DEWS2004 6-A-02

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

平手　勇宇† 　岩橋　永悟†† 　山名　早人††

† 早稲田大学理工学部　〒 169-8555 東京都新宿区大久保 3-4-1
†† 早稲田大学大学院理工学研究科　〒 169-8555 東京都新宿区大久保 3-4-1
E-mail: †hirate@yama.info.waseda.ac.jp, ††eigo@yama.info.waseda.ac.jp, ††yamana@waseda.jp

あらまし　データマイニング分野での頻出パターン抽出手法は、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パターンを抽出する手法である。最小サポート値から抽出される頻出パターン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 k パターンを抽出する Top-k Mining コンセプトが近年提案されている。しかし、Top-k Mining コンセプトとして提案されているアルゴリズムは、k の値が大きくなればなるほど指数関数的に実行時間が増加し、ユーザは頻出上位 k 個のパターンの抽出が終了するまで待たなければならない。これに対し本稿では、FP-growth を基にして、Top-k Mining コンセプトで、サポート値の高いパターンから計算していき、計算でき次第順次ユーザへ返すことによって、ユーザ応答時間を短くしたアルゴリズム (Pipelined Top k FP-growth) を提案する。

キーワード　データマイニング、頻出パターン、Top-k、FP-growth

Mining Top-k frequent patterns algorithm focusing on the reaction time to users

Yuu HIRATE†, Eigo IWAHASHI††, and Hayato YAMANA††

† Faculty of Science and Engineering, Waseda University　3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555 Japan
†† Graduate School of Science and Enginnering, Waseda University　3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555 Japan
E-mail: †hirate@yama.info.waseda.ac.jp, ††eigo@yama.info.waseda.ac.jp, ††yamana@waseda.jp

Abstract　In Data Mining researches, frequent patterns mining algorithms require some user-specified minimum support, and mine frequent patterns whose support values are higher than the minimun support. In recent years, since it is difficult to predict how many frequent patterns are mined with a minimum support, Top-k Mining concept is proposed. Top-k Mining concept is based on the algorithm mining frequent patterns without a minimum support and with the number of most k frequent patterns (ordered by their support values). However, Top-k Mining concept algorithms are slow when k is large. Therefore, users have to wait until the finish of mining the most k frequent patterns. In this paper, We propose a new Top-k Mining concept algorithm based on FP-growth, called "Pipelined Top k FP-growth". The pipelined Top k FP-growth mines patterns with the decending order of their suport values. Then, it returns frequent patterns to users sequentially to shorter the response time.

Key words　DataMining,Frequent Pattern,Top-k,FP-growth

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、大容量化に伴い、大量のデータが蓄積することが一般的になってきた。しかし、集められた大量のデータは記号の列に過ぎない。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。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パターン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パターン、即ち頻出パターンを効率よく抽出する手法の研究が行われている。

2000 年までに提案されてきた相関ルール抽出アルゴリズムは、94 年に発表された、Apriori [1] に基づくアルゴリズムであった。Apriori は生成する候補アイテムセットの数が多く、メモリを多く消費してしまう問題や、データベースに対して繰り返しスキャンをしてしまう問題点があり、頻出パターンを算出するまでに時間がかかってしまう。しかし、2000 年に Apriori をベースとしない、FP-growth [2] が提案された。FP-growth は、候補アイテムセットを生成せずに、2 回のデータベーススキャンだけで、頻出アイテムセットを算出することができる。結果、FP-growth はこれまでの Apriori をベースとしたアルゴリズムと比べて、革新的な速度を実現した。

高速に頻出パターンを算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全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上での頻出パターン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の例として Maximal Pattern Mining [3] [5]、Closed Pattern Mining [6] [5]、Top-k Mining [7] [8]、Incremental Mining [9] が挙げられる。Maximal Pattern Mining と Closed Pattern Mining は、頻出アイテムセット集合の中で $subset$ となるアイテムセットを除外し、$superset$ のみを抽出するマイニング手法である。Top-k Mining は、最小サポート値を必要とせず、単純にサポート値の高いパターンから k 個だけを抽出するマイニング手法である。Incremental Mining は、データセットの変化、最小サポート値の変化に対して、二度目以降のマイニング速度を向上させたマイニング手法である。

データマイニングの実際の応用を考えた時、サポート値降順 k パターンを抽出する 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 Ada Wai-chee Fu らによって提案された Apriori [1] ベースの Itemset-Loop / Itemset-iLoop [7] と、2002 年に Han らによって提案された FP-growth ベースの TFP [8] である。Itemset-Loop / Itemset-iLoop は、パターンの要素数の上限を指定し、上限以下の要素数で構成されたサポート値降順 k パターン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ってしまう。TFP は、パターン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 Closed Pattern [6] のみを対象として、サポート値降順 k パターンを抽出するアルゴリズムである。

