最適化から見たデータマイニング

宇野　毅明

機械学習とデータマイニングは，データから知識を見つけ出すという意味において同じようなものとしてとらえられることが多いが，実態としては，その方向性に大きな違いがある．機械学習が全体を俯瞰した知識を少量得るのに対し，データマイニングは局所的な構造を大量に得ることを目的とし，特に大規模なデータに焦点を当てている．本稿では OR 的，特に最適化的な視点からデータマイニング，特にパターンマイニングを解釈し，その技術を紹介する．列挙的な計算での大規模データの取り扱い方と，実データ持つ構造に対するアルゴリズムの挙動，大規模データでの効率性の秘密もあわせて紹介する．

キーワード：パターン発見，列挙，大規模データ

1. はじめに

データマイニングは，大量のデータから有用な規則性やパターンを見つけるための手法である．統計学や機械学習と，根っこの部分は同じようなものであるが，その表面に現れる手法としてのスタンスは大きく異なるところがある．現在，データマイニングという言葉はビジネスシーンを始め多様な分野で使われているため，元来の意味より多少多くの事柄を含んでいるように思われる．

データマイニングの出発点は，データの中にある局所的な特徴やパターンといったものを，効率よく取り出したい，というものであった．従来の統計的な手法や，その後発展する機械学習は，データの分布や少数のグループへのクラスタリングなど，データの大域的な特徴をとらえることを目的としている．対してデータマイニングでは，比較的少数の項目に共通する特徴や，あちらこちらに現れる基礎的なパターンといった，局所的な構造を探し出すことを目的としている．特に，機械学習は，データを最適な形でモデルに当てはめる，という最適化問題を解くことが多いことに対して，データマイニングは，ある種の特徴を持つ構造を列挙するという，列挙問題として定式化されることが多い．

データマイニングは，巨大なデータを扱う，ということを基本理念としているため，その手法も巨大データを対象として設計されていることが多い．なお，テキストマイニングや評判分析と呼ばれるものは，巨大データを対象とはしているものの，基本的には自然言語解析に礎を持つ問題である．また，株価マイニング，のような呼ばれ方をする問題は，統計的な手法を用いた，予測技術であることが多い．

データマイニング研究は，1994 年の Agrawal らの結合規則発見に関する研究を契機として，急速に発展した．結合規則とは，あるアイテムの集合を含むならば，このアイテムを含むことが多い，というような，データに含まれる項目のアイテムの包含に関する傾向を示すものである．この結合規則発見の鍵となる基本的な技術が頻出アイテム集合発見である．この 10 年間で，結合規則発見にとどまらず，深さ優先探索法の提案や，飽和集合発見，構造データへの拡張など，さまざまな形で盛んな研究が続いており，データマイニングの主要なトピックの一つとなっている．

本稿では，**頻出集合発見アルゴリズム**の基礎について解説する．特に，最適化やアルゴリズム研究の立場から，その基本的な構造と，筆者らによる実装コンテスト FIMI'04 での優勝プログラム LCM の開発経験に基づいた，実際のデータのための高速な実装法について解説する．本解説がこれらのアルゴリズムを実際の問題に適用する場合に参考になれば幸いである．

2. 頻出パターン

頻出パターンは，データマイニングの代表的な問題である．多数の項目 (record) が集まってできているデータベースに対して，その中の（与えられた）閾値以上の項目に含まれるようなパターンが頻出パターンであり，頻出パターンをすべて見つける列挙問題が頻出パターンマイニングである．パターンはデータベースの構造によりさまざまなものが考えられる．各項目がグラフであれば，パターンもグラフであるのがある種自

うの　たけあき
国立情報学研究所
〒 101–8430 東京都千代田区一ツ橋 2–1–2

A: 1,2,5,6,7,9	３つ以上に含まれる
B: 2,3,4,5	アイテム集合
C: 1,2,7,8,9	{1} {2} {7} {9}
D: 1,7,9	{1,7} {1,9}
E: 2,3,7,9	{2,7} {2,9} {7,9}
F: 2,7,9	

