行列操作としての頻出アイテム集合列挙

Frequent Itemset Enumeration as Matrix Operation

上原子 正利◆ 小柳 滋◆

Masatoshi KAMIHARAKO Shigeru OYANAGI

頻出アイテム集合列挙はデータマイニングの分野で頻繁に研究されてきた問題である．一般にこの問題はアイテムとトランザクションに対する操作として考えられるが，本論文ではこの問題を行列操作として考える．行列操作としてのこの問題は2値行列から一定の条件を満たす密度1部分行列の列集合を発見することに相当し，この問題専用のデータ構造は汎用行列データ構造の変形と見なせる．さらに我々は，汎用行列データ構造を用いて行列操作としての頻出アイテム集合列挙アルゴリズムが定義できることを示す．

Frequent itemset enumeration is a problem which has been studied frequently in datamining field. This problem is generally seen as operation on items and transactions. On the other hand, we review this problem as matrix operation. We can see this problem as finding column sets of a certain type of density 1 submatrices from a binary matrix, and the data structure for this problem as deformed general matrix data structure. We also show that we can define algorithms which use general matrix data structures and represent this problem as matrix operation.

1. はじめに

データマイニングの分野で頻繁に研究されてきた問題に頻出アイテム集合列挙(Frequent Itemset Enumeration，以下FISE)がある．この問題ではアトミックなデータをアイテム，その集合をトランザクションとそれぞれ呼ぶ．この問題の目的は，与えられたトランザクション集合中の一定数以上のトランザクションに含まれるアイテム集合の列挙である．

FISEに関する初期の議論は，対象となるデータ集合がディスク上に存在すると想定し，データ集合へのアクセスの単位をトランザクションとしていた．それに対して2000年頃からのアルゴリズムは，対象データ集合を主記憶上に保持し，トランザクションとアイテムのアクセスを併用できるデータ構造を用いている．これらのアクセスの併用はFISEの性質を反映するため，このようなデータ構造は重要な進歩である．

その一方で，この問題に対する視点は90年代から変わっていない．その視点は，対象データを当初の表現であるアイテムとトランザクションのまま捉えるものである．FISEアルゴリズムの処理対象はデータベースやトランザクションといったデータに限らないため，この表現は必然的ではない．にもかかわらず，この問題の議論はこの視点に制約され，より一般的な表現との対応が重視されない．その結果，関連する概念が無関係な言葉で表現され，不適切な用語が利用され，議論が必要以上に複雑になる．

本論文の目的はFISEの単純な表現方法を示すことである．そのような表現方法として本論文では行列を用いる．行列の利点は3つある．まず，行列は一般的な概念であるため，「トランザクション」のような特殊な用語を排除できる．また，行列は図で示せるため，人間にとって直感的である．さらに，行列は全ての要素にアドレスを付けた表現であり，概念的にデータ構造と近く，計算機にとって自然な表現である．

本論文ではまずFISEの問題定義を行列で記述し直す．次に既存のFISEアルゴリズムとそのデータ構造を行列の観点から再考する．最後に汎用行列データ構造を用いた新しいFISEアルゴリズムを定義する．なお，本論文の議論では集合列挙木[1]の概念を前提とする．

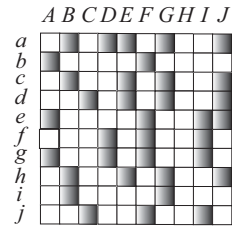
2. 行列と FISE の問題定義

本章ではFISEの問題定義を行列表現で記述する．一般的な表現については[2]を参照されたい．

FISEの各トランザクションはアイテムの非順序集合である．そのため，何らかの基準でアイテムの順序を決めれば，それに従って各トランザクションを並べ替え，データ集合を1つの2値行列として表現できる．図1(a)は一般のアイテムとトランザクションに基づくデータ集合，図1(b)はその行列表現である．以下では「行集合」で行番号の集合を，「列集合」で列番号の集合を指す．個々の行ベクトルに1要素を持つ列をその行の「1要素列」と呼ぶ．「1要素行」も同様である．