しかし、従来の Top-k Mining は抽出するパターン数 k が大きいと、実行時間が指数関数的に増加する。したがって k の値を大きくして実行した場合、応答時間の長時間化という点においてユーザビリティが低いと考えられる。本稿では、応答時間の短縮に着目した Top-k Mining として、2000 年に Han らによって考案された FP-growth を基にし、抽出したい最大アイテムセット数 k を指定して、サポート値降順に頻出アイテムセットを求め、順次ユーザに表示する手法を提案する。

以下、第 2 節では、本論文で用いる用語を定義する。第 3 節では、関連研究として頻出パターン抽出アルゴリズムを述べる。第 4 節では提案アルゴリズムについて述べる。第 5 節では、第 4 節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、第 6 節でまとめを行う。

2. 用語定義

本論文で用いる用語を定義する。

サポート (support)　アイテム集合を $I = \{i_1, i_2, \ldots, i_m\}$、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。パターン X のサポート $sup(X)$ は、T 全体に対して X を含むトランザクションの割合を表す。

頻出パターン　頻出パターンとは、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パターンである。

Top-k 頻出パターン　アイテムセットをサポート値降順に並び替え、k 番目に配置されたパターンのサポート値を α と置く。Top-k 頻出パターンとは、α 以上のサポート値を持つパターンである。

k-itemset　k-itemset とは、k 個のアイテムから構成されるアイテム集合である。

3. 関連研究

本節では、頻出パターン全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 Maximal Pattern Mining 手法、Closed Pattern Mining 手法、Top-k Mining 手法、Incremental 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 [1]

Agrawal らによって 1994 年に提案された Apriori アルゴリズムは、効率的に全ての相関ルールを発見できる手法である [1]。Apriori は、後述する FP-growth [2] が発表された 2000 年まで、多くの派生的な改良を生んだベーシックなアルゴリズムである。Apriori は、「長さ k の非頻出アイテムセットを含む長さ $k+1$ のアイテムセットは、必ず非頻出アイテムセットである」という理論の基、ボトムアップに頻出アイテムセットを数え上げるアルゴリズムである。

Apriori アルゴリズムの欠点は、候補アイテムセットの数が莫大になることである。また、候補アイテムセットをチェックするためにデータベースを繰り返しスキャンする必要がある。

3.1.2 FP-growth [2]

Apriori のように候補アイテムセットを生成すると、パターンが多く存在する場合、候補アイテムセットを格納するために必要となるメモリ容量が大きくなる。この問題を解決するために、候補アイテムセットを生成しない FP-growth アルゴリズムが 2000 年に Han らによって提案された [2]。このアルゴリズムは、特殊なデータ構造である FP-tree 構造を利用しており、FP-tree 構造を参照するだけで全ての頻出パターンを数え上げることができる。FP-tree は巨大なデータベースが小さく圧縮されたデータ構造であり、スキャンの繰り返しを減らすことができる。

a) FP-tree の構築

FP-tree 構造は、頻出パターンを抽出するために必要な情報を

全て保持する。頻出アイテムを抽出するために、TDB を一度スキャンする必要がある。抽出された頻出アイテムを頻度が降順になるように並び替える（並び替えたリストを $F-list$ とする）。そして空（null）のラベルを持つ木のルートを作る（この木を T とする）。2 回目のデータベーススキャンでは、各トランザクションごとに以下の処理を行う。

（1） $F-list$ に含まれるアイテムのみを、トランザクションから抽出し、抽出したアイテムを、サポート値降順に並び換える。これを、Frequent Items と呼ぶ。

（2） Frequent Items（サポート値順）のアイテム一つずつ順番に、T と比較する。T が Frequent Items の要素である子ノードを持っていれば、その子ノードのカウントを 1 増やす。T が Frequent Items の要素である子ノードを持っていないときは、新しくカウント 1 を持つ子ノードを作る。

（3） Frequent Items の最後の要素まで (1)、(2) の操作を繰り返す。

全てのトランザクションで上記の処理を終えたら、同じ名前（アイテム ID）を持つノードにリンクを付ける。さらに、各頻出アイテム a_i に対して、頻出アイテム a_i 自身と、a_i のノードリンクのヘッドを格納するヘッダテーブルを生成する。

$min\_sup = 3$ として、表 1 のような TDB から作成した、FP-tree を図 1 に示す。

表 1　サンプル 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, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

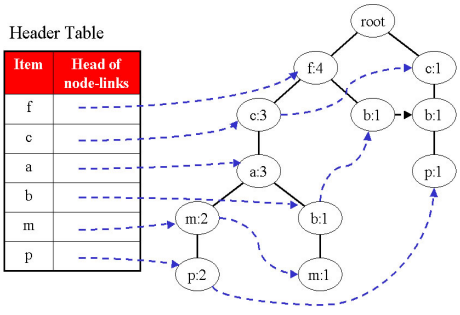

図 1　表 1 から構築された FP-tree 構造 (文献 [2] より引用)

b）　FP-growth

FP-growth は、構築した FP-tree を基に頻出パターンを抽出する手法である。全ての頻出パターンを抽出するために、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アイテム 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、FP-tree 内のアイテム a_i を格納しているノードを全てたどることができる。また、アイテム 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 を含む全ての頻出パターンを抽出する。