図 1 トランザクションデータベースと頻出集合

然であるが，そこから頂点ラベルの組合せ，というパターンを見つける問題も，場合によっては重要となるだろう．本稿では，パターンマイニングでも特に基礎的である，頻出アイテム集合マイニングの解説を行う．

2.1 記法と用語

レコード，つまり項目の集合となっているデータベースの中で，トランザクションデータベース(transaction database) とは，各レコード (record) がトランザクション (transaction) と呼ばれるアイテムの集合になっているものである．言い換えれば，すべてのアイテムの集合 $\sum = \{1, \ldots, n\}$ に対して，トランザクションは $\sum$ の部分集合であり，トランザクションデータベースはトランザクションの集合族である．トランザクションデータベースには，同一のトランザクションが複数含まれてもよい．$\sum$ の部分集合をここでは**アイテム集合**(itemset) と呼ぶ．

図 1 の左にトランザクションデータベースの例がある．すべてのアイテムの集合は $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ であり，トランザクションの集合は {A, B, C, D, E, F} である．例えば，スーパーなどの売り上げデータは，各レコードが「1 人のお客さんがどの商品を買ったか」という記録になっているので，スーパーの全商品の集合をアイテム集合と思えば，トランザクションデータベースになっている．

このほか，アンケート結果のデータも，1 つのレコードが，ある人がチェックを入れた項目の集合になっていると考えればよいし，それぞれの項目について，1 から 5 の点数を与えるようなものであれば，「質問と回答の組」（Q1 に 5 がついている，など）がアイテムであると考えることで，トランザクションデータベースとみなせる．このほか，実験データなどもこの範疇に入り，非常に多くのデータベースがトランザクションデータベースであるとみなせる．

2.2 頻出アイテム集合を使ったデータベース分析

さて，このデータを解析しようとすると，どのような視点があるだろうか．視点として，最も多く含まれるアイテムは何か，トランザクションの数はいくつか，トランザクションの平均的な大きさはどの程度か，あるいは大きさの分散はどの程度か，といったものが思いつくだろう．これらは，いわば，データを全体的にとらえて特徴を見いだす手段だと言える．

では逆に，データの局所的な特徴をとらえることはできないだろうか．局所的というと，1 つ 1 つのレコードを見る方法が考えられるが，これでは各レコードの個性に解析が引きずられてしまう．また，トランザクションの大きさ，どのアイテムを含むか，といった特徴では，局所的な個性を語るには弱すぎる．その意味では，「どのようなアイテム集合が多くのトランザクションに含まれるか」という点に注目することは，両方の要求を満たしており，都合が良い．また，多くのレコードに含まれる組合せに注目することで，局所的だが全体にある程度共通する構造を見つけることができる．データの中から，特殊だが顕著な例を見つけるのには適しているだろう．

このように，データベースの中によく現れるアイテム集合を**頻出アイテム集合 (frequent itemset)**，あるいは単に**頻出集合**とよぶ．正確には，**最小サポート**と呼ばれる閾値 σ を用いて，σ 個以上のトランザクションに含まれるようなアイテム集合として定義される．図 1 では，右側に示した集合はどれも少なくとも 3 つ以上のトランザクションに含まれるため，最小サポート 3 に対する頻出集合となっている．アイテム集合 X に対して，それを含むトランザクションを**出現 (occurrence)** と呼ぶ．また，出現の集合を**出現集合 (occurrence set)**，出現の数を**頻出度 (frequency)**，あるいは**サポート (support)** といい，それぞれ **Occ(X)** と **frq(X)** と表記する．図 1 の例では，アイテム集合 $\{2, 5\}$ の出現は 1 行目と 2 行目のトランザクションである．頻出集合は，1993 年に Agrawal, Imielinski, Srikant らによって，結合規則の発見に関連して提案されたのが最初である [1].

