高次知識を獲得するリソース指向型オンラインマイニング法の開発

山本　泰生

ストリームデータを対象とするデータストリームマイニングの研究が活発に進められている．データストリームマイニングは大規模なストリームデータの新しい知的価値を創出する基本的な技術であり応用分野は多岐にわたる．本稿ではその中の基礎問題の一つであるトランザクションストリーム上の頻出パターンマイニングに焦点を当てながら，さきがけ研究の中で取り組んでいるオンライン近似アルゴリズムとパターンの圧縮に関する研究内容を概説する．

キーワード：ストリームデータ，データストリームマイニング，リアルタイム処理，近似圧縮

1. はじめに

クラウドサービスやIoTの発展に伴い，気象，小売，製造，インフラ，観光，医療，スポーツなどの多岐にわたる分野において，多種多様なストリームデータが生み出されている．ストリームデータとは，複数のエッジノードから高速に生成される無限長のデータ系列のことであり，さまざまな応用可能性をもつとされる．たとえば，プラント制御や気象観測，交通システム，計算クラスタなどの監視系では，各種イベントの発生傾向の変化や異常をいち早く検知することが求められる．このようなリアルタイム分析は，スマートシティやBDD (Big Data and Disaster) などの新しいビッグデータ分野でも活用が期待されている．

高速に流れ続けるストリームデータの特徴は，時間経過とともに蓄積されるデータ総量が急速に増加する点にある．たとえばWalmartでは毎年数十億の販売履歴データが生成され，Twitterでは毎日100 TBのWebログデータが蓄積される [1]．よって，ストリームデータをリアルタイム分析するタスクでは，ディスク上のデータ走査は極力避けながら，新規データを逐次的にインメモリ処理することが求められる．このようなオンライン処理はストリームデータの管理・運用技術として重要な研究課題の一つと位置づけられる．

この中で本稿ではトランザクションストリーム上の頻出パターンマイニング（frequent pattern mining from a transactional stream，以降，FPM-TSと略す）に焦点を絞り，高速・省メモリなオンライン近似アルゴリズムを概説する．構成は次のとおりである．2節で背景を述べ，計算タスクとしてのFPM-TSの意義や難しさを説明する．3節では著者らが提案するオンライン近似圧縮の手法とその性質を紹介する．4節で提案法の拡張について述べた後，5節でまとめる．

2. 背景

図1のように複数のエッジノードからデータが随時到着するストリームを考える．各データは解析対象となるイベントやカテゴリ属性変数に相当しアイテムと呼ばれる．アイテムの全種類数[1]をuとすると，アイテムの全体集合Iは$\{x_1, x_2, \ldots, x_u\}$と書ける．ある時間区間内に到着するアイテムの集合をトランザクションと呼ぶと，ストリーム$\mathcal{S}_n$は可変長のトランザクション列$\langle t_1, t_2, \ldots, t_n\rangle$と表すことができる．ただし$t_i$は$i$番目（以降，$i$を時刻と呼ぶ）に到着するトランザクションであり，$t_i \subseteq I$かつ$t_i \neq \emptyset$を満たすものとす

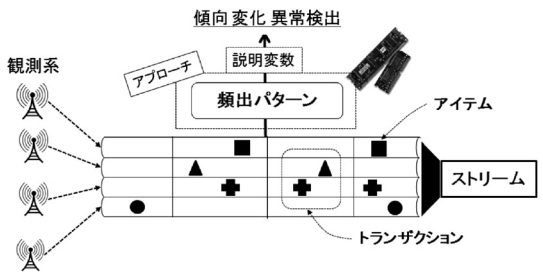

図1　ストリームデータマイニングとリアルタイム分析

[1] 本稿では離散構造をもつデータを想定する．連続値をとるデータの場合，符号化 [2] などの前処理が必要となる．

やまもと　よしたか
山梨大学大学院総合研究部
〒400–8511 山梨県甲府市武田 4–3–11
独立行政法人科学技術振興機構，さきがけ
〒332–0012 埼玉県川口市本町 4–1–8
yyamamoto@yamanashi.ac.jp

