頻出・飽和・極大頻出集合の効率的な列挙アルゴリズムとその実装

宇野 毅明 [1]　　清見 礼 [1]　　有村 博紀 [2]

[1] 国立情報学研究所, 〒 101-8430 東京都千代田区一ツ橋 2-1-2, Tel/FAX 03-4212-2544
e-mail:uno@nii.jp, masashi@grad.nii.ac.jp
[2] 北海道大学情報科学研究科, 〒 060-0814 札幌市北区北 14 条西 9,
Tel 011-706-7678, FAX: 011-706-7890, e-mail:arim@ist.hokudai.ac.jp

抄録: トランザクションデータベースの，一定数以上のトランザクションに含まれるアイテムの集合を頻出集合とよぶ．集合の意味で極大な頻出集合を極大頻出集合とよび，同一のトランザクション集合に含まれる頻出集合の中で極大なものを頻出飽和集合とよぶ．データベースからこれら頻出集合を全て見つけ出す問題は，データマイニング分野での基礎的な問題であり，多くの研究がある．実用上の観点から，高速な求解アルゴリズムの必要性も高い．本稿では，これら 3 つの頻出集合列挙問題に対して，高速なアルゴリズム LCM (Linear time Closed itemset Miner), LCMfreq, LCMmax を提案する．さらに，他のアルゴリズムとの比較計算機実験によって，これらアルゴリズムの有効性を示す.

Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets

Takeaki Uno[1]　　Masashi Kiyomi[1]　　Hiroki Arimura[2]

[1] National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, JAPAN, 101-8430,
e-mail:uno@nii.jp, masashi@grad.nii.ac.jp
[2] Information Science and Technology, Hokkaido University Kita 14-jo Nishi 9-chome, 060-0814
Sapporo, JAPAN, e-mail:arim@ist.hokudai.ac.jp

Abstract: For a transaction database, a frequent itemset is an itemset included in at least a specified number of transactions. A frequent itemset K is maximal if it is included in no other frequent itemset, and closed if it is included in no other itemset included in the same transactions as K. The problems of finding these frequent itemsets are fundamental in data mining, and from the applications, fast implementations for solving the problems are needed. In this paper, we propose efficient algorithms LCM (Linear time Closed itemset Miner), LCMfreq and LCMmax for these problems. We show the efficiency of our algorithms by computational experiments compared with existing algorithms.

1　はじめに

近年，データマイニングの分野で，データの中から頻出する部分構造を見つけ出す問題が脚光を浴びている．頻出する部分構造は，アソシエーションルール発見などに応用があるほか，データの特徴を捉えるためにも有効だからである．この種の問題の中で最も古くから研究されており，かつ最も重要な問題が，頻出集合列挙問題である．これは，トランザクションデータベースの，一定数以上のトランザクションに含まれる集合 (頻出集合) を全て発見する問題である．また，極大な頻出集合を列挙する問題，含まれるトランザクションが等しい集合の中の極大元 (飽和集合) を列挙する問題も研究されている．これらの問題に対しては，非常に多くのアルゴリズムが提案されている [1, 2, 3, 8, 9, 14, 18, 17, 20, 21]．実装の研究も多く，速度を比較するコンテスト, FIMI03 (Frequent Itemset Mining Implementations) も，昨年開催された [6]

近年，筆者らは極大 2 部クリークの列挙アルゴリズム [16, 19] を用いた頻出飽和集合の高速列挙アルゴリズム LCM (Linear time Closed itemset Miner) を提案した [18, 17]．このアルゴリズムは，理論計算量が既存のアルゴリズムより小さく，さらに計算実験でも，良い結果を出している [6]．筆者らは，このアルゴリズムを改良し，頻出集合，および極大頻出集合を列挙するアルゴリズムを構築したが，改良が不十分であり，特に極大頻出集合列挙においては，他のアルゴリズムよりも悪い結果となっている．

そこで本研究では，LCM アルゴリズムにさらなる改良を加え，より高速な実装を実現した．[17] の研究では，既存の方法に改良を加えたデータベース縮約法と，高速飽和性判定法を導入し，さらに今回は，極大頻出集合の列挙に対する新しい枝狩り法と，極大性の高速判定法を導入した．これら手法の有効性は，FIMI03 で優秀な成績を残したアルゴリズムとの計算機実験による比較で検証した．

2 記法と問題の導入

$E = \{1, ..., n\}$ を**アイテム**の集合，$\mathcal{T}$ を E 上の**トランザクション**の集合からなるデータベースとする．$\mathcal{T}$ の全てのトランザクションに含まれるアイテムの総数，つまりデータベースの大きさを $||\mathcal{T}||$ と表記する．E の部分集合 K に対して，K を含む $\mathcal{T}$ のトランザクションを**出現**と呼ぶ．K の出現の集合を $Occ(K) = \{F | K \subseteq F, F \in \mathcal{T}\}$ と定める．与えられた定数 α に対して，$|Occ(K)| \geq \alpha$ を満たす K は**頻出集合**とよばれる．α を**サポート**とよぶ．頻出集合 K が他の頻出集合に含まれないならば，K を**極大頻出集合**とよぶ．また，K が $Occ(K') = Occ(K)$ となるような K' を全て含むとき，K は**飽和集合**であるという．任意のアイテム集合 K に対して，それを含むような唯一の飽和集合 K' が存在し，かつ $K' = \bigcap_{T \in Occ(K)} T$ であることが知られている [12, 13]．そのような飽和集合 K' を K の**閉包**とよび，$clo(K)$ と表記する．