TID	Items
a	*B, D, E, G, J*
b	*A, F*
c	*B, E, G, J*
d	*C, E, G, J*
e	*A, F, I, J*
f	*D, F, I*
g	*A, D, F, I*
h	*B, E, G, J*
i	*B, G*
j	*C, F, I*

(a)



(b)

図1 データ集合の例

Fig.1 An Example of Dataset

行列で考えると，FISE は次の条件を満たす列集合の列挙操作となる．その条件は，その列集合の全ての列に1要素を持つ行の数が所与の自然数 *minsup* 以上というものである．以下では列集合 C の全ての列に 1 要素を持つ全ての行からなる集合を「C の密度1行集合」と呼ぶ．ある列集合 C の密度1行集合 R について，その要素数が *minsup* 以上なら，R と C が作る部分行列を頻出アイテム集合部分行列と呼ぶ[3]．この部分行列では列集合が決まれば行集合も一意に決まるため，これを指定するには元の行列と列集合で十分である．*minsup* はこの部分行列に要求される最小行数である．

逆単調性は次のように表現できる．図1の列集合$\{E\}$の密度1行集合は$\{a,c,d,h\}$だが，拡大した列集合$\{B,E\}$の密度1行集合は$\{a,c,h\}$であり，逆単調性は列集合が拡大すると

◆ 正会員 立命館大学 m7i@mail.goo.ne.jp
◆ 正会員 立命館大学 oyanagi@cs.ritsumei.ac.jp

密度1行集合が縮退することを意味する．あるいは，列集合の間に包含関係が成立すればそれらの密度1行集合の間には逆方向の包含関係が成立すると言える．図2の2つの頻出アイテム集合部分行列 $R1\times C1$ と $R2\times C2$ はこれを示し，直感的に表現すると「横に伸ばせば縦に縮む」となる．

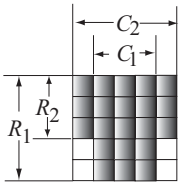


図2　逆単調性：横に伸ばせば縦に縮む

Fig.2　Anti-Monotone Property : Stretched in Width and Shrinking in Height

なお，FISEはしばしばfrequent itemset/pattern miningと呼ばれるが，ここでの「pattern」は集合を指し，「mining」は列挙を指す．本論文ではより単純な表現を選び，「集合」と「列挙」を用いる．

3. 行列とFISEのデータ構造とアルゴリズム

FISEの問題定義を行列で表現できることは，既存のFISEアルゴリズムも行列で表現できることを意味する．本章ではまず汎用の行列データ構造を紹介し，次に既存のFISEアルゴリズムとそのデータ構造を行列の観点から再考する．

汎用の行列データ構造で最も単純なものは，全ての行列要素を記録した配列である．このデータ構造では任意の行列要素に定数時間でアクセスできるが，行列が疎になると不要なゼロ要素が増え，記憶効率と非ゼロ要素への選択的なアクセスの効率が落ちる．疎行列では一般に配列とリストを組み合わせたデータ構造が用いられる[4][5]．これは一般に隣接リストと呼ばれるが，本論文ではALM（配列・リスト行列）と呼ぶ．ALMの難点は列アクセスの効率が悪いことである．すなわち，ある列の1要素行を知るには全ての行リストを調べる必要がある．以下ではこの操作を全行スキャンと呼ぶ．

ALMの難点を解決する行列データ構造に[6]のものがある．これを要素アクセスの観点から見ると，図3のように行のALMと列のALMを組み合わせたものに等しい．以下ではこれをALDM（配列・リスト双対行列）と呼ぶ．ALDMでは行集合を表す配列を行エントリ，列集合を表す配列を列エントリと呼ぶ．

ALDMを単純化したものが[3]のデータ構造である．これはALDMの行と列を分離し，リストをソート済み配列に換えたものである．以下ではこれをAADM（配列・配列双対行列）と呼ぶ．図4は図1(b)のAADMである．

これらの汎用データ構造のうち，ALMはApriori[7]の行列データ構造と見なせる．Apriori はデータ集合のデータ構造を明示しないが，データ集合に対する操作が個々のトランザクションを取り出すものであるため，そのデータ構造はALMと見なせる．

