Incremental intersection for frequent sequential pattern mining

Yoshitaka Yamamoto 1,2 Natsumi Yamauchi 3 Koji Iwanuma 1

1 Graduate Faculty of Interdisciplinary Research, University of Yamanashi
2 JST, Presto
3 Department of Computer Science, University of Yamanashi

Abstract: We propose a novel method for frequent sequence mining over transaction stream (FSM-TS). Unlike itemset mining, the task of FSM-TS is required to handle sequences of item sets and thus involved in the highly combinatorial explosion of mining objects. The proposed method addresses this problem by extending two key techniques, called incremental intersection and resource-oriented approximation, in the context of FSM-TS. In this paper, we briefly describe the validness of it and show a preliminary result in the experiment obtained using real datasets.

1 Introduction

FSM-TS is one of the most general mining tasks in streaming data mining. The target data is streaming transactions with variable length L where the well-studied single stream is the specific case of L = 1. FIM-TS is used to derive co-occurrences and sequences appeared frequently in transactions, and thus regarded as an extension of such fundamental tasks as frequent itemset mining (FIM) [8] and frequent sequence of items mining (FISM) [7]. This task is widely applicable to streaming data analysis especially for extracting explanatory variables with the discrete structure. Indeed, it is essential to find causalities (episode discovery [1, 5]) in real-time over the input text data (e.g., twitters, newspaper and SNS). However, compared with other fundamental tasks that have been intensively studied (i.e., FIM and FISM), it is still poorly investigated to establish a reasonable solution for FSM-TS due to the huge search space.

There is a recent work to integrate the compression and approximation techniques in FIM [3, 6, 9]. The literature presents an one-pass approximation algorithm for obtaining a lossy-compressed form of the frequent itemsets. This method is composed of two key techniques: the exact mining, called incremental intersection, for finding the closed itemsets and so-called resource-oriented approximation mining. In this paper, we reconstruct them in the context of FSM-TS and propose an one-pass algorithm for efficiently finding the frequent sequences of itemsets. We next show the validness of the output as well as a preliminary result in the experiment. For the space limitation, we omit the proof of theorems and select the experimental result to be put in the paper.

2 Notion and terminologies

We briefly review the notations and terminology in the paper. Let I = \{x_1, \ldots, x_n\} be the universe set of items. An itemset is a non-empty subset of I. A sequence \( \alpha \) is an ordered list of itemsets, denoted by \( \langle s_1 \cdots s_n \rangle \), where \( s_i \) (1 \( \leq i \leq n \)) is an itemset \( \{a_1, a_2, \ldots, a_m\} \). It is abbreviated by \( (a_1 a_2 \cdots a_m) \) for simplicity. We call \( n, s_n \) and \( s_1 \) by the width, head and tail of \( \alpha \), respectively. Let \( \alpha = \langle s_1 \cdots s_n \rangle \) and \( \beta = \langle t_1 \cdots t_m \rangle \) be two sequences. \( \alpha \) is a subsequence of \( \beta \) if there exist integers \( i_1 < \cdots < i_n \) such that \( s_k \subseteq t_{i_k} \) for each \( k \) (1 \( \leq k \leq n \)). \( \alpha \circ \beta \) denotes the concatenation of \( \beta \) to \( \alpha \).
A transaction stream $S$ is an unbounded single sequence of itemsets (i.e., transactions) with variable-length. $S$ is written as $\langle t_1 \cdots t_n \rangle$ when the output is requested at time $n$. Given $S$, we aim at seeking for the frequent subsequences in it. We use so-called head frequency measure [1, 4] to define the frequency of a sequence over the stream.