例として図 1 中の FP-tree よりアイテム p を含んだ頻出パターン抽出過程を述べる。図 1 中の FP-tree におけるヘッダテーブルより頻出パターン (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、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、たった一本の枝 $(c:3)$ しか得られない。ゆえにたったひとつの頻出パターン $(cp:3)$ しか得られない。したがって、p を含む頻出パターンは $(p:3), (cp:3)$ である。

3.2　コンセプトマイニング手法

3.2.1　Maximal Pattern Mining

頻出パターン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算出する。しかし、必要以上に多くの頻出アイテムセットから、有用な知識を発見することは困難である。Maximal Pattern Mining は、このような問題を解決するために、min_sup を満たす頻出アイテムセットの中から **Maximal Frequent Itemset** のみを抽出するマイニングのことを指す。Maximal Pattern Mining アルゴリズムとして、Max-Miner [3]、FPmax [4] などが提案されている。

なお、アイテムセット X が **Maximal Frequent Itemset** であるということは、「アイテムセット X のサポート値が min_sup 以上であり、かつ X のスーパーセットである任意の X' が、min_sup 未満のサポート値である」ことである。

3.2.2　Closed Pattern Mining

Max Pattern Mining と同じように、Closed Pattern Mining 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、**Frequent Closed Itemset** のみを抽出するマイニングである。Closed Pattern Mining アルゴリズムとして、CLOSET [6],FP-close [5] などが提案されている。

なお、アイテムセット X が **Frequent Closed Itemset** であるということは、「アイテムセット X のサポート値が min_sup 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 の全てののスーパーセット X' が、min_sup 未満のサポート値である」ことである。

3.2.3　Top-k Mining

一般に最小サポート値から、結果として抽出される頻出パターン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パターンが抽出され、逆に最小サポート値が少しでも大きいと、ほとんど頻出パターンは抽出されない。このように、適切な min_sup を選択することは、一般的に難しい。このため、min_sup を閾値として必要とせず、Top-k 頻

出パターンを抽出する Top k Mining が提案されている [7] [8]。

3.2.4 Incremental Mining

一般に頻出パターン抽出において、適切なアイテムセットを抽出することができるまで、試行錯誤をして min_sup の値を変えなければならない。さらに、一般社会において、頻出パターン抽出対象の TDB も、トランザクションレコードの追加、削除、更新などをおこない絶えず変化をしている。しかし、従来の頻出パターン抽出アルゴリズムは、min_sup の値を変えたり、TDB が変化するとはじめからやり直さなくていけないアルゴリズムであった。

このような背景から、min_sup の変化、TDB の変化に対して、最初にマイニングを行ったときに算出される資源を 2 度目以降のマイニングに適用し、2 度目以降のマイニングを高速に行うことに主眼を置いた、FELINE [9] などが、Incremental Mining として提案されている。

3.2.5 コンセプトマイニングまとめ

コンセプトマイニングアルゴリズムの特徴の一覧を、表 2 に示す。

表 2　コンセプトマイニングアルゴリズムの特徴

アルゴリズム	特徴
Maximal Pattern Mining	
Max-Miner('98) [3]	・深さ優先探索を行う。 ・Maximal Pattern の最大アイテムセット上を l とおくと、***TDB*** へのスキャン回数は、$l+1$ 回である。
FPmax('03) [4]	・深さ優先探索を行う。 ・FP-growth を基にしたアルゴリズムである。
Closed Pattern Mining	
CLOSET('00) [6]	・FP-growth を基にして、Closed Pattern を抽出する。 ・***TDB*** を複数の探索空間に分割して、各探索空間ごとに FP-tree を構築する。 ・***TDB*** へのスキャン回数は 2 回である。
FP-close('03) [5]	・FP-growth を基にして、Closed Pattern を抽出する。 ・深さ優先探索を行う。
Top-k Mining	
Itemset-Loop / Itemset-iLoop('00) [7]	・Apriori を基にしたアルゴリズムである。 ・最大アイテムセット長 m 以下の頻出パターンの中でサポート値降順 k 個を抽出する。
TFP('02) [8]	・FP-growth を基にしたアルゴリズムである。 ・アイテムセット長 l 以上の CLosed Pattern のうち、サポート値降順 k 個を抽出する。
Incremental Mining	
FELINE('03) [9]	・FP-growth を基にしたアルゴリズムである。 ・***TDB*** の情報を、全て CATS-tree に保持してマイニングを行う。

4. 提 案 手 法

4.1 従来手法の問題点

データマイニングの実際の応用を考えた時、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 年に提案された Itemset-Loop / Itemset-iLoop と、2002 年に提案された FP-growth ベースの TFP である。従来のマイニング手法において、min_sup を小さくすると、マイニング実行時間が指数関数的に増加するのと同様に、Top-k Mining アルゴリズムも、k の値が大きくなると、マイニング実行時間が増加する。したがって、k の値が大きい場合、応答時間の長時間化という点においてユーザビリティが低い。

