疑似頻出アイテム集合の多項式遅延列挙アルゴリズム

Efficiently Mining Pseudo Frequent Itemsets in Polynomial Time Delay

宇野 毅明 [1*]　有村 博紀 [2]
Takeaki Uno[1] and Hiroki Arimura[2]

[1] 国立情報学研究所
[1] National Institute of Informatics
[2] 北海道大学情報科学研究科
[2] Graduate School of Information Science and Technology, Hokkaido University

Abstract: Mining frequently appearing patterns in a database is a basic problem in recent informatics, especially in data mining. Particularly, when the input database is a collection of subsets of an itemset, the problem is called frequent itemset mining problem, and has been extensively studied. In the real world use, one of popular difficulties of frequent itemset mining is that data is often incorrect, or missing some parts. It causes that some records which should/may include a pattern become not including the pattern. Thus, in real world problems, it is valuable to use an ambiguous inclusion relation and find patterns which is "almost" included in many records. However, from the difficulty on the computation, such kind of problems have not been actively studied. In this paper, we use an alternative inclusion relation in which we consider an itemset P is included in an itemset T if at most k items of P are not included in T, i.e., $|P \setminus T| \leq k$. We address the problem of enumerating frequent itemset under this inclusion and propose an efficient polynomial delay polynomial space algorithm. To skip so many small non-valuable frequent itemsets, we propose an algorithm for directly enumerating frequent itemsets of a certain size.

1　はじめに

与えられたデータベースの多くの項目に含まれるパターンを発見する問題は，頻出パターン発見問題とよばれ，データマイニングの中心的な問題である．特に，データベースの各項目がアイテム集合の部分集合であり，かつパターンとして同じくアイテム集合の部分集合を用いた問題は，頻出集合発見問題とよばれ，多くの研究がなされてきた．正確には，各項目がアイテム集合 I の部分集合となっているようなデータベース D に対して，アイテム集合の部分集合 P の**頻出度**を，P を含む D の項目数で定義する．頻出度が**最小サポート**とよばれる閾値 σ 以上である部分集合を**頻出集合**と定義し，入力したデータベース D と閾値 σ に対して，D の全ての頻出集合を見つける問題を**頻出集合発見問題**，あるいは**頻出集合列挙問題** という [1, 4, 10, 12, 11].

頻出パターン発見は，データマイニングを始めとする巨大なデータを解析する問題でよく使われている．人間がざっと眺めただけでは特徴が捉えられないような巨大なデータベースから，頻出するパターンを見つけることで特徴を抽出するというアプローチが行えるからである．そのため，頻出パターン発見は現実問題に対して実際に使われることが多い．しかし，現実問題に適用する場合，いくつかの問題が出てくる．その１つは，データベースがデータの欠損や不完全性を持つことが多く，本来特徴として出てくるであろうパターンが，データの欠損のために頻出でなくなる，あるいは頻出度が低下して，重要でないとみなされてしまう可能性があることである．このような問題に対処するためには，パターンと項目の包含関係をあいまいにし，完全には含まれていない場合でもほとんどの部分が含まれていたら含まれる，とするような包含関係が必要となる．

このような包含関係にあいまいさを導入したパターンマイニングとして，アイテムがトランザクションに含まれないということをエラーととらえ，エラーの数が少ないようなパターンを見つけるというものがある。このようなパターンは、ときにフォールトトレラント頻出アイテム集合とよばれている [5, 6, 7, 8, 14]．包含

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

関係をあいまいにする例として、アイテム集合 P のうち、閾値 θ 以上の割合がトランザクションに含まれるときに、包含関係が成り立つ、つまり $|P \cap T|/|P| \geq \theta$ となる場合に包含関係が成り立つとするものがある [14]。これはごく自然なあいまいさの定義であるが、頻出集合の族が単調性を満たさなくなり、アプリオリなどの既存手法で効率良く列挙できなくなる。

また、エラーが少ないようなアイテム集合、トランザクション集合の組を直接見つける、あるいは先の包含関係にさらに制約を入れてより良いパターンを見つける、という研究も行われている。この問題も同様に単調性を持たず、完全に列挙することは簡単でない。そのため、これらの問題に対するアルゴリズムは、ヒューリスティックを用いて探索を行う、列挙の完全性を保証しないものが多い。一方，系列パターンマイニング，文字列マイニングの分野では，あいまい検索の技術を用いて，エラーを許したマッチングでパターンの出現を定義するというモデルが考えられており，いくつかのアルゴリズムが提案されている [13, 9].