集合 K とアイテム $e \notin K$ に対し，$K \cup \{e\}$ が頻出集合であれば，e を**頻出アイテム**とよび，逆に頻出でなければ**非頻出**とよぶ．集合 K に対し，K に含まれるアイテムで添え字最大のものを K の**末尾**とよび，$tail(K)$ と表記する．K に含まれ，添え字が i より小さなアイテムの集合を $K(i)$ と表記する．

3 既存の列挙手法

頻出集合の任意の部分集合は，やはり頻出集合である．つまり集合が頻出である，という性質は単調である．そのため，頻出である限り空集合にアイテムを 1 つずつ加える，という方法で，任意の頻出集合を生成できる．既存の頻出集合列挙アルゴリズムは，大きく 2 種類に分かれるが，両者ともにこの性質を用いている．

1 つ目の方法は，apriori，あるいは level-by-level と呼ばれる方法である [1, 2]．大きさが k である頻出集合の集合を $\mathcal{D}_k$ と表記する．apriori は最初 $\mathcal{D}_0$, つまり $\{\emptyset\}$ を求め，そして $\mathcal{D}_{k-1}$ から $\mathcal{D}_k$ を求める，という作業を $k = 1$ から昇順に行う．$\mathcal{D}_k$ は $\mathcal{D}_{k-1}$ の頻出集合にアイテムを 1 つ加えて得られるため，$\mathcal{D}_{k-1}$ の頻出集合にアイテムを加えて得られる集合を全て生成し，頻出集合のみを集めると $\mathcal{D}_k$ が得られる．この操作を k を 1 つずつ大きくして行い, $\mathcal{D}_k$ が空となったら終了するのが，apriori の基本的な仕組みである．

apriori は幅優先探索型の探索を行うアルゴリズムである．空集合を出発地点とし，そこからハミング距離が短い (すなわちアイテム数が小さい) 頻出集合から順々に発見する．それに対して，バックトラック法は，深さ優先的な探索を行うアルゴリズムである [3, 18, 17]．バックトラック法は，再帰呼び出しを用いた反復的なアルゴリズムである．各反復は現在解を受け取り，それに各アイテム e を追加して集合 $K \cup \{e\}$ を生成し，それが頻出集合であれば，$K \cup \{e\}$ を渡して再帰呼び出しを行う．頻出であるという性質が単調であることから，この方法で全ての頻出集合を発見できる．バックトラック法では，解を重複して発見することを防ぐため，各反復で現在解 K の末尾 $tail(K)$ よりも添え字が大きいアイテムのみを追加する，という工夫を行っている．これは，分枝限定法的に探索を行い，辞書順で頻出集合を発見していると解釈できる．バックトラック法を実行すると，反復の再帰構造を表した計算木が得られる．ある反復 I に対し，この計算木上で子孫に対応する反復を，I の子孫とよぶ．

apriori 型のアルゴリズムは，解集合を保持するために，多くのメモリを消費する．しかし，バックトラック型のアルゴリズムは，現在解しか保持しないため，メモリの消費が少ない．また，解集合の管理に関する計算時間もかからないため，求解速度は一般に短かい．しかし，apriori 型のアルゴリズムには，既発見解を用いて頻出度計算を工夫できるという利点もある．本研究の LCM はバックトラック法アルゴリズムであるが，より高速な頻出度計算方法を用いているため，既発見解をメモリに保持せずとも高速な計算を実現している．

4 本研究での手法

本節では，LCM が用いている計算手法を，他のアルゴリズムの手法と比較して解説する．本研究で新たに追加した改良は，データベースの縮約，極大頻出集合の列挙法，極大性・飽和性の判定である．

4.1 データベースの保持

既存研究では，データベースの保存や保持には，FP-tree (Frequent Pattern tree) と呼ばれる prefix tree を用いるのが良いと言われている [8]．prefix tree を用いることで検索が高速になり，また，共有されている prefix を 1 つにまとめて保持できるため，メモリの節約ができる．また，FP-tree を構築することで，同一なトランザクションを検出できる．頻出度や極大性の計算では，同一なトランザクションを 1 つにまとめることで計算時間が短縮できるため，この点でも FP-tree は推奨されている．