4.2 提案手法の概要

提案手法は、サポート値の大きいパターンから計算していき、順次ユーザに結果を返すことによって、k の値が大きい場合でも応答時間を短縮化させた手法 (Pipelined Top k FP-growth) である。以下では、まず Pipelined Top k FP-growth のベースとなる Top-k Mining である Top k FP-growth について 4.3 で提案する。次に、Top k FP-growth を拡張し、サポート値上位の頻出パターンから順次ユーザに返すことによって、応答時間を短縮化させた提案手法である Pipelined Top k FP-growth について 4.4 で提案する。

4.3 Top k FP-growth

Top k FP-growth は、Pipelined Top k Fp-growth のベースとなる Top-k Mining アルゴリズムであり、本論文で提案する手法である。本手法は、FP-growth [2] をベースとして、min_sup 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、全ての Top-k 頻出パターンを抽出する。

4.3.1 FP-growth からの拡張

Top k FP-growth のパラメータは min_sup 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 頻出パターンを抽出するために 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 からのパターン生成数の削減、パターン生成後の出力がある。

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。即ち、1-itemset 集合の中で、$Border\_sup$ 以上のサポート値を持つアイテムは、k 個存在する。

a) 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 Frequent Items を、min_sup を超えるサポート値を持つ 1-itemset でなく、$Border\_sup$ を超えるサポート値を持つ 1-itemset とする。

［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、全ての Top-k 頻出パターン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 $\{\beta\}$ と置くと、

$$Sup(\{\alpha,\beta\}) \leq Sup(\{\alpha\}) < Border\_Sup$$

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、サポート値降順に k 個の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。

例えば、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、$k = 6$ の時の $Border\_sup$ は、3 となる。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。

b) FP-tree からのパターン生成数の削減

構築した FP-tree から全てのアイテムに関してノードリンクをたどり、パターンを生成すると k パターン以上抽出される。したがって Top k FP-growth では、Reduction Array という配列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パターン生成数を削減することによって、計算量を削減する。

Top k FP-growth では、FP-tree から生成されたパターンと、パターンのサポート値を、逐一 Reduction Array と呼ばれる配列に格納していく。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。

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 からのパターン生成開始時には 0 であるが、k 個以上のパターン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、あるアイテム α 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アイテム β のノードリンクをたどる前に、β のカウント値 (=$sup(\beta)$) と、上記に示した $Boundary\_sup$ を比較を行う。そして、$sup(\beta) \geq Boundary\_sup$ であれば、β のノードリンクをたどりパターンを生成していくが、もし $sup(\beta) < Boundary\_sup$ であれば、FP-tree からのパターン生成を終了する。

$Boundary\_sup$ を下回るサポート値を持つアイテムに関して、ノードリンクをたどってパターン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 β のノードリンクをたどって抽出された β を含む任意のパターンは、$sup(\beta)$ 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、β を含む任意のパターンは、$Boundary\_sup$ 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 β よりも下に配置されている任意のアイテムセット γ と、γ をたどって抽出されるパターンも、$sup(\beta)$ よりも小さな値を持つ。よって、γ を含む任意のパターンについても、$Boundary\_sup$ よりも小さなサポート値を持つこととなる。以上より、$sup(\beta) < Boundary\_sup$ の時、FP-tree からのパターン生成を終了しても、過不足なく全ての Top-k 頻出パターンが生成される。

例えば、表 1 の TDB に対して、$k = 6$ で FP-tree を構築し、アイテム $\{a\}$ のノードリンクをたどり終えた時の Reduction Array を図 2 に示す。図 2 で、Reduction Array の 6 番目に配置されているアイテムセット $\{f, c, a\}$ のサポート値が 3 であるため、$Boundary\_sup = 3$ とおかれる。次に、Header Table においてアイテム $\{a\}$ のノードリンクをたどり終えた状態を考える。この時、次のアイテム $\{b\}$ のノードリンクをたどる前に、アイテム $\{b\}$ 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、アイテム $\{b\}$ のノードリンクをたどり、パターンを生成するが、$sup(b) < Boundary\_sup$ であればノードリンクをたどらない。このようにして、パターン生成を行う。

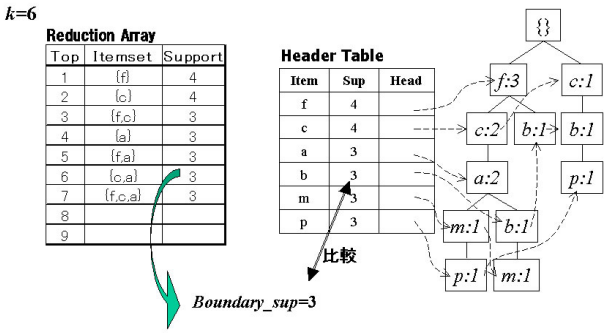

図 2　$k = 6$ の時のアイテム $\{a\}$ のノードリンクをたどり終えた後の Reduction Array

c) パターン生成後の出力

上記の a)b) の拡張を行っても、FP-tree から抽出したパターンは、k 個以上存在する。したがって、FP-tree からパターンを抽出した後、Reduction Array よりサポート値降順 k アイテムセットをマイニング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 サポート値降順上位 k 個の頻出アイテムセット

