TF^{2}P-growth：閾値設定を必要としない頻出アイテムセット抽出アルゴリズム

平手勇宇[†] 岩橋永悟[†,☆] 山名早人[††]

データマイニング分野での頻出アイテムセット抽出手法は，最小サポート値を与えて，最小サポート値以上のサポート値を持つアイテムセットを抽出する手法である．与えられる最小サポート値から抽出される頻出アイテムセット数を予測することは困難であることから，最小サポート値を必要とせず，頻出上位数 k を指定して，サポート値降順に k アイテムセットを抽出する Top-k Mining コンセプトが近年提案されている．しかし，Top-k Mining コンセプトも閾値として k を指定する必要があり，ユーザはマイニングプロセス開始時に，解析に必要なアイテムセット数を予測しなければならない．本稿では，最小サポート値や閾値をユーザが指定する必要のない頻出アイテムセット抽出法として TF^{2}P-growth アルゴリズムを提案する．TF^{2}P-growth は，短時間でサポート値降順にアイテムセットを抽出しユーザに返すアルゴリズムである．

TF^{2}P-growth: Frequent Itemset Mining Algorithm without Any Thresholds

Yu Hirate,[†] Eigo Iwahashi[†,☆] and Hayato Yamana[††]

Conventional frequent itemset mining algorithms require some user-specified minimum support, and then mine frequent itemsets with support values that are higher than the minimum support. As it is difficult to predict how many frequent itemsets will be mined with a specified minimum support, the Top-k mining concept has been proposed. The Top-k Mining concept is based on an algorithm for mining frequent itemsets without a minimum support, but with the number of most k frequent itemsets ordered according to their support values. However, the Top-k mining concept still requires a threshold k. Therefore, users must decide the value of k before initiating mining. In this paper, we propose a new mining algorithm, called "TF^{2}P-growth," which does not require any thresholds. This algorithm mines itemsets with the descending order of their support values without any thresholds and returns frequent itemsets to users sequentially with short response time.

1. はじめに

近年，ネットワーク環境の整備・記憶装置の低価格化，大容量化にともない，大量のデータが蓄積されるようになってきた．しかし，集められた大量のデータは記号の列にすぎない．大量のデータの中から新しい知識となる情報を抽出することは，人間では不可能であることから，データマイニング技術が注目されている．

データマイニング技術の重要な問題として相関ルールの抽出がある[1)]．相関ルールとは，複数のアイテムまたはアイテムセットの間の相関関係を表すものである．相関ルール抽出問題は，データベースからアイテムセットの出現頻度が高いアイテムセットを抽出する問題に置き換えることができる．データマイニング分野においては，巨大なデータベースに対してアイテムセットの出現頻度が高いアイテムセット，すなわち頻出アイテムセットを効率良く抽出する手法の研究が行われている．従来の頻出アイテムセット抽出アルゴリズムは，Candidate-generation-and-test アプローチと，Pattern-growth アプローチに分類することができる[2)]．

Apriori[3)] に代表される Candidate-generation-and-test アプローチは，候補アイテムセットを生成

† 早稲田大学大学院理工学研究科
Graduate School of Science and Engineering, Waseda University
†† 早稲田大学理工学術院
Science and Engineering, Waseda University
☆ 現在，株式会社日立製作所情報・通信グループソフトウェア事業部
Presently with Software Division, Information & Telecommunication Systems, Hitachi, Ltd.

し，生成した候補アイテムセットの出現率を逐一計算しながら頻出アイテムセットを抽出する手法である．しかし，生成する候補アイテムセット数が膨大になってしまうことと，候補アイテムセットの出現率を計算するたびにデータセットのスキャンを必要とすることの 2 つの理由で，頻出アイテムセット抽出処理に時間がかかることが問題となっていた．

FP-growth[4] に代表される Pattern-growth アプローチは，データセットを特殊なデータ構造に変換し，変換したデータ構造に対して頻出アイテムセット抽出を行う手法である．Pattern-growth アプローチはデータセットのスキャン回数を数回に抑え，候補アイテムセットを生成することなく頻出アイテムセットを抽出するため，多くの場合において，Candidate-generation-and-test アプローチよりも高速に頻出アイテムセットを抽出することが可能となる．Pattern-growth アプローチは，2000 年以降に提案された多くの頻出アイテムセット抽出アルゴリズムに採用されている[2),5),6)]．しかし，これらの従来アルゴリズムは最小サポート値の設定をする必要がある．したがって，ユーザが不適切な最小サポート値の設定をすると，結果が返ってくるまでに時間がかかってしまったり，結果に少数の頻出アイテムセットしか含まれておらず解析ができなかったりするような事態が発生する．

高速に頻出アイテムセットを抽出するアルゴリズムが提案されている一方，単純に最小サポート値を超えるアイテムセットすべてを結果として出すのではなく，いろいろな条件を付加したうえでの頻出アイテムセットを抽出するコンセプトマイニングという手法が提案されてきている．コンセプトマイニング手法の中には，極大アイテムセット抽出手法，飽和アイテムセット抽出手法，Top-k Mining 手法がある．

極大アイテムセット抽出手法，飽和アイテムセット抽出手法を利用することにより，ユーザは頻出アイテムセットの中でのサブセットを除くスーパセットのみを抽出することができる．Top-k Mining 手法を利用することにより，ユーザは最小サポート値を設定することなく，抽出アイテム数 k を指定して，抽出される頻出アイテムセット数を制御することができる．

データマイニングの実際の応用を考えた場合，サポート値降順に k 個のアイテムセットを抽出する Top-k Mining はユーザビリティを高めるという観点において重要である．Top-k Mining として現在までに提案されている手法は，2000 年に Fu らによって提案された Apriori[3] ベースの Itemset-Loop/Itemset-iLoop[13] と，2002 年に Han らによって提案された FP-growth ベースの TFP[14] である．Itemset-Loop/Itemset-iLoop は，アイテムセットの要素数の上限を指定し，上限以下の各要素数で構成されたサポート値降順の k アイテムセットを抽出するアルゴリズムであるが，Apriori ベースのためスキャン回数が多く実行時間が長くなる．TFP は，アイテムセットの要素数の下限を指定し，下限以上の要素数で構成される飽和頻出アイテムセット[11] のみを対象として，サポート値降順に k アイテムセットを抽出するアルゴリズムである．

Top-k Mining アルゴリズムを実行する場合，ユーザはアルゴリズム実行時に抽出アイテム数 k を指定する必要がある．しかし，ユーザにとって解析に必要十分であるアイテムセット数 k を指定することは困難である．したがって，Top-k Mining アルゴリズムを利用して頻出アイテムセットを抽出する場合，ユーザは適切な k の値が設定できるまで繰り返し Top-k Mining アルゴリズムを実行しなければならない．繰り返し Top-k Mining アルゴリズムを実行する場合，そのつど k の値を指定して実行しなければならないというユーザビリティを損なう点と，対象とするデータセットを繰り返し読み込むことに起因する実行時間の長時間化という点の 2 つの問題点が存在する．