しかし，本研究では FP-tree は用いず，単なる配列を用いている．LCM は，頻出度や極大性の判定時にデータベースの検索を必要としないため，検索を高速にする努力が必要ないからである．また，FP-tree を構築するためには，$O(||\mathcal{T}|| + |\mathcal{T}| \log |\mathcal{T}|)$ の時間が必要である．本研究の LCM は，各トランザクションを配列で保持し，データベース中のトランザクションをアイテムの辞書順でソートすることで同一なトランザクションを発見している．辞書順ソートは Radix sort によりデータベースサイズの線形時間で行えるため，LCM のデータベース構築時間は $O(||\mathcal{T}||)$ である．一般に，データマイニングで扱われるデータベースは，疎であり，かつトランザクション数が大きい．そのため，$||\mathcal{T}||$ よりも $|\mathcal{T}| \log |\mathcal{T}|$ が大きくなることが多く，LCM に利がある．FP-tree が用いる 2 分木構造は計算量の係数が大きいため，この点においても FP-tree を用いない LCM が有利である．

既存研究ではその他にも，非頻出なアイテムをデータベースから取り除いてデータベースを縮小するという方法が良く使われている．LCM は，さらに「全てのトランザクションに含まれるアイテムを取り除く (フラグを立てて，必ず入っているとみなす)」という操作を加えた縮約を行っている．既存研究で，特にサポートが大きな場合，データベースの縮約が高速化に大きく貢献することが知られている．

4.2 頻出度計算

一般に，頻出集合列挙アルゴリズムにおいて，最も計算コストがかかる部分は新たに生成したアイテム集合の頻出度の計算である．アイテム集合 K に対して，$|Occ(K)|$ の計算を単純に行うと，基本的にデータベースを全てスキャンする必要がある．そのため，頻出集合列挙アルゴリズムの多くは，K の部分集合の出現から，$Occ(K)$ を効率良く計算することで，高速な頻出度計算を実現している．$K = K_1 \cup K_2$ であれば，$Occ(K) = Occ(K_1) \cap Occ(K_2)$ が成り立つ．2 つの集合の共通部分は，それぞれの集合を添え字順でソートしておけば，線形時間で計算できる．そのため，$Occ(K)$ は $\mathrm{O}(|Occ(K_1)| + |Occ(K_2)|)$ 時間で計算できる．この方法は，**down project** とよばれている．さらに，$|Occ(K_1)| < \alpha$ か $|Occ(K_2)| < \alpha$ であることが確認できれば，その時点で K は頻出集合でないと結論づけられる．

バックトラック型のアルゴリズムでは，各反復で現在解 K に各アイテム e を追加し，その頻出度を計算する．追加するアイテムの集合を H とすれば，計算時間は $O(\sum_{e \in H}(|Occ(K)| + |Occ(\{e\})|))$ となる．apriori 型のアルゴリズムは，大きさ k の頻出集合を全て保持しているため，$K_1 \cup K_2 = K \cup \{e\}$ であり，かつ $|Occ(K_1)| + |Occ(K_2)|$ が小さくなるような K_1, K_2 をうまく選ぶことにより，計算時間を $|Occ(K \cup \{e\})|$ に近づけることができる．また，$K \cup \{e\}$ が頻出集合でない場合，共通部分をとらずに非頻出と結論付けられる可能性があることも利点である．

また，データベースをビットマップで持つことにより，共通部分をとる演算を高速に行う，という方法もある [5]．ビットマップで保持した集合の共通部分をとるには，$\mathrm{O}(|\mathcal{T}|)$ 時間かかるのであるが，例えば 32 ビットコンピュータであれば，32 個のアイテムについて同時に共通部分をとれるため，単純に考えて 32 倍程度の高速化が見込まれる．しかし，データベースが疎な場合には効果が弱いこと，「同一のトランザクションを 1 つにまとめる」縮約操作との相性が悪い点が難点である．FIMI03 での計算実験からは，縮約操作の方に分があるように思われる．

LCM は，K に各アイテム $e \in H$ を追加して $Occ(K \cup \{e\})$ を計算する際に，これらの方法とはまったく異なる**出現配布法**を用いている．まず，追加する各アイテム $e \in H$ に対して，空のバケツを用意する．次に，$Occ(K)$ の各出現 T に対して，$T \cap H$ の各アイテム e のバケツに T を入れる，という作業を行う．操作終了後，アイテム e のバケツは $Occ(K \cup \{e\})$ と等しくなる．LCM はバックトラック的に探索を行うため，$H = \{tail(K) + 1, ..., |E|\}$ であり，$T \cap H$ のアイテムは T のアイテムを添え字降順でたどることで $\mathrm{O}(|T \cap H|)$ 時間で見つけることができる．そのため，H の全てのアイテムに対する計算時間は，$\mathrm{O}(\sum_{T \in Occ(K)} |T \cap H|) = \mathrm{O}(|Occ(K)| + \sum_{e \in H} |Occ(K \cup \{e\})|)$ となり，down project よりも計算量が小さくなり，高速化が期待できる．

apriori 型のアルゴリズムは，$K \cup \{e\}$ が非頻出であることを非常に短時間で結論付けられる可能性を持っているため，非頻出なアイテム e に関する頻出度計算，$\mathrm{O}(|Occ(K \cup \{e\})|)$ の分だけ計算時間が高速になる可能性がある．LCM は，これら非頻出なアイテムに関わる計算時間を短縮するため，アイテム e の添え字を，$\{e\}$ の頻出度の昇順でソートしている．この添え字のソートにより，非頻出なアイテムに関する計算時間を，全体の計算時間の 1/4 程度に減らすことに成功した．よって，apriori のような工夫を