METHOD

（1） TDB をスキャンして、全ての 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 を基にして、FP-tree を構築する。

（4） 構築した FP-tree からパターンを生成する。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、パターン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 3 に示す。

4.4 Pipelined Top k FP-growth

Top k FP-growth は、従来の Top-k Mining と同様に、k の値が大きくなると実行時間が増加する。k の値が大きい場合、応答時間の長時間化によって、ユーザは長い時間待たされ解析を始めることができないことが問題となる。Pipelined Top k FP-growth は、Top FP-growth を基にしたアルゴリズムで、$n = 10^i, i \geq 2$

表 3　FP-growth [2] と Top k FP-growth の違い

	FP-growth [2]	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 からのパターン生成対象	FP-tree のノード全て	$Boundary\_sup$ によって生成数削減
パターン生成後	生成したアイテムセット全てを結果として抽出	サポート値順に並び換えて、上位 k アイテムセットを結果として抽出

（デフォルト値(注1)）を満たす Top n を順番に抽出していき、最終的に Top k を抽出するアルゴリズムである。n の値はユーザが自由に設定できる。Pipelined Top k FP-growth は、計算でき次第サポート値上位のパターンをユーザに返すことによって、ユーザ応答時間を短くできる。

デフォルトの値である $n = 10^i, i \geqq 2$ の場合においてのマイニングプロセスは、次のようになる。k が $100 < k \leqq 1000$ の場合は、はじめに Top100 を抽出してから Top k を抽出する。$1000 < k \leqq 10000$ の場合は Top100,Top1000 を抽出してから、Top k を抽出する。Pipeline Top k FP-growth の利点は、以下に挙げるとおりである。

例えば、$k = 3000$ としたとき、Top 3000 の算出に必要な時間は、Top 100 の算出に必要な時間よりも長い。このため、Top 3000 を一度に算出する時間、ユーザは待たされることとなる。しかし、100 個を早い段階で算出していけば、Top 100 が算出できた時点でユーザは結果の解析を始めることができる。したがって、ユーザによる解析まで含めた合計時間は、Pipelined Top k FP-growth の方が、Top FP-growth よりも短くなると考えられる。

4.4.1　Pipelined Top k FP-growth のアルゴリズム

Piprlined Top k FP-growth の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 サポート値降順上位 k 個の頻出アイテムセット
METHOD

（1） TDB をスキャンして、全ての 1-itemset のサポート値をカウントする。

（2） $i = 1$ とする。

（3） $k > 10^{i+1}$ の場合は、$n = 10^{i+1}$ とし、$k \leqq 10^{i+1}$ の場合は、$n = k$ とする。

（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 を基にして、FP-tree を構築する。

（6） 構築した FP-tree からパターンを生成する。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、パターン生成数を低減させる。

（7） 抽出した頻出アイテムセットを、サポート値降順にソートし、上位 n アイテムセットを結果として返す。

（8） $k = n$ の場合は、マイニングを終了し、$k > n$ の場合は、i をインクリメントして、(3) に戻る。

(注1)：一度結果をユーザに返せば、ユーザは解析を開始できるので、2 度目以降の応答時間はそれほど重要でなく、アルゴリズム全体の速度を向上させるために、デフォルト値 $n = 10^i, i \geqq 2$ を選んだ。

5. 性能評価

第 4 節で述べた Pipelined Top k FP-growth を実装し、ユーザ応答時間、従来手法である FP-growth との実行時間比較、大規模データベースに対してのスケーラビリティを評価した。

5.1　実験環境

第 4 節で提案した手法を、Intel Pentium4 プロセッサ 2.4GHz と、1GB のメモリを搭載した計算機 1 台で実行した。なお、実装した手法を評価するデータセットとして、IBM の人工データ生成プログラム [10] を用いてデータセットを生成した。生成したデータセットは、T10I4D1000k である。T10I4D1000k はトランザクション総数が 1000,000(1000k)、トランザクションあたりのアイテム数が 10、パターンの平均最大長が 4 というパラメータを持つデータセットである。

5.2　ユーザ応答時間の評価

Pipelined Top k FP-growth において k の値を $100 \leqq k \leqq 10000$ の範囲で、500 ずつ増やして実行時間を測定した。抽出したアイテムセット数 (Top k) と実行時間の関係を図 3 に、$Border\_sup$ と実行時間の関係を図 4 に示す。