る．t_i の要素数を t_i のトランザクション長と呼び，ストリーム上の最大トランザクション長を L とする．また n は現在の時刻を表す．

FPM-TS にはさまざまなバリエーションがある．本稿ではその一般クラスとしてアイテム集合系列を対象とする問題を取り上げる．ストリーム $\mathcal{S}_n$ が与えられた時，$P = \langle s_{k_1}, s_{k_2}, \ldots, s_{k_r} \rangle$ を $\mathcal{S}_n$ 上の集合系列と呼ぶ．ただし各 k_j は時刻 $(1 \leq k_j \leq n)$ であり，$k_1 < k_2 < \cdots < k_r$ を満たす．また各 s_{k_j} はトランザクション t_{k_j} の空でない部分集合とする．s_{k_1} と s_{k_r} をそれぞれ P の先頭と末尾と呼び，r を P の系列長と呼ぶ．簡単のため系列中の集合間のコンマは略記する．

例 1. アイテムの全体集合 I を $\{a, b, c, d\}$ とし，ストリーム $\mathcal{S}_4^1$ を $\langle \{d\}\{a, b, c\}\{b, c\}\{a, c, d\} \rangle$ とする．$P = \langle \{a, b, c\}\{d\} \rangle$ と $Q = \langle \{a\}\{b, c, d\} \rangle$ を考えると，二つのうち $\mathcal{S}_4^1$ 上の集合系列は P のみである．

時刻 i $(1 \leq i \leq n)$ において P が $\mathcal{S}_i$ 上に出現するとき，P は時刻 i でサポートされるという．ただし同一の出現を別時刻で重複して数え上げないよう，P の末尾は必ず最も新しいトランザクションに出現するものとする（すなわち $s_{k_r} \subseteq t_i$）．このとき，P をサポートする時刻の総和を P の頻度と呼び，$sup(P, n)$ と書く[2]．最小サポート σ $(0 < \sigma \leq 1)$ に対し，$sup(P, n) > \sigma \times n$ を満たす P を頻出パターンと呼ぶ．FPM-TS の計算タスクは各時刻の全頻出パターンを求めることである．

例 2. 例 1 の $\mathcal{S}_4^1$ と P を考えると，P は時刻 4 でのみサポートされるため頻度は 1 である．また $\mathcal{S}_4^1$ に 3 回以上出現する（たとえば $\sigma = 0.7$ に対する）頻出パターンは，$\langle \{d\}\{c\} \rangle$ と $\langle \{c\} \rangle$ の二つである．

系列長を 1 に限定するとき，このタスクは頻出アイテム集合マイニングに相当する．頻出アイテム集合は，いわゆる「おむつ」と「ビール」の例のように，アイテム間の共起関係を表す．ストリームデータのコンテキストでは，複数のエッジノード間で共起して発生するアイテム集合に相当し，これはノード間の相関性を示す「空間的な特徴」と捉えられるかもしれない．集合系列ではさらに時間情報が加わる．たとえばファイルシステムのログ記録を考えると，ファイルを書き込む操作の前には必ずファイルを開く操作がなされる．このような「時系列的な因果」を含め，FPM-TS はストリームデータの時間的，空間的な特徴を形式知として構成できる．また頻出パターンは説明変数の構成が難しい場合のデータ素性として，しばしば利用されており [4–6]，ストリームデータの素性合成器としても幅広い応用可能性をもつ．

他方，このタスクでは解候補の組み合わせ爆発現象が大きな問題となる．またリアルタイム応答などオンライン処理を基本とするストリームデータマイニング共通の技術的制約と難しさを含んでいる．

これらの課題を克服するためのアプローチとして，本稿ではオンライン近似圧縮と呼ばれる手法 [7, 8] を紹介する．この手法は，解候補集合の圧縮表現を逐次的に求めていくもので，抽出対象である任意の頻出パターンの頻度をある誤差を許しながら復元することができる．提案法は元々頻出アイテム集合マイニング用に開発されたものだが，近年集合系列への拡張も可能であることがわかってきた [9]．次節ではまずはじめに頻出アイテム集合マイニングの場合を扱う．

