RF-006

負の重みに対応した高速頻出集合発見プログラムの開発

Efficient Algorithm and Implementation for Frequent Itemset Mining with Negative Weights

宇野 毅明†
Takeaki Uno

1．まえがき

近年の巨大データの出現により，データベースの分析を通じた半自動的な学習・知識発見は科学・産業の中で重要な位置を占めている．巨大データを扱うデータマイニングの分野では，データベースに多く現れる頻出パターンを見つける問題が多く研究されている．基礎的な問題であるが故に実装も多いが，アルゴリズムの一工夫による基本的な拡張機能を行なっていないものが多く，実用上では大きな弊害となっている．本稿ではそのようなアルゴリズムの一工夫として負の重みを扱う頻出パターン発見問題に対するアルゴリズムの提案を行なう．著者らが開発した高速頻出集合発見アルゴリズムである LCM にこのアルゴリズムを組み込むことで，高速な実装を開発した．その有効性を計算実験により検証し，アルゴリズムの高速性を示した．

1. 導入

近年の IT 技術の発達により，様々なデータを半自動的に収集することが可能となった．これにより，いろいろな場面，分野で様々な種類の巨大データが集積されるようになった．このことは，データの入手を容易にする反面，自動的に収集できないデータが相対的に入手しづらくなり，また，データの巨大さが解析を困難にしてきている．このような巨大データは，医学・工学・理学などの学術的な分野では，観測データからの知識発見や実験結果の検証などに，工業・経営などの産業の分野ではマーケティングや故障診断などに用いられているが，巨大データの半自動的な解析は大きな課題となっている．

データマイニングは，このような巨大なデータから半自動的にデータの特徴を抽出する手法である．データマイニングの中でも基礎的な手法の一つに頻出パターン発見がある．データベースの多くに含まれるパターンを頻出パターンとよび，与えられたデータベースに対して頻出パターンをすべて見つける問題が頻出パターン発見である．パターンは何らかの構造のクラスで定義され，例えば各項目がアイテムの集合となっているデータベースに対して，パターンをアイテムの集合で定めると，頻出パターンは多くの項目（たいていはある与えられた定数 σ 以上の項目）に含まれるアイテムの組合せ，となる．この問題は特に頻出集合発見[1]とよばれ，単に頻出パターン発見というと，この問題を指すことも多い．図 1 に頻出集合の例を示した．各トランザクションの 1 番左の数値はトランザクションの重みであるので，頻出集合の例では無視されている．

† 国立情報学研究所 〒101-8430 東京都千代田区一ツ橋 2-1-2, uno@nii.jp

同様にして，各項目が時系列データ，文字列，グラフ，ベクトルといった構造であるデータベースに対して，パターンのクラスを系列，ワイルドカード付き文字列，木やサイクルなどで定義し，包含関係を含まれる，埋め込める，部分的に含まれる，似ている，といった条件で定義することで，様々な頻出パターン発見の問題が定義でき，これらは学術・産業の多くの分野で応用を持つ．頻出パターンはデータベースの多くの部分に共通する性質を明らかにすることができるため，巨大データを俯瞰する上で重要な手法なのである．

頻出パターン発見は，直接的な方法で行うと非常に計算コストを要する作業である．パターンは通常組合せ的な構造をもつため，頻出パターンの候補，つまり可能なパターンの数は指数的に大きく，また各パターンの頻出度（いくつの項目に含まれるか）の計算も，データベースの項目全てに対してそのパターンが含まれるかどうかを調べる必要があるため，非常に時間がかかる．包含関係に対しては，キーワード検索のような上手な索引作りが難しいこともその一因である．

ところが，2000 年ごろより比較的高速なアルゴリズムが多く提案されてきており(例えば[2])，巨大なデータでも現実的な時間で解けるようになってきた．その技術の中でも中心的なものは，①頻出パターンの作る族が単調であることを利用して山登り的な探索を行うことにより，出力する解の数に比例した数しか解の候補（頻出であるかを調べる解のこと）を生成しない，② 山登りの探索が先のレベルに進むたびに，再帰的にデータの中から不要な部分を削除し，（通常は指数的に膨らむ）探索末端での計算コストを軽減する，というものである．現在では頻出パターン発見問題はそれほど難しくないと位置付けられているようであり，特に頻出集合発見は高速で解けている．高速化される理由のイメージを図 2 右に示した．