しても，計算スピードは高々1.4 倍程度にしかならず，むしろ複雑な計算を行うことによる計算時間のロスが大きくなると考えられる．

いくつかのアルゴリズムは，各反復で頻出度計算のためのデータベースを縮約し，計算時間を短縮する工夫を行っている．ある反復で現在解に追加されるアイテムの集合を H とすると，この反復から再帰呼び出しされた反復で追加されるアイテムは，必ず H に含まれる．よって，データベースのアイテムを H に制限できる．その上で，K を含まないトランザクション，非頻出なアイテムを取り除き，同一なトランザクションを１つにすると，各反復でデータベースが劇的に小さくなる．そのため，列挙アルゴリズムの再帰構造の下のレベル，つまり多くの反復を抱える部分での計算時間が短縮される．この方法を**反復型データベース縮約法**とよぶ．

再帰呼び出しを行って，何回か反復型データベース縮約法を用いると，データベース全体が CPU キャッシュ上に収まるほど小さくなる．そのため，深さがある程度深い反復の計算速度は大きく増加し，縮約によるデータベースの縮小割合よりも大きな速度向上が行われる．

4.3 超立方体分解法

LCM は，頻出集合列挙を行う際に，いくつかの頻出集合をまとめて発見する，という手法を用いている．まとめて発見される頻出集合が，アイテム集合の 01 束上で超立方体を構成するため，この手法を**超立方体分解法**とよぶ．頻出集合 K に対し，$tail(K)$ より添え字が大きく，かつ $Occ(K) = Occ(K \cup \{e\})$ となるようなアイテム e の集合を $H(K)$ とする．このとき，$H(K)$ の任意の部分集合 H に対して，$Occ(K \cup H) = Occ(K)$ が成り立ち，よって $H \cup K$ は頻出集合となる．LCM はこの性質を利用し，K に関する反復で，$K \cup H(K)$ に含まれ，かつ K を含むようなアイテム集合を一度に全て出力する．これにより，およそ $2^{|H(K)|}$ 回の頻出度計算を省略できる．

$H(K)$ に含まれるアイテムに関して再帰呼び出しを行うと，解を重複して出力してしまうため，超立方体分解法では $H(K)$ のアイテムを追加しての再帰呼び出しは行わない．代わりに，他の再帰呼び出しに $H(K)$ を入力として渡し，再帰呼び出し中の各反復で，$H(K)$ の各部分集合を加えた解を出力する．任意のアイテム集合 H' と $H(K)$ の部分集合 H に対して，$Occ(K \cup H \cup H') = Occ(K \cup H')$ が成り立つため，これらの反復で $H(K)$ の部分集合を加えて出力されるアイテム集合も，全て頻出集合となる．以下に，超立方体分割法の詳細を示す．

ALGORITHM Hypercube_decomposition (K:current solution, S:itemset)
$S := S \cup H(K)$
Output all itemsets including K and included in $K \cup S$
For each item $e \in E, e > tail(K)$ **do**
If $|Occ(K \cup \{e\})| \geq \alpha$ **then**
bf Call Hypercube_decomposition $(K \cup \{e\}, S)$
End for

4.4 Prefix 保存飽和拡張

既存の頻出飽和集合の列挙アルゴリズムは，頻出集合の列挙を基礎とするものが多い．つまり，頻出集合を列挙し，その中で飽和集合となっているもののみを出力する，というものである．この方法は，頻出集合数と頻出飽和集合数に大きな違いがない場合は有効であるが，両者の差が大きいと，飽和集合とならない頻出集合を大量に生成することになり，効率が悪くなる．そのためにいくつかのヒューリスティックな枝狩り法が提案されており，計算速度の向上に貢献している．しかし，枝狩りは完全ではないため，計算量は頻出飽和集合数の線形とはならず，解数の増加に伴い，計算時間は線形より真に大きな割合で増加する可能性がある．

LCM は，飽和集合の列挙に **prefix 保存飽和拡張**という手法を用いている．飽和集合 K に対して，その**飽和末尾アイテム** $clo\_tail(K)$ を $clo(K(i)) = K$ が成り立つような添え字最小のアイテムとする．$clo\_tail(K)$ は必ず K に含まれる．このとき，K' が K の Prefix 保存飽和拡張であるとは，あるアイテム $e > clo\_tail(K)$ に対して $K' = clo(K \cup \{e\}), K'(e-1) = K(e-1)$ が成り立つことをいう．ただし，$e-1$ は e より添え字が 1 小さいアイテムとする．$Occ(K') \neq \mathcal{T}$ である任意の飽和集合 K' は，K' よりも頻出度が真に大きい飽和集合 K の prefix 保存飽和拡張となっており，かつそのような K は唯一的である．そのため，$Occ(K_0) = \mathcal{T}$ となる頻出飽和集合 K_0 から出発し，再帰的に prefix 保存飽和拡張を生成することにより，全ての頻出飽和集合を深さ優先的に発見できる．

prefix 保存飽和拡張を用いた飽和集合列挙法の計算量は，頻出飽和集合数に対して線形である．そのため，計算時間は解数の増加に対して，線形より大きく増加することはないことが理論的に保証される．prefix 保存飽和拡張の詳細については [17] を参照されたい．