3. オンライン近似圧縮

3.1 基本アイデア：Δ-圧縮

I をアイテムの全体集合，$\mathcal{S}_n$ をストリーム，Δ を非負整数とし，次のアイテム集合族の圧縮表現を考える．

定義 1. [Δ カバー] P と Q を I 上のアイテム集合とする．P が Q を Δ カバーする（$P \succeq_\Delta Q$ と表す）とは以下の条件を満たすときまたそのときに限る：

$$P \supseteq Q \text{ かつ } sup(Q, n) \leq sup(P, n) + \Delta$$

定義 2. [Δ-圧縮] F と T を I 上のアイテム集合の族とする．F が T を Δ カバーする（$F \succeq_\Delta T$ と表す）とは以下の条件を満たすときまたそのときに限る：

$$\forall Q \in T \; \exists P \in F \; s.t. \; P \succeq_\Delta Q$$

また $F \succeq_\Delta T$ を満たすとき，F を T の Δ-圧縮と呼ぶ．

例 3. 図 2 のような四つのアイテム集合 P_1, P_2, P_3, P_4 を含む族 T を考える．このとき $\{P_1, P_3, P_4\}$ は T の 0-圧縮，$\{P_3, P_4\}$ は 1-圧縮，$\{P_4\}$ は 2-圧縮である．

集合族 T の Δ-圧縮が求まれば，それから T の元のアイテム集合の頻度を復元することができる．ただし復元される頻度は最大 Δ だけの誤差を含む．例 3 の

[2] この頻度尺度は特に系列末尾頻度と呼ばれ [3]，逆単調性を満たすことが知られている．

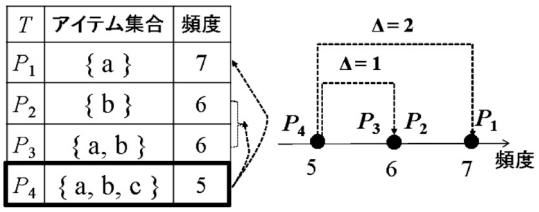

図 2　Δ-圧縮の例

T とその 2-圧縮である $\{P_4\}$ を考えると，P_1,P_2,P_3 は P_4 の部分集合であり逆単調性から少なくとも頻度は 5 以上である．この P_4 の頻度を復元頻度とすると，実際 P_1 において最大 2 の誤差を含むこととなる．

オンライン近似圧縮法では，頻出アイテム集合族のかわりに，その Δ-圧縮を抽出対象とする．すなわち，圧縮による誤差 Δ を許容することで，管理すべき解候補（本節ではアイテム集合）を絞るアプローチをとる．Δ-圧縮は近似削除と漸近交差と呼ばれる技法に基づき求められる．以下，これらの基本技法を説明する．

3.2　近似削除

近似削除は，ストリームデータマイニングのオンライン近似アルゴリズム [10–14] の枠組み[3]の中でよく利用されるテクニックである．これらのアルゴリズムでは解候補となる各アイテム集合の頻度を誤差を含む見積頻度として管理する．すなわち，管理対象のアイテム集合を P とし，P の見積頻度とその誤差をそれぞれ $c(P)$ と $e(P)$ と表すと，$c(P) - e(P) \leq sup(P,n) \leq c(P)$ が成り立つ．これらの三つ組 $\langle P, c(P), e(P)\rangle$ をエントリと呼び，時刻 i に管理される全エントリの集合を $TS(i)$ と表す．$k(i)$ は $TS(i)$ のエントリ数とする．

近似削除の基本アイデアは低見積頻度のエントリを積極的に削除していくことにある．メモリ消費抑制のため多くの手法で採用されているが，削除の戦略に応じてパラメータ指向型 [10–12] とリソース指向型 [13, 14] の二つに大別される．前者は誤差パラメータ ϵ $(0 \leq \epsilon \leq \sigma)$ が与えられ，各時刻 i で見積頻度が $\epsilon \times i$ 未満であるものが削除される．誤差が必ず $\epsilon \times n$ 以下となることが保証される反面，エントリ数が少なくとも $\mathrm{O}(\frac{2L}{\epsilon})$（$L$ は最大トランザクション長）だけ必要となることがわかっており [14]，メモリを大量に消費する危険がある．これに対し，後者はサイズ定数 k $(k > 0)$ が与えられる．各時刻 i でもしエントリ数 $k(i)$ が k を超えるならば，見積頻度が低いものから順に超過している個数 $(k(i) - k)$ だけエントリを削除する．ここで削除されたエントリの最大見積頻度が誤差となる．パラメータ指向型の近似削除とは異なり，リソース指向型ではメモリ消費の上限を設定できる反面，最終的な誤差を直接コントロールすることが難しくなる．このようにパラメータ指向型とリソース指向型の近似削除にはそれぞれ一長一短の特徴がある．