Apriori は頻出アイテム集合部分行列を列集合だけで保持する．要素数 k の列集合を構成するには，まず要素数 $k-1$ の列集合から可能な全ての組み合わせを作る．次に，それらの密度1行集合の要素数が *minsup* 以上あるかを全行スキャンで確認する．

Apriori にはALMに起因する2つの難点がある．その1

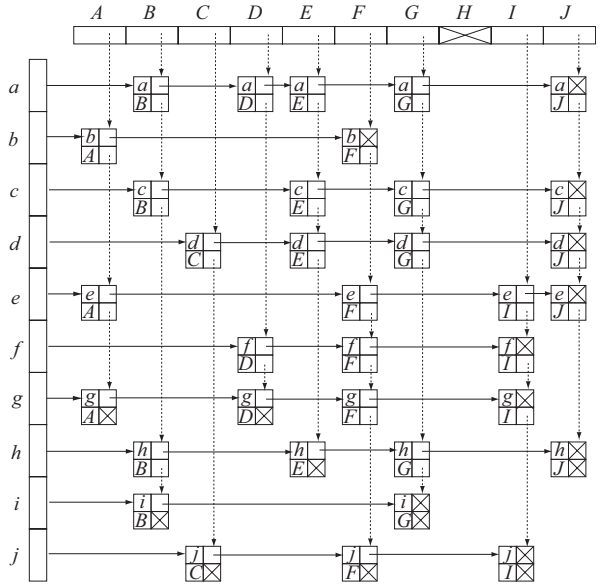


図3　ALDM（配列・リスト双対行列）

Fig.3　ALDM (Array-List Dual Matrix)

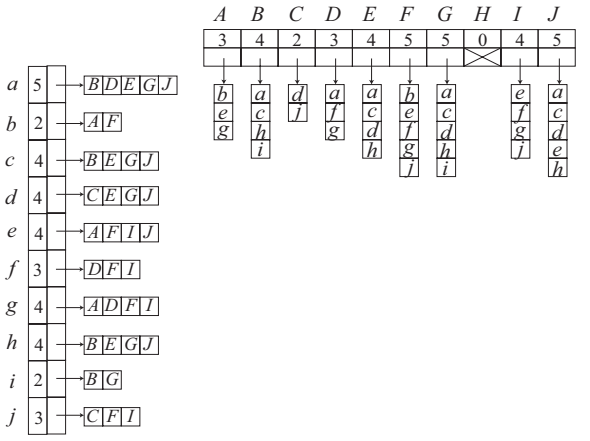


図4　AADM（配列・配列双対行列）

Fig.4　AADM (Array-Array Dual Matrix)

つは，要素数 $k-1$ の列集合に列番号を追加して要素数 k の列集合を作る際，追加後の列集合を頻出とする列がわからず，とりあえず追加してからその密度1行集合の要素数を全行スキャンで調べなければならないことである．この方法は「候補生成と検査 (candidate generation-and-test)」と呼ばれる．この方法を行列で考えると，列集合から列集合を構成し，その後に行列要素を確認する「列集合-列集合-行列要素」の方法と言える．もう1つの難点は，処理時間を消費する全行スキャンの回数を減らすため，集合列挙木を記憶量の効率が悪い幅優先で探索することである．

これらの難点を解消したアルゴリズムにFP-growth[8]がある．これは行列に対して行と列の両方からアクセスできるデータ構造を用いることで，全行スキャンをせず追加すべき列を決定し，集合列挙木を深さ優先で探索できる．また，その部分行列は列集合と行列要素を組み合わせている．

FP-growth が用いるデータ構造の FP 木は，名称と異なり木ではなく，ALDM の変形と見なせる．この変形は列の選択および順序変更と行の圧縮からなる．図 5 は *minsup* を 3 として図 1 から構成した FP 木である．ここでの行エントリの要素は 2 つだけだが，これは行が圧縮されたためである．