図 3、図 4 の凡例について説明する。

First Result Pipelined Top k FP-growth で、Top 100 を抽出するまでの時間

Secound Result Pipelined Top k FP-growth で、Top 1000 を抽出するまでの時間

Total Result Pipelined Top k FP-growth で、最終的に Top k を抽出し終えるまでの時間

Direct 一度に Top k を抽出するまでの時間、即ち Top k FP-growth の実行時間

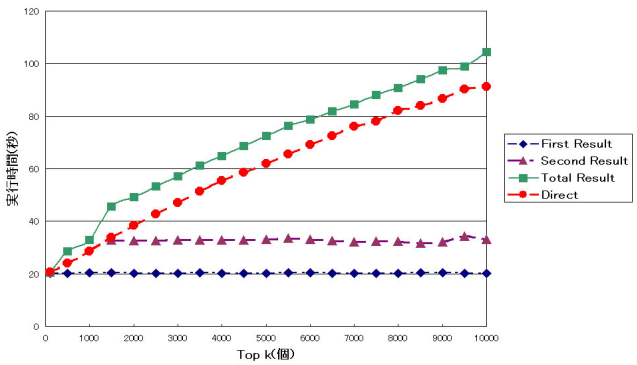

図 3　Top k と Pipelined Top k FP-growth の実行時間

また、Direct の時間を基準にして、First Result, Second Result, Total Result の時間の割合と抽出したアイテムセット数 (Top k) の関係を図 5 に示し、同じく Direct の時間を基準にして、First Result, Second Result, Total Result の時間の割合と $Border\_sup$(%)

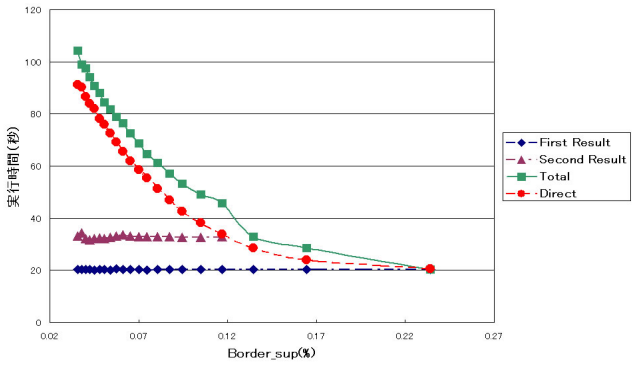

図 4　$Border\_sup$ と Pipelined Top k FP-growth の実行時間

の関係を図 6 に示す。図 5、図 6 の凡例について説明する。

First Result

$$\frac{Pipelined Topk FP-growth \text{ で } Top100 \text{ を抽出するまでの時間}}{Direct \text{ で算出した時間}} \times 100$$

Secound Result

$$\frac{Pipelined Topk FP-growth \text{ で } Top1000 \text{ を抽出するまでの時間}}{Direct \text{ で算出した時間}} \times 100$$

Total Result

$$\frac{Pipelined Topk FP-growth \text{ で } Topk \text{ を抽出するまでの時間}}{Direct \text{ で算出した時間}} \times 100$$

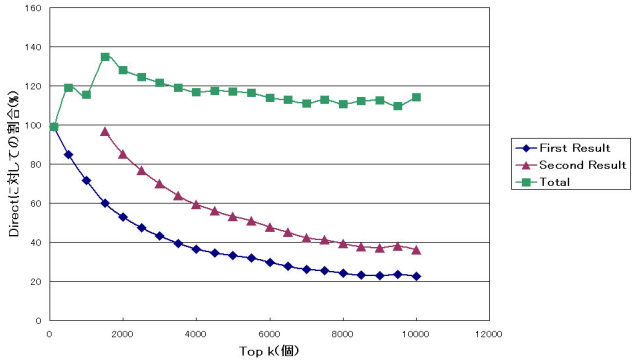

図 5　Direct を基準とした実行時間の割合と Top k の関係

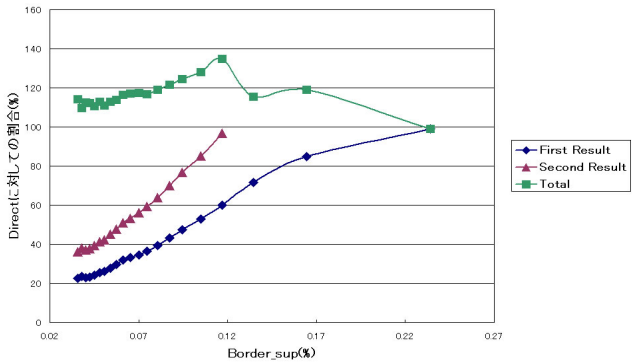

図 6　Direct を基準とした実行時間の割合と $Border\_sup$ の関係

図 3、図 4 より、Pipelined Top k FP-growth の全体実行時間よりも Top FP-growth の実行時間の方が早いことがわかる。これは、Pipelined Top k FP-growth の全体実行時間は、Top k を抽出する時間に加え、Top 100, Top 1000 を抽出する実行時間が加算されることに起因する。また、$k = 100$ から $k = 500$ に変化する時、Pipeline Top k FP-growth の全体実行時間の増分が大きくなってしまっている。これは、$k = 100$ の時は TDB を読み込み Top 100 パターンを抽出してマイニングが終了するのに対し、$k = 500$ の時は、TDB を読み込み、一度 Top 100 パターンを抽出した後、Top 500 パターンを抽出することでマイニングを終了することに起因する。