**Definition 1 (Window)** Let $S = \langle t_1 \cdots t_n \rangle$ be the stream and $k$ be the maximal width of target subsequences to be searched. The window at time $i$, denoted by $\text{win}(i)$, is the subsequence $\langle t_{e(i)} \cdots t_i \rangle$, where

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

**Definition 2 (Head frequency)** Let $\alpha = \langle s_1 \cdots s_m \rangle$ be a sequence. The head frequency of $\alpha$ at time $n$ wrt $S$ is defined as $\text{sup}(\alpha, n) = \sum_{i=1}^{n} \text{include}(\text{win}(i), \alpha)$, where the function $\text{include}(\text{win}(i), \alpha)$ is given as:

$$w(\alpha, i) = \begin{cases} 1 & \text{if } s_m \subseteq t_i \text{ and } \alpha \text{ is a subsequence of } \text{win}(i) \\ 0 & \text{otherwise} \end{cases}$$

The head frequency focuses on the occurrence under the set-inclusion condition between two head elements (i.e., $s_m$ and $t_i$). Thus, there is no overlap in the occurrences captured by this measure. Besides, the head frequency ensures anti-monotonicity criterion with respect to the following inclusion relation [4].

**Definition 3** Let $\alpha = \langle s_1 \cdots s_m \rangle$ and $\beta = \langle t_1 \cdots t_m \rangle$ be two sequences. $\alpha$ includes $\beta$, denoted by $\alpha \supseteq \beta$, if $s_n \subseteq t_m$ and $\alpha$ is a subsequence of $\beta$. $\alpha$ properly includes $\beta$, denoted by $\alpha \supset \beta$, if $\alpha \supset \beta$ but $\alpha \nsubseteq \beta$.

**Lemma 1** [4] If $\alpha \supseteq \beta$, $\text{sup}(\alpha, n) \geq \text{sup}(\beta, n)$.

Let $\text{sup}(\alpha, i)$ be the support of $\alpha$ at time $i$ ($i \leq n$). A sequence $\alpha$ is frequent if $\text{sup}(\alpha, n) \geq \sigma n$ for a minimal support threshold $\sigma$ ($0 < \sigma \leq 1$). The FSM-TS task is finding the frequent sequences over the stream $S$. Anti-monotonicity brings the concept of closedness compression. Let $F(i)$ be the frequent sequences at time $i$. A sequence in $F(i)$ is closed if there is no sequence in $F(i)$ that properly includes it. $CF(i)$ denotes the closed frequent sequences. $F(i)$ can be restored from $CF(i)$ without generating any compression error (i.e., lossless compression). Due to memory efficiency, it is desirable to seek for $CF(i)$.

The difficulty lies in how to efficiently obtain it for each time $i$ in online processing. Unlike statistic database, it is required to process streaming transactions, continuously arriving at high speed. In this paper, we develop an incremental way to manage the closed sequences in one-pass approximation setting.

### 3 Incremental intersection

The key idea is based on the technique of incremental intersection, which has been proposed in context of frequent itemset mining (FIM) [2, 10], where the mining object is an itemset in $I$. The literature [2, 10] has revealed a cumulative and incremental way to update the closed itemsets based on the following theorem:

**Theorem 1** [10] Let $T(i)$ be the closed itemsets at time $i$, and $t_{i+1}$ be the transaction at $i+1$. $T(i+1) = T(i) \cup \{\beta \mid \beta = \alpha \cap t_{i+1}, \beta \neq \emptyset, \alpha \in T_i \}$.

By Theorem 1, the closed itemsets at the next time can be derived by computing the intersection of each stored itemset with the new transaction. The incremental interect is applicable to FSM. Figure 1 describes the intuition how the closed sequences $TS(i)$ can be incrementally updated for each time $i$. Note that our mining objects are sequences of itemsets. Hence, it is necessary to define the intersection of a sequence with the window. Since the window is also a sequence, it is equivalent to define the intersection of two sequences. However, it is not obvious to an-
Definition 4 Let $\alpha$, $\beta$ and $\gamma$ be sequences. $\gamma$ is an intersection between $\alpha$ and $\beta$ if $\gamma \subseteq \alpha$, $\gamma \subseteq \beta$ and there is no $\gamma'$ such that $\gamma' \subset \gamma$, $\gamma' \subseteq \gamma$. 

Example 1 Let $\alpha = \langle (a)(b)(c) \rangle$ and $\beta = \langle (ab)(bc) \rangle$ be two sequences. There are two intersections between $\alpha$ and $\beta$, that is, $\langle (a)(c) \rangle$ and $\langle (b)(c) \rangle$.