ほとんどが含まれる，という条件のモデル化はいくつか考えられるが，ここでは，定数 k を設定し，パターン P のアイテムのうち T に含まれないものが k 個以下であるとき，P は T に含まれると定義する．最小サポートが大きい，例えば 90%となるような場合，この問題は，アイテム集合 $\{i_1, \ldots, i_h\}$ で，データベースのほとんどの項目に，この中の $h-k$ 個が含まれるようなものを見つける，という問題に相当する．これは，ある種のデータベースの特徴付けと考えられ，機械学習などの分野に応用できる．

本稿では，与えられたトランザクションデータベース，最小サポート，定数 k に対して，この包含関係の意味で頻出である集合を全て見つける問題を扱い，この問題を解く多項式時間遅延多項式空間アルゴリズムを提案する．著者の知る限り，この問題に対する多項式時間アルゴリズムはこれが初めてである．

ただし，実際の問題では，大きさが小さく，興味のないアイテム部分集合のうち，非常に多くがこの包含関係の意味で頻出集合となってしまうことがある．この問題は，この包含関係を用いた場合避けて通れないものである．そこで，本稿では，大きさ l の，この意味での頻出集合をより直接的に見つける問題に対する効率的なアルゴリズムも，合わせて提案する．

本論文は以下のように構成される．まず，次節で記法と問題を定義し，3 節で基本的なアルゴリズムを解説する．また，4 節で大きさ l の k 擬似頻出集合を直接的に見つけるアルゴリズムを解説し，5 節で本稿をまとめる．

2 記法

データベース $\mathcal{D}$ の各項目がアイテムの集合 $I = \{1, ..., n\}$ の部分集合となっているとき，D を**トランザクションデータベース**，$\mathcal{D}$ の各項目を**トランザクション**とよぶ[1]．$|\mathcal{D}|$ で $\mathcal{D}$ のトランザクション数を表記する．$||\mathcal{D}||$ を $\mathcal{D}$ のトランザクションの大きさの総和とトランザクション数を足したもの，つまり $\sum_{T \in \mathcal{D}}(|T|+1)$ とし，**データベース $\mathcal{D}$ の大きさ**とよぶ．データベースの大きさにトランザクション数を加えている理由は、大きさが 0 のトランザクションを処理する時間を $O(1)$ とするためである．以下，特に断らない限り，$\mathcal{D}$ は固定されているとする．

I の部分集合を**パターン**とよぶ．パターン P の中で最大のアイテムを P の**末尾**とよび，$tail(P)$ と表記する．$\mathcal{D}$ のトランザクションで P を含むものを P の**出現**とよぶ．P の出現の集合を $Occ(P)$ と表記する．P の頻出度 $frq(P)$ を $|Occ(P)|$ で定義する．$\mathcal{D}$ と 閾値 σ に対して，頻出度が σ と同じか，あるいはそれ以上であるパターンを**頻出集合**とよぶ．頻出度はしばしば**サポート**とよばれ，このとき，閾値は**最小サポート**とよばれる．トランザクションデータベース $\mathcal{D}$ と閾値 σ を入力して，$\mathcal{D}$ の頻出集合を全て見つけ出す問題を**頻出集合列挙問題**とよぶ[2]

定数 k に対して，トランザクション T とパターン P の k **擬似包含関係**を，$|P \setminus T| \leq k$ が成り立つことと定義し，$P \subseteq_k T$ と表記する．k 擬似包含関係を総称して**擬似包含関係**とよぶ．パターン P に対して，$P \subseteq_k T$ が成り立つトランザクションを k **擬似出現**とよび，k 擬似出現の集合を $Occ_{\leq k}(P)$ と表記する．また，$|P \setminus T| = k$ が成り立つようなトランザクションの集合を $Occ_{=k}(P)$ と表記する．$Occ(P), Occ_{=k}(P) \subseteq Occ_{\leq k}(P)$ である．$Occ_{\leq k}(P)$ の大きさを k **擬似頻出度**とよび，$frq_k(P)$ と表記する．図 2 に例を示した．k 擬似頻出度が最小サポート σ と同じかそれより大きなパターンを k **擬似頻出集合**とよぶ．ここで，擬似頻出集合列挙問題を以下のように定義する．

擬似頻出集合列挙問題
入力: トランザクションデータベース $\mathcal{D}$，最小サポート σ, k
出力: $\mathcal{D}$ の全ての k **擬似頻出集合**

解を列挙するアルゴリズムが，入力の大きさと出力の大きさの和に対して多項式時間で終了するとき，そ

[1] 正確には，トランザクションは，項目を示す ID と I の部分集合の組で与えられるが，ここでは問題設定上 ID が意味を持たないため，省略した．

[2] この問題はしばしば頻出集合発見問題とよばれるが，発見という言葉は完全性，つまり解の全てを見つけ出すという条件が付いていない問題に使われることが多いため，ここでは，完全性を必要とする問題に使われる列挙という言葉を用いた．