本稿では，閾値設定を必要としない新しい頻出アイテムセット抽出アルゴリズム “TF^{2}P-growth (Threshold Free FP-growth)” を提案する．“TF^{2}P-growth” は，閾値を設定することなくサポート値降順に頻出アイテムセット集合を求め，順次ユーザに表示する手法である．TF^{2}P-growth は，ベースとして Top-k Mining アルゴリズムを用いているが，サポート値降順 k 頻出アイテムセットを抽出する手法は提案されていない．このため本稿では，“TF^{2}P-growth” のベースとなる Top-k Mining 手法である “Top-k FP-growth” アルゴリズムもあわせて提案する．“TF^{2}P-growth” は，Top-k Mining アルゴリズムを繰り返し実行する場合に比べ，データセット読み込み回数を減らし高速化も実現している．

以下，2 章では，本稿で用いる用語を定義する．3 章では，関連研究として頻出アイテムセット抽出アルゴリズムを述べる．4 章では提案アルゴリズムである Top-k FP-growth と TF^{2}P-growth のアルゴリズムについて述べる．5 章では，4 章で述べた提案アルゴリズムを実装した実験結果について述べる．最後に，6 章でまとめを行う．

2. 用語定義

アイテム集合を $I = \{i_1, i_2, \ldots, i_m\}$ とする．アイ

テムセット X は，アイテム集合 I のサブセットである．m 個のアイテムによって構成されているアイテムセットを m-itemset とする．トランザクションデータベース（TDB）を $TDB = \{t_1, t_2, \ldots, t_n | t_i \subseteq I\}$ とする．T の各要素 t_i をトランザクションとする．

TDB が与えられた場合，アイテムセット X のサポート $sup(X)$ は，TDB 全体に対して X を含むトランザクションの割合を表す．頻出アイテムセットとは，ユーザが与えた最小サポート値（min_sup）以上のサポート値を持つアイテムセット集合である．アイテムセットをサポート値降順に並べ替え，k 番目に配置されたアイテムセットのサポート値を α と置く．Top-k 頻出アイテムセットとは，α 以上のサポート値を持つアイテムセット集合である．

3. 関 連 研 究

本章では，頻出アイテムセットをすべて抽出する従来のマイニング手法について述べた後，コンセプトマイニングである極大アイテムセット抽出手法，飽和アイテムセット抽出手法，Top-k Mining 手法について述べる．

3.1 従来のマイニング手法

3.1.1 Apriori [3)]

Apriori [3)] は，基本的な幅優先探索アルゴリズムである．Apriori の理論は，「非頻出 k-itemset を含む $(k+1)$-itemset は，必ず非頻出アイテムセットである」である．この理論のもと，Apriori は頻出 k-itemset から候補 $(k+1)$-itemset を生成し，候補 $(k+1)$-itemset の頻度を計算する（$k \geq 1$）．

Apriori が提案された後，Apriori をもとに効率化を図ったさまざまな派生アルゴリズムが提案された．候補アイテムセットの生成数を削減しメモリ使用量を抑えた DHP アルゴリズム[7)]，TDB のスキャン回数を減らした Partition アルゴリズム[8)] などがあげられる．

3.1.2 FP-growth [4)]

2000 年に Han らによって提案された FP-growth アルゴリズムは，最初の Pattern-growth アプローチに基づくアルゴリズムである．FP-growth は，TDB を FP-tree と呼ばれるデータ構造に変換し，FP-tree を参照することにより頻出アイテムセットを抽出する．FP-tree 構造は，Prefix-Tree 構造を拡張したデータ構造であり，頻出アイテムセットを抽出するために必要な情報を，圧縮して格納する．

3.1.2.1 FP-tree 構造

FP-tree 構造は完全な頻出アイテムセットを抽出するために必要なすべての情報を保持する．FP-tree 構造は頻出 1-itemset の Prefix-tree と，頻出 1-itemset のヘッダテーブルで構成される．Prefix-tree のそれぞれのノードは，1-itemset 名，カウント値，そしてノードリンクの 3 つのフィールドで構成されている．

- 1-itemset 名フィールドには，1-itemset の名前を格納する．
- カウント値フィールドには，Prefix-tree のルートノードから該当ノードまでのパスで表現されるアイテムセットを含むトランザクション数を格納する．
- ノードリンクフィールドには，FP-tree 中に存在する同一 1-itemset 名フィールドのノードへのリンクを格納する．

ヘッダテーブルのそれぞれのエントリには，1-itemset 名とノードリンクのヘッドの 2 つのフィールドが存在する．

- 1-itemset 名フィールドには，1-itemset の名前を格納する．
- ノードリンクのヘッドフィールドには，Prefix-tree 中の同一 1-itemset 名を格納するノードのうち，先頭に配置されているノードへのリンクを格納する．

3.1.2.2 FP-tree 構造の構築

FP-growth アルゴリズムは，FP-tree 構造を構築するために TDB を 2 回スキャンする必要がある．1 度目のスキャンで FP-growth は，頻出 1-itemset を抽出し，サポート値降順になるように頻出 1-itemset を並べ替える．並べ替えた頻出 1-itemset のリストを，F-list と呼ぶ．2 度目のスキャンの前に，空（null）のラベル R をルートノードとする木 T を作る．2 度目のスキャンでは，TDB 中のすべてのトランザクション t について以下に示す処理を行う．

(1) t の要素であっても，F-list にない 1-itemset は無視する．
(2) F-list に存在する頻出 1-itemset のみ，F-list と同じ順番でソートする．
(3) ソートしたアイテムセットを，$[p|P]$ と表現する．p は，ソートしたアイテムセットの中で最もサポート値が高い 1-itemset であり，P は，それ以外のアイテムセットで構成され，サポート値降順でソートされている．
(4) 後述する関数 insert_tree$([p|P], R)$ を実行し，FP-tree を構築する．

関数 insert_tree$([p|P], R)$ は，トランザクション $[p|P]$ 情報をルートノードが R である木 T に付け加える．insert_tree$([p|P], R)$ の擬似コードを**図 1** に

```
function insert_tree([p|P], R){
  let N be a direct child node of R,
    such that N's item-name=p's item-name.
  if(R has a direct child node N){
    increment N's count by 1.
  }
  else{
    create a new node M linked under the R.
    set M's item-name equal to p.
    set M's count equal to 1.
  }
  if(P ≠ φ){
    let p' be the first 1-itemset in P.
    let P' be the remaining 1-itemset list in P.
    call insert_tree([p'|P'], R).
  }
  else {
    exit.
  }
}
```

図 1 関数 insert_tree($[p|P], R$) 擬似コード
Fig. 1 Pseudo code of insert_tree($[p|P], R$).

