモリ不足で結果を得ることができなかった. 表 6 より, ほぼ全ての場合で, 支持度計算なし手法が高速であることが分かる. これは, 支持度計算なし手法では, 部分集合が生成子であるか否かの判定がハッシュ表の探索のみで行えるためである. データセット connect は, 飽和アイテム集合と極小生成子の平均アイテム数の差が大きく上昇型抽出法が有利であるが, この connect に対しても支持度計算なし手法が優れていることが分かった. 本研究では, 高速化のために支持度は飽和アイテム集合の集合の垂直配置表を用いて計算をしている.

表 4: 抽出数の概要

データセット	ms	#(FIS)	#(CIS)	#(MG)
retail	0.0001	240,852	189,077	191,265
	0.0005	19,242	19,144	19,144
	0.001	7,589	7,572	7,572
mushroom	0.01	90,751,401	51,640	103,377
	0.05	3,755,511	12,843	21,146
	0.1	574,431	4,885	7,615
connect	0.5	88,316,367	130,101	130,101
	0.6	21,250,671	68,343	68,343
	0.7	4,129,839	35,875	35,875
pumsb	0.6	19,537,365	1,075,015	2,854,005
	0.7	2,698,265	241,196	658,379
	0.8	142,156	33,295	67,835

表 5: 飽和集合と極小生成子のサイズの概要

データセット	ms		最大	最小	平均
retail	0.0001	CIS	12	1	2.8926
		MG	8	1	2.7782
	0.0005	CIS	6	1	2.2599
		MG	6	1	2.2532
	0.001	CIS	5	1	2.0704
		MG	5	1	2.0681
mushroom	0.01	CIS	19	1	9.8921
		MG	9	1	4.7047
	0.05	CIS	17	1	7.7304
		MG	8	1	4.1751
	0.1	CIS	16	1	6.8061
		MG	7	1	3.9241
connect	0.5	CIS	21	1	13.0742
		MG	8	1	5.5468
	0.6	CIS	20	1	12.1706
		MG	8	1	5.3307
	0.7	CIS	18	1	11.0001
		MG	7	1	5.1645
pumsb	0.6	CIS	22	1	9.9759
		MG	17	1	8.9568
	0.7	CIS	18	1	9.1607
		MG	14	1	8.1535
	0.8	CIS	14	1	6.9548
		MG	11	1	6.2035

そこで, 表 7 に垂直配置表およびハッシュ表のエントリ数, 表 8 に垂直配置に格納されている飽和アイテム集合の ID のリストの長さの平均値を示す. 垂直配置表の空間 (メモリ) 使用量は, 概ね (エントリ数) * (ID リストの平均長) で見積もれる. 例えば, connect の $ms = 0.5$ の上昇型では, $(1.3 * 10^5) * (1.1 * 10^3) = 1.4 * 10^8$ であり, ハッシュ表よりも概ね大きい.

表 6: 抽出時間

データセット	ms	上昇型 [s] (既存手法)	下降型 [s] (既存手法)	支持度計算なし [s]
retail	0.0001	4.10	5.92	0.17
	0.0005	0.07	0.12	0.01
	0.001	0.01	0.02	0.01
mushroom	0.01	87.34	N/A	349.48
	0.05	1.74	16.53	4.14
	0.1	0.22	1.59	0.46
connect	0.5	706.00	N/A	537.64
	0.6	164.55	695.79	53.45
	0.7	24.72	54.17	6.49
pumsb	0.6	N/A	N/A	13,208.67
	0.7	3,504.23	785.46	30.56
	0.8	1.62	1.29	0.14

表 7: 垂直配置表及びハッシュ表のエントリ数

データセット	ms	上昇型	下降型	支持度なし
retail	0.0001	191,265	167,143	240,852
	0.0005	19,114	12,390	19,242
	0.001	7,572	5,012	7,589
mushroom	0.01	103,377	N/A	90,751,401
	0.05	21,146	3,754,069	3,755,511
	0.1	7,615	573,884	574,431
connect	0.5	130,101	N/A	88,316,367
	0.6	68,343	21,248,568	21,250,671
	0.7	35,875	4,128,619	4,129,839
pumsb	0.6	N/A	N/A	19,537,365
	0.7	658,379	2,686,527	2,698,264
	0.8	67,835	139,011	142,156

5 まとめ

本研究では, 極小生成子の飽和アイテム集合からの支持度計算なし抽出手法を検討した. 支持度計算なし手法と既存手法の比較を行い, 実験結果より, おおむね支持度計算なし手法が優れていることが確認できた.

極小生成子を抽出する際に, 探索したアイテムをハッシュ表に保持しなければならず, メモリを膨大に使用する. 今後の課題として, ハッシュの代用となるデータ構造の検討がある.

表 8: IDList の平均長

データセット	ms	上昇型	下降型
connect	0.5	1,167.29	N/A
	0.6	818.69	68.76
	0.7	507.92	60.10
pumsb	0.7	318.87	143.95
	0.8	81.27	56.23

謝辞

本研究は一部, ISPS 科学研究費補助金 (No.16K00298) および JST さきがけの援助を受けている.

参考文献

[1] J. Han, J. Pei and Y. Yin: Frequent Patterns without Candidate Generation. *Proc. ACM-SIGMOD' 00*, pp.1-12, (2000)

[2] X. Wu, C. Zhang, and S. Zhang: Efficient Mining of Both Positive and Negative Association Rules. *ACM Trans. on Information Systems*, Vol.22(3), pp.381-405, (2004)

[3] H. Wang, X. Zhang and G. Chen: Mining a Complete Set of Both Positive and Negative Association Rules from Large Databases. *Proc. the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining (PAKDD'08)*, pp.777-784, (2008)

[4] M. Zaki: Mining Non-Redundant Association Rules. *Data Mining and Knowledge Discovery*, Vol. 9, pp. 223-248, (2004)

[5] 井出典子, 岩沼宏治, 山本泰生: 負の相関ルールを抽出する高速トップダウン型アルゴリズム, 人工知能学会論文誌, 29 巻 4 号, pp. 406-415, (2014)

[6] 岩沼宏治, 佐生隼一, 黒岩健歩, 山本泰生: 負の相関ルール集合の極小生成子に基づく圧縮表現, 情報処理学会論文誌, 57 巻 8 号, pp. 1845-1849, (2016)

[7] 宇野 毅明, 有村 博紀: LCM 公開プログラム, <http://research.nii.ac.jp/uno/ dodes-j.htm>

[8] 佐生 隼一, 岩沼 宏治, 山本 泰生, 黒岩 健歩: 負の相関ルールマイニング効率化のための極小生成子の抽出計算, 人工知能学会第 103 回人工知能基本問題研究会, SIG-FPAI, B5-03, pp. 61-66, (2017)

[9] 谷島健斗, 岩沼宏治, 黒岩健歩, 佐生隼一, 山本泰生: 極小生成子を用いた負ルール抽出計算の効率化, 第 31 回人工知能学会全国大会, 4A1-3in1 , (2017)

[10] 谷島健斗, 岩沼宏治, 山本泰生: 負の相関ルールマイニングの効率化のための飽和アイテム集合からの極小生成子の高速抽出, 人工知能学会第 112 回知識ベースシステム研究会, SIG-KBS, B5-02, pp. 17-24, (2017)

[11] Frequent Itemset Mining Dataset Repository, <http://fimi.ua.ac.be/>(2017-2-28)