3.3　漸近交差法

Δ-圧縮は飽和性 (closeness) に基づく圧縮の拡張とみなすことができる．F をアイテム集合の族とする．あるアイテム集合 P が F に関して飽和であるとは，$P \in F$ であり，任意のアイテム集合 $Q \in F$ に対し，$Q \supset P$ かつ $sup(P,n) = sup(Q,n)$ を満たすものが存在しないときまたそのときに限る．時刻 n までにストリーム上に出現した全アイテム集合族を $I(n)$ とし，$I(n)$ に関する飽和アイテム集合の族を $CI(n)$ とすると，$CI(n)$ は $I(n)$ の 0-圧縮になっていることに気づく．漸近交差法 (incremental intersection) は，この飽和アイテム集合族を次の漸化式をもとに逐次的に更新する手法である [16, 17].

$$CI(i+1) = CI(i) \cup \{t_{i+1}\} \cup \{\beta \mid \beta = \alpha \cap t_{i+1}, \beta \neq \emptyset, \alpha \in CI(i)\}$$

漸化式によれば，次時刻の飽和アイテム集合は，(1) 新規トランザクション，(2) 現時刻の飽和アイテム集合，もしくは (3) それらの共通部分のいずれかに相当する．(1) と (2) はすでに現時刻で管理しているので (3) の共通部分を求めるだけで更新を行うことができる．二つのアイテム集合の共通部分を求める時間計算量は $\mathrm{O}(L)$ なので，時刻 i で $k(i)$ 個のエントリを更新する時間計算量は $\mathrm{O}(L \times k(i))$ となる．実際には，t_{i+1} と共通要素をもたないエントリへの無駄なアクセスを防ぐため，アイテムごとにそれを含むエントリをまとめて管理する垂直配置型の索引データ構造（*cid_list* と表す）がしばしば利用される [17].

例 4.　次のストリーム $\mathcal{S}_3^2 = \langle\{a,b\}\{b,c\}\{a,c\}\rangle$ を考える．各時刻 i までに出現した飽和アイテム集合を漸近交差法により求めると図 3 のとおりとなる．時刻 1 のエントリ集合 $TS(1)$ には t_1 が含まれる．*cid_list* にはアイテム a, b の行が作成され，エントリ P_1 が登録される．時刻 2 では t_2 自身と $t_2 \cap \{a,b\}$ のエントリ P_3 が $TS(2)$ に追加される．ここで共通部分に相当する P_3 の見積頻度は 1 だけ増加する．時刻 3 で $t_3 = \{a,c\}$ が到着したとき *cid_list* を参照すれば，t_3 と共通部分をもつのは P_1 と P_2 のみであることがわかる．

[3] 本稿では決定的アルゴリズムに焦点を絞るが，サンプリングなどの確率アルゴリズム [15] も活発に研究されている．

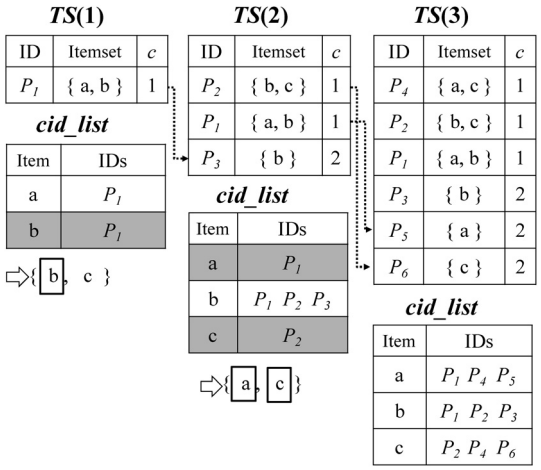

図 3　漸近交差法の例

3.4　アルゴリズムと諸性質

この漸近交差法に近似削除を組み込むことで，頻出アイテム集合族の Δ-圧縮を求めるオンラインアルゴリズムが構成される．Algorithm 1 にその擬似コードを示す．6 行目と 7〜10 行目の処理がそれぞれ漸近交差と近似削除の処理に相当する．関数 *intersect* では，はじめに新規トランザクション t_{i+1} が $TS(i)$ に登録されているかどうかチェックする．未登録の場合，t_{i+1} の誤差を $\Delta(i)$ とし登録する．これは t_{i+1} が現在までに高々 $\Delta(i)$ 回だけ出現していた可能性があるためである．次に $TS(i)$ の各エントリと t_{i+1} との共通部分を計算し，その結果を一時作業領域 C に保存する（5〜11 行目）．別のエントリから同じ共通部分が計算される場合，見積頻度の大きいほうを C に登録する（10〜11 行目）．その後，C をもとに $TS(i)$ を更新する（12〜17 行目）．なおエントリ id のアイテム集合，見積頻度，誤差をそれぞれ $p(id)$, $c(id)$, $e(id)$ と表している．