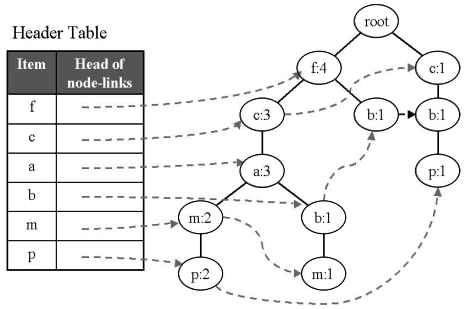

図 2 表 1 から構築された FP-tree 構造
Fig. 2 FP-tree structure constructed by sample TDB which shown in Table 1.

表 1 サンプル TDB
Table 1 Sample TDB.

TID	Items	Frequent Items
100	$f,\ a,\ c,\ d,\ g,\ i,\ m,\ p$	$f,\ c,\ a,\ m,\ p$
200	$a,\ b,\ c,\ f,\ l,\ m,\ o$	$f,\ c,\ a,\ b,\ m$
300	$b,\ f,\ g,\ i,\ o$	$f,\ b$
400	$b,\ c,\ d,\ k,\ p$	$c,\ b,\ p$
500	$a,\ f,\ c,\ e,\ l,\ p,\ m,\ k$	$f,\ c,\ a,\ m,\ p$

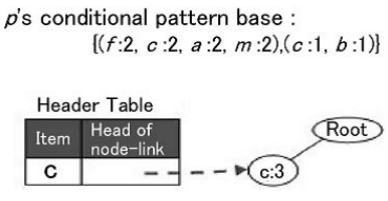

図 3 p-conditional FP-tree 構造
Fig. 3 p-conditional FP-tree structure.

示す.

FP-tree 構造の例を **図 2** に示す．この FP-tree 構造は，**表 1** に示すような TDB から最小サポート値を 3 としたときに構築された場合の構造である．図 2 では，すべてのノードは (1-itemset 名：カウント値) で表現されている．リンクは矢印（点線）で表現されている．

3.1.2.3 FP-growth

FP-growth は，構築した FP-tree をもとに頻出アイテムセットを抽出する手法である．すべての頻出アイテムセットを抽出するために，FP-growth は，FP-tree のヘッダテーブル中のノードリンクのヘッドからノードリンクをたどる．ヘッダテーブルの任意の 1-itemset $\{a_i\}$ のノードリンクのヘッドからノードリンクをたどることにより，1-itemset $\{a_i\}$ を含むすべての頻出アイテムセットを抽出することができる．また，1-itemset $\{a_i\}$ のノードリンク上のノードから root ノードまでのパスからなる prefix-path を a_i 条件付きパターンベース（a_i conditional pattern base）と呼ぶ．a_i conditional pattern base から $\{a_i\}$ を含む条件の下での FP-tree（a_i conditional FP-tree）を構築し，$\{a_i\}$ を含むすべての頻出アイテムセットを抽出する．

例として 図 2 中の FP-tree より 1-itemset $\{p\}$ を含んだ頻出アイテムセット抽出過程を述べる．図 2 中の FP-tree におけるヘッダテーブルより頻出アイテムセット (p:3) が得られ，$\{p\}$ のノードリンクより FP-tree の 2 つのパス（$< f:4,\ c:3,\ a:3,\ m:2,\ p:2 >$, $< c:1,\ b:1,\ p:1 >$）が得られる．$\{p\}$ とともにどのアイテムが出現しているかを見ることによって，$\{p\}$ の prefix-path$< f:2,\ c:2,\ a:2,\ m:2 >$ を数えることができる．同様に 2 つ目のパスは $(c,\ b,\ p)$ が 1 度出現し，$\{p\}$ の prefix-path が $< c:1,\ b:1 >$ であることを表す．

したがって，**図 3** のように，p conditional pattern base は（$< f:2,\ c:2,\ a:2,\ m:2 >$, $< c:1,\ b:1 >$）となり，カウント値が 3 以上あるのはアイテム $(c:3)$ のみなので，p conditional FP-tree（conditional FP-tree）は，たった 1 本の枝 $(c:3)$ しか得られない．ゆえにたった 1 つの頻出アイテムセッ

ト $(cp:3)$ しか得られない．したがって，$\{p\}$ を含む頻出アイテムセットは $(p:3),(cp:3)$ である．

3.2 コンセプトマイニング手法

3.2.1 極大アイテムセット抽出手法

頻出アイテムセット抽出アルゴリズムは，しばしば必要以上に多くの頻出アイテムセットを抽出する．しかし，必要以上に多くの頻出アイテムセットから，既知でない知識に結び付くアイテムセットを発見することは困難である．極大アイテムセット抽出手法は，このような問題を解決するために，最小サポート値を満たす頻出アイテムセットの中から**極大頻出アイテムセット**のみを抽出するマイニングのことを指す．極大アイテムセット抽出アルゴリズムとして，Max-Miner [9]，FPmax [10] などが提案されている．

なお，アイテムセット X が**極大頻出アイテムセット**であるということは，「アイテムセット X のサポート値が最小サポート値以上であり，かつ X のスーパセットである任意の X' が，最小サポート値未満のサポート値である」ことである．

3.2.2 飽和アイテムセット抽出手法

極大アイテムセット抽出手法と同じように，飽和アイテムセット抽出手法は，結果として出されるアイテムセット数を減らす目的で，**飽和頻出アイテムセット**のみを抽出するマイニングである．飽和アイテムセット抽出アルゴリズムとして，CLOSET [11]，FP-close [6] などが提案されている．

なお，アイテムセット X が**飽和頻出アイテムセット**であるということは，「アイテムセット X のサポート値が最小サポート値以上であり，かつ X と同一のトランザクション上にある X のすべてのスーパセット X' が，最小サポート値未満のサポート値である」ことである．

3.2.3 Top-k Mining 手法

一般に最小サポート値から，結果として抽出される頻出アイテムセットの個数を予測することは困難である．最小サポート値が小さいと大量の頻出アイテムセットが抽出され，逆に最小サポート値が大きいと，ほとんど頻出アイテムセットは抽出されない．このように，適切な最小サポート値を選択することは難しい．このため，最小サポート値を閾値として必要とせず，抽出したいアイテムセット数 k を指定して，Top-k 頻出アイテムセットを抽出する手法が提案されている [13),14)]．