4.5 飽和性の判定

飽和集合を列挙するには，アイテム集合が飽和集合であるかどうか判定する必要がある．この操作を**飽和性の判定**とよぶ．集合 K の飽和性の判定には，2種類の方法がある．1つは K に各アイテムを追加し，頻出度を計算するものである．どのアイテムを追加しても，頻出度が真に小さくなるならば，またそのときに限り，K は飽和集合である．2つ目は，今までに発見した頻出集合を，出現集合が同一なものをグループ化してメモリに保存する方法である．各グループの中で極大なものが飽和集合になりうるので，極大でないものを破棄し，計算終了後に各グループに残った極大な集合が，頻出飽和集合である．

LCM は，prefix 保存飽和拡張を行う際に，閉包を計算している．閉包の計算方法は，飽和性を判定する1つ目の方法とほぼ同じで，現在解 K に対して，追加しても出現集合 $Occ(K)$ が変化しないようなアイテムを全て追加している．その際，データベースの縮約を行うことで計算時間の増大を防いでいる．LCMmax のある反復 I での現在解を K とする．LCM は頻出度計算のために，各トランザクションから不要なアイテムを取り除き，その上で同一なトランザクションを1つにまとめる，という作業を各反復で行っている．そして，縮約したデータベースを子反復に渡している．同様に，I も親から縮約されたデータベースを受け取る．このデータベースがトランザクション $S_1, ..., S_h$ で構成されているとしよう．また，トランザクション S_l は，もとのデータベースのトランザクション $T_1^l, ..., T_k^l$ がまとめられたものだとする．この反復で追加されるアイテムの集合を H とすると，任意の l に対して，$T_1^l \cap H = T_2^l \cap H =, ..., = T_k^l \cap H$ が成り立つ．

ここで，$S \in \{S_1, ..., S_h\}$ に対して，その**内共通部分** $In(S_l)$ を $In(S_l) = \bigcap_{T \in S} T$ で定める．K の閉包は $\bigcap_{T \in S, S \in \{S_1,...,S_h\}} T$ であり，$\bigcap_{T \in S, S \in \{S_1,...,S_h\}} T = \bigcap_{S \in \{S_1,...,S_h\}} In(S)$ を満たすので，あらかじめ内共通部分を各 $S_1, ..., S_h$ に対して計算しておくことで，高速に閉包の計算ができる．さらなるデータベースの縮約を行う際の共通部分の更新計算は，$\bar{S}_l$ の共通部分を求めることにより，高速に行える．

4.6 極大頻出集合の列挙法

既存の極大頻出飽和集合の列挙法の多くは，頻出集合列挙を基礎としている．深さ優先的，あるいは幅優先的に頻出集合を発見し，その中で極大であるもののみを出力する，というものである．ただし，多くの場合，頻出集合数に比べて極大頻出集合数ははるかに小さいため，非極大な頻出集合を多く探索することになる．そのため，非頻出集合の生成を回避，あるいは無駄な分枝を切除するよう，ヒューリスティックな枝狩りの方法が提案されている．

本稿の LCMmax は，バックトラック法で頻出集合を発見し，それが極大頻出集合であれば出力する，という手法を用いている．それに加え，既存の手法と同じく，ヒューリスティックを用いた枝狩りで高速化している．ある反復での現在解を K とし，この反復で K に追加されるアイテムの集合を H とする．また，K' を，K を含む極大頻出集合とする．このとき，H に含まれるアイテムの添え字を，K' のアイテムが $H \setminus K'$ のアイテムよりも大きな添え字を持つように付け直す．添え字順の変更は，列挙法の正当性に影響を与えないため，添え字を変更した後も，通常の再帰呼び出しで頻出集合を列挙できることを注意しておく．

e を $K' \cap H$ に含まれるアイテムとする．ここで $K \cup \{e\}$ に関する再帰呼び出しを考えると，この再帰呼び出し中に出力される頻出集合は，全て K' の部分集合である．なぜなら，子孫で追加されるアイテムは，必ず e より添え字が大きく，添え字の並び替えにより，それらのアイテムは全て K' に含まれるからである．

LCMmax の各反復では，まずアイテム e を1つ選び，$K \cup \{e\}$ に関する再帰呼び出しを行って，$K \cup \{e\}$ を含み，$|K \cap H|$ が最大となる集合を求め，それを K' としている．これにより，K を含む極大頻出集合を見つける手間を省くと同時に，より大きな頻出集合を見つけることで，計算時間短縮の効果を大きくしている．さらに LCMmax は，頻出飽和集合列挙を基礎とすることで，基本的な反復数を減少させている．これは，頻出集合数と頻出飽和集合数の差が大きいときに効果がある．

