A-012

ストリームデータからの△-強飽和集合のオンライン抽出

Complete Online Mining of Strongly Closed Itemsets from Data Streams

日向 涼 † 岩沼 宏治 ‡ 仁科 拓巳 † §
Ryo Hinata Koji Iwanuma Takumi Nishina

1 はじめに

現在, プラントや交通情報などのセンシングデータ, Web ログデータなどは急速に膨大で無限に増え続けている. このような, 常に高速に流れ続けるデータを**データストリーム**と呼ぶ. このデータストリームから有用な知見を高速に得るオンライン処理技術が, 現在盛んに研究されている. その手法の一つとしてオンライン型ストリームマイニングが挙げられる.

仁科らは, 頻出飽和アイテム集合をトランザクションストリームから高速に抽出するために飽和集合を拡張 FP 木を利用して圧縮保持し, 更に圧縮したままで集合積演算を行う効率的なオンライン近似アルゴリズム [9] を提案した. ただ疎なデータ上の頻出アイテム集合は, 効果的に圧縮できないという課題があった. これに対して, 飽和集合をさらに一般化した Δ-飽和集合 [2] が提案されている. Δ-飽和集合はアイテム集合の頻度情報を完全に保存することはできないが, 標準的な飽和集合よりもコンパクトにアイテム集合を圧縮し, また対象となるデータの変化に対して安定であることが知られている. これまで提案されている Δ-飽和集合の処理形式は、オフライン [2] や準オンライン [3] であり, 完全オンライン型の抽出法は提案されていなかった. 本研究では, 先行研究 LC-CloStream[9] を用いた完全オンライン型の Δ-飽和集合抽出法を提案する. LC-CloStream は飽和集合を抽出するが, これをプレフィルタとして用いて飽和集合を抽出し, 抽出された飽和集合の集合の中から Δ-飽和集合を選別抽出していく. Δ-飽和集合を用いることで, 疎なストリームデータに関しても効果的な圧縮を行うことができる.

本論文の構成は以下の通りである. 第 2 章は準備である. 第 3 章では先行研究での LC-CloStream と Δ-飽和集合を示す. 第 4 章では, LC-CloStream を用いたオンライン型の Δ-飽和集合抽出アルゴリズムを新しく提案する. 第 5 章は性能確認のための実証実験の結果と考察を示す. 第 6 章はまとめである.

†山梨大学大学院医工農学総合教育部工学専攻コンピュータ理工学コース Computer Science and Engineering Course, Integrated Graduate School of Medicine, Engineering and Agricultural Sciences, University of Yamanashi
‡山梨大学大学院総合研究部 Interdisciplinary Graduate School, University of Yamanashi
§現在の所属は 明電システムソリューション (株)

2 準備

本稿での表記法と用語の定義を以下に示す.

定義 1 アイテムの全体集合を $I = \{e_1, e_2, \cdots, e_r\}$ とするとき, I の部分集合 A を**アイテム集合**もしくは**トランザクション**と呼ぶ. トランザクションの列 $\langle A_1, A_2, \cdots, A_N \rangle$ を**トランザクションストリーム**と呼び, N を**ストリーム長**と呼ぶ.

以下ではアイテムを $a, b, \cdots$ で, アイテム集合を $X, Y, \cdots$ で表す. またアイテム集合がトランザクションを表すときは, $A_1, A_2, \cdots$ と表記する. またアイテム集合 $\{e_1, e_2, \cdots, e_m\}$ を $e_1e_2\cdots e_m$ と略記する. また集合 X の要素数を $|X|$ で表記する.

定義 2 $\mathcal{S} = \langle A_1, \cdots, A_N \rangle$ をストリーム, X をアイテム集合とする. 各時刻 t $(1 \leq t \leq N)$ における多重集合 $\mathcal{K}(X,t)$ を, $\mathcal{K}(X,t) = \{A_j \in \mathcal{S} \mid X \subset A_j,\ 1 \leq j \leq t\}$ とする. このとき X の時刻 t の**頻度** $\sup(X,t)$ を $\sup(X,t) = |\mathcal{K}(X,t)|$ と定める. **相対最小頻度**をユーザが与える閾値 σ $(0 < \sigma \leq 1)$ とするとき, X が $\mathcal{S}$ 上の**時点 t で頻出**であるとは, $\sup(X,t) \geq \sigma \cdot t$ なる場合を言う. また X が $\mathcal{S}$ 上の**時刻 t で飽和**しているとは, $X \subsetneq Y$ で以下の式を満たすアイテム集合 Y が存在しない場合を言う.

$$\sup(X,t) - \sup(Y,t) = 0$$

$t = N$ が明らかである場合は, 単に**頻出**, **飽和**と略記する. 次に文献 [2] で導入されている Δ-飽和集合を本論文で取り扱うストリームに適した形で導入する.