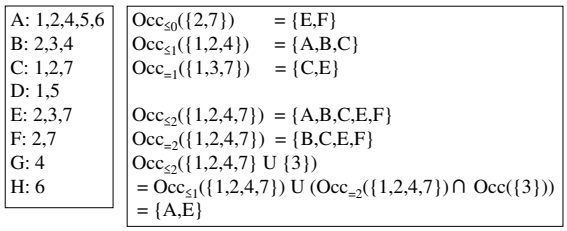

図 1: 擬似出現と，アイテムの追加によるその更新の例

のアルゴリズムは**出力多項式時間**とよばれる．特に，任意の 2 つの連続する出力の間にかかる計算時間が入力の大きさの多項式時間で抑えられるとき，**多項式時間遅延**とよばれる．多項式時間遅延であれば，計算時間は出力数に対して線形となり，出力数に対しては最適なアルゴリズムとなる．

3 基礎アルゴリズム

頻出集合列挙問題は，アルゴリズム理論的には，比較的取り扱いやすい問題である．その理由は，頻出集合の任意の部分集合は頻出である，という単調性が成り立つため，頻出集合にアイテムを追加する，という作業を再帰的に繰り返すだけで，任意の頻出集合を構築することが可能だからである．単純にアイテムの追加を行うと，同一の頻出集合を複数回生成してしまうが，これは末尾拡張を使うことで回避できる．アイテム集合 P に，P の末尾よりも大きなアイテムを追加して得られるアイテム集合を**末尾拡張**とよぶ．アイテム集合の生成を末尾拡張のみに限定することで，任意の頻出集合 P は，その末尾を除去して得られる頻出集合からのみ生成されることになり，重複は回避される．この末尾拡張を再帰的に行うバックトラックアルゴリズムにより，多項式時間遅延のアルゴリズムが比較的容易に構築できる．正確には，頻出集合1つあたり，入力したデータベースの大きさに対して線形の時間，つまり $O(||\mathcal{D}||)$ となる．メモリの使用量も最適であり，$O(||\mathcal{D}||)$ である．

しかし，現実問題で頻出集合を列挙する場合，入力するデータベースは巨大であることが多く，1つあたりに $||\mathcal{D}||$ に比例する時間をかけていては，現実的な時間内に計算が終了しない．そのため，現実問題での計算時間を減少させる技術が多く開発されてきた．その中でも有用な技術が，データベースの縮約である．

データベースから，$\{i\}$ が頻出集合とならないようなアイテム i，および全てのトランザクションに含まれるアイテム i を取り除き，その後同一となったトランザクションを1つにまとめるという作業をすると，データベースが小さくなる．さらに，アイテム集合を文字

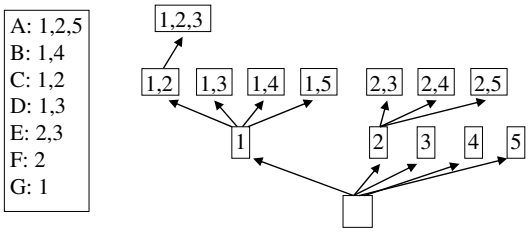

図 2: 最小サポート σ を 4 とした場合のバックトラック法の実行例

列とみなし， prefix tree，trie を使って共通する接頭辞の情報を共有することでデータ量を小さくできうる．このような操作を**データベース縮約**とよぶ．現実問題では，特に最小サポートが大きい場合，データベース縮約は大きな効果を発揮する．また，各反復で必要な限定されたデータベースに対して再帰的にこの操作を適用することで，各反復の計算時間を再帰的に短縮できる．この操作は**反復的データベース縮約**とよばれ，さらなる速度の向上が可能である．

末尾拡張による列挙と，関連するアルゴリズム技法は，k 擬似頻出集合列挙にも適用可能である．まず，単調性を保証するための以下の性質を見よう．

性質 1 任意の 2 つパターン $P, P', P \subseteq P'$ に対して，$Occ_{\leq k}(P') \subseteq Occ_{\leq k}(P)$ が成り立つ．

この性質より，k 擬似頻出集合の族は単調性を満たすことがわかる．単調性から，以下のバックトラックアルゴリズムを構築することが可能である．このアルゴリズムは，**バックトラック** $(\emptyset)$ を呼び出すことで，全ての k 擬似頻出集合を出力する．以下，最小サポートは，1 以上 $|\mathcal{D}|$ 以下であると仮定し，空集合は必ず k 擬似頻出集合であるとする．また，k 擬似頻出集合 P に対して，$CHD(P)$ を末尾拡張 $P \cup \{i\}$ が k 擬似頻出集合となるようなアイテム i の集合とする．