通常，頻出集合はある種の知識やルールの候補であるため，1 つだけ見つけてもあまり意味がない．よくある最適化的なアプローチよりは，列挙的なアプローチを用いて全体を俯瞰するような手法なのである．与えられたトランザクションデータベースと閾値 σ に対して頻出集合をすべて見つける問題は**頻出集合発見問題 (frequent itemset mining)**，あるいは**頻出集合の列挙 (enumeration)** と呼ばれる．頻出度はサポート (support) とも呼ばれるため，閾値 σ は**最小サポート (minimum support)** と呼ばれる．

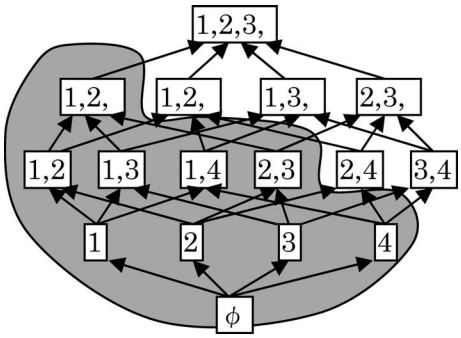

図 2　アイテム集合が作る束と頻出集合の領域

頻出集合は σ の大きさによりその数が変化する．計算の立場からは，σ が大きければ解となる頻出集合の数は小さく，有利であるが，モデルの立場からは，有用な知識をもらさず見つけ出すため，σ を小さめに設定することが望ましい．ここにはトレードオフがある．σ を上手に設定する，あるいは頻出度の高いものから k 個頻出集合を見つけることで，解の数を爆発させずに有用な知識を取り出すことが重要である．

3. 頻出集合列挙アルゴリズム

3.1 計算時間の評価

実用上，頻出集合発見を用いるような場合，入力するデータベースは巨大であることが多く，また出力する解の数も大きい．そのため，計算時間が爆発的に増加する解法は現実的には使用できない．現実的な時間内に求解を終了させるためには，アルゴリズム的な技術が欠かせない．

一般に列挙アルゴリズムの計算時間の大きな変化は，解数の変化によってもたらされる．そのため，**出力の大きさ**，すなわち，解数に対する評価が重要となる．計算時間が [入力の大きさ + 出力の大きさ] の多項式時間となっているアルゴリズムは**出力多項式時間 (output polynomial)** と呼ばれ，計算効率の 1 つの指標となっている．また，1 つの解を出力してから次の解を出力するまでの時間が，入力のみの多項式時間となっているとき**多項式遅延 (polynomial delay)** と呼ばれる．

実用的には，マイニング問題は巨大な入力，巨大な出力を持つことが多く，このような多項式性の議論だけでは，現実的な効率性を担保できない．解 1 つ当たりの計算時間が入力の低い次数のオーダー，あるいは定数になっていることが必要である．興味深いことに，今までに知られている多項式時間の列挙アルゴリズムは，ほぼすべてこの条件を満たす．加えて，後に述べる末広がり性を考慮した実装を行うことで，解 1 つ当たりの計算時間が定数に近くなる．つまり，出力多項式時間の解法が存在すれば，一工夫加えることで効率良い実装が得られると考えてよい．

メモリの使用量も，列挙アルゴリズムでは大きな問題である．解が多いため，発見した解をすべてメモリに蓄えるアルゴリズムは，多大なメモリを消費することとなる．発見した解をメモリに保存する必要があるかないか，という点が，メモリの点での計算効率に関する重要なポイントである．

3.2 アプリオリ法

あるトランザクション t が頻出集合 X を含むならば，t は X の任意の部分集合をも含む．このことは，X の部分集合の出現集合は X の出現集合を含み，頻出度は X より同じか大きくなるということを示す．そのため，頻出集合の族は**単調性**を満たす．これは，図 2 で示すようなアイテム集合の束（部分集合間の包含関係を表したグラフ）の上で，頻出集合は下部に連結に存在しているということである．

そのため，空集合から出発し，アイテムを 1 つずつ追加していくことで任意の頻出集合を得ることができる．この操作を網羅的に行うことでアイテム束上の探索を行えば，すべての頻出集合が見つかる．この探索を幅優先的に行うものが**アプリオリ (apriori)**[1, 2]，深さ優先的に行うものが**バックトラック法 (backtracking)** [4, 5, 6, 7,15] である．