定義 3 アイテム集合 X がストリーム S 上の**時刻 t で Δ-飽和**しているとは, 以下の条件を満たす Y が存在しない場合を言う.

1. $X \subsetneq Y$
2. $\sup(X,t) - \sup(Y,t) < \Delta$

Δ を**飽和強度**と呼ぶ. Δ=1 のときの Δ-飽和集合は通常の意味の飽和集合であることを注意して頂きたい.

3 先行研究

3.1 漸近的集合積法と LC-CloStream

ストリーム上の飽和集合は，以下のように集合積を用いて特性化できる．

命題 1 $\mathcal{S} = \langle A_1, \cdots, A_N \rangle$ をストリームとする．アイテム集合 X が時刻 t で飽和していることと，$X = \bigcap_{A_i \in \mathcal{K}(X,t)} A_i$ が成り立つことは同値である．

命題 1 より，出現トランザクション集合の集合積をとることにより，要素数の大きい飽和集合を効率よく抽出できる．また命題 1 より以下の漸化式が成り立つ．以下では，ストリーム $\mathcal{S}$ 上の飽和アイテム集合の族を $\mathcal{CIS}(\mathcal{S})$ で表し, 列 S_1 と S_2 の連結を $S_1 \circ S_2$ と表す．

命題 2 $\mathcal{S} = \langle A_1, \cdots, A_N \rangle$ をストリームとする．このとき以下が成り立つ．

$$\begin{aligned} \mathcal{CIS}(\langle A_1 \rangle) &= \{A_1\} \\ \mathcal{CIS}(\mathcal{S}_t \circ \langle A_{t+1} \rangle) &= \mathcal{CIS}(\mathcal{S}_t) \cup \{A_{t+1}\} \cup \\ &\quad \{C \mid \exists B \in \mathcal{CIS}(\mathcal{S}_t) : C = B \cap A_{t+1}\} \end{aligned}$$

但し，$\mathcal{S}_t$ $(1 \leq t \leq N-1)$ は $\langle A_1, \cdots, A_t \rangle$ である．

命題 2 の漸化式を利用することにより，飽和集合をオンライン抽出することができる. 以下, これを**漸近的集合積法** [1] と呼ぶ.

Clo-Stream [5] は, この漸近的集合積を用いた飽和アイテム集合を抽出するオンライン型の厳密解法である. Clo-Stream の計算時間と空間使用量を改善するために岩沼 [8] は Lossy-Counting の ϵ-近似削除を導入した近似アルゴリズム LC-CloStream を提案した.

頻度表とは, 飽和集合 X と許容誤差を ϵ とした場合, $\langle X, f(X), \delta(X) \rangle$ の 3 項組で記録した表である．ここで, $f(X)$ は X が登録された時点 t_X から，現時刻 t までの出現回数であり，$\delta(X)$ は登録された時点での最大誤差 $\Delta = \lfloor \epsilon \cdot (t_X - 1) \rfloor$ である．$f(X) + \delta(X)$ を X の**見積もり頻度**と呼ぶ．また ϵ-近似削除を行うため，LC 法に倣った頻度表の管理を行った. 即ち時刻 t のとき, 見積もり頻度が $\lfloor \epsilon \cdot t \rfloor$ 未満の飽和集合を表から削除する. 頻度表は非常に巨大になり, その保持には多大なコストがかかる．そのため仁科ら [9] は, スキップ走査 FP 木による圧縮手法を提案した.

3.2 スキップ走査 FP 木による飽和集合圧縮

図 1 がスキップ走査 FP 木である. 頻度表を FP 木で圧縮して保持し, 更に圧縮したままで集合積演算を行うことで, 効率的に飽和集合を求めるオンラインアルゴリズム [9] を提案した.

FP 木 [4] は, トランザクションをアイテムとその頻度を保持した頂点によるトライ木で保持し，更に各アイテムの出現を BList で索引化している. FP 木により, アイテム集合の出現トランザクションを高速に求めることができる．仁科ら [9] は, この FP 木を拡張し, スキップ走査を用いて対象となるブランチのみを辿りながら, 効率的に集合積計算を線形時間で行えるアルゴリズムを提案した.