バックトラック (P)
1. P を出力する
2. $CHD(P)$ を計算する
3. **for each** $i \in CHD(P)$ **call バックトラック** $(P \cup \{i\})$

バックトラック法の実行例を図 3 に示した．このアルゴリズムの正当性は明らかとしてよいだろう．各反復では，k 擬似頻出集合を受け取り，それを出力し，その末尾拡張を全て生成し，生成した末尾拡張の中で k 擬似頻出集合となっているもののみについて再帰呼び出しを行う．そのため，このアルゴリズムは 1 反復で 1 つの k 擬似頻出集合を出力し，k 擬似頻出集合1つあたりの計算時間は 1 反復の計算時間と等しくなる．このアルゴリズムを単純に実装すると，step 2 で，P の各末尾

拡張の k 擬似頻出度を計算するところで $O(||\mathcal{D}||)$ の時間を消費し，そのため，1 反復の計算時間は $O(n||\mathcal{D}||)$ となる．これは，以下で述べる方法で短くできる．まず，以下の性質を確認しよう．

性質 2 トランザクション $T \in Occ_{=h}(P)$ が i を含むときは $T \in Occ_{=h}(P \cup \{i\})$ であり，i を含まないときは $T \in Occ_{=h+1}(P \cup \{i\})$ である．

この性質から，以下の性質が観察できる．

性質 3
(1) $Occ(P \cup \{i\}) = Occ(P) \cap Occ(\{i\})$
(2) $h \geq 1$ に対して，$Occ_{=h}(P \cup \{i\}) = (Occ_{=h}(P) \cap Occ(\{i\})) \cup (Occ_{=h-1}(P) \setminus Occ(\{i\}))$

性質 3 から，ある i に対する $Occ_{=0}(P \cup \{i\}), ..., Occ_{=k}(P \cup \{i\})$ は，$Occ_{=0}(P), ..., Occ_{=k}(P)$ を用いて $O(|Occ_{\leq k}(P \cup \{i\})| + |Occ(\{i\})|)$ 時間で計算できることがわかる．同じく性質 3 から，各 i に対して $Occ_{=k}(P) \cap Occ(\{i\})$ を計算することで，各 $P \cup \{i\}$ の k 擬似頻出度が計算できることがわかる．$i > tail(P)$ なる全ての i に対して $Occ_{=k}(P) \cap Occ(\{i\})$ を計算するには**振り分け** [2, 10, 12, 11] とよばれる手法が効率良い．

まず，各アイテム i に対して空のバケツを用意する．次に，$Occ_{=k}(P)$ の各トランザクション T に対して，T に含まれるアイテム $i > tail(P)$ それぞれに対して i のバケツに T を挿入する，という作業を行う．この作業を全ての T について行った後，バケツ i の中身は $Occ_{=k}(P \cup \{i\})$ となる．以下に，振り分けのアルゴリズムを記述する．このアルゴリズムは，引数としてトランザクションの集合 $\mathcal{S}$ と $tail(P)$ を渡すと，各 $i > tail(P)$ について $\mathcal{S} \cap Occ(\{i\})$ を計算して $bucket[i]$ に格納する．アルゴリズムの実行前に，各バケツ $bucket[i]$ は空集合に初期化されているとする．

振り分け $(\mathcal{S}, tail(P))$
1. **for each** $T \in \mathcal{S}$ **do**
2. **for each** $i \in T, i > tail(P)$ **do**
3. insert T into $bucket[i]$
4. **end for**
5. **end for**

振り分けの実行例を図 3 に示した．トランザクションの集合 $\mathcal{S}$ と添え字 h に対して，$\mathcal{S}_{>h} = \{T \cap \{h+1, \ldots, n\} \mid T \in \mathcal{S}\}$ とする．以後，各トランザクションは，データベースを入力した際にアイテムの昇順でソートされているとする．このソートは，全てのトランザクションを同時にバケツソートすることで $O(||\mathcal{D}|| + n)$ 時間で実行できる．

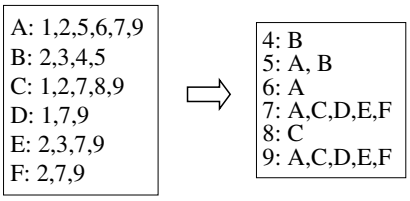

図 3: 振り分けの実行例

性質 4 アルゴリズム**振り分け** は，$O(||\mathcal{S}_{>tail(P)}||)$ 時間で各 $i > tail(P)$ について $\mathcal{S} \cap Occ(\{i\})$ を計算する．