近似削除にはパラメータ指向型 (PO) とリソース指向型 (RO) の二つのタイプがある．これらを併用することもできるが，ここでは利用タイプとそれに対応するパラメータの組を選択して利用するものとする．関数 *checkDeletion* では各エントリを見積頻度昇順で取り出しながら，各タイプの削除基準に合致するかどうか調べる．削除する場合，削除エントリの見積頻度が誤差として $\Delta(i)$ に反映される（9 行目）．以降，パラメータ指向型（resp. リソース指向型）のアルゴリズムを PO-ComStream (resp. RO-ComStream) と呼ぶ．

例 5.　先の例の $\mathcal{S}_3^2$ にそれぞれのアルゴリズムを適用する．$\epsilon = 0.5$ としたとき，時刻 2 の時点で P_1 と P_2 が削除され $\Delta(2) = 1$ となる．これに合わせて，t_3 に相当するエントリ E が $\langle\{a, c\}, 2, 1\rangle$ となる．一方，$k = 3$ のとき，時刻 3 の時点で見積頻度の低いエントリ P_1, P_2, P_4 が削除される．よって最小サポート $\sigma = 0.5$ の場合，前者の PO-ComStream は E と P_3 を，後者の RO-ComStream は P_3, P_5, P_6 をそれぞれ出力する．

$TS(i)$ に未登録のアイテム集合に対しても，$TS(i)$ をもとに時刻 i での頻度を見積ることができる．

定義 3.　P を $TS(i)$ に登録されていないアイテム集合とする．いま $P \subseteq Q$ を満たす $TS(i)$ 中の任意の Q のエントリ集合を $TS_P(i)$ と表す（すなわち $TS_P(i) \subseteq TS(i)$ である）．$TS_P(i)$ 中のエントリで最大の見積頻度をもつものを R_P と書き，P の代表と呼ぶ．このとき P の見積頻度 $c(P)$ を次のとおり与える（ただし $\Delta(i)$ は時刻 i での最大誤差計数である）：

$$c(P) = \begin{cases} c(R_P) & TS_P(i) \neq \emptyset \\ \Delta(i) & TS_P(i) = \emptyset \end{cases}$$

例 6.　再び例 5 を用いる．時刻 3 での処理終了時点で PO-ComStream における $TS(3)$ は $\{E, P_3\}$ であった．いまアイテム集合 $\{c\}$ を考えると，$\{c\}$ のエントリは $TS(3)$ に存在しないが，$TS_{\{c\}}(3)$ はエントリ $E = \langle\{a, c\}, 2, 1\rangle$ を含む．よって $c(\{c\}) = 2$ となる．$sup(\{c\}, 3) = 2$ なので，この見積頻度と一致している．

定義 3 の見積頻度は逆単調性が成り立ち，次の性質を満たすことが知られている [7]：

定理 1.　任意のアイテム集合 P と時刻 i において，

$$c(P) - \Delta(i) \leq sup(P, i) \leq c(P)$$

系 1.　$\Delta(n) \leq \sigma n$ であるとき，ComStream の出力は全頻出アイテム集合族の $\Delta(n)$-圧縮となる．

定理 2.　PO-ComStream は各時刻 i で $\Delta(i) \leq \epsilon \times i$ を保証する．RO-ComStream は各トランザクションを $O(kL + k\log(k))$ の時間と $O(k)$ の領域で処理する．

Δ-圧縮の Δ は，本来圧縮による誤差を指したが，系 1 では近似削除による誤差 $\Delta(n)$ と統合されている．このように圧縮誤差と削除誤差を同一視し利用する点がオンライン近似圧縮の特徴といえる．