ストリーム $\mathcal{S} = \langle abcd, abc, bcde, bde, ace, ce, e \rangle$ としたとき，時刻 7 における FP 木 (頻度表) は図 1 のようになる．各アイテム集合のアイテムを根頂点から辞書順に登録している．その頂点には, そこから根までの経路で表現される飽和集合の出現頻度と最大誤差が記録されている．抽出した飽和集合を FP 木で保持することで共通の要素を持つような飽和集合を圧縮して保持することができる．図 1 の木に格納されている飽和集合とその頻度値は $abcd : 1$, $abc : 2$, $ace : 1$, $ac : 3$, $bcde : 1$, $bcd : 2$, $bc : 3$, $bde : 2$, $bd : 3$, $b : 4$, $ce : 3$, $c : 5$, $e : 5$ となる.

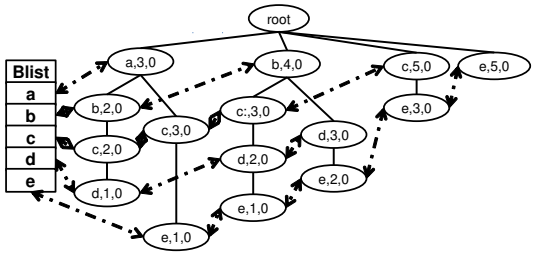

図 1: 時刻 7 における FP 木による頻度表

3.3 オフライン型 Δ-飽和集合抽出

Boley らは, オフライン処理で Δ-飽和集合を抽出する手法 [2] を提案した. 2 分探索木を用いて, 重複なく効率的に Δ-飽和集合を算出する工夫を行っている. Δ-飽和集合を全て列挙する基本的なアイデアは, あるアイテム i を含む Δ-飽和集合と i を含まない Δ-飽和集合を 2 分することを繰り返して Δ-飽和集合を列挙することである.

4 オンライン型 Δ-飽和集合抽出

本研究では, LC-CloStream[9] をプレフィルタとして用いて飽和集合を抽出し, 抽出された飽和集合の集合の中から Δ-飽和集合を選択していく. 定義より, $\Delta > 1$ のときの Δ-飽和集合は, 必ず飽和集合となる. このため Δ-飽和集合は, 1-飽和集合の中から選別抽出すれば十分である. 以下では, 1-飽和集合を**標準飽和集合**と表現する.

4.1 Δ-飽和集合の列挙

標準飽和集合の集合から Δ-飽和集合 ($\Delta > 1$) の選別列挙をするが, Δ-飽和集合を列挙する工夫として, 以下の補題 1 の見積もり頻度の逆単調性 [8] を利用することで標準飽和集合であるか否かのチェックを行う範囲を大幅に削減することができる. 以下では, 時刻 t のときの頻度木（FP 木）に記録されている X の 頻度値と最大誤差を $f(X,t)$, $\delta(X,t)$ と表記する.

補題 1 (見積り頻度の逆単調性) $\mathcal{S}$ を長さ $\mathcal{N}$ のストリームとする. 各時刻 t $(1 \leq t \leq N)$ の処理が終了したと時点において, 任意の $X \subset Y$ となる $X \in TS(t)$ と $Y \in TS(t)$ に対して以下が成り立つ.

$f(X,t) + \delta(X,t) \geq f(Y,t) + \delta(Y,t)$

補題 2 ストリーム S 上の時刻 t でアイテム集合 X に対して以下の性質を満たす $e \in I$ が存在しない場合, X は S 中の **Δ**-飽和集合 である.

$$\sup(X,t) - \sup(X \cup \{e\}, t) < \Delta$$

LC-CloStream で抽出される標準飽和集合の集合は FP 木で保持されている. $X \cup \{e\}$ が飽和集合ではない場合, FP 木上には存在しない. そこで閉包（$closure$）を利用する. アイテム集合 X の**閉包** [6] を $clo(X)$ と表現し，$clo(X) = \bigcap_{A_i \in \mathcal{K}(X,t)} A_i$ と定義する. 閉包について，次の性質が知られている.

補題 3 任意のアイテム集合に対して以下が成り立つ.

$clo(X)$ は X を含む唯一の最小な飽和集合.