性質 4 より，以下の補題を得る．

補題 1 パターン P に対して，k 擬似頻出度が非ゼロであるアイテム集合 $P \cup \{i\}, i > tail(P)$ の k 擬似頻出度を，$O(||Occ_{=k}(P)_{>tail(P)}||)$ 時間で計算できる．

この補題より，$CHD(P)$ は $O(||Occ_{=k}(P)_{>tail(P)}||) = O(||\mathcal{D}||)$ 時間で計算できることがわかった．バケツに必要なメモリは，以下の性質からデータベースの線形サイズで抑えられる．

性質 5 任意のトランザクション集合 $\mathcal{S} \subseteq \mathcal{D}$ とアイテム i に対して，振り分け実行後のアイテム i のバケツの中にあるトランザクションの数は $|Occ(\{i\})|$ を超えない

アルゴリズム**バックトラック**の再帰の深さは高々 n であるため，メモリ使用量の合計は $O(n||\mathcal{D}||)$ となる．ここで，以下の定理を得る．

定理 1 アルゴリズム**バックトラック**は $O(n||\mathcal{D}||)$ メモリを使用し，k 擬似頻出集合を *1* つあたり $O(||\mathcal{D}||)$ 時間で列挙する．

系 1 アルゴリズム**バックトラック**は k 擬似頻出集合を列挙する多項式時間遅延多項式空間アルゴリズムである．

4 アルゴリズムの効率化

この節では，前節で提案したアルゴリズムの簡素化および省メモリ化を行う．基本的なアイディアは，バケツを各反復で再利用することでメモリを節約するというものである．まず，計算の単純化を行うべく以下の開設を行う．

$Occ_{=h}(P)$ のトランザクション T を，ペア (T, h) で表す．$Occ_{\leq k}(P)$ の全ての トランザクションをこのペアの形に変換した集合を $Occ'_{\leq k}(P)$ とする．$Occ'_{\leq k}(P)$ に対して振り分けを行うと，各バケツ i は，$Occ'_{\leq k}(P \cup \{i\})$

になるため，計算の効率が上がる．$Occ'_{\leq k}(P)$ を h に関してソートして保持すれば，$Occ_{=k}(P)$ も，その大きさの線形時間で取り出すことができる．振り分け後のバケツ内のトランザクションは，振り分け前のトランザクション集合の順序を保つため，得られる $Occ'_{\leq k}(P \cup \{i\})$ も，h に関して同様にソートされた状態となる．

バケツの再利用に関しては，**右側掃きだし**とよばれる手法 [10] が効果的である．まず，以下の性質を確認しよう．

性質 6 パターン P を受け取る反復は，その反復の実行開始から終了時まで，内部で呼び出される反復も含め，$i \leq tail(P)$ であるアイテム i のバケツを使用することはない

以上の性質より，$P \cup \{i\}$ に関する再帰呼び出しを行う際には，$j < i$ なるアイテム j のバケツは，再帰呼び出しの終了時までそのまま保存されることがわかる．そこで，再帰呼び出しの実行順序を添え字の降順にした，以下のアルゴリズム PFIM (Pseudo Frequent Itemset Minor) を考える．

PFIM(P, $Occ'_{\leq k}(P)$)
1. P を出力する
2. $Occ'_{\leq k}(P)$ に対して振り分けを行う
3. **if** $|Occ_{\leq k-1}(P)| \geq \sigma$ **then** $L := \{tail(P)+1, \ldots, n\}$
4. **else** $L := \{i \mid |Occ'_{\leq k}(P \cup \{i\})| > 0\}$
   L から $CHD(P)$ に含まれない i を除去し，
   i のバケツを初期化する
5. **end if**
6. L をアイテムの降順でソートする
7. **while** $L \neq \emptyset$ **do**
8. 　L の先頭を取り出し，i に代入する
9. 　**call PFIM** $(P \cup \{i\}, Occ'_{\leq k}(P \cup \{i\}))$
10. 　i のバケツを初期化する
11. **end while**

このアルゴリズムは，バケツを再利用するため，一見，各反復の開始時にバケツの中身が初期化されていないように思える．もし，各反復の実行開始時に，その反復で使用する全てのバケツが初期化されていれば，このアルゴリズムの正当性は明らかであろう．そこで，以下の補題を証明する [10].