Algorithm 1: ComStream

Data: 近似タイプ $t \in \{po,\ ro\}$ とパラメータ ϵ or k,
最小サポート σ，ストリーム $\mathcal{S}_n = \langle t_1, \ldots, t_n \rangle$
Result: 頻出アイテム集合族の Δ-圧縮

set i as 1 ($i \leftarrow 1$) ▷ i：時刻
$\Delta(0) \leftarrow 0$ ▷ $\Delta(i)$：時刻 i での最大誤差計数
initialize $TS(0)$ ▷ $TS(i)$：時刻 i でのエントリ集合
while $i \leq n$ **do**
  read t_i
  $TS(i) \leftarrow intersect(TS(i-1), t_i)$
  while $checkDeletion(TS(i), t)$ **do**
    $m \leftarrow getMin(TS(i))$ ▷ m は最小エントリ
    $\Delta(i) \leftarrow c(m)$ ▷ $\Delta(i)$ の更新
    $delete(TS(i), m)$ ▷ エントリ m を削除
  $i \leftarrow i + 1$
for *each id in* $TS(n)$ *s.t.* $c(id) > \sigma n$ **do**
  output id ▷ id は Δ-圧縮を構成するエントリ

Function `intersect`$(TS(i),\ t_{i+1})$

initialize C ▷ C：登録候補のエントリ集合
$id \leftarrow get(t_{i+1}, TS(i))$ ▷ t_{i+1} のエントリを取得
if *id is null* **then**
  add $\langle t_{i+1}, \Delta(i), \Delta(i) \rangle$ to $TS(i)$
for *each entry id in* $TS(i)$ **do**
  $\beta \leftarrow p(id) \cap t_{i+1}$ ▷ 共通部分計算
  $id_\beta \leftarrow get(\beta, C)$ ▷ β のエントリを取得
  if $\beta \neq \emptyset$ *and* id_β *is null* **then**
    add $\langle \beta,\ c(id)+1,\ e(id) \rangle$ to C
  else if $\beta \neq \emptyset$ *and* $c(id_\beta) < c(id) + 1$ **then**
    $c(id_\beta) \leftarrow c(id) + 1,\ e(id_\beta) \leftarrow e(id)$
for *each entry* id_C *in* C **do**
  $id_{TS_i} \leftarrow get(p(id_C), TS(i))$
  if id_{TS_i} *is null* **then**
    ▷ id_C は $TS(i)$ に含まれていない
    add id_C to $TS(i)$
  else
    ▷ id_C は $TS(i)$ に含まれている
    replace id_{TS_i} with id_C
return *TS(i)*

Function `checkDeletion`(T_i, t)

if $t = po$ **then**
  $m \leftarrow getMin(T_i)$ ▷ m は最小エントリ
  if $c(m) \leq \epsilon \times i$ **then**
    return *true* ▷ パラメータ指向型削除
else
  if $k < k(i)$ **then**
    return *true* ▷ リソース指向型削除
return *false*

最後に ComStream の性能について簡単に述べる．図 4 は，誤差パラメータ ϵ に対するメモリ消費の抑制効果と平均処理時間（1 トランザクション当たり）を示している．データは FIMI リポジトリ[4]の小売データ (retail) を用いている．細線はパラメータ指向型の従来手法 (StreamMining) [11]，太線は PO-ComStream に相当する．図 4 より ϵ が増加するに伴い処理時間とメモリ消費ともに低く抑えられていくことがわかる．また図 5 は，サイズ定数 k に対する RO-ComStream の最終的な誤差 $\Delta(n)$ と平均処理時間を示している．データは FIMI リポジトリのクリックストリームデータ (kosarak) を用いている．細線はリソース指向型の従来手法 (Skip LC-SS) [14]，太線は RO-ComStream に相当する．図 5 より k が増加するに伴い誤差は低下する反面，処理時間は線形的に増すことがわかる．