頻出集合発見は，データマイニングを始めとする諸分野の中でも，実装（理論を実現したプログラムのこと）が手に入りやすい問題である．2003,2004 年に FIMI (Frequent Itemset Mining Implementation)[4] という実装のコンペティションを行うワークショップが開かれたこともその要因となっており，現在でも多くの高性能で安定的に動く実装が手に入る．しかし多くの実装が，このコンテストで要求されている機能しか持っていない，という課題もあり，現実問題を解く際にこれらの実装を利用しようとしても，ちょっとした機能の不足から適用が困難となることも多い．

そのような「ちょっとした機能」の一つに，項目の重みの考慮がある．通常，頻出パターン発見においては，データの項目はすべて等価であり，重要性などのファクターは考えないことが多い．しかし，現実問題では項目が重みを持つことが多く，例えばマーケッティングの場合は大量の商品を購入する顧客と少量の商品しか購入し

D =

```
 1: 1, 2, 5, 6, 7, 9
-1: 2, 3, 4, 5
 1: 1, 2, 7, 8, 9
-1: 1, 7, 9
 1: 2, 7, 9
-1: 2
```

3 つ以上に含まれるもの

{1} {2} {7} {9} {1, 7}
{1, 9} {2, 7} {2, 9} {7, 9}
{1, 7, 9} {2, 7, 9}

重み合計が 2 以上であるもの

{7} {9} {2, 7} {2, 9} {7, 9}
{2, 7, 9} {1, 2, 7, 9}

図 1: 重み付きトランザクションデータベースと頻出度 3 以上のアイテム集合，および含まれるトランザクションの重みの合計が 2 以上であるアイテム集合

ない顧客では重要度が異なるのが当然であるし，データによっては，項目に信頼度のようなものがついていることもある．信頼度の高い項目に含まれていることに対して，より高い得点を与えたい，というモデルである．

頻出集合発見アルゴリズムは，上述の②の技術を用いる際に，「同一の項目は 1 つにまとめて，まとめた個数をまとめられた項目の重みとして表現する」と操作を行うことでデータの圧縮をしているものが多い[5,6,8]．つまり，内部的に重みを扱う機能は備えているのである．それにも関らず，インターフェースを持たないがゆえに，外部から重みを指定することができないのである．

上述した例は，項目に「正の重み」を与えるものであるが，応用例によっては，項目に「負の重み」を与えることで効率的な解析が行えるものもある．例えば，都市部の顧客と郊外の顧客のデータを比較し，郊外の顧客に顕著（頻出）であるが，都市部の顧客には顕著でないパターンを見つける，という問題を考えよう．この問題を直接的に頻出パターン発見問題で解こうとすると，郊外の顧客のデータに顕著なパターンをすべて見つけ出し，その中から都市部のデータでは顕著でないパターンを探し出す，ということになる．しかし，負の重みを使ったモデルだと，郊外の顧客のデータの項目に正の重みを，都市部の顧客のデータの項目に負の重みを与えることで，「あるパターンを含む項目の重みの合計が（正の）閾値より大きい」ということが「そのパターンは郊外の顧客データの項目に多く含まれ，都市部の顧客データの項目にはあまり含まれない」ということを示すことになる．この評価値が大きなパターンはコントラストパターンとよばれる．図 1 の右側にコントラストパターンの例を示した．トランザクションの重みは，各トランザクションの 1 番左側の数値である．

同様に 2 つのデータベースでの含まれ方に差異があるパターンを評価するものとして，エマージングパターンがある．あるパターン P に対して，そのパターンを含む郊外の顧客データの項目数を，そのパターンを含む都市部の顧客データの項目数で割ったものを考える．この数値が大きいものがエマージングパターンであり，偏りが大きいパターンと判断される．

コントラストパターンやエマージングパターンは，データを分ける特徴とみなすことができるため，学習や分類の分野でよく用いられる．例えば，コントラストアイテム集合を使って画像分類をおこなう Nowozin らの研究[7]や遺伝子の分類を行なう西郷らの研究[10]がある．その他にも，購買行動を行う顧客と行わない顧客の分類，故障を起こす事例と起こさない事例の分類，発現する遺伝子の分類としない遺伝子の分類，というように，多くの応用がある．巨大なデータに対して，このようなパターンが効率良く発見できれば，学習・分類をはじめとするデータ解析の諸分野に与える影響は大きいであろう．