アプリオリはとても有名な手法であり，耳にされたことのある読者も多いことと思う．空集合から出発し，大きさが 1 の頻出集合をすべて見つけ，それらにアイテムを 1 つ追加することで大きさ 2 の頻出集合を見つける，というように，頻出集合を大きさ順で，幅優先的に見つけ出すアルゴリズムである．**アプリオリ**のアルゴリズムを以下に示す．

アプリオリ

1. D_1 = 大きさ 1 の頻出集合の集合，k:= 1
2. while $D_k \neq \phi$
3. 大きさが k+1 であり，D_k の 2 つのアイテム集合の和集合となっているものを全て D_{k+1} に挿入する
4. frq(S)<σ となる S を全て D_{k+1} から取り除く
5. k :=k + 1
6. end while

図 3 にアプリオリの実行例を示した．このアルゴリズムでは，k 回目のループが始まるとき，D_k は大きさが k である頻出集合の集合になっている．そして，ステップ 3 で D_k の集合 2 つを組み合せてできる大きさ

k+1 のアイテム集合をすべて D_{k+1} に挿入する．頻出集合の単調性から，大きさが k+1 の任意の頻出集合は必ず D_{k+1} に含まれる．そのため，ステップ 4 で D_{k+1} から頻出でないアイテム集合を取り除くことで，大きさ k+1 の頻出集合の集合が得られる．

アプリオリのステップ 4 で D_{k+1} の各アイテム集合の頻出度を計算する際に，次のアルゴリズムを使うと効率的である．

1. for each $S \in D_{k+1}$ do frq(S) := 0
2. for each transaction $t \in T$ do
3. for each $S \in D_{k+1}$ do
   if $S \subseteq t$ then frq(S) := frq(S)+1
4. end for

まずステップ 1 ですべてのアイテム集合 S について frq(S) を 0 に設定し，ステップ 2 で 1 つずつトランザクションを選び，それが含むアイテム集合 S に対して frq(S) を 1 増加させる．すべてのトランザクションに対してこの処理をすると，frq(S) は S の頻出度となる．これにより，データベースへのアクセス数を少なくすることが可能で，ディスクにデータ保存されているような場合，効率性が高くなる．

アプリオリの計算時間は，（候補の総数）×（データベースの大きさ）に比例し，候補の総数は頻出アイテム集合の数の n 倍（n はアイテムの数）を超えないことを考えると，頻出集合 1 つ当たり，最悪でも（データベースの大きさ）× n に比例する時間となることがわかる（本稿では，データベースの大きさとはデータベースの各トランザクションの大きさを合計したものとする）．頻出集合をすべてメモリに保持する必要があるため，メモリ使用量は見つける頻出集合の数に比例する．アプリオリは，候補を生成する時間が入力の多項式時間とならないため，多項式遅延とはならない．

アプリオリの高速化する方法として，**プルーニング (pruning)** がある．X を D_{k+1} のアイテム集合とする．もし X の大きさ k の部分集合で D_k に含まれないものがあれば，X は頻出集合でないアイテム集合を含むということがわかる．よって，X は頻出集合となることはない．この検査を D_{k+1} のすべてのアイテム集合に対して行えば，頻出度の検査対象を少なくすることができる．例えば，図 2 で D_3 に最初に含まれている $\{1,2,7\}$ は，D_2 に含まれないアイテム集合 $\{1,2\}$ を含んでいる．このことから，$\{1,2,7\}$ は頻出集合ではないと，頻出度の計算をする前に結論づけられるので，ただちに D_3 から取り除くことができる．