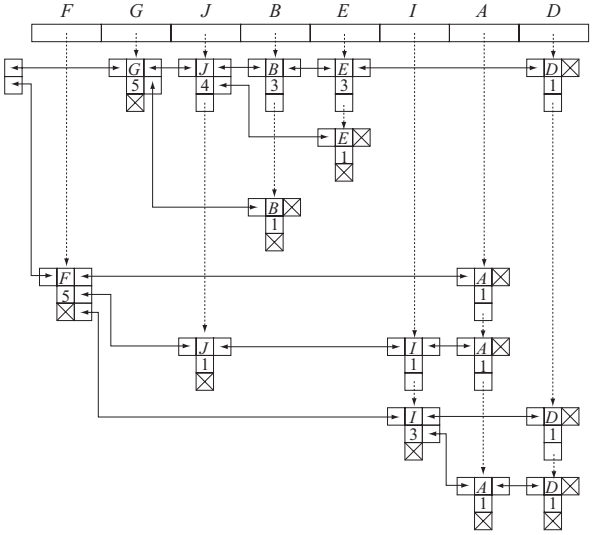


図 5　ALDM の変形としての FP 木

Fig.5　FP-tree as Deformed ALDM

FP-growth の列集合構成方法は「パターン拡大（pattern growth）」と呼ばれる．これを行列で考えると，列集合から行列要素を確認して新しい列集合を作る「列集合-行列要素-列集合」の方法と言える．これが可能であるのは，FP 木によって行と列の両方から行列要素へアクセスできるためである．なお,「候補生成と検査」の Apriori も「パターン拡大」と異なる方法で「パターン」を拡大する．両者の相違点は不要な拡大「パターン」を生成するかどうかである．

4. Belone：汎用行列データ構造を用いる列集合-行列要素-列集合 FISE アルゴリズム

FISE は行列操作として記述でき，FISE 専用データ構造は行と列の両方からアクセスできる特殊な行列データ構造と見なせる．それならば，議論を単純化して，行と列の両方からアクセスできる汎用の行列データ構造を用いて，列集合-行列要素-列集合の FISE アルゴリズムを定義できるはずである．本章ではこのようなアルゴリズムの例として Belone（ビローン）を定義する．

Belone は行列のデータ構造として概念も実装も単純な AADM を用いる．AADM を用いることで Belone は列集合-行列要素-列集合と集合列挙木の深さ優先探索を実行できる．ただし，AADM は要素を行と列の二重に記憶するため，密行列に対しては全要素配列の行列と比べて記憶量で劣る．FP-growth と比較すると，Belone は行圧縮を行うがその効率は FP 木より劣り，主記憶上の行列データ構造の列は非頻出アイテムも含む．このうち前者は単純さを重視した結果であり，後者は Belone が与えられた行列を操作するだけで行列の構成を含まないためである．

Belone の基本的な行列操作は，集合列挙木の節点のヘッドとその密度 1 行集合からなる部分行列を図 2 の $R1\times C1$ から $R2\times C2$ のように横に引き伸ばすものである．すなわち、処理過程でヘッドは拡大し，その密度 1 行集合は縮退してゆく．Belone という名称は，何らかの物を引き伸ばす際の日本語の擬態語に由来する．

Belone の行列操作を図 6 の木で説明する．アルゴリズムの細部と AADM 以外のデータ構造については[9]を参照されたい．この木は図 1(b)から最小行数 *minr* を 3 として実行した場合のものである．図中の数字は構成された順序を示し，(11)(12)(16) より下と(14)より右は省略している．