負の重みを考慮することは，計算上の困難性を 1 つ発生させる．それは，頻出パターンの作る族の単調性を破壊し，山登り的な探索と単純な枝刈りでは解を見つけ損なう可能性があるということである．項目の重みが全て非負であれば，パターンに要素を追加して大きくしたときに頻出度は必ず増加しない．そのため，パターンに要素を追加しつつ，頻出でなくなったところで探索を打ち切る，という探索手法で頻出パターンのみを探索できたが，負の重みを持つ項目が存在すると，パターンに要素を追加した際に頻出度が増加することがあり，頻出でないパターンに対して探索の打ち切りができなくなる．

本稿では，近年の高速頻出集合発見アルゴリズムを，負の重みをも扱えるようにする比較的単純な手法を提案する．著者らは第 2 回の FIMI において LCM ver2 というアルゴリズムを提案し[11,12]，その年の最優秀実装賞を獲得している．本稿では LCM を改良し，単純な仕組みで負の重みを扱えるようにする技術を提案する．また，その実装を作成し，計算実験より評価を行なうと共に，著者のホームページで実装を細部にわたる利用法のドキュメントと共に公開した．実装とドキュメントは http://research.nii.ac.jp/~uno/codes-j.html を参照されたい．

本稿の貢献は，負重みの扱いに関する計算技術を，既存の高速化手法との親和性を重視して詳細に設計したことと，高機能高性能なソフトウェアを開発して利用可能とし，その評価を行なったことにある．本稿で提案する手法は，既存手法に対して非常に単純な改良を行なうものである．これは，アルゴリズム技術的な観点から見ると重要度が低いが，ソフトウェア的な視点からすると，既存の高速化技術との親和性が高い，実現性が高い，といった実用面での利点が非常に高くなっている．頻出パターン発見は，計算技術に対する基礎的な研究は一通り行なわれた感があり，今後は実用面での基礎を固めるべく，効率良い拡張法や利用法を整理する研究が中心になると考えられる．その意味では，本稿のような研究の重要性は高くなるであろう．

以下，2 節で記法の定義を行ない，3 節で LCM の概略と負重みへの対応法を記述する．4 節で実験の結果を示し，5 節でまとめる．

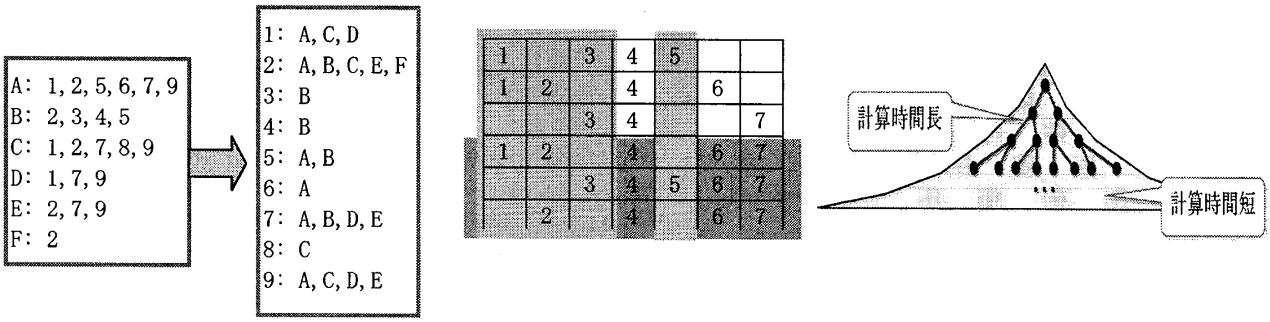

図 2：（左）P=φである場合の振り分けの実行例．振り分けは，トランザクション A に対し，A が含む全てのアイテム i に対して A を occ(i)に挿入する．右側の各行は occ(i)に対応し，振り分けの結果，1 を含むトランザクションの集合となっている．（中）最小サポートσが 3 であり tail(P)が 3 である場合のデータベース圧縮の例．アイテム 5 が頻出度が 2 であるため除去され，下 3 つのトランザクションが同一であるため 1 つにまとめられる．（右）指数的に広がる反復（図の下部）での計算時間を短くすることで，全体の計算時間が短くなる

2. 記法

I = {1,2,...,n} をアイテムの集合，D を，各項目が I の部分集合となっているデータベースとする．D の項目はトランザクションとよばれ，D は 2 つ以上の同一なトランザクションを含んでよいとする．‖D‖ を D に含まれるトランザクションの大きさの合計とし，これを D の大きさとする．