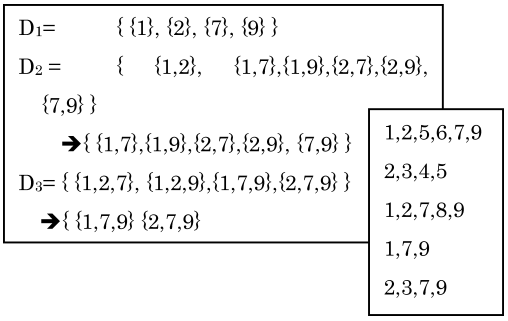

図 3 $\sigma = 3$ でのアプリオリの実行例

3.3 バックトラック法

アプリオリが幅優先的な探索を行うのに対し，**バックトラック法**は深さ優先的な探索を行う．アイテム集合 X に対して，その中の最大のアイテムを**末尾 (tail)** と呼び，tail(X) と表記する．ただし tail(ϕ) は $-\infty$ とする．バックトラック法の各反復では，X にその末尾よりも大きなものだけを加えて，より大きなアイテム集合を生成する．これにより，重複無くすべての頻出集合が探索できる．下記のアルゴリズムは，バックトラック (ϕ) を呼び出すことで，すべての頻出集合が列挙する．

バックトラック (X)

1. output X
2. for each i>tail(X) do
3. if frq(X∪{i})σ then call バックトラック (X∪{i})

バックトラック法の長所は，すでに発見した解をメモリに保存しておく必要がないため，入力データベースを保持するメモリがあれば動く点である．計算量はアプリオリと同じで，1 つ当たり（データベースの大きさ）×n に比例する時間がかかる．また，各反復で解を必ず 1 つ出力するため，多項式遅延となる．ただし，データベースがディスクに格納されている場合は，各反復でデータベースをスキャンすることになるために非効率となる．

バックトラック法は，空集合から出発し，アイテムを 1 から順に追加していく．まず 1 を追加した際に，frq($\{1\}$) $\geqq 3$ が成り立つので再帰呼び出しを行う．$\{1\}$ を引数として呼び出された反復では，tail($\{1\}$) $= 1$ より大きなアイテムを追加する．$2, \ldots, 6$ までは，追加しても頻出集合とはならないため，再帰呼び出しは行われず，7 を追加した際に frq($\{1,7\}$) $\geqq 3$ が成り立つため，再帰呼び出しが行われる．以後同様に処理が進められる．{1,7} に関する再帰呼び出しが終了した後，

{1} に関する反復では，次に 8 を追加する．これは頻出ではないので，再帰呼び出しを行わず，次に 9 を追加する．これは頻出となるため，再帰呼び出しが行われる．

4. 高速化の技術

前述のアプリオリ，バックトラックともに，計算量の上では出力数依存のアルゴリズムであり，出力数多項式時間を達成している．しかし，実際にはデータベース全体をスキャンするという作業は非常に時間がかかり，素直な実装ではとても現実的な時間内に求解できない．そのため，いくつかの高速化技法が提案されている．

4.1 データベース縮約

最初の技法は，データベース縮約と呼ばれるものである．これは，入力するデータベースから，あらかじめ不要な部分を除去することでデータベースを小さくし，メモリ使用量とデータアクセスのコストを短縮する．

アイテム i を含むトランザクションの数が σ 未満であるとき，i を含む任意のアイテム集合の頻出度は σ 未満になり，頻出集合とならない．よって i は追加するアイテムの候補からはずしてよく，さらには，データベースからアイテム i をすべて除去しても計算結果に変化はない．また，すべてのトランザクションに含まれるアイテムは，任意のアイテム集合に追加しても頻出度を変化させないため，同様に不要であることがわかる．これらを取り除くことで，データベースを縮小できる．さらに，もし同一のトランザクションが k 個あれば，それらは 1 つに併合し，併合された数 k を重みのような形で保存することができる．この 2 つの操作により，現実的なデータは，特に最小サポートが大きい場合，大幅に縮小することができる．

図 4 に最小サポートを 3 としてデータを縮約した例を示した．左端のデータベースから 3 つ未満のトランザクションにしか含まれないものを除いたものが真ん中のデータベースであり，さらに同一のトランザクションを併合したものが右である．

4.2 振り分け

振り分け (occurrence-deliver) はデータベースを転置することで各アイテムを追加した場合の頻出度を一度に計算して高速化する技法である [8, 11, 10]．これは，異なるアイテムの種類が非常に多い場合など，疎なデータベースにおけるバックトラック法の高速化において有用な技法である．

t を X の出現とする．t がアイテム i を含むなら，またそのときに限り，X∪{i} は t に含まれる．そこで，t ＼ X に含まれるアイテムすべてについて，t に含まれる，というマークをつけよう．この作業を，X の出現すべてに対して行うと，アイテム i についたマークの集合は，X∪{i} の出現集合になる．これが**振り分け**であり，次のアルゴリズムとなる．