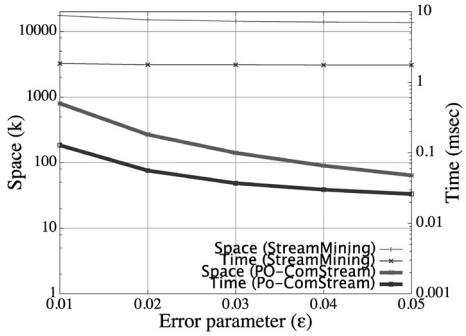

図 4 PO-ComStream の性能

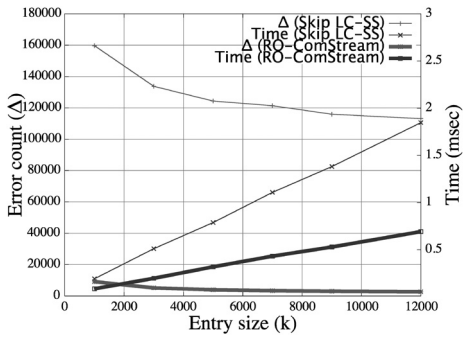

図 5 RO-ComStream の性能

4. 集合系列マイニングへの拡張

前節まで頻出アイテム集合を扱ってきたが，オンライン近似圧縮は集合系列の場合にも有効である．ここでは集合系列への拡張法について要点を絞り概説する．いま探索対象の集合系列の最大時間幅を w とする．w は先頭と末尾間の時刻の最大許容差を意味する．このとき以下のウィンドウの概念を考えることができる．

定義 4. $\mathcal{S}_n = \langle t_1, t_2, \ldots, t_n \rangle$ をストリームとする．このとき $\mathcal{S}_n$ 上の集合系列 $\langle t_{e(i)}, t_{e(i)+1}, \ldots, t_i \rangle$ を時

[4] http://fimi.ua.ac.be/data/

刻 i のウィンドウ（$win(i)$ と書く）と呼ぶ．ただし

$$e(i) = \begin{cases} i - w + 1 & if\ i \geq w \\ 1 & otherwise \end{cases}$$

抽出対象の集合系列は必ずあるウィンドウ内に出現する．よって新規ウィンドウが到着する度に現在管理している集合系列のエントリを更新していけばよい．すなわち集合系列の漸近交差ではトランザクションではなくウィンドウとの共通部分を考える（図 6）．飽和系列 α をサポートする時刻 i までのウィンドウの集合を $W(\alpha, i)$ と書く．時刻 i における二つの飽和系列の（極大）共通部分 γ を次のように定義する．

定義 5. α, β を時刻 i における飽和集合系列とする．任意のウィンドウ $win \in W(\alpha, i) \cup W(\beta, i)$ に対し，γ が win によりサポートされるとき，γ を時刻 i における α と β の共通部分と呼ぶ．γ を真に含む系列で時刻 i における α と β の共通部分となるものが存在しないとき，γ を時刻 i における α と β の極大共通部分と呼ぶ．時刻 i における α と β の全極大共通部分の集合を $int(\alpha, \beta, i)$ と表す．

例 7. 例 1 のストリーム $\mathcal{S}_4^1$ を考える．ウィンドウ幅を 3 として $\mathcal{S}_4^1$ 上の時刻 4 における二つの飽和集合系列 $\alpha = \langle\{d\}\{bc\}\rangle$ と $\beta = \langle\{abc\}\{c\}\rangle$ を考える．$W(\alpha, 4) = \{win(2), win(3)\}$, $W(\beta, 4) = \{win(4)\}$ なので時刻 4 における α と β の極大共通部分は $\langle\{c\}\rangle$ となる．よって $int(\alpha, \beta, 4) = \{\langle\{c\}\rangle\}$ である．

アイテム集合の場合と同様，集合系列でも以下の定理に基づき漸近交差法を考えることができる：

定理 3. $TS(i)$ を時刻 i までに出現した飽和集合系列の族，$win(i+1)$ を次のウィンドウとする．このとき

$$TS(i+1) = TS(i) \cup \{win(i+1)\} \cup \{\beta \mid \beta \in int(\alpha, win(i+1), i+1), \alpha \in TS(i)\}.$$

漸化式より $TS(i)$ を逐次的に更新できることがわかる．なお [9] では交差表と呼ばれる 2 次元表を用いて極大共通部分を列挙する手法を提案している．

5. まとめ

本稿では，データストリームマイニングの基礎問題の一つである FPM-TS を取り上げ，高速・省メモリに頻出パターンを抽出するオンラインアルゴリズムを紹介した．ComStream は漸近交差法に近似削除を組み込んだシンプルな手法だが，頻出パターンの Δ-圧縮表現を抽出することができる．これにより誤差 Δ 以内で任意の頻出パターンの頻度復元が可能となる．また近似削除にはパラメータ指向型とリソース指向型の二つの戦略があり，それに応じて誤差 Δ とメモリ消費の挙動が変わることを示した．最後に漸近交差法を集合系列へ拡張するアイデアを紹介した．