アイテム集合 P に対して，P を含むトランザクションの集合を Occ(P) と表記し，frq(P) = |Occ(P)| とする．tail(P) を P に含まれるアイテムの中で最大のものとする．P が空集合である場合，tail(P)は-1 であるとする．与えられた，最小サポートとよばれる定数 σ に対して，frq(P) ≧ σ を満たすアイテム集合 P を頻出集合とよぶ．また，トランザクション t の重みを w(t) とする．

3 LCM の概略と負重みへの対応法

導入の節で述べたとおり，頻出集合の列挙は山登り探索と単純な枝刈りにより効率良く行える．厳密には，さらに重複を避けるために，末尾拡張という手法が使われている．これは，現在探索中のアイテム集合に対して追加するアイテムは，そのアイテム集合中，最大のアイテムよりも大きなもののみに限定する，というものである．さらに LCM は，振り分けという技法により，現在のアイテムに 1 つアイテムを追加して得られるアイテム集合全てに対して Occ と frq を一度に効率良く計算する手法を使っている．LCM を簡単に記述するとこのようになる．

LCM (*D*, *P*, occ)
1. **output** *P*
2. *D'*,occ = **DatabaseReduction** (*D*, occ) // reduce the database
3. **call Delivery** (occ) // set occ(*i*) := Occ(*P*+*i*) for all *i*>tail(*P*)
4. **for** *i* := maximum item **to** tail(*P*)+1 **step** -1
5. **if** frq(*i*) ≧ σ **then call LCM**(D', *P*+*i*, occ(*i*))
6. occ(*i*) := φ , frq(*i*) :=0
7. **end for**

LCMmain ()
1. load database *D*
2. sort items in each transaction
3. set occ(*i*) := φ for all *i*
4. call **LCM** (*D*, φ , {*t* | *t*∈*D*})

ここで，occ(i) が P+i を含むトランザクションの集合，frq(i)が P+i の頻出度を表す変数である．6. で occ と frq を初期化しているため，各反復の最初では i>tail(P) である i に対して常に occ(i)= φ , frq(i)=0 が成り立っており，変数の再利用が可能である(rightmost sweep というテクニックである)．データベースが疎であるときには，4. のループを occ(i) が非空な i についてのみループを行なうようにすることで探索の無駄を省ける．2. が再帰的にデータを圧縮するところであり，3. が振り分けという手法による，各候補パターンを含むトランザクションの計算である．振り分けは，疎な行列の転置を計算する手法と等価であり，以下のように記述される．ここでは詳しい解説は省略する．

Delivery (occ)
1. **for each** *t* **in** *P*
2. **for each** *j*>tail(*P*) **in** *t*
3. **insert** *t* **to** occ(*j*), frq(*j*) := frq(*j*)+1
4. **end for**
5. **end for**

2. のループでは，あらかじめ各トランザクションのアイテムをソートしておくことにより，j>tail(P)を満たす j のみを効率良くたどることができる．図 2 に振り分けの実行例を示した．以下にデータベース圧縮の擬似コードを示す．

DatabaseReduction (*D*, occ)
1. *D'* := all transactions in occ
2. remove all items *i*≦tail(*P*), from all transactions *t* in *D'*
3. **Delivery** (occ)
4. remove all items *i* from all transactions *t* s.t. frq(*i*) < σ
5. unify all identical transactions into one
6. return (*D'*, {*t* | *t*∈*D'*})