振り分け (X)
1. for each t∈Occ(X) do
2. for each i∈t, i>tail(X) do
3. i に t のマークをつける
4. end for

4.3 再帰的データベース縮約と末広がり性

バックトラック法のためのもう 1 つの有効な高速化技法が，再帰的データベース縮約である．バックトラック法の各反復では，必ず tail(X) より大きなアイテムのみが追加されるため，再帰的に呼び出された反復の中では tail(X) より小さいアイテムは不要である．X の各出現から tail(X) より小さなアイテムをすべて除去すると，スキャンすべき部分のみを持つデータベースができる．これを**条件付データベース (conditional database)** と呼ぶ．

条件付データベースの中では，各アイテム集合の頻出度はもともとのデータベースでの頻出度と同じかまたは小さくなる．そのため，条件付データベースをさらにデータベース縮約すると，より小さいデータベースが得られる．この作業を各反復で行うと，再帰的にデータを小さくできる．これを**再帰的データベース縮約 (recursive database reduction)** と呼ぶ．

バックトラックのような列挙形のアルゴリズムは，一般に各反復で再帰呼び出しを複数回行う．そのため，再帰呼び出しが作る計算木の構造は，根の部分が小さく，葉のほうへ進むほど頂点数が多くなる末広がり的な構造を持つ．つまり，深いレベルの反復での計算時間が全体の計算時間のほとんどの部分を占めるようになる．そのため，上記の手法のように，反復が深くなれば深くなるほど計算時間が縮小されるような工夫を加えることで，大きな計算時間の改善が行えるのである．こ

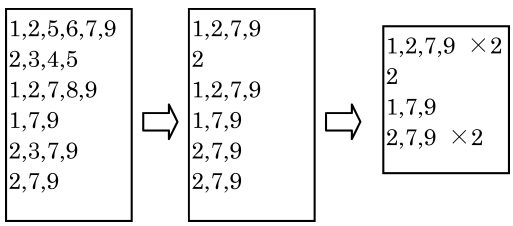

図 4 データベース縮約

の性質は多くのパターンマイニングアルゴリズムを始め，列挙アルゴリズム全般に成り立つ性質である．

実験的には，このような改良を行わない場合，解の増加とともに計算時間は増加していくが，再帰的に計算時間を減少させると，ある程度解数が増えるまでは初期化の部分がボトルネックとなり，計算時間の増加はみられず，解が十分に大きくなったとしても，解の数に比例はするものの，ゆるやかに計算時間が増大していく．そのため，解数が十分大きくなると，解数のみに計算時間が依存するようになり，現実的に非常に優れた解法となる．

5. 実装の比較

さて，これらのアルゴリズムと技術の組合せで，実際のデータに対して有用なものはどれであろうか．この疑問を解決すべく，FIMI (frequent itemset mining implementations) ワークショップ (http://fimi.cs.helsinki.fi/) にていくつもの実装の比較が行われているので，その結果を紹介しよう．

FIMI ワークショップで用いられたデータセット（**FIMI データセット**と呼ぶ）は，現実問題から得られたものが中心である．POS データ，web のクリックデータなどがあり，多くがトランザクションの平均サイズが 10 未満と疎であるが，パワー則を満たし，大きなトランザクションもいくつか含まれている．また，密なデータセットもあり，特に機械学習のベンチマークからきた問題は密である．

まず，全体的な傾向として，頻出集合数がデータベースの大きさに対して十分小さい場合，計算のボトルネックは問題の入力と初期化になる．そのため，複雑なデータ構造や処理を用いない，シンプルなアルゴリズムが高速となる．しかし解数の増加とともに両者はすぐに逆転し，入力と出力がほぼ等しい大きさになるころには両者の差は圧倒的に開く．