4.7 極大性の判定

極大頻出集合列挙では，発見した頻出集合が極大であるかどうか判定する必要がある．この判定を**極大性の判定** とよぶ．既存の研究では，極大性の判定は，すでに発見した解の中で極大なもののみをメモリに保持し，新しく発見した解をメモリに加えた際に，極大でなくなったものを破棄する，という方法で行われている．この操作を実現するには，「すでに発見した解の中で，新しい解を含むもの（および含まれるもの）を見つける」という操作が必要である．現在の技術では，通常の2分検索のように，解集合数の log のオーダーでこの操作を実行することはできない．いくつかのヒューリスティックな高速化方法があるが，解集合の大きさにある程度依存してしまうことには変わりがない．

LCMmax は，解集合を保持する代わりに，$K \cup \{e\}$ が頻出になるかどうかを全てのアイテム e について調べることで，極大性のチェックを行っている．その際，飽和性の判定と同様に，データベースの縮

約を行うことで計算時間の増大を防いでいる．LCMmax のある反復 I での現在解を K とし，I が受け取った縮約されたデータベースがトランザクション $S_1, ..., S_h$ で構成されているとしよう．また，トランザクション S_l は，もとのデータベースのトランザクション $T_1^l, ..., T_k^l$ がまとめられたものだとする．

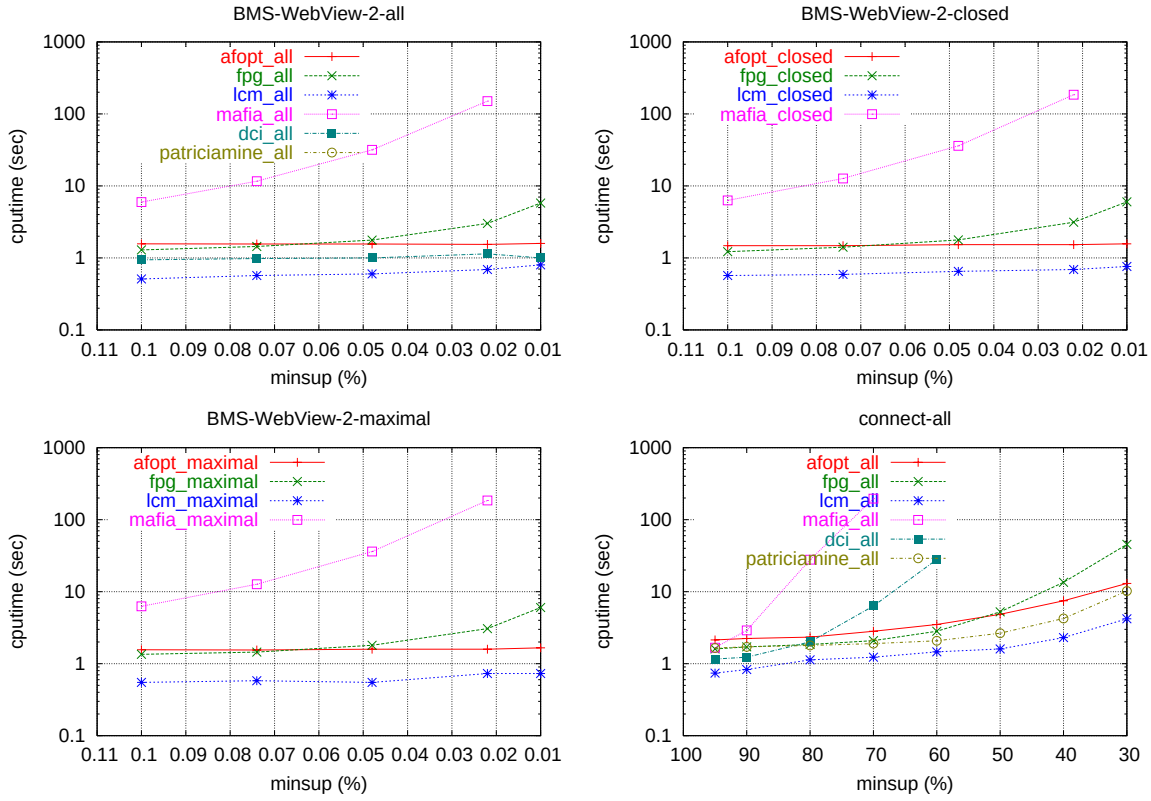
アイテム e と 縮約されたトランザクション S_l に対して，重み $w(e, S_l)$ を，$T_1^l, ..., T_k^l$ で e を含むものの数とする．すると，$K \cup \{e\}$ の頻出度は $\sum_{S \in \{S_1,...,S_h\}} w(e, S)$ で表わされる．このように，重みを使って頻出度の計算を行えば，縮約されたトランザクションに比例した大きさで，高速に頻出度を計算できる．この手法は，トランザクションデータベースを，各アイテムに重みがついたデータベースに縮約し，頻出度計算を高速化していると解釈できる．通常のデータベースの縮約と同様，非頻出なアイテムは重みのついたデータベースからも取り除けるため，さらにデータベースを縮約できる．