このルーチンは P を含むトランザクションを集めたデータベースから，今後追加することのない，非頻出なア

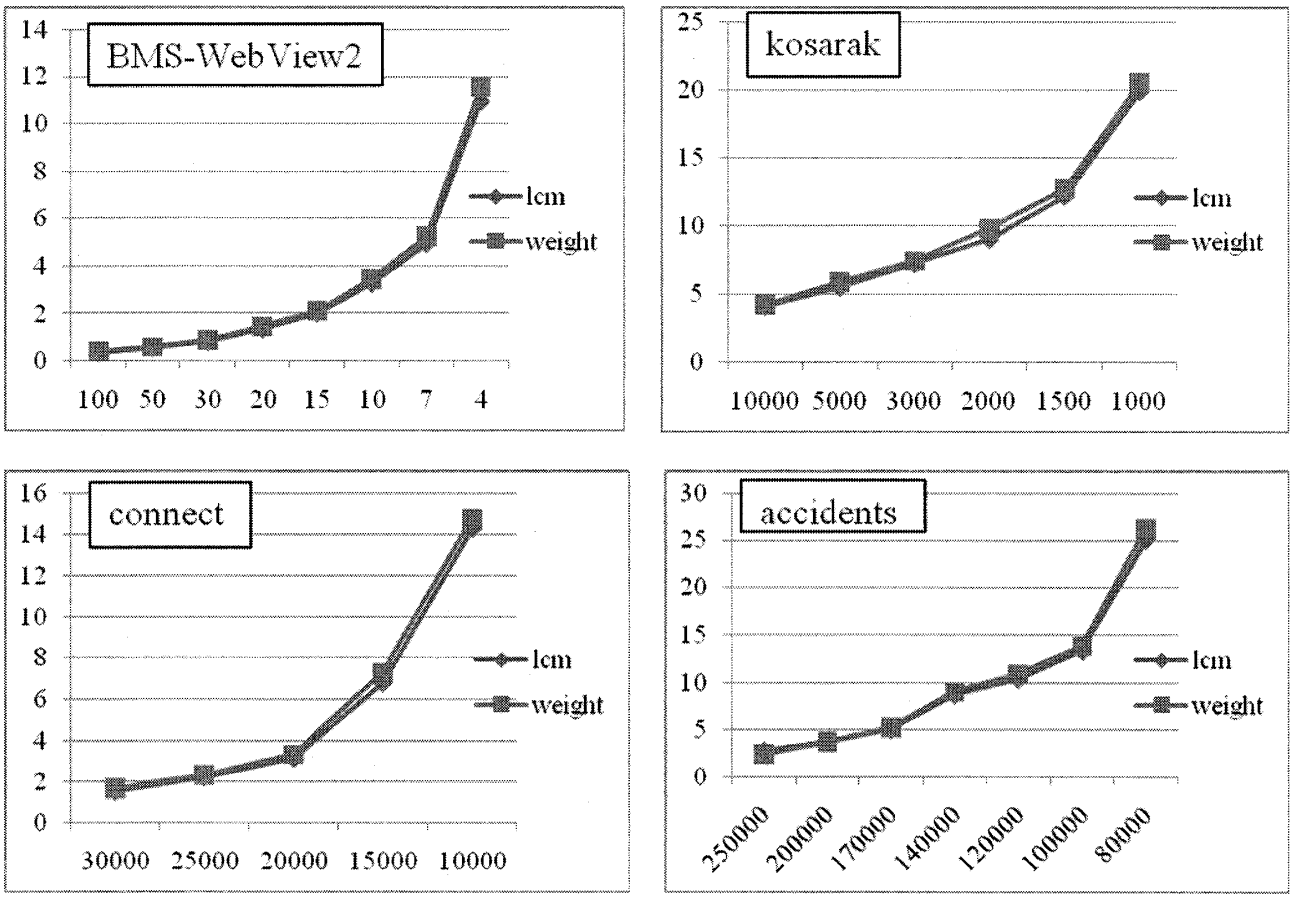

図３： 通常の LCM(lcm)と負重みを考慮した LCM(weight)での計算時間の比較

イテムと tail(P) より小さなアイテムを除去し，同一なトランザクションをまとめて得られるデータベース D' を計算して，occ を更新するルーチンである．2. で tail(P)より大きくないアイテムを取り除き，4. で P+i が頻出でないアイテム i を取り除き，その結果同一となったトランザクションを 5 で一つにまとめている．まとめたトランザクションには，まとめた個数を重みの形で記憶し，frq(i) を計算する際にはこの値を参照することとする．**DatabaseReduction** の実行例も図 2 に示した．

次に，負の重みが付けたトランザクションを扱う場合に対する変更点を考えよう．まず，トランザクションの重みに対応するためには，frq(P) を，P を含むトランザクションの個数で定義するのではなく，P を含むトランザクションの重み和で定義し直すことが必要である．これに伴い，**Delivery** の 3. での frq の計算を，$frq(j) := frq(j) + w(t)$ とする必要がある．また，**DatabaseReduction** では，同一なトランザクションの数を重みにするのではなく，同一なトランザクションの重み合計を新たな重みとする．