4. 提 案 手 法

4.1 既存手法の問題点

3.1 節，3.2.1，3.2.2 項で示した頻出アイテムセット抽出手法の場合，ユーザは閾値として最小サポート値を設定したうえで抽出プロセスを実行しなければならない．しかし，一般的にユーザは最小サポート値から抽出される頻出アイテムセット数を予測できないため，最小サポート値を設定することはユーザにとって困難となる．

これに対して，Top-k Mining 手法は，実行時の閾値が抽出アイテムセット数 k であるので，ユーザビリティを高めるという点において重要である．しかし，ユーザにとって解析に必要十分であるアイテムセット数 k を指定することは困難である．したがって，Top-k Mining アルゴリズムを利用して頻出アイテムセットを抽出する場合，ユーザは適切な k の値が設定できるまで繰り返し Top-k Mining アルゴリズムを実行しなければならない．繰り返し Top-k Mining アルゴリズムを実行する場合，そのつど k の値を指定して実行しなければならないというユーザビリティを損なう点と，対象とするデータセットを繰り返し読み込むことに起因する実行時間の長時間化という点の 2 つの問題点が存在する．

4.2 提案手法の概要

我々の提案する “TF^{2}P-growth” アルゴリズムは，閾値を必要とせず，サポート値降順にアイテムセットを抽出し続ける手法である．さらに，サポート値の高いアイテムセットほど短時間でユーザに返す．まず 4.3 節で，FP-growth をもとにした Top-k Mining 手法である Top-k FP-growth アルゴリズムを提案する．Top-k FP-growth アルゴリズムは，閾値 k を与え，サポート値降順 k 個の頻出アイテムセット集合を抽出する手法である．次に，4.4 節で，“Top-k FP-growth” をベースとした閾値を必要としないアルゴリズム “TF^{2}P-growth” を提案する．

4.3 Top-k FP-growth

Top-k FP-growth は，TF^{2}P-growth のベースとなる Top-k Mining アルゴリズムである．本手法は，FP-growth [4] をベースとして，最小サポート値を与えることなく，抽出したいアイテムセット数 k を指定して，Top-k 頻出アイテムセットを抽出する．以下では，これまでのワークショップ [15] や国際ミーティング [16] での議論をふまえ，Top-k FP-growth を提案する．

4.3.1 FP-growth からの拡張

Top-k FP-growth のパラメータは，最小サポート値ではなく k である．したがって k の値から，少ない計算量で Top-k 頻出アイテムセットを抽出するために FP-growth を拡張する．FP-growth からの拡張点として，新たな閾値 $Border\_sup$ の設定，FP-tree

からのアイテムセット生成数の削減，アイテムセット生成後の出力がある．

4.3.1.1 *Border_sup* の設定

Top-k FP-growth は，閾値として *Border_sup* を用いて FP-tree を構築する．具体的には，FP-tree 構築対象の頻出 1-itemset を，*Border_sup* を超えるサポート値を持つ 1-itemset とする．*Border_sup* は内部の閾値であるため，*Border_sup* の値の設定は自動的に行われる．*Border_sup* の定義は以下のとおりである．

Border_sup *Border_sup* とは，サポート値降順にソートされた 1-itemset の中で，k 番目に配置された 1-itemset のサポート値のことである．すなわち，1-itemset 集合の中で，*Border_sup* 以上のサポート値を持つ 1-itemset は，k 個存在する．

補題 4.1 *Border_sup* 以上のサポート値を持つ 1-itemset を，FP-tree 構築の対象と限定しても，すべての Top-k 頻出アイテムセットを過不足なく抽出できる．

［証明］ *Border_sup* を下回るサポート値を持つ任意の 1-itemset を $\{\alpha\}$，任意のアイテムセットを $\{\beta\}$ とおくと，

$$sup(\{\alpha,\beta\}) \leq sup(\{\alpha\}) < Border\_sup$$

を満たすので，$\{\alpha\}$ をサブセットとして含む任意のアイテムセット $\{\alpha,\beta\}$ は，必ず *Border_sup* を下回る．*Border_sup* 以上のサポート値を持つアイテムセットは，*Border_sup* の定義から必ず k アイテムセット以上存在する．したがって，Top-k 頻出アイテムセットを抽出するには，*Border_sup* を上回るサポート値を持つ 1-itemset を FP-tree 構築対象とすればよい．

これにより，**図 4** に示すように FP-tree 構築に必要なメモリスペースを削減することができる．なお，図 4 は簡素化のためノードリンクを省略してある．たとえば，表 1 の *TDB* において，1-itemset の 6 番目に高いサポート値は 3 であるため，$k = 6$ のときの *Border_sup* は，3 となる（図 4 中のヘッダテーブル）．したがって，1-itemset である $\{f\}$，$\{c\}$，$\{a\}$，$\{b\}$，$\{m\}$，$\{p\}$ を対象として FP-tree を構築する．

4.3.1.2 FP-tree からのアイテムセット生成数の削減

4.3.1.1 で構築した FP-tree からすべてのアイテムに関してノードリンクをたどり，アイテムセットを生成すると k 個以上の頻出アイテムセットが抽出される．したがって Top-k FP-growth では，Reduction Array という配列と *Boundary_sup* という内部閾値を用いてヘッダテーブルからたどるノードリンク数を削減し

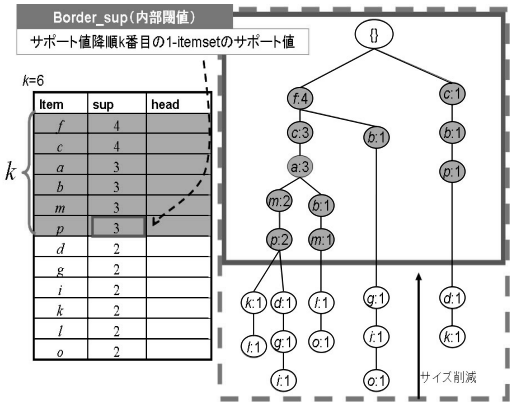

図 4 *Border_sup* を定義しての FP-tree サイズ削減
Fig. 4 Ruduction of memory space by using *Border_sup*.