補題 3 より閉包は必ず標準飽和集合になるため FP 木上に存在する. また $clo(X \cup \{e\})$ は, $X \cup \{e\}$ と頻度が等しいため補題 2 を以下のように変形することができる.

補題 4 ストリーム S 上の時刻 t でアイテム集合 X に対して以下の性質を満たす $e \in I$ が存在しない場合, X は S 中の **Δ**-強飽和集合 である.

$$\sup(X,t) - \sup(clo(X \cup \{e\}), t) < \Delta$$

4.2 FP 木の探索手順

FP 木上の標準飽和集合が Δ-飽和集合であるか否かを調べる疑似コードを Algorithm 1 に示す. Algorithm 1 では, $X \cup \{e\}$ 自身が標準飽和集合である場合, 7 行目で FP 木を根から辿りで頻度を取得する. $X \cup \{e\}$ 自身が標準飽和集合でない場合, $X \cup \{e\}$ は $clo(X \cup \{e\})$ の部分集合であるため, 14-21 行目で, FP 木の Blist を辿り $clo(X \cup \{e\})$ の頻度を取得する. $X \cup \{e\}$ を含む飽和集合は複数あるが, $clo(X \cup \{e\})$ はその中の頻度最大のものとなる. 28 行目で標準飽和集合 X に含まれないアイテムの集合の全てが, Δ-飽和集合の条件を満たした場合に X は Δ-飽和集合となる.

図 2 では, X を $\{c\}$, 追加するアイテムを $\{d\}$ とした $\{c\} \cup \{d\}$ の頻度を取得する手順を示す. FP 木の $\{c\}$ を先頭に持つ枝には $\{c\} \cup \{d\}$ は存在しないため, $\{c\} \cup \{d\}$ 自身は標準飽和集合ではないことがわかる. そのため次に Blist を辿り, $clo(\{c\} \cup \{d\})$ を探索する. Blist で全ての c を図 2 の二重破線のように辿り, 葉に $\{d\}$ が存在するか色付きの箇所を探索し, $clo(\{c\} \cup \{d\})$ の頻度を取得する. 図 2 の $clo(\{c\} \cup \{d\})$ の頻度は, 2 となる.

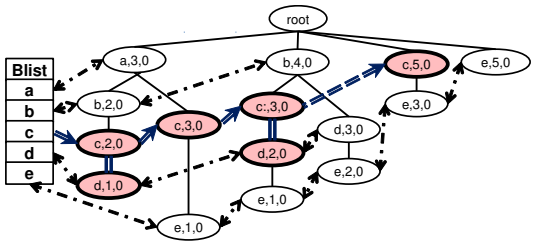

図 2: Blist を用いたアイテム $\{c\} \cup \{d\}$ の探索

Algorithm 1 Δ-飽和集合のチェック

Input: X:チェック対象のアイテム集合,
$\prec$:アイテムの順序関係, $delta$:強飽和集合の設定値
Output: GET_LABEL(X)

```
1: tset ← I \ X                    ▷ X に含まれないアイテムの集合
2: while tset ≠ ∅ do                ▷ tset が空になるまで探索
3:     Select e from tset           ▷ tset から 1 つ選択する
4:     Y ← X ∪ {e}
5:     SORT(Y, ≺)                   ▷ X を ≺ の昇順にソート
6:     freq ← 0                     ▷ 頻度変数の初期化
7:     freq ←FIND(TS_root, Y)       ▷ Y を探索
8:     if freq ≠ 0 then             ▷ Y が発見された場合
9:         if f(X) − freq < delta then
10:            break                ▷ 補題 3 を満たさず探索打ち切り
11:        end if
12:    end if
13:    if freq = 0 then             ▷ Y が発見されなかった場合
14:        bnode ← Node(Blist[Y[0]])
15:        while bnode ≠ NULL do    ▷ Blist を利用
16:            tmp ← FIND(bnode, Y) ▷ Y を発見した場合
17:            if freq < tmp then
18:                freq ← tmp
19:            end if
20:            bnode ←NEXT(bnode)
21:        end while
22:    end if
23:    if f(X) − freq < delta then
24:        break                    ▷ 補題 3 を満たさず探索打ち切り
25:    end if
26:    tset ← tset \ {e}            ▷ 探索済みの e を tset から除く
27: end while
28: if tset = ∅ then                ▷ tset が空になった場合
29:    ADDLABEL(X, ΔCIS)            ▷ X は Δ-飽和集合である
30: end if
```