しかし、Pipelined Top k FP-growth では、最初にユーザに結果を返す時間を k の値をいくつに設定しても、20 秒程度である。例として、$k = 3000$ を考える。一度にサポート値降順 3000 パターンを抽出するには、約 47 秒かかるのに対し、Pipelined Top k FP-growth では、約 20 秒最初の結果をユーザに返すことができる。

図 5、図 6 を見ると、Pipelined Top k FP-growth は、Top k FP-growth と比べて 120%前後の時間がかかってしまうという欠点が存在するが、サポート値降順 100 パターンは、Direct に対して $k = 1000$ の時で約 70%、$k = 3000$ の時で約 40%、$k = 10000$ の時で、約 20%の時間でユーザに返すことが可能となる。また、サポート値降順 1000 パターンは、Direct に対して $k = 3000$ の時で約 70%、$k = 5000$ の時で約 50%、$k = 10000$ の時で約 40%の時間でユーザに返すことが可能となった。

5.3　FP-growth との実行時間比較評価

Pipelined Top k FP-growth との比較のために、FP-growth アルゴリズムを $0.09 \leqq min\_sup \leqq 0.23$ の範囲で、min_sup を 0.01 ずつ増やして、実行時間を測定した。抽出したアイテムセット数 (Top k) と実行時間の関係を図 7 に、$Border\_sup$ と実行時間の関係を図 8 に示す。FP-growth は、min_sup 以上のサポート値を持つ全ての頻出パターンを抽出するアルゴリズムであるため、Pipelined Top k FP-growth と単純に比較することはできない。したがって、図 7、図 8 においての FP-growth 実行時間のプロット方法を説明する。

- 図 7 においては、各 min_sup で、抽出されたパターン数を横軸（Top k）にとり、実行時間を縦軸にプロットした。
- 図 8 においては、各 min_sup で抽出された頻出パターン数から、Top k FP-growth で抽出するために必要な最大の $Border\_sup$ を求める。求めた $Border\_sup$ を、横軸 ($Border\_sup$) にとって実行時間を縦軸にプロットした。

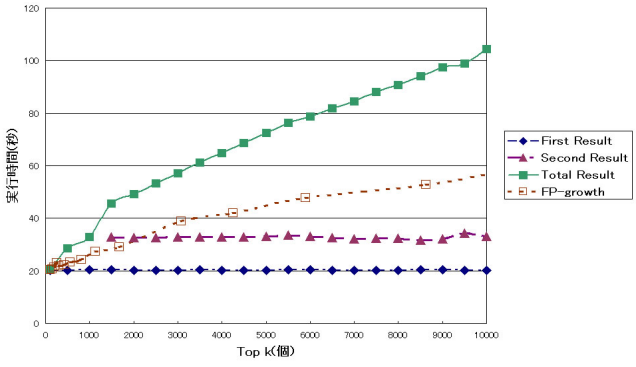

図 7　Top k を基準とした Pipelined Top k FP-growth と FP-growth の比較

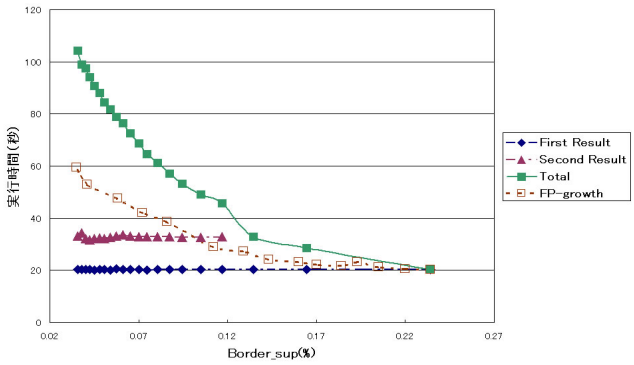

図 8　$Border\_sup$ を基準とした Pipelined Top k FP-growth と FP-growth の比較

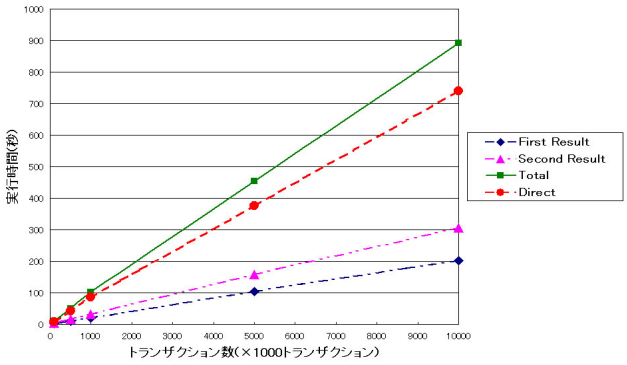

図 9　トランザクション数と実行時間の関係

図 7、図 8 を見ると、FP-growth は、Pipelined Top k FP-growth(Total) の約 60%の実行時間で頻出パターンを抽出する。これは、同じパターン数を抽出するにあたって、Pipelined Top k Fp-growth の方が、