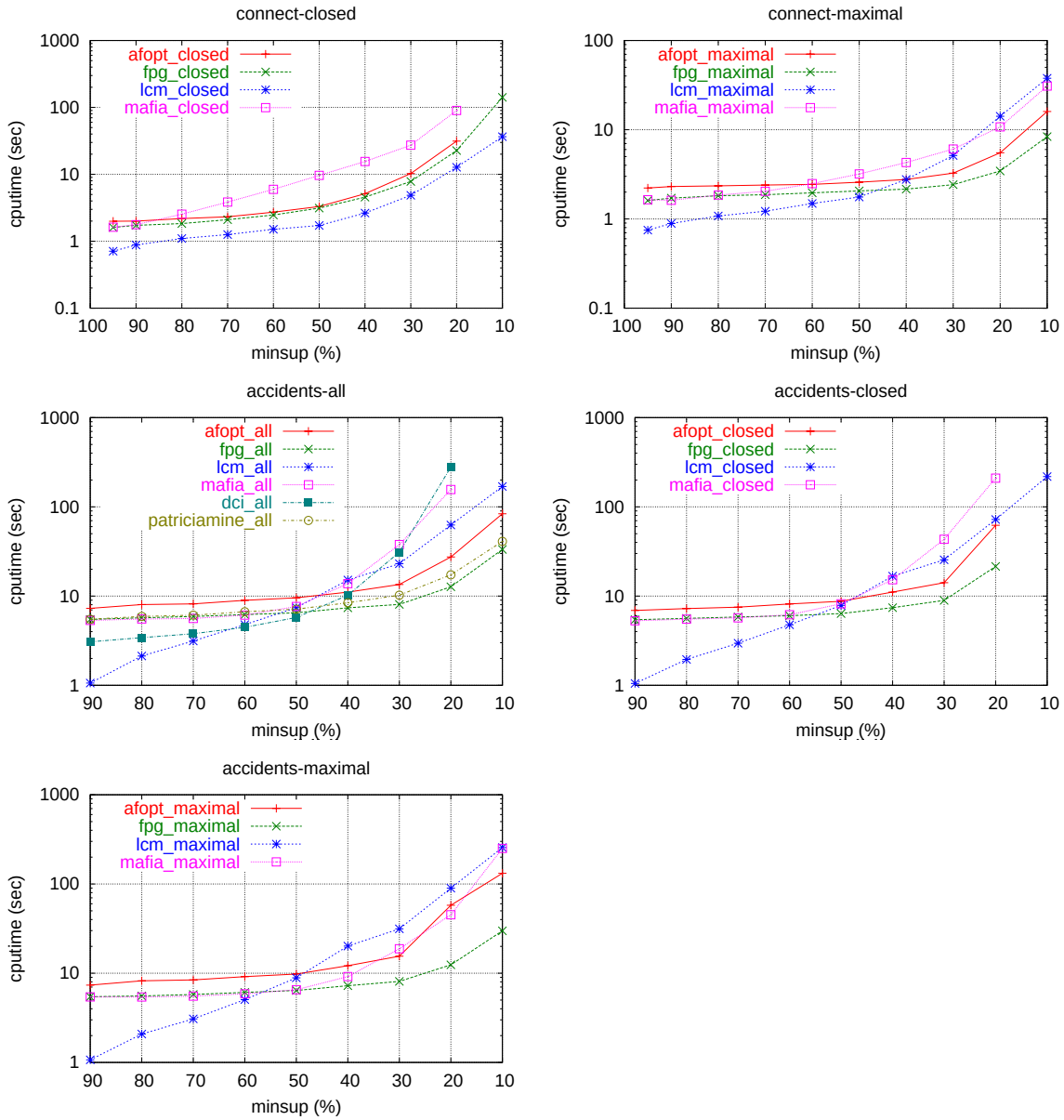
LCM での極大性の判定にかかる計算時間は，データベースの大きさに依存し，極大解の数には依存しない．そのため，極大解が少ないような問題では，既存の方法が有利であるが，解数が大きく，時間がかかる問題では，LCM が有利である．

5 計算実験

本節では，LCM アルゴリズムの計算機実験の結果を示す．使用機器は AMD athron XP 1600+, 256MB のメモリを搭載したノート PC，プログラム言語は C，OS は Linux である．実験データは FIMI03[6] で用いられた accidents (何らかの実データ), connect(学習理論でのベンチマーク問題), BMS-WebView2(web のデータ) の 3 種類を用いた．accidents はアイテム数が小さい割りにトランザクション数が大きく，非常に密なデータベースであり，逆に BMS-WebView2 は，アイテム数・トランザクション数がともに大きく，疎である．connect は両者の中間であるが，人工的な構造があり，頻出集合に規則性がある．

比較するアルゴリズムは，FIMI03 で優秀な成績を残した fpgrowth[7], afopt[10], MAFIA[5], kDCI[11], PATRICIAMINE[15] である．なお，kDCI は頻出集合列挙のみの実装である．それぞれの問題で，サポートを変化させて実験を行った．以下にその結果を示す．