アイテムセット生成数を削減することによって，計算量を削減する．ここで，Reduction Array とは，FP-tree から生成されたアイテムセットと，そのサポート値を格納するテーブルである．また，*Boundary_sup* の定義を以下に示す．

Boundary_sup *Boundary_sup* は，Reduction Array の k 番目に配置してあるアイテムセットのサポート値のことである．*Boundary_sup* は FP-tree からのアイテムセット生成開始時には 0 であるが，k 個以上のアイテムセットが抽出された時点から，値が大きくなる性質を持っている．

Top-k FP-growth では，FP-tree から生成された頻出アイテムセットと，頻出アイテムセットのサポート値を，逐一 Reduction Array に格納していく．この配列はノードリンクをたどり終えるたびにサポート値降順にソートされる．

Top-k FP-growth では，ある 1-itemset $\{\alpha\}$ のノードリンクをたどり終えて，ヘッダテーブルの次に配置されている 1-itemset$\{\beta\}$ のノードリンクをたどる前に，$\{\beta\}$ のカウント値 ($=sup(\{\beta\})$) と，上記に示した *Boundary_sup* との比較を行う．そして，$sup(\{\beta\}) \geq Boundary\_sup$ であれば，$\{\beta\}$ のノードリンクをたどりアイテムセットを生成していくが，もし $sup(\{\beta\}) < Boundary\_sup$ であれば，FP-tree からのアイテムセット生成を終了する．

Boundary_sup を下回るサポート値を持つ 1-itemset に関して，ノードリンクをたどってアイテムセット生成をしなくてもよい理由を示す．FP-tree 構造のヘッダテーブル中にあるアイテム $\{\beta\}$ のノードリンクをたどって抽出された $\{\beta\}$ を含む任意のアイテムセットは，$sup(\{\beta\})$ よりも小さなサポート値を持つ．よって，$sup(\{\beta\}) < Boundary\_sup$ を満たすとき，

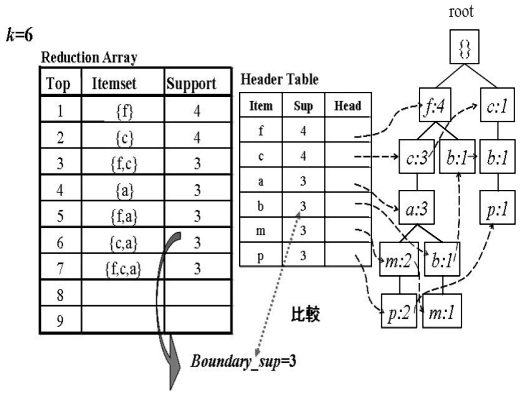

図 5 $k = 6$ のときのアイテム $\{a\}$ のノードリンクをたどり終えた後の Reduction Array

Fig. 5 Reduction Array when Top-k FP-growth finished traversing item $\{a\}$'s node-link ($k = 6$).

$\{\beta\}$ を含む任意のアイテムセットは，$Boundary\_sup$ よりも小さなサポート値を持つことになる．さらに FP-tree のヘッダテーブルで $\{\beta\}$ よりも下に配置されている任意のアイテムセット $\{\gamma\}$ と，$\{\gamma\}$ をたどって抽出されるアイテムセットも，$sup(\{\beta\})$ よりも小さな値を持つ．よって，$\{\gamma\}$ を含む任意のアイテムセットについても，$Boundary\_sup$ よりも小さなサポート値を持つこととなる．以上より，$sup(\{\beta\}) < Boundary\_sup$ のとき，FP-tree からのアイテムセット生成を終了しても，過不足なくすべての Top-k 頻出アイテムセットが生成される．

たとえば，表 1 の TDB に対して，$k = 6$ で FP-tree を構築し，1-itemset $\{a\}$ のノードリンクをたどり終えたときの Reduction Array を **図 5** に示す．図 5 で，Reduction Array の 6 番目に配置されているアイテムセット $\{c, a\}$ のサポート値が 3 であるため，$Boundary\_sup = 3$ とおかれる．このとき，次の 1-itemset $\{b\}$ のノードリンクをたどる前に，1-itemset $\{b\}$ のサポート値（$=3$）と，$Boundary\_sup$ を比較する．この場合は，$sup(\{b\}) = Boundary\_sup$ であるので，1-itemset $\{b\}$ のノードリンクをたどり，アイテムセットを生成する．もし，$sup(\{b\}) < Boundary\_sup$ であった場合は，$\{b\}$ のノードリンクをたどらない．このようにして，アイテムセット生成を行う．

4.3.1.3 アイテムセット生成後の出力

上記の 4.3.1.1，4.3.1.2 項の拡張を行っても，FP-tree から抽出したアイテムセットは，k 個以上存在する．したがって，FP-tree からアイテムセットを抽出した後，Reduction Array より Top-k 頻出アイテムセットを抽出結果としてユーザに返す．

4.3.2 Top-k FP-growth アルゴリズム

Top-k FP-growth アルゴリズムを示す．

INPUT TDB，抽出したいアイテムセット数 k
OUTPUT Top-k 頻出アイテムセット
METHOD

(1) TDB をスキャンして，すべての 1-itemset のサポート値をカウントする．
(2) 1-itemset のサポート値降順 k 番目アイテムセットのサポート値を $Border\_sup$ として，F_list を作成する．
(3) 作成した F_list をもとにして，FP-tree を構築する．
(4) 構築した FP-tree からアイテムセットを生成する．その際，新たなノードリンクをたどるたびに $Boundary\_sup$ を計算し，アイテムセット生成数を削減させる．
(5) 抽出した頻出アイテムセットのうち，上位 k アイテムセットを結果として返す．

FP-growth と Top-k FP-growth の違いを **表 2** に示す．

4.4 TF^{2}P-growth

3.2.3 項で紹介した従来の Top-k Mining 手法と同様に，Top-k FP-growth も閾値としてアイテムセット数 k を指定して，実行しなければならない．しかし，ユーザにとって解析に必要十分であるアイテムセット数 k を指定することは困難である．したがって，Top-k Mining アルゴリズムを利用して頻出アイテムセットを抽出する場合，ユーザは適切な k の値が設定できるまで繰り返し Top-k Mining アルゴリズムを実行しなければならない．繰り返し Top-k Mining アルゴリズムを実行する場合，そのつど k の値を指定して実行しなければならないというユーザビリティを損なう点と，対象とするデータセットを繰り返し読み込むことに起因する実行時間の長時間化という点の 2 つの問題点が存在する．

この問題を解決するために，本節では Top-k FP-growth をもとにした “TF^{2}P-growth” アルゴリズムを提案する．TF^{2}P-growth は，閾値を必要とせずサポート値降順にアイテムセットを抽出するアルゴリズムであり，Top-k FP-growth を繰り返し実行する場合よりもデータセットのスキャン回数を減らすことにより高速化を実現したアルゴリズムである．n_c をチャンクサイズとし，抽出された頻出アイテムセットは，n_c アイテムセットずつ結果としてユーザに返される．ユーザが n_c を設定しなくても TF^{2}P-growth を実行

表 2 FP-growth[4] と Top-k FP-growth の違い
Table 2 Difference between FP-growth[4] and Top-k FP-growth.