これらのデータはメモリに入るため，アプリオリの利点は活かせず，アプリオリは他のアルゴリズムより大きく遅い．振り分けを使うことの効果は認められ，数倍の高速化を達成しており，特に疎なデータでその差が大きい．データベース縮約は，最小サポートが大きくかつ解数が小さい場合に有効であるが，疎なデータなど，最小サポートが 10–20 と非常に小さい場合には効果がない．再帰的なデータベース縮約も同じであるが，特に密なデータで解数が大きくなったときの計算時間の改善率は劇的であり，使用していない実装では計算が終了しないような問題でも解くことができる．

高速な実装では，計算のボトルネックは，ほぼどの問題でも解の出力部分であり，その意味では，十分解の数が大きければ，頻出集合列挙は，現実的にはほぼ最適な時間で行えると考えて良いだろう．

6. まとめ

本稿では，頻出アイテム集合発見問題を，アルゴリズムを中心にして解説した．飽和集合マイニングや一般のグラフマイニング等，本稿で触れられなかった話題に関しては，宇野 [12]，浅井 [3], Washio & Motoda [13]，鷲尾 [14] らの解説を参照されたい．

頻出集合はデータマイニングの基礎であり，実際にツールとして使う機会も多い．本稿で紹介したアルゴリズムの多くは FIMI のサイトに実装と解説論文があり，またベンチマーク問題も手に入る．また，著者のホームページ [9] には，いくつかの頻出パターン発見の実装と問題変換など，各種のツールがある．

参考文献

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

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo: Fast Discovery of Association Rules. *Advances in Knowledge Discovery and Data Mining*, pp. 307–328 (1996)

[3] 浅井達哉，有村博紀：半構造データマイニングにおけるパターン発見技法，データ工学論文特集，電子情報通信学会論文誌，Vol. J87-D-I, No. 2, pp. 79–96 (2004)

[4] R. J. Bayardo Jr.: Efficiently Mining Long Patterns from Databases. *Proc. SIGMOD Conference 1998*, pp. 85–93 (1998)

[5] J. Han, J. Pei, Y. Yin: Mining Frequent Patterns without Candidate Generation. *Proc. SIGMOD Conference 2000*, pp. 1–12 (2000)

[6] S. Morishita, A. Nakaya: Parallel Branch-and-Bound Graph Search for Correlated Association Rules. *Large-Scale Parallel Data Mining*, pp. 127–144 (1999); S. Morishita: On Classification and Regression. *Proc. Discovery Science 1998*, *LNAI 1532*, pp. 40–57 (1998)

[7] S. Morishita, J. Sese: Traversing Itemset Lattice with Statistical Metric Pruning. *Proc. PODS 2000*, pp. 226–236 (2000)

[8] J. Pei, J. Han, B. Pinto, Q. Chen, U. Dayal, M. Hsu: PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. *Proc. IEEE ICDE 2001*, pp. 215–224 (2001)

[9] 宇野毅明のホームページ，
http://research.nii.ac.jp/~uno/index-j.html

[10] T. Uno, M. Kiyomi, H. Arimura, LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets. *Proc. FIMI'04* (2004)

[11] T. Uno, T. Asai, Y. Uchida, H. Arimura: An Ef-

ficient Algorithm for Enumerating Closed Patterns in Transaction Databases. *Proc. Discovery Science 2004*, *LNAI 3245*, pp. 16–31 (2004)
[12] 宇野毅明，有村博紀：データインテンシブコンピューティング その 2 頻出アイテム集合発見アルゴリズム，レクチャーシリーズ「知能コンピューティングとその周辺」，（編）西田豊明，第 2 回，人工知能学会誌，2007 年 5 月号 (2007)
[13] T. Washio, H. Motoda: State of the Art of Graph-Based Data Mining. *SIGKDD Explorations*, 5(1): pp. 59–68 (2003)
[14] 鷲尾隆：データインテンシブコンピューティング その 1 離散構造マイニング，レクチャーシリーズ「知能コンピューティングとその周辺」，（編）西田豊明，第 1 回，人工知能学会誌，2007 年 3 月号 (2007)
[15] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li: New Algorithms for Fast Discovery of Association Rules. *Proc. KDD 1997*, pp. 283–286 (1997)