多くの問題において，LCM アルゴリズムが短時間で計算を終了している．特に、サポートが大きい場合に高速なのは，データベース縮約にかかる時間が FP-tree を用いたアルゴリズムより高速だからである．また，サポートが小さなところでは、超立方体分解，prefix 保存飽和拡張が有効に働き，計算時間を短縮している．accidents のサポートを小さくした場合のみ，他のアルゴリズムより時間がかかっているが，これは解数がトランザクション数に比べてそれほど大きくないため，既存の手法のほうが極大性の判定に時間がかからないためと，accidents では FP-tree の圧縮効率が良く，各反復で入力するデータベースがおよそ 1/2-1/3 程度に圧縮されているためと考えられる．メモリの使用量に関しても，LCM アルゴリズムは他のアルゴリズムと異なり，解数が増えても，メモリの使用量は増加しない．

6 まとめ

本稿では，prefix 保存飽和拡張を用いた飽和集合列挙アルゴリズム LCM に改良を加え，頻出集合・極大頻出集合を列挙するアルゴリズム LCMfrqe, LCMmax とその高速実装を提案した．計算実験の結果，多くの問題において，本研究で行った改良 LCM の実装が他のアルゴリズムよりも良い結果を出した．しかし，極大頻出集合列挙で，データベースの大きさに対して解数が小さい場合，既存のアルゴリズムよりも極大性の判定に計算時間を費やすことがあった．より高速な極大性判定アルゴリズムの開発を行うこと，また，解数が少ない場合には既存の極大性判定手法を用いることなどによって，さらなる高速化が行える可能性がある．

7 謝辞

この研究は，国立情報学研究所共同研究費の補助を受けた．

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *In Proceedings of VLDB '94,* pp. 487–499, 1994.

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

[3] R. J. Bayardo Jr., "Efficiently Mining Long Patterns from Databases", In *Proc. SIGMOD'98*, pp. 85–93, 1998.

[4] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino, "On the Complexity of Generating Maximal Frequent and Minimal Infrequent Sets," *STACS 2002*, pp. 133-141, 2002.

[5] D. Burdick, M. Calimlim, J. Gehrke, "MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases," In *Proc. ICDE 2001*, pp. 443-452, 2001.

[6] B. Goethals, *the FIMI'03 Homepage*, `http://fimi.cs.helsinki.fi/`, 2003.

[7] G. Grahne and J. Zhu, "Efficiently Using Prefix-trees in Mining Frequent Itemsets," In *Proc. IEEE ICDM'03 Workshop FIMI'03*, 2003. (Available as CEUR Workshop Proc. series, Vol. 90, http://ceur-ws.org/vol-90)

[8] J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns without Candidate Generation," *SIGMOD Conference 2000*, pp. 1-12, 2000

[9] R. Kohavi, C. E. Brodley, B. Frasca, L. Mason and Z. Zheng, "KDD-Cup 2000 Organizers' Report: Peeling the Onion," *SIGKDD Explorations,* 2(2), pp. 86-98, 2000.

[10] Guimei Liu, Hongjun Lu, Jeffrey Xu Yu, Wang Wei, and Xiangye Xiao, "AFOPT: An Efficient Implementation of Pattern Growth Approach," In *Proc. IEEE ICDM'03 Workshop FIMI'03*, 2003. (Available as CEUR Workshop Proc. series, Vol. 90, http://ceur-ws.org/vol-90)

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

[12] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, *Efficient Mining of Association Rules Using Closed Itemset Lattices*, *Inform. Syst.*, 24(1), 25–46, 1999.

[13] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, *Discovering Frequent Closed Itemsets for Association Rules*, In *Proc. ICDT'99*, 398-416, 1999.

[14] J. Pei, J. Han, R. Mao, "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets," *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery 2000*, pp. 21-30, 2000.

[15] A. Pietracaprina and D. Zandolin, "Mining Frequent Itemsets using Patricia Tries," In *Proc. IEEE ICDM'03 Workshop FIMI'03*, 2003. (Available as CEUR Workshop Proc. series, Vol. 90, http://ceur-ws.org/vol-90)

[16] S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, "A New Algorithm for Generating All the Maximum Independent Sets," *SIAM Journal on Computing,* Vol. 6, pp. 505–517, 1977.

[17] T. Uno, T. Asai, Y. Uchida, H. Arimura, "An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases," to appear in Proc. of Discovery Science 2004, 2004.

[18] T. Uno, T. Asai, Y. Uchida, H. Arimura, "LCM: An Efficient Algorithm for Enumerating Frequent Closed Item Sets," In *Proc. IEEE ICDM'03 Workshop FIMI'03*, 2003. (Available as CEUR Workshop Proc. series, Vol. 90, http://ceur-ws.org/vol-90)

[19] 宇野 毅明, "大規模グラフに対する高速クリーク列挙アルゴリズム," 電気通信学会コンピュテーション研究会, 京都大学, 2003.

[20] M. J. Zaki, C. Hsiao, "CHARM: An Efficient Algorithm for Closed Itemset Mining," *2nd SIAM International Conference on Data Mining (SDM'02)*, pp. 457-473, 2002.

[21] Z. Zheng, R. Kohavi and L. Mason, "Real World Performance of Association Rule Algorithms," *KDD 2001*, pp. 401-406, 2000.