	FP-growth[4]	Top-k FP-growth
FP-tree 構築対象	min_sup 以上の頻出 1-itemset	$Border\_sup$ 以上の 1-itemset
FP-tree からのアイテムセット生成対象	FP-tree のノードすべて	$Boundary\_sup$ によって生成数削減
アイテムセット生成後	生成したアイテムセットすべてを結果として抽出	サポート値順に並べ替えて，上位 k アイテムセットを結果として抽出

することが可能である．したがって，TF^{2}P-growth を実行するときは，ユーザは対象となるデータセットのみを指定すればよい．これが，TF^{2}P-growth の 1 つ目の利点である．ただし，5.3 節で後述するように，ユーザが頻出アイテムセット抽出時間間隔を調節したい場合は，自由に設定することができる☆．また，n_c は，TF^{2}P-growth 実行中にもユーザが変更できる．

サポート値降順で n_c 個ごとに連続して頻出アイテムセットを抽出するため，最終的に抽出されるアイテムセット集合は n_c の値に関係なく同一となる．これが，TF^{2}P-growth の 2 つ目の利点である．これに対し，Top-k FP-growth の閾値である k は，サポート値降順で k 個の頻出アイテムセットを抽出するのみであるため，k を変えると最終的に抽出されるアイテムセット集合が異なる．このため，ある k によって得られる頻出アイテムセット集合にユーザが満足できない場合，k の値を変えて何度も Top-k FP-growth を実行しなければならない．

n_c がデフォルト値の 1000 である場合（ユーザが n_c を設定しない場合）の TF^{2}P-growth の動作を以下に示す．ユーザは何も閾値を定めることなく TF^{2}P-growth を実行する．実行された TF^{2}P-growth は，まず Top-1000 頻出アイテムセットをユーザに返し，その後 Top-1001〜2000 頻出アイテムセット，Top-2001〜3000 頻出アイテムセット，... と返す．ユーザは Top-1000 頻出アイテムセットを取得した後から，抽出されたアイテムセットの解析を始めることができる．そして，ユーザが解析に十分な数のアイテムセットを取得したときに，抽出プロセスを停止させることができる．いい換えると，ユーザは好きなときに抽出プロセスを停止させることができる．

4.4.1 TF^{2}P-growth アルゴリズム

TF^{2}P-growth アルゴリズムを示す．

INPUT TDB
OUTPUT（サポート値降順）頻出アイテムセット
METHOD

(1) TDB をスキャンして，すべての 1-itemset のサポート値をカウントする．
(2) $n_0 = 0$，$i = 1$ とする．
(3) $n = n_0 + n_c \times i$ とする（n_c のデフォルト値 1000）．
(4) 1-itemset のサポート値降順 n 番目アイテムセットのサポート値を $Border\_sup$ として，F_list を作成する．
(5) 作成した F_list をもとにして，FP-tree を構築する．
(6) 構築した FP-tree からアイテムセットを生成する．その際，頻出アイテムセットを生成するたびに $Boundary\_sup$ を計算し，アイテムセット生成数を削減させる．
(7) 抽出されたアイテムセットのうち，(n_0+1) 番目から $(n_0 + n_c \times i)$ 番目のアイテムセットを返す．
(8) n_c がユーザによって変更されているかどうかをチェックし，変更されている場合は (9) へ，変更されていない場合は (10) へ進む．
(9) $n_0 = n0 + n_c \times i$，$i = 1$ とした後，n_c を新しい値に更新する．(3) に戻る．
(10) i をインクリメントして，(3) に戻る．

5. 性能評価

本章では，TF^{2}P-growth アルゴリズムの性能評価を示す．TF^{2}P-growth アルゴリズムの性能評価は，(1) FP-growth[4]，Top-k FP-growth，TF^{2}P-growth の実行時間と抽出された頻出アイテムセット数の関係の比較，(2) TF^{2}P-growth の大規模データベースに対してのスケーラビリティ，(3) チャンクサイズ n_c の設定による実行時間の違いの 3 つの視点で評価した．データセットは IBM 人工データセット生成プログラム[18] によって生成した T10I4D1000k データセットを利用した．T10I4D1000 k データセットとは，トランザクション総数が 1000,000 (1000 k)，トランザクションあたりのアイテム数が 10，頻出アイテムセットの平均最大長が 4 というパラメータを持つデータセッ

☆ デフォルト値として 1000 に設定されており，ユーザが n_c を設定しない場合，自動的に n_c=1000 となる．なお，デフォルト値が 1000 である理由は特にないので，デフォルト値自体をユーザが変更して問題ない．

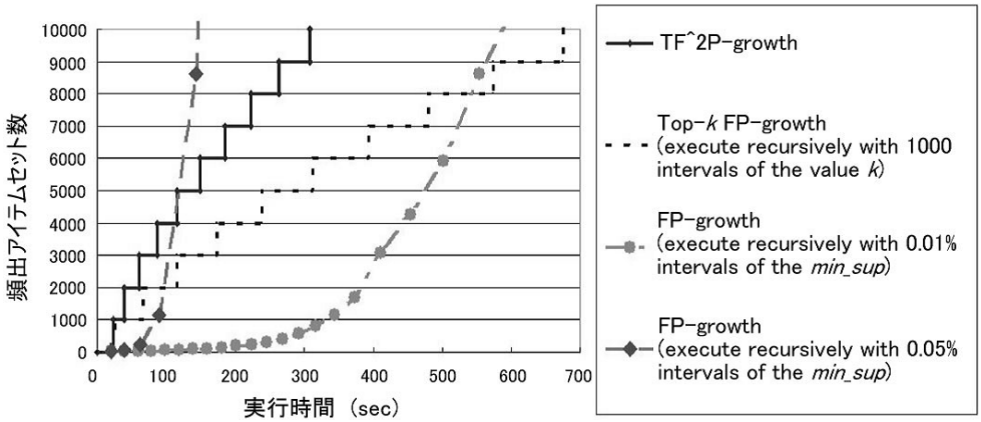

図 6 実行時間と頻出アイテムセット数の関係
Fig. 6 Execution time vs. number of mined frequent itemsets.

表 3 実験環境
Table 3 Experiments environment.