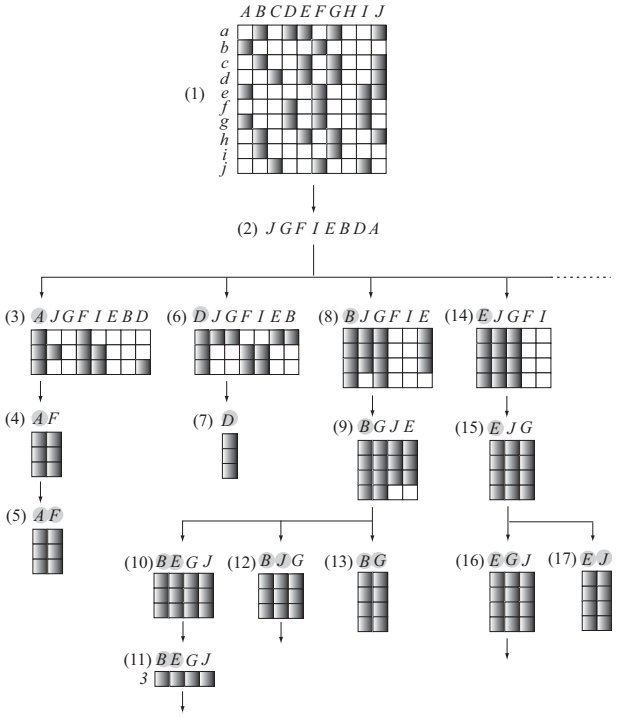


図 6　Belone の行列操作木

Fig.6　A Matrix Operation Tree of Belone

Belone はまず行列(1)の列を調べ，1 要素数が *minr* 以上の列集合を 1 要素数で降順ソートした順序集合(2)を作る．この集合の要素を「有効列」と呼ぶ．次の(3)は，有効列配列の終端要素 *A* を集合列挙木の節点のヘッドとし，それ以外をテイルとしたものである．図中ではヘッドを網掛けで示す．行集合はヘッドの密度 1 行集合である．次にテイルの各列について，その列が行集合内に持つ 1 要素の数を調べる．この数はテイルの列ベクトルと行集合内の全ての行に 1 要素を持つ架空の列ベクトルとの内積に等しい．内積が *minr* 未満の列は逆単調性より不要であるため排除し，(4)を作る．この時点でヘッドは頻出アイテム集合と判明しているため出力する．

次にテイルの終端要素をヘッドに移動して(5)を作る．このような要素の移動は一般に行集合の縮退を伴うが，ここでは発生していない．図中で行縮退が発生するのは(9)から(10)と(12)への遷移である．(5)では頻出であることが判明している{*A, F*}を出力し，テイルが無くなったため処理は後戻りする．この時点で *A* を含む頻出アイテム集合が全て列挙された．

(10)から(11)では行圧縮を行っている．(10)の部分行列では 3 つの行が同じ列に 1 要素を持つため，これらをまとめ，まとめた行に圧縮数 3 を割り当てる．この数を用いると，(11)の継続節点での列の内積計算時に 1 つの行の確認で 3 つの行の確認と同じ結果が得られる．この圧縮は 1 要素列が完全に一致する行だけを対象とするため，プレフィックスの一致で

圧縮する FP 木より圧縮効率が低い．

Belone を定義した目的は FISE を行列操作として記述するアルゴリズムの存在を示すことであり，本論文の主な内容は以上で全て示された．以下は本論文の主要な目的ではないが，Belone の性質を調べるため，性質の異なる 2 つのデータ集合を用いて FP-growth と処理速度を比較する．

データ集合にはFIMIのウェブサイト[1]で公開されているT40I10D100K（以下T40）とchessを用いる．T40 は疑似乱数で作られた人工の疎行列であり，chessはゲームの状態から作られた[1]自然な密行列である．FP-growthの実装はChristian Borgeltによる[2]．この実装は一般のFP-growthに加えて枝刈り[10]を利用できるため，ここでは枝刈りを行う場合と行わない場合の両方で速度を計測した．我々の実装はC++による[3]．計算機はCPUがPentium IV 3.2GHz，RAMが1GBのPCで，OS はLinux，コンパイラはGCCである．

処理時間の計測結果を図 7 に示す．T40 での枝刈りなし FP-growth は非常に遅く，枝刈りありの FP-growth でも Belone より 3 倍程度遅い．chess では逆に FP-growth が 2 倍程度速い．データによる優劣の違いの原因として考えられるものは FP 木の性質である．FP 木は自然な意味の無い人工の疎行列をうまく圧縮できない一方，chess のように自然な密行列は効率良く圧縮できると考えられる．