補題 2 **PFIM***($P, Occ'_{\leq k}(P)$)* は，アイテム $i \in \{tail(P)+1, \ldots, n\}$ のバケツが全て初期化された状態で呼び出されれば，その終了時には，アイテム $i \in \{tail(P)+1, \ldots, n\}$ のバケツは全て初期化されている．

証明：この補題を，木の葉に近いレベルから遡る帰納法で証明しよう．このアルゴリズムの計算木の葉に対応する反復，つまり再帰呼び出しを行わない反復を考える．このような反復では，step 4 で L が空集合となり，step 2 の振り分けで非空となったバケツは全て，step 4 で初期化される．そのため，題意は成り立つ．

次に，計算木上で子孫の葉からの距離が h 以下である反復で題意が成り立つと仮定し，距離が $h+1$ の反復を考えよう．その反復が受け取るパターンを P とし，アイテム $i \in \{tail(P)+1, \ldots, n\}$ のバケツは全て初期化されていると仮定する．step 2 の振り分けによって非空となったバケツのうち，$P \cup \{i\}$ が k 擬似頻出とならない i のバケツは step 4 で初期化される．step 7 から step 11 ではループが実行され，再帰呼び出しが行われる．このとき，L がアイテムの降順でソートされていることから，L の先頭のアイテム i より大きなアイテムのバケツは全て初期化されている．そのため，$P \cup \{i\}$ に関して再帰呼び出しを行う際には，$tail(P \cup \{i\})$ より大きなアイテムのバケツは初期化されており，帰納法の仮定から，その再帰呼び出し終了後，i 以後のバケツが初期化されているという状態が保たれる．

その後，i のバケツを初期化し，i は L から除去されているため，L の新しい先頭のアイテムより大きなアイテムに関して，バケツは初期化された状態になる．以後，L のアイテムを降順で取り出して再帰呼び出しすることから，再帰呼び出しを行う際，帰納法の仮定は満たされており，よって L の全てのアイテムについて再帰呼び出しを行った後，$tail(P)$ より大きなアイテムについては，バケツが初期化されている．よって，帰納法から題意は成り立つ．∎

この補題から，アルゴリズムの実行時間について以下の定理を得る．

定理 2 アルゴリズム **PFIM** は，$O(||\mathcal{D}||)$ のメモリを使い，$O(\sum_{P \in \mathcal{F}}(||Occ_{\leq k}(P)_{>tail(P)}|| + \log n)) = O(|\mathcal{F}| \times ||\mathcal{D}||)$ 時間で k 擬似頻出集合を列挙する．ただし，ここで $\mathcal{F}$ は全ての k 擬似頻出集合からなる集合である．

証明：アルゴリズムの正当性は，アルゴリズム**バックトラック**の正当性と補題 2 より導かれる．メモリに関する題意は，バケツを再利用することから明らかである．

次に，パターン P を受け取る反復の計算時間を考えよう．step 2 は $O(||Occ_{\leq k}(P)_{>tail(P)}||)$ 時間，step 6 は $O(|CHD(P)| \log n)$ 時間，残る部分については $O(|CHD(P)|)$ 時間で実行できる．これを全ての k 擬似頻出集合 P について合計を取ると，$O(\sum_{P \in \mathcal{F}}(||Occ_{\leq k}(P)_{>tail(P)}|| + \log n)) = O(|\mathcal{F}| \times ||\mathcal{D}||)$ となる．∎

本稿で提案したアルゴリズムの構造は，頻出集合列挙アルゴリズムである LCM [10, 12, 11] と等しい．データベース縮約などの，実践的な効率化の手法もそのまま適用可能である．よって，現実問題に対しても，ほぼ同じような挙動を示すと思われ，実用上は十分な性能を発揮できると期待できる．

5 実用上の高速化

本稿では，擬似頻出集合に用いる緩和した包含関係として k 擬似包含関係を採用した．これは，通常の包含関係を自然に緩和したモデルであるが，現実問題への適用時に１つ大きな問題がある．それは，大きさの小さいパターンの多くが k 擬似頻出集合となることである．例えば，大きさが k 以下のパターンは全て k 擬似頻出集合であるし，$k-1$ 擬似頻出集合 P に任意のアイテムを付け加えたものが k 擬似頻出集合になる．実際には，このような大きさの小さいパターンに興味があることは少ないと思われるので，それら小さいパターンは出力しないことが望ましい．

このような問題を解決するために良く使われる手法が，極大なパターンのみを出力する，というものである．しかし，これら小さいパターンの多くが最小サポートに近い k 擬似頻出度を持つと考えられるため，その多くが極大になると考えられる．また，大きなパターンでかつ極大でないものを失うことになる．そこで本研究では，ある定数 l の k 擬似頻出集合を直接的に見つける方法を考えることにする．