負の重みを考慮すると，単純な枝刈りが無効となることを前で説明した．その問題を回避するため，アイテム集合に対して正頻出度 $frq^+(P)$ を定義する．$frq^+(P)$ は P を含むトランザクションの中で，正の重みを持つもののみの重みを合計したものである．このとき，以下の式が成り立つ．

$$frq^+(P) \geqq frq(P)$$
$$frq^+(P+i) \leqq frq^+(P)$$

つまり，$frq^+(P)$ は頻出度の上界になっており，かつ frq^+ に関しては，関数値の単調性が成り立っている，つまりアイテムの追加により frq^+ が増加することはないのである．このことから，$frq^+(P)$ が σ より小さければ，任意のアイテム集合 S に対して，$frq(P \cup S) \leqq frq^+(P \cup S) < \sigma$ が成り立つことがわかり，よってこのような場合には P に関する探索を打ち切って良い．

frq^+ を効率良く計算するためには，**DatabaseReduction** に変更が必要である．負の重みを持つトランザクションと正の重みを持つトランザクションが同一である場合，両者の重みを足してして新たなデータベースを作ると，そのデータベースでは $frq^+(i)$の値に変化が生じてしまう．そのため，各トランザクション t に正重み pw(t) と通常の重み w(t) を持たせることとし，初期化の際には $pw(t) := \max(w(t),0)$ としておく．トランザクションをまとめる際には，pw(t) の合計を取ることで pw(t)も更新する．すると，常に $frq^+(P)$ は P を含むトランザクション t の pw(t) の値を合計したものになる．

frq^+ を使った枝刈りを行なうには，**LCM** の 5 での条件判定を $frq^+(i) \geqq \sigma$ にすれば良い．frq^+ を効率良く計算するには，**Delivery** の 3. を以下のように書き換える．

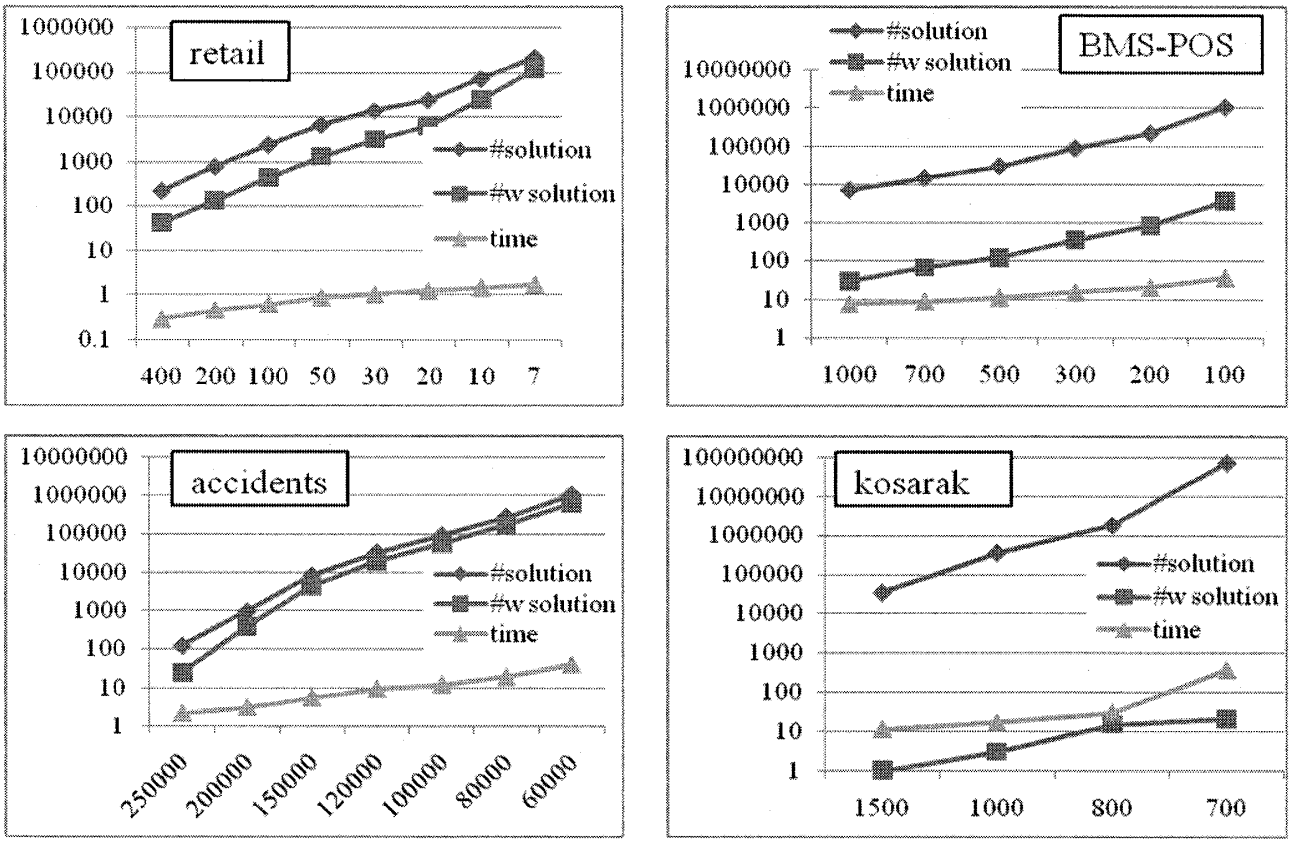

図 4： (#solution) 半分(retail, BMS-POS) (あるいは 9 割(accidents, kosarak))のトランザクションの重みを 0，残りを 1 (あるいは 9) とした場合の頻出集合数，(#w solution) 同じく重みを 0 でなく -1 とした場合の頻出集合数と (time) その計算時間