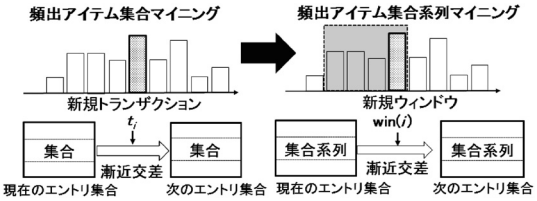

図 6 集合系列への拡張のアイデア

謝辞 本研究は JST さきがけの援助を受けている．本稿を執筆する機会と貴重なコメントをいただいた中原孝信先生，宇野毅明先生ならびに編集委員の皆様に感謝申し上げます．また本稿の研究内容については岩沼宏治先生より多くの助言をいただいている．

参考文献

[1] D. C. Anastasiu, J. Iverson, S. Smith and G. Karypis, "Big data frequent pattern mining," *Frequent Pattern Mining*, C. C. Aggrawal and J. Han (eds.), Springer, pp. 225–260, 2014.
[2] J. Lin, E. Keogh, L. Wei and S. Lonardi, "Experiencing SAX: A novel symbolic representation of time series," *Data Mining and Knowledge Discovery*, **15**, pp. 107–144, 2007.
[3] 岩沼宏治，"テキスト系列マイニングにおける有用性尺度について，" 人工知能学会論文誌，**27**, pp. 136–145, 2012.
[4] H. Cheng, X. Yan, J. Han and C.-W. Hsu, "Discriminative frequent pattern analysis for effective classification," In *Proceedings of 2007 IEEE 23rd International Conference on Data Engineering*, pp. 716–725, 2007.
[5] T. Quack, V. Ferrani and L. V. Gool, "Video mining with frequent itemset configurations," *Image and Video Retrieval*, **4071**, pp. 360–369, 2006.
[6] K. R. Seeja, "Feature selection based on closed frequent itemset mining: A case study on SAGE data classification," *Neurocomputing*, **151**, pp. 1027–1032, 2015.
[7] 岩沼宏治，山本泰生，福田翔士，"ストリーム中の頻出飽和集合を抽出するオンライン型 ϵ-近似アルゴリズムの完全性，" 人工知能学会論文誌，**31**, pp. 1–10, 2016.
[8] Y. Yamamoto and K. Iwanuma, "Online pattern mining for high-dimensional data streams," In *Proceedings of 2015 IEEE International Conference on Big Data (Big Data)*, pp. 2615–2617, 2015.

[9] 山本泰生，山内夏美，岩沼宏治，“漸近交差法に基づくオンライン頻出系列パターンマイニング，”人工知能学会第 100 回人工知能基本問題研究会，2016.
[10] H.-F. Li, M.-K. Shan and S.-Y. Lee, “DSM-FI: An efficient algorithm for mining frequent itemsets in data streams,” *Knowledge and Information Systems*, **17**, pp. 79–97, 2008.
[11] R. Jin and G. Agrawal, “An algorithm for in-core frequent itemset mining on streaming data,” In *Proceedings of the Fifth IEEE International Conference on Data Mining*, pp. 210–217, 2005.
[12] G. S. Manku and R. Motwani, “Approximate frequent counts over data streams,” In *Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 346–357, 2002.
[13] A. Metwally, D. Agrawal and A. E. Abbadi, “Efficient computation of frequent and top-k elements in data streams,” *Database Theory–ICDT 2005*, pp. 398–412, 2005.
[14] Y. Yamamoto, K. Iwanuma and S. Fukuda, “Resource-oriented approximation for frequent itemset mining from bursty data streams,” In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pp. 205–216, 2014.
[15] R. C. Wong and A. W. Fu, “Mining top-k frequent itemsets from data streams,” *Data Mining and Knowledge Discovery*, **13**, pp. 192–217, 2006.
[16] C. Borgelt, X. Yang, R. N. Cadenas, P. C. Saez and A. P. Montano, “Finding closed item sets by intersecting transactions,” In *Proceedings of the 14th International Conference on Extending Database Technology*, pp. 367–376, 2011.
[17] S.-J. Yen, C.-W. Wu, Y.-S. Lee, V. S. Tseng and C.-H. Hsieh, “A fast algorithm for mining frequent closed itemsets over stream sliding window,” In *Proceedings of 2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*, pp. 996–1002, 2011.