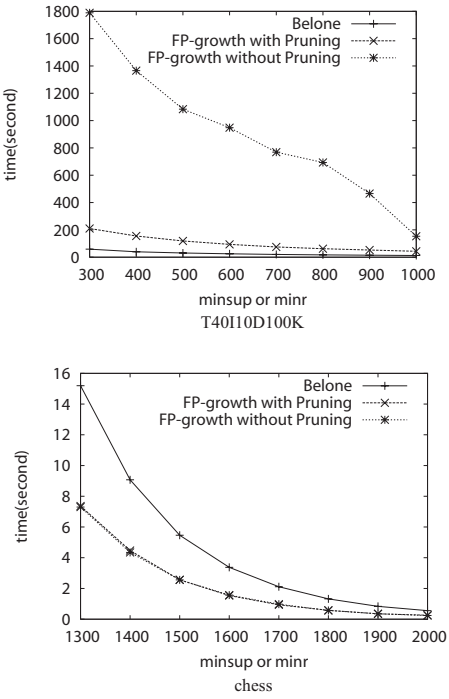


図 7　Belone と枝刈りあり/なしの FP-growth の処理時間

Fig.6　Processing Times of Belone and FP-growth with/without Pruning

5. おわりに

本論文では FISE を行列操作として捉え，逆単調性などの FISE の概念を行列の観点から記述した．また，FISE のデータ構造を汎用の行列データ構造と関連付け，FISE アルゴリズムを行列操作の観点から記述した．さらに，汎用行列データ構造を用いて FISE アルゴリズム Belone を定義した．

本論文で議論できなかった問題として，関連問題の極大頻出アイテム集合列挙[1]などがある．このような問題も行列操作として記述できると考えられるため，今後このような拡張を Belone に導入できる可能性がある．

また，行列の性質によって Belone と FP-growth の優劣が変わる理由についてさらに考える必要がある．実験結果を見ると，行列の圧縮可能性のようなデータの性質が処理速度に大きく影響していると考えられ，また，この変化がアルゴリズムではなく実装に起因する可能性もある．これらの点を明確にすることは，データの種類に合わせたアルゴリズムの選択を可能にし，効率の良い問題解決を実現する鍵になると考えられる．

[1] http://fimi.cs.helsinki.fi/data/

[2] http://fuzzy.cs.uni-magdeburg.de/~borgelt/fpgrowth.html, version 1.8 , 2005.12.06

[3] 著者までご連絡くだされればBeloneのソースコードをお送りします．

[文献]

[1] J.R.J. Bayardo: “Efficiently Mining Long Patterns from Databases”, Proceedings of the ACM SIGMOD, Seattle, WA, pp.85-93 (1998).

[2] 福田剛志，森本康彦，徳山豪: “データマイニング”，共立出版 (2001).

[3] 上原子正利，小柳滋: “内積縮退 MC:類似行の検出と類似列の検出を組み合わせたマトリクスクラスタリングアルゴリズム”，情報処理学会論文誌：データベース, 45, SIG 7 (TOD22), pp.151-162 (2004).

[4] A.V. Aho and J.D. Ullman: “Foundations of Computer Science C Edition”, Computer Science Press (1995).

[5] ジョン・ベントリー（小林健一郎訳）: “珠玉のプログラミング　本質を見抜いたアルゴリズムとデータ構造”，ピアソン・エデュケーション (2000).

[6] 小柳滋，久保田和人，仲瀬明彦: “Matrix Clustering : CRM 向けの新しいデータマイニング手法”，情報処理学会論文誌, 42, 8, pp.2156-2166 (2001).

[7] R. Agrawal, R. Srikant: “Fast Algorithms for Mining Association Rules”, Proceedings of the 20th VLDB Conference, pp.487-499 (1994).

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

[9] 上原子正利，小柳滋: “行列操作としての頻出アイテム集合列挙”, DEWS2006

[10] C. Borgelt: “An Implementation of the FP-growth Algorithm”, Workshop Open Software for Data Mining (OSDM'05, Chicago, IL) (2005).

上原子 正利　Masatoshi KAMIHARAKO

立命館大学研究員．1997 年京都大学工学部情報工学科卒業．1999 年同大学大学院工学研究科情報工学専攻修士課程修了．日本データベース学会会員．

小柳 滋　Shigeru OYANAGI

立命館大学教授．昭和 47 年京都大学工学部数理工学科卒業．昭和 52 年京都大学大学院工学研究科数理工学専攻博士課程修了．同年（株）東芝入社．平成 14 年より現職．工学博士．データマイニング，並列処理，コンピュータアーキテクチャに関する研究に従事．電子情報通信学会，ACM，IEEE-CS

各会員.