3. insert t to occ(j), frq(j) := frq(j)+w(t), $\text{frq}^+(j) := \text{frq}^+(j) + \max(w(t),0)$

また，変数の再利用のため，**LCM** の 6 に $\text{frq}^+(i) := 0$ を追加する．

これの処理は **LCM** の基本アルゴリズムに重みの計算を 2 種類追加しただけであるので，計算オーダーに変化は無く，根本的な時間のロスを伴うことはない．そのため，各反復の計算時間は通常の頻出集合発見と同程度の速度が期待できる．

3 計算実験

本節では LCM に前節の改良を加えたものに対して計算実験を行なったその結果を解説する．なお，同様の問題を解く適当な実装が見つからなかったため，ここでは主に同程度の難しさの頻出集合発見を解くのに必要となる時間との比較を中心に行なった．

プログラムは C でかかれ，gcc でコンパイルされ，cygwin 上で実行された．実験に用いた PC はノートブック PC，CPU は Core2Duo U7500 1.06GHz，メモリは 1GB である．

実験に用いたデータは FIMI repository [4] よりダウンロードした．BMS-WebView2，retail は非常に疎なデータ，BMS-POS と kosarak は疎でかつトランザクションが多いデータ，connect はある種の規則を持って機械的に作られた密なデータ，accidents は非常に密でかつトランザクション数が大きい．各データの詳細を以下に記す．

	トランザクション数	アイテム数	トランザクションの平均の大きさ
BMS-WebView2	77512	3340	4.62
retail	88162	16470	10.3
kosarak	990002	41271	8.09
BMS-POS	515597	1657	6.53
connect	67557	130	43
accidents	340183	469	33.8

まず，通常の LCM と負重みに対応した LCM の計算時間の差を図 3 に示す．4 つのベンチマーク問題について，複数の最小サポートについて実験を行い，計算時間を示した．lcm が従来の LCM，weight が負重みに対応した LCM の計算時間である．重みは全て 1 としてある．横軸が最小サポート，縦軸が計算時間(秒)に対応する．いずれの場合でも計算時間に大きな差は見られず，今回の改良は計算の負荷をそれほど増やしていないことがわかる．

次に，負の重みを考慮した場合の解数の様子を調べた．図 4 は 4 つのデータセットにそれぞれに対し，いくつかの最小サポートに対する解の数と計算時間(秒)を示したものである．retail と BMS-POS に関しては，真ん中より前にあるトランザクションの重みを-1，残りのトランザクションの重みを 1 とした場合の解数を #w solution，その計算時間を time，重み-1 を 0 にした場合の解の数を #solution で示している．最後のものは，真ん中より後ろにあるトランザクションに対する通常の頻出集合の数に相当する．最小サポートが大きいと，両者の数に開きがあり，最小サポートが小さくなるにつれ差が縮まることがわかる．これは，自然な結果であろう．

負重みを許したとき，発見されるパターンの数と実際に探索したパターンの数は一致しなくなる．この場合，探索されるパターンは，正の重みを持つトランザクションのみからなるデータベースでの頻出集合（飽和集合列挙の場合は頻出飽和集合）である．コントラストパターン・エマージングパターン列挙においては，本稿で用いた枝刈り以外の枝刈りを効果的に用いることは難しく，実際に探索されるパターン数が問題の難しさの本質となる面がある．そのため，この正のトランザクションに限定したデータベースでの頻出集合数を問題の難しさの指標とするのが適当だと考えられる．この尺度では，LCM はほぼ最適な速度を達成している．エマージングパターンなどを列挙する他の手法においては，このような評価尺度の点にふれているものは少なく，パターン数や計算時間のみが報告されており，探索における処理の効率性や，問題の難しさと速度との関連がつかみにくいものが多い．