CPU	Intel Pentium4 2.4 GHz
メモリ	1,024 MB
OS	RedHat9.0 Linux (kernel2.4.20)
コンパイラ	gcc3.2.2
コンパイルオプション	−O6

トである．実験環境を **表 3** に示す．

5.1 実行時間比較

本節では，TF^{2}P-growth，FP-growth，Top-k FP-growth の実行時間と抽出されたアイテムセット数の関係の比較を行った．データマイニングの実際の利用では，ユーザは，はじめに適切な最小サポート値 k の値を設定することは難しい．したがって，ユーザは最小サポート値 k の値を変えながら繰り返し実行しなければならない．

本実験では，TF^{2}P-growth はデフォルトの状態（n_c=1000）で実行する．FP-growth は，最小サポート値をはじめ 0.3%として実行し，2 度目以降 (1) 0.01%ずつ，(2) 0.05%ずつ減少させながら繰り返し実行する，2 つの方法で計測した．Top-k FP-growth は，TF^{2}P-growth との比較のため，k の値をはじめ 1000 として実行し，2 度目以降 1000 ずつ増加させる方法で計測した．実験結果を **図 6** に示す．

図 6 より以下に示す 2 つのことが確認できる．1 つめは，FP-growth の場合，適切な最小サポート値を選択することが難しいので，ユーザは最小サポート値をはじめは高い値に設定しなければならない点である．もし，ユーザが最小サポート値と抽出される頻出アイテムセット数の関係を予測できるのであれば，図 6 の 0.05%ずつ減少させる場合のように，FP-growth のほうが TF^{2}P-growth よりも多くのアイテムセットを抽出できる．しかし，ユーザは最小サポート値と抽出される頻出アイテムセット数の関係を予測できないのが，一般的である．したがって，ユーザは最小サポート値を 0.01%のような微小な値を間隔として減少させながら繰り返し実行しなければならない．しかし，間隔が 0.01%の場合に代表されるように，最小サポート値を微小な値ずつ減少させながら繰り返し実行した場合，時間が経過してもユーザが得られるアイテムセット数は増えない．反面，TF^{2}P-growth は，n_c の設定をしなかった場合（n_c がデフォルトの 1000 である場合）でも，最小サポート値を 0.01%ずつ減少させた場合の FP-growth と比べて，同一時間により多くの数のアイテムセットをユーザが取得できることが確認できる．なお，n_c は実行途中でユーザが自由に変更できるため，最初はデフォルト値で実行を開始し，抽出される頻出アイテムセットの抽出時間間隔が長いと感じた場合は，n_c を小さく調整することができる．このように，n_c の設定はユーザが容易に行うことができる．

2 つめは，TF^{2}P-growth を実行すると，同一時間で Top-k FP-growth を繰り返し実行する手法よりも多くの頻出アイテムセットを抽出できる点である．これは，TF^{2}P-growth は，最初の TDB へのスキャン時に完全な F-list を生成することによって，TDB へのスキャン回数を減らしたことに起因する．

これらの結果は，TF^{2}P-growth が，閾値を設定しなくてもよいという利点に加え，Top-k FP-growth よりも同一時間でより多くの頻出アイテムセットを抽出できることを意味する．さらに，ユーザが抽出したいアイテムセット数に対応する最小サポート値を設定できない場合，TF^{2}P-growth は，FP-growth よりも同

表 4 データセットサイズ
Table 4 Dataset size.

T10I4D1000 k	49 MB
T10I4D5000 k	245 MB
T10I4D10000 k	490 MB

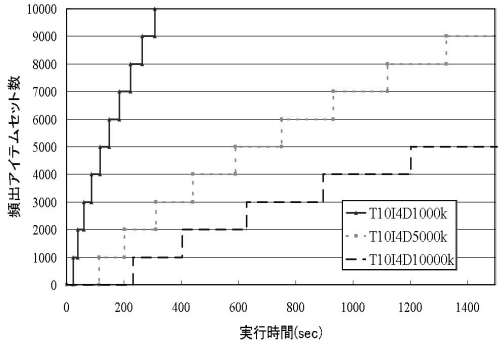

図 7 トランザクション数の違うデータセットにおいての実行時間
Fig. 7 Execution time with different numbers of transactions.

一時間でより多くの頻出アイテムセットを抽出できることを意味する．

5.2 スケーラビリティ

我々は，TF^{2}P-growth のスケーラビリティも評価した．この評価では，トランザクション数が違う複数のデータセット，T10I4D5000 k，T10I4D10000 k に対して，TF^{2}P-growth の実行時間（n_c=1000 とする）を計測した．なお，それぞれのデータセットのサイズを **表 4** に示す☆．実験結果を**図 7** に示す．また各データセットに対し同一 Top 頻出アイテムセット数でグループ化し，抽出時間をプロットしたものを **図 8** に示す．

図 7，8 より，TF^{2}P-growth はデータセットのトランザクション数が大きくなると抽出時間が線形に増加することが分かる．

5.3 n_c の設定

5.1，5.2 節でチャンクサイズ n_c はデフォルト値で 1000 として評価したが，データセットが大きい場合，前節での考察のとおり，最初の結果が返ってくるまでに長い時間がかかる．本節では，大規模データセット（T10I4D10000 k）を例にとり，チャンクサイズ n_c による実行時間の違いについて計測した．n_c を 100，500，1000，2000 と設定したときの，n_c の違いによる実行時間と抽出アイテムセット数の関係を **図 9** に

☆ IBM 人工データセット生成プログラムでは，データセットのフォーマットとしてバイナリフォーマットと，アスキーフォーマットが存在するが，ここではバイナリフォーマットを利用した．

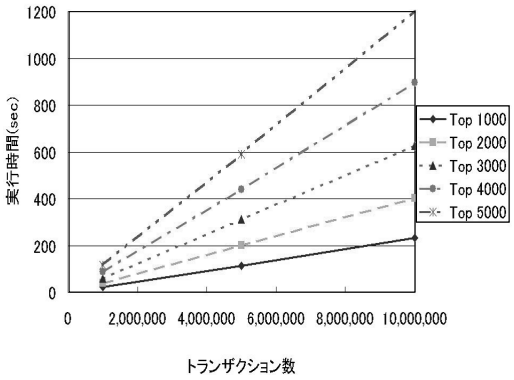

図 8 同一 Top 数におけるトランザクション数と実行時間
Fig. 8 The relation between number of transactions and execution time with the same number of frequent patterns.

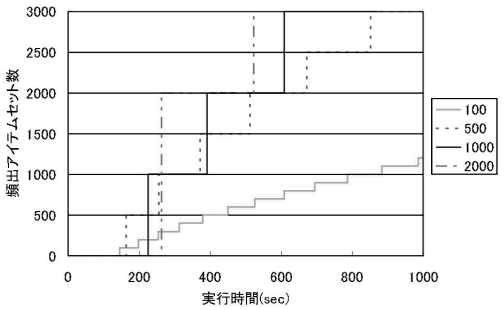

図 9 n_c の違いによる実行時間と抽出アイテムセット数の関係
Fig. 9 The relation between execution time and the number of frequent itemsets with different chunk size.