The (sequence) intersection in FSM is not necessarily unique, which is unlike the (itemset) intersection in FIM. int$(\alpha, \beta)$ denotes the set of intersections between $\alpha$ and $\beta$. The validity of incremental intersection for FSM is described as follows:

Theorem 2 Let $TS(i)$ be the closed sequences at time $i$ and $\text{win}(i+1)$ be the next window. Then,

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

4 Finding the intersections

We need to develop an efficient way to compute int$(\alpha, \text{win}(i+1))$. At first, it is infeasible to generate every candidate subsequence of $\alpha$ and $\text{win}(i+1)$, and then check if it is closed or not. Indeed, there are $O(2^{KL})$ candidates to be checked for the maximal transaction length $L$ and sequence width $k$. To avoid such generate-and-test approach, we focus the search only on the following cross table:

Definition 5 Let $\alpha = \langle s_1 \cdots s_n \rangle$ and $\beta = \langle t_1 \cdots t_m \rangle$ be two sequences ($n \leq m$). The cross table of $\alpha$ with $\beta$ is the two-dimensional table $T$, where for each $i$ ($1 \leq i \leq n$) and $j$ ($1 \leq j \leq m$), the $i$-th column and $j$-th row element, denoted by $e(i, j)$, corresponds to $s_i \cap t_j$. For some integers $i$ and $j$, the area $A(i, j)$ is the set of elements $\{T[u][v] | i \leq u < n, j \leq v < m\}$.

Definition 6 Let $A(i, j)$ be an area in the cross table. A path wrt $A(i, j)$ is a list $\langle e(i_1, j_1) \cdots e(i_t, j_t) e(n, m) \rangle$ such that $e(i_u, j_u) \in A(i, j)$ ($1 \leq u \leq l$), $i_1 \leq i_2, \cdots, i_t < n$ and $j_1 \leq j_2, \cdots, j_t < m$. Given a path $p$, $S(p)$ denotes the sequence of $p$ obtained by removing the empty elements from $p$. Let $p$ and $q$ be two paths. We say $p$ includes $q$ if $S(p) \supseteq S(q)$. A path $p$ is maximal wrt $A(i, j)$ if $p$ is a path wrt $A(i, j)$ and there is no path $q$ wrt $A(i, j)$ such that $S(q) \supseteq S(p)$.

Example 2 Consider the following two sequences:

$\alpha = \langle (abc)(cd)(c)(bc) \rangle, \beta = \langle (a)(ac)(c)(abcd)(b) \rangle$.

There are four intersections $I_1$, $I_2$, $I_3$ and $I_4$ between $\alpha$ and $\beta$ in Figure 3, each of which corresponds to one maximal path wrt $A(1, 1)$.

Proposition 1 Let $p = \langle e(i_1, j_1) \cdots e(i_t, j_t) e(n, m) \rangle$ be a maximal path wrt $A(1, 1)$. Then, the subpath $\langle e(i_u, j_u) \cdots e(n, m) \rangle$ is also maximal wrt $A(i_h, j_h)$, for every $u$ ($1 < u \leq t$).

By Proposition 1, we can extend the path $p$ with the new element $e(x, y)$ if and only if (1) $p$ is maximal wrt $A(i_1, j_1)$ where $e(i_1, j_1)$ is the prior tail of $p$ and (2) the extended path is also maximal wrt $A(x, y)$. Every next element of the prior tail $e(i_1, j_1)$ is included either in the following two sets:

$C_h(i_1, j_1) = \{e(x, y) | 1 \leq x < i_1, y = j_1 - 1\}$,

$C_v(i_1, j_1) = \{e(x, y) | x = i_1 - 1, 1 \leq y < j_1 - 1\}$.

$C_h$ (resp. $C_v$) is called the horizontal (resp. vertical) set. Any maximal path can be generated by continuing to select the next tail element from either $C_h$ or $C_v$. Figure 3 sketches the search process: we first select the element $e(i, j)$ for $C_h(n, m)$ and select the next from $C_v(i, j)$. These step-wise-selected elements are concatenated one by one.

Both horizontal and vertical sets include some elements that are prohibited or redundant to be selected. Let $t = e(i, j)$ be the tail element of the prior path.