表 1: 実験に使用したデータベース

データベース	#(item)	#(trans.)	ave(item)
mushroom	119	8,124	23.0
retail	16,469	88,162	10.3
earthquake	1,229	16.768	2.5

5 実験結果及び考察

実験には，Frequent Itemset Mining Dataset Repository[7] から 4 種の実データセットを使用した．各データセットの詳細を表 1 に示す．mushroom は密 (dense) なデータセットであり，retail，earthquake は疎 (sparse) なデータセットである．#(item) はデータセット中に含まれるアイテムの種類数を示し，#(trans.) はトランザクションの総数，ave(item) は各トランザクション中に出現するアイテムの平均数である．相対飽和強度 Δ^r を $\Delta^r=\frac{\Delta}{|S|}$ とする．パラメータは最大許容誤差 ϵ を決め打ちとし, 相対飽和強度 Δ^r を変化させた．

表 2: 実データでの各実験結果

データベース	相対飽和強度 Δ^r	許容誤差 ϵ	Δ-飽和集合数	実行時間 (sec.)
mushroom	0.0002	0.01	66,709	35
	0.01	0.01	13,601	3,477
	0.1	0.01	16	142
retail	0.00002	0.001	405,928	527
	0.001	0.001	10,378	43,327
	0.01	0.001	31	18
erathquake	0.0001	0.001	52,374	2
	0.001	0.001	5,224	2,501
	0.01	0.001	34	8

実験結果を表 2 に示す．相対飽和強度 Δ^r が mushroom において 0.0002, retail において 0.00002, earthquake において 0.0001 の時, 飽和強度は Δ=1 となり, 標準飽和集合が抽出されている．密なデータと疎なデータ共に, 標準飽和集合から強飽和集合へ有効な圧縮結果を得ることができた．一方で標準飽和集合の集合から Δ 飽和集合の抽出に時間がかかった．これは扱うデータのアイテム種類数に依存し, アイテム種類数が多いと, チェックすべきアイテム集合が増加することが原因として挙げられる．

6 まとめ

本研究では, LC-CloStream[9] をプレフィルタとして用いて標準飽和集合を抽出し, 抽出された標準飽和集合の集合の中から Δ-飽和集合を抽出を提案した．密なデータと疎なデータ共に, 標準飽和集合から強飽和集合へ有効な圧縮結果を得ることができた．今後は プレフィルタを利用せずに Δ-飽和集合を直接求めるような工夫を行っていきたい．

謝辞

本研究の一部は，JSPS 科学研究費補助金（No.19K12096）の援助を受けている．

参考文献

[1] C. Borgelt, X. Yang, R. Nogales-Cadenas,P. Carmona-Saez and A. Pascual-Montano:Finding closed frequent item sets by intersecting transactions. *Proc. EDBT 2011*, pp.367–376 (2011)

[2] M. Boley, T. Horvath and S. Wrobel: Efficient Discovery of Interesting Patterns Based on Strong Closedness. *Statistical Analysis and Data Mining*, Vol.2 (2009)

[3] D. Trabold and T. Horvath: Mining Strongly Closed Item sets from Data Streams. *Discovery Science*, , LNAI10558, pages 251-266 (2017)

[4] J. Han, J. Pei and Y. Yin: Mining Frequent Patterns without Candidate Generation. *Proc ACM int'l Conf. on Management of data*, pp1-12 (2000)

[5] S. Yen, C. Wu, Y. Lee, V.S. Tseng and C. Hsieh: A fast algorithm for mining frequent closed itemsets over stream sliding window. *IEEE Int'l Conf. on Fuzzy Systems*, pp.27–30 (2011)

[6] M. Zaki.: Mining Non-Redundant Association Rules. *Data Mining and Knowledge Discovery*, Vol. 9, pp. 223-248 (2004)

[7] Frequent Itemset Mining Dataset Repository, <http://fimi.ua.ac.be/>(2020-6-1)

[8] 岩沼 宏冶, 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 巻 5 号 B, pp.1-10 (2016)

[9] T. Nishina, K. Iwanuma and Y. Yamamoto: A Skipping FP-Tree for Incrementally Intersecting Closed Itemsets in On-Line Stream Mining. *IEEE Int'l Conf. on Big Data and Smart Computing (BigComp)*, pp.1-4 (2019)