- 多くの 1-itemset を対象にして FP-tree を構築していること
- FP-tree からのパターン生成において、多くのパターンを抽出していること

が挙げられ、また Pipelined Top k FP-growth は、FP-growth に比べ、余分に

- FP-tree のノードリンクをたどり終えるたびに、生成したパターンをサポート値降順に並び替えること

を行っているからである。

しかし、Pipelined Top k FP-growth アルゴリズムは、あらゆる Top 数、あらゆる $Border\_sup$ においても、FP-growth よりも短時間で最初の結果をユーザに返すことが分かる。

5.4　大規模データベースに対してのスケーラビリティの評価

Pipelined Top k FP-growth の大規模データベースに対してのスケーラビリティを評価するために、同じく IBM の実行データ生成プログラム [10] を用いて、トランザクション数の違う新たなデータセット T10I4D100k、T10I4D500k、T10I4D5000k、T10I4D10000k を生成した。これら 4 つのデータセットと、T10I4D1000k の 5 つのデータセットを対象に、$k = 10000$ での Pipeline Top k FP-growth の実行時間を測定した。図 9 にトランザクション数と実行時間の関係を示す。凡例は図 3 と同じである。図 9 を見ると、First Result、Second Result、Total の実行時間が、トランザクション数に比例していることが分かる。これは、データセットが大きくなればなるほど、Total の実行時間と、First Result の実行時間の差が大きくなることを意味する。即ち、データセットが大きくなればなるほど、Pipelined Top k FP-growth のユーザ応答時間を短くする目的が達成できる。

6.　おわりに

本稿では、2000 年に Han らによって考案された FP-growth を基にし、抽出したい最大アイテムセット数 k を指定して、サポート値降順に頻出アイテムセットを求め、順次ユーザに表示する手法を提案した。本手法を T10I4D1000k に対して適用したところ、サポート値上位 3000 パターンを FP-growth により一度に求める場合（予め上位 3000 パターンに対応する最小サポート値が既知と仮定）に比較し、最初の上位 100 パターンを約 50%の処理時間で、また全 3000 パターンを約 140%の処理時間でユーザに提示することができ、ユーザ応答性を向上させることができた。

本稿で提案した手法は、ボトルネックとなっていた I/O 関係の処理を並列に実行することで、高速化を期待することができる。今後は、Pipeline Top k FP-growth を並列化して、性能を評価していきたい。

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」によるものである。

文　　献

[1] R.Agrawal, R.Srikant. “ Fast algorithms for mining association rules ”. In VLDB ’94, pp. 487-499, Santiago, Chile, Sept. 1994.

[2] J. Han, J. Pei, and P.S. Yu,“ Mining frequent Patterns without Candidate Generation, ”In Proceedings of the ACM SIGMOD Conference on Management of Data pp.1-12, 2000.

[3] Bayard, R.J. ”Efficiently mining long patterns from databases”, In Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 85-93, 1998.

[4] G.Grahne and J.Zhu, ”High performance mining of maximal frequent itemsets,” In SIAM’03 Workshop on High Performance Data Mining, May 2003.

[5] G. Grahne and J. Zhu, ”Efficiently Using Prefix-trees in Mining Frequent Itemsets,” In Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Inplementations, 2003.

[6] J.Pei, J.Han, and R.Mao, ”CLOSET: An efficient algorithm for mining frequent closed itemsets,” In DMKD’00, 2000.

[7] A.W.-C Fu. R.W.-W. Kwong, and J.Tang, ”Mining n-most interseting itemsets,” In Proceedings of the ISMIS’00, 2000.

[8] J. Han, J. Wang, Y. Lu, and P. Tzvetkov, ”Mining Top-K Frequent Closed Patterns without Minimum Support,” In Proceedings of IEEE ICDM Conference on Data Mining, 2002.

[9] W. Cheung and Osmar R. Zaiane, ”Incremental Mining of Frequent Patterns Without Candidate Generation or Support Constraint,” 7th International Database Engineering and Applications Symposium (IDEAS 2003), Hong Kong, China, July 16-18, 2003.

[10] IBM Quest Data Mining Project. Quest synthetic data generation code. http://www.almaden.ibm.com/cs/quest/syndata.html

[11] 岩橋永悟, 山名早人:”FP-growth の並列化による頻出パターン抽出の高速化”, 情処研報 (DBS),Vol.2003,No.71,pp.327-334 (2003.7).