パターン P のアイテム i に対して，$Occ^*_{=k}(P,i)$ を $\{T \mid T \in Occ_{=k}(P), i \notin T\}$ とする．

補題 3 大きさ l の任意の k 擬似頻出集合 P に対して，そのアイテムの並び順 $(i_1, i_2, \ldots, i_{|P|})$ が存在して，任意の y について $|Occ_{\leq k-1}(\{i_1, \ldots, i_y\})| \geq |Occ_{\leq k}(P)|\frac{|P|-y}{|P|}$ が成り立つ．

証明：$(i_1, i_2, \ldots, i_{|P|})$ を，$|Occ^*_{=k}(P, i_y)|$ の昇順，つまり任意の $1 \leq y < |P|$ について $|Occ^*_{=k}(P, i_y)| \leq |Occ^*_{=k}(P, i_{y+1})|$ を満たす列とする．このとき，$y \leq |P|$ について，アイテム集合 $\{1, \ldots, y\}$ の $k-1$ 擬似頻出度を考える．任意の $j > y$ に対して $\{1, \ldots, y\}$ は $Occ^*_{=k}(P, i_j)$ のトランザクションに $k-1$ 擬似包含関係の意味で含まれる．$|Occ^*_{=k}(P, i_j)|$ の平均値は $|Occ_{=k}(P)|\frac{k}{|P|}$ 以下であり，１つのトランザクションは高々 k 個の j に対してのみ $Occ^*_{=k}(P, i_j)$ に含まれることをあわせると，$\bigcup_{j=y+1}^{|P|} Occ^*_{=k}(P, i_j)$ の大きさは $(|P|-y) \times |Occ_{=k}(P)|\frac{k}{|P|}/k = |Occ_{=k}(P)|\frac{|P|-y}{|P|}$ 以上となる．$|Occ_{\leq k-1}(\{i_1, \ldots, i_y\})| = |Occ_{\leq k-1}(P)| + |\bigcup_{j=y+1}^{|P|} Occ^*_{=k}(P, i_j)|$ であるので，$(i_1, \ldots, i_{|P|})$ は題意を満たす列である．∎

以後，定数 l とアイテム集合 $P, |P| < l$ に対して，条件 $|Occ_{\leq k-1}(P)| \geq \sigma\frac{l-|P|}{l}$ を部分頻出度条件とよぼう．補題 3 より，大きさが l である任意の k 擬似頻出集合は，アイテムを逐次的に追加するアルゴリズムで，部分頻出度条件を満たすパターンのみを通過して，全て列挙できることがわかる．部分頻出度条件を満たす k 擬似頻出集合の総数は，全体に比べてはるかに少ないと考えられるため，速度の向上が見込まれる．

ただし，部分頻出度条件を満たすパターンのみを生成しようとすると，末尾拡張は使用できない．なぜならば，そのようなパターン P に対して，$P \setminus \{tail(P)\}$ が先の性質を満たすとは限らないからである．そのため，末尾より小さなアイテムも加える必要がある．しかし，無条件に追加を行うと，１つの k 擬似頻出集合 P を異なる $P \setminus \{i\}, P \setminus \{j\}$ から 2 回，さらにはより多数生成する可能性がある．そこで，以下のルールを用いて重複を避ける．

生成ルール: 部分頻出度条件を満たすパターン P は，$|Occ_{k-1}(P \setminus \{i\})|$ が最大である $P \setminus \{i\}, i \in P$ からのみ生成する．タイがある場合は，アイテム集合が辞書順で最小のものを選ぶ．

補題 4 生成ルールを用いてアイテムを逐次的に追加することで，大きさ l の任意の k 擬似頻出集合がちょうど *1* 回生成される．

このような生成ルールを用いた列挙手法を逆探索という [3]．以下に，この生成ルールを用いた，大きさが l の k 擬似頻出集合を列挙するアルゴリズムを記述する．

部分頻出度条件逆探索 (P)
1. **if** $|P| = l$ **then output** P ; **return**
2. **for** $i \notin P$ **do**
3. **if** $|Occ_{\leq k}(P \cup \{i\})| \geq \sigma$ **then**
// k pseudo frequency check
4. **if** $|Occ_{\leq k-1}(P \cup \{i\})| \geq \frac{\sigma}{l}(l - |P|)$ **then**
// partial frequency check
5. **if** $P, P \cup \{i\}$ が生成ルールを満たす **then**
call 部分頻出度逆探索 $(P \cup \{i\})$
6. **end for**