大きなデータ(kosarak)，密なデータ(accidents)に関しては，最小サポートが大きいものしか実用的な時間内では解けず，結果この設定のもとでは解が一つも見つからないという結果になった．そこで，重みの与え方を，最初の 1 割のトランザクションに重み 9 を与え，残りのトランザクションに重み-1 を与える，として実験を行なってみた．正の重みを持つトランザクションの数を減らすことで，顕著なものが出てきやすいようにしたのである．

今回のデータは顕著な差があるとは考えにくい状況で行なったため，負重みを用いた場合と用いない場合で，頻出集合の間に顕著な差を見ることはできなかったが，少なくとも計算時間の面では，巨大なデータにおいても実時間内に計算が終了できることがわかった．これは，実用上は十分な結果であると考えている．異なる問題を解いているために直接の比較は難しいが，emerging pattern を見つける他の実装[3]では，トランザクション数 10,000 のデータベースから十数個のパターンを見つけるために，同程度の PC で 20 秒の時間を費やしている．また[8]では，トランザクション数 1000 程度から 1000 程度のパターンを見つけるのに 900 秒程度の時間を費やしており，本稿の実装が 2 桁以上高速であろうことが予想される．

5 まとめ

データマイニングの基礎的な問題である頻出集合発見に対して，それぞれのトランザクションに重みがついている場合を考え，負の重みを許す場合でもある程度効率良く枝刈りを行なう，既存の手法との相性の良いアルゴリズムを開発した．またその高速な実装を開発し，Web 上にて公開すると共に，計算実験によりその有効性を示した．今後はこのような少々の工夫により有効に解け，かつ応用を広く持つような追加的機能の調査とアルゴリズムの開発，および LCM のツールとしての完成度を高めることが課題である．この実装がデータマイニングをはじめとする諸分野の研究に役立てば幸いである．

参考文献

[1] R. Agrawal, H. Mannila , R. Srikant , H. Toivonen and A. I. Verkamo, Fast Discovery of Association Rules, Advances in Knowledge Discovery and Data Mining, MIT Press, 307-328 (1996).

[2] R. J. Bayardo Jr., Efficiently Mining Long Patterns from Databases, Proceedings of SIGMOD'98, pp. 85-93 (1998).

[3] G. Dong and J. Li. Efficient mining of emerging patterns: discovering trends and differences. In Proc. 5th ACM SIGKDD international conference on Knowledge discovery and data mining (SIGKDD'99), pp. 43–52, (1999).

[3] B. Goethals, the FIMI Homepage, http://fimi.cs.helsinki.fi/ (2003).

[4] G. Grahne and J. Zhu, Efficiently Using Prefix-trees in Mining Frequent Itemsets, Proceeding of IEEE ICDM'03 Workshop FIMI'03, (Available as CEUR Workshop Proceedings Series, Vol. 90, http://ceur-ws.org/vol-90) (2003).

[5] G. Liu, H. Lu, J. X. Yu, W. Wei, and X. Xiao, AFOPT: An Efficient Implementation of Pattern Growth Approach, proceeding of IEEE ICDM'03 Workshop FIMI'03 (Available as CEUR Workshop proceedings Series, Vol. 90, http://ceur-ws.org/vol-90) (2003).

[6] Sebastian Nowozin, Koji Tsuda, Takeaki Uno, Taku Kudo, Gökhan and H. Bakir, Weighted Substructure Mining for Image Analysis. CVPR 2007 (2007).

[7] S. Orlando, C. Lucchese, P. Palmerini, R. Perego and F. Silvestri, kDCI: a Multi-Strategy Algorithm for Mining Frequent Sets, proceeding of IEEE ICDM'03 Workshop FIMI'03 (Available as CEUR Workshop proceedings Series, Vol. 90, http://ceur-ws.org/vol-90) (2003).

[9] X. Qian, J. Bailey, and C. Leckie , Mining Generalised Emerging Patterns, Lecture Notes in Computer Science 4304, 295-304 (2006).

[10] H. Saigo, T. Uno, K. Tsuda, Mining complex genotypic features for predicting HIV-1 drug resistance, Bioinformatics 23(18), pp. 2455-2462 (2007).

[11] T. Uno, T. Asai, Y. Uchida and H. Arimura, An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases, Lecture Notes in Artificial Intelligence 3245, pp. 16-31 (2004).

[12] T. Uno, M. Kiyomi, and H. Arimura, LCM ver.2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets, Proceeding of IEEE ICDM'03 Workshop FIMI'04 (Available as CEUR Workshop Proceedings Series) (2004)