示す．

図 9 に示すように，n_c を小さく設定することにより抽出結果を返すまでの実行時間間隔の短縮化が実現できる．これは，TF^{2}P-growth の対象を大規模データセットとした場合の，5.2 節で議論した抽出時間間隔が長くなる問題に対して有効である．

すなわち，短い抽出時間間隔でアイテムセット集合を抽出したい場合，n_c を小さい値にすればよいことが分かる．逆に，抽出時間間隔は長くても，単位時間あたりの抽出アイテムセット数を多くしたい場合は，n_c を大きな値にすればよいことが分かる．なお，実行開始前にある n_c に対して，どの程度の抽出時間間隔になるかを予測することは一般に難しいが，ユーザは実行中に n_c を自由に変更して抽出時間間隔を調整することができる．この点が，TF^{2}P-growth の利点である．

6. おわりに

本稿では，閾値を設定することなくサポート値降順に頻出アイテムセットを抽出する TF^{2}P-growth アル

ゴリズムを提案した．さらに，TF^{2}P-growth のベースとなる Top-k Mining 手法である Top-k FP-growth アルゴリズムも提案した．TF^{2}P-growth を評価した結果，我々は 2 つの利点を確認することができた．1 つめは，閾値を設定せずに頻出アイテムセット抽出を実行できることである．2 つめは，ユーザが実行時に n_c を変更することにより，抽出時間間隔を調節することができることである．

一般的に，最小サポート値と抽出されるアイテムセット数の関係を予測することが困難である．さらに，抽出されたアイテムセットの解析に必要十分なアイテムセット数を予測することも困難である．これらの 2 つの困難に対し，インタラクティブに頻出アイテムセットを抽出していく TF^{2}P-growth アルゴリズムは，有用であると考える．

本稿では，閾値設定を必要としない，抽出時間の短縮化という点においてのユーザビリティの向上を実現したアルゴリズムを提案したが，近年の相関ルール抽出分野の研究では，極大頻出アイテムセット，飽和頻出アイテムセット抽出手法によって，アイテムセット数を減らすことによるユーザビリティの向上を実現するものが多い．したがって，今後の課題は，本手法を極大頻出アイテムセット，飽和頻出アイテムセット抽出手法に適用させることや，並列化することである．

謝辞 本研究の一部は，文科省 21 世紀 COE「プロダクティブ ICT アカデミア」および科学技術振興費「e-Society」プロジェクトによるものである．

参 考 文 献

1) Agrawal, R., Imielinski, T. and Swami, A.: Mining Association Rules between Sets of Item in Large Databases, *Proc. ACM SIGMOD '93*, pp.207–216 (1993).
2) Pei, J., Han, J., Lu, H., Nishio, S., Tang, S. and Yang, D.: H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases, *Proc. IEEE ICDM'01*, pp.441–448 (2001).
3) Agrawal, R. and Srikant, R.: Fast Algorithms for Mining Association Rules, *Proc. VLDB'94*, pp.487–499 (1994).
4) Han, J., Pei, J. and Yu, P.S.: Mining Frequent Patterns without Candidate Generation, *Proc. ACM SIGMOD'00*, pp.1–12 (2000).
5) Liu, J., Pan, Y., Wang, K. and Han, J.: Mining Frequent Item Sets by Opportunistic Projection, *Proc. ACM SIGKDD'02*, pp.229–238 (2002).
6) Grahne, G. and Zhu, J.: Efficiently Using Prefix-trees in Mining Frequent Itemsets, *Proc. IEEE ICDM Workshop FIMI'03* (2003).
7) Park, J.S., Chen, M. and Yu, P.S.: An Effective Hash-Based Algorithms for Mining Association Rules, *Proc. ACM SIGMOD'95*, pp.175–186 (1995).
8) Savasere, A., Omiecinski, E. and Navathe, S.: An Efficient Algorithm for Mining Association Rules in Large Databases, *Proc. VLDB'95*, pp.423–444 (1995).
9) Bayardo, R.J.Jr.: Efficiently Mining Long Patterns from Databases, *Proc. ACM SIGMOD'98*, pp.85–93 (1998).
10) Grahne, G. and Zhu, J.: High Performance Mining of Maximal Frequent Itemsets, *Proc. SIAM Workshop on High Performance Data Mining* (2003).
11) Pei, J., Han, J. and Mao, R.: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets, *Proc. ACM SIGMOD Workshop on Data Mining and Knowledge Discovery* (2000).
12) Wang, J., Han, J. and Pei, J.: CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets, *Proc. ACM SIGKDD'03*, pp.236–245 (2003).
13) Fu, A.W.-C., Kwong, R.W.-W. and Tang, J.: Mining N-most Interesting Itemsets, *Proc. Int. Symposium on Methodologies for Intelligent Systems* (*ISMIS*) (2000).
14) Han, J., Wang, J., Lu, Y. and Tzvetkov, P.: Mining Top-K Frequent Closed Patterns without Minimum Support, *Proc. IEEE ICDM'02*, pp.211–218 (2002).
15) 平手勇宇，岩橋永悟，山名早人：ユーザへの応答時間を重視した最頻出 k パターン抽出アルゴリズム，第 15 回データ工学ワークショップ（DEWS2004），6A-02 (2004).
16) Hirate, Y., Iwahashi, E. and Yamana, H.: An Efficient Algorithm for Mining Top-k Frequent Patterns with a Small Response Time, *CORS/INFORMS Joint International Meeting*, TC18-3 (2004).
17) Goethals, B. and Zaki, M.J.: Advances in Frequent Itemset Mining Implementations: Introduction to FIMI03, *Proc. IEEE ICDM Workshop on FIMI'03* (2003).
18) IBM Quest Data Mining Project. Quest synthetic data generation code. http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html

(平成 16 年 12 月 20 日受付)
(平成 17 年 4 月 6 日採録)

(担当編集委員 大森 匡)

平手 勇宇（学生会員）

1980 年生．2004 年早稲田大学理工学部情報学科卒業．2005 年同大学院理工学研究科情報・ネットワーク専攻修士課程修了．同年同大学院同専攻博士課程に入学，現在に至る．データマイニング等の研究に従事．ACM SIGMOD 日本支部，日本データベース学会，ACM 各会員．

岩橋 永悟

1980 年生．2003 年早稲田大学理工学部情報学科卒業．2005 年同大学院理工学研究科情報・ネットワーク専攻修士課程修了．同年株式会社日立製作所情報・通信グループソフトウェア事業部入社，現在に至る．日本データベース学会会員．

山名 早人（正会員）

1964 年生．1987 年早稲田大学理工学部電子通信学科卒業．1989 年同大学院修士課程修了．1993 年同大学院博士課程修了．博士（工学）．1989〜1993 年早稲田大学情報科学研究教育センター助手．1993〜2000 年通商産業省工業技術院電子技術総合研究所．2000 年早稲田大学理工学部助教授．2005 年早稲田大学理工学術院教授，現在に至る．1995 年情報処理学会山下記念研究賞，2000 年情報処理学会ベストオーサ賞を受賞．情報検索等の研究に従事．電子情報通信学会，IEEE，ACM 各会員．