補題 5 アルゴリズム逆探索の *1* 反復の計算時間は $O(||D||)$ 時間である．

証明：アルゴリズムの計算時間で重要な部分は step 3,4,5 である．このうち，3,4 については，$O(||D||)$ 時間で計算できることを，前節で解説した．次に step 5 での生成ルールを満たすかどうかの確認であるが，この部分では，$|Occ_{\leq k-1}(P \cup \{i\} \setminus \{j\})|$ を全ての $j \in P$ について計算する必要がある．この計算を素直に行うと $O(||D|| \times |P|)$ 時間かかる．そのため，以下で計算時間を減少する工夫を述べる．まず，$Occ_{\leq k-1}(P \cup \{i\} \setminus \{j\}) = Occ^*_{=k}(P \cup \{i\}, j) \cup Occ_{\leq k-1}(P \cup \{i\})$ であることを観察しよう．今，$Occ_{\leq k-1}(P \cup \{i\})$ は短時間で計算できることがわかっているため，$Occ^*_{=k}(P \cup \{i\}, j)$ が短時間で得られれば，短時間で $|Occ_{\leq k-1}(P \cup \{i\} \setminus \{j\})|$ の計算ができる．$Occ^*_{=k}(P \cup \{i\}, j) = Occ^*_{=k}(P, j) \cap Occ(\{i\})$ であることから，全ての $i \notin P$ について $Occ^*_{=k}(P \cup$

$\{i\}, j)$ を計算する時間は，$Occ^*_{=k}(P, j), j = 1, \ldots, |P|$ が $Occ_{=k}(P)$ の分割になっていることを考えると，振り分けを使うことで $O(||D||)$ 時間でできる．以上より，題意は示された．∎

6 まとめ

本稿では，頻出パターン発見問題に対して，包含関係にあいまいさ・エラーを導入するというモデル化を考案し，アイテムのうちいくつかを含まなくて良いという緩和された包含関係による頻出集合をモデル化した．また，その緩和された包含関係による頻出集合を列挙する問題に対して，多項式時間遅延多項式空間アルゴリズムを提案した．このアルゴリズムは，既存の頻出集合列挙アルゴリズムの構造を継承しており，実用的に高いパフォーマンスを示すことが期待される．また，大きさが小さい頻出集合の数が爆発的に増えることを考慮し，ある程度以上の大きさの頻出集合を直接的に列挙するアルゴリズムも合わせて提案した．今後，実問題での効率を検証するため，アルゴリズムの実装と計算機実験が必要である．また，他の頻出パターン発見問題に対する拡張も将来的な課題であろう．

謝辞

当研究の一部は，文部科学省科学研究補助費「大規模ゲノムデータ処理に対する高速高精度アルゴリズムの開発」，国立情報学研究所共同研究からサポートを受けて行われた．

参考文献

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

[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, S. Arikawa, Efficient Substructure Discovery from Large Semi-structured Data. *SDM 2002*, 2002.

[3] D. Avis and K. Fukuda, Reverse Search for Enumeration, *Discrete App. Math.,* 65, pp. 21–46, 1996.

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

[5] J. Besson, C. Robardet, and J. F. Boulicaut, Mining Formal Concepts with a Bounded Number of Exceptions from Transactional Data, *KDID 2004, Lecture Notes in Computer Science* 3377, pp. 33–45, 2005.

[6] J. Liu, S. Paulsen, W. Wang, A. Nobel, J. Prins, "Mining Approximate Frequent Itemsets from Noisy Data," 5th IEEE International Conference on Data Mining (ICDM'05), pp. 721-724, 2005

[7] J. K. Seppanen and H. Mannila, "Dense Itemsets", In SIGKDD 2004.

[8] W. Shen-Shung and L. Suh-Yin, "Mining Fault-Tolerant Frequent Patterns in Large Databases", ICS2002, 2002.

[9] M. Takeda, S. Inenaga, H. Bannai, A. Shinohara, and S. Arikawa, Discovering Most Classificatory Patterns for Very Expressive Pattern Classes, *In Proc. of Discovery Science 2003, Lecture Notes in Computer Science* 2843, pp. 486–493, 2003.

[10] 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.

[11] T. Uno, T. Asai, Y. Uchida, 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, H. Arimura, LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets, In *Proc. IEEE ICDM'04 Workshop FIMI'04*, 2004.

[13] J. T. L. Wang, G. W. Chirn, T. G. Marr, B. Shapiro, D. Shasha and K. Zhang, Combinatorial pattern discovery for scientific data: some preliminary results, *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pp. 115–125, 1994

[14] C. Yang, U. Fayyad, P. S. Bradley, "Efficient Discovery of Error-Tolerant Frequent Itemsets in High Dimensions," In SIGKDD 2001.