Definition 7 (Nearest superset) $e(u, j)$ ($u < i$) is the nearest superset of $t$ in horizontal, denoted by $n_h(t)$, if $e(u, j) \supseteq e(i, j)$ and there is no $u'$ such that $u < u' < i$ and $e(u', j) \supseteq e(i, j)$. In turn, $e(i, u)$ ($u < j$) is the nearest superset of $t$ in vertical, denoted by $n_v(t)$, if $e(i, u) \supseteq e(i, j)$ and there is no $u'$ such that $u < u' < j$ and $e(i, u') \supseteq e(i, j)$.

Based on the above notion, we define the selectable elements in $C_v(i, j)$ and $C_h(i, j)$ as follows.
Algorithm 1 Finding the prime paths

Input: $\alpha = \langle s_1 \cdots s_n \rangle$ and $\beta = \langle t_1 \cdots t_m \rangle$

Output: the set $U$ of prime paths (PPs) wrt $A(1,1)$

1: initialize the candidate path $\phi$ and set $U$ of PPs
2: create the cross table $T$ of $\alpha$ with $\beta$
3: if $e(n,m) \neq \emptyset$ then
4: \hspace{1cm} terminate \hspace{1cm} $\triangleright$ the head is empty
5: else
6: \hspace{1cm} $\phi := (e(n,m)) \circ \phi$
7: \hspace{1cm} if $n == 1$ or $m == 1$ then
8: \hspace{2cm} add $\phi$ to $U$
9: else
10: \hspace{2cm} compute the non-redundant elements $N$
11: \hspace{2cm} for each element $e(x,y)$ in $N$ do
12: \hspace{3cm} $\phi := (e(x,y)) \circ \phi$
13: \hspace{3cm} call intersect($T$, $x$, $y$, $\phi$, $U$)
14: \hspace{2cm} end for
15: \hspace{1cm} end if
16: \hspace{1cm} return $U$
17: end if

\begin{function} \text{intersect}(T, u, v, \phi, U) \end{function}

18: search the nearest supersets $n_h$ and $n_v$
19: if $u == 1$ and $n_v$ does not exist then
20: \hspace{1cm} add $\phi$ to $U$
21: else if $v == 1$ and $n_h$ does not exist then
22: \hspace{1cm} add $\phi$ to $U$
23: else
24: \hspace{1cm} compute the selectable elements $S(u,v)$
25: \hspace{1cm} for each element $e(x,y)$ in $S$ do
26: \hspace{2cm} if the condition in Lemma 5 holds then
27: \hspace{3cm} $\phi := (e(x,y)) \circ \phi$
28: \hspace{3cm} call intersect($T$, $x$, $y$, $\phi$, $U$)
29: \hspace{2cm} end if
30: \hspace{1cm} end for
31: \hspace{1cm} end if
32: \hspace{1cm} return
33: \end{function}

Definition (Selective elements) Let $e(i,j)$ be the tail element $t$. $e(x,y)$ in $C_h(i,j)$ (resp. $C_v(i,j)$) is prohibited wrt $t$ if there is the nearest superset $n_h(t) = e(u,j)$ (resp. $n_v(t) = e(i,u)$) of $t$ and $x < u$ (resp. $y < u$), $e(x,y)$ in $C_h(i,j)$ (resp. $C_v(i,j)$) is redundant wrt $t$ if there is an element $e(x',y)$ (resp. $e(x,y')$) such that $x < x' < i$ (resp. $y < y' < j$) and $e(x,y) \not\subseteq e(x',y)$ (resp. $e(x,y') \not\subseteq e(x,y)$). $e(x,y)$ in $C_h(i,j)$ (resp. $C_v(i,j)$) is selectable wrt $t$ if $e(x,y)$ is neither prohibited nor redundant.

Definition (Prior path) Let $p$ be a path wrt $A(i,j)$ whose tail is $e(i_p,j_p)$. $p$ is prior in horizontal (resp. vertical) if there is no path $q$ whose tail is $e(i_q,j_q)$ such that $S(q) = S(p)$, $i_p > i_q$ (resp. $j_p > j_q$) and $j_p = j_q$ (resp. $i_p = i_q$). We say that $p$ is prior, if $p$ is prior both in horizontal or vertical.

We call by a prime path (PP) such a path that is prior and maximal wrt $A(i,j)$. It is sufficient to seek only for the prime paths (PPs) wrt $A(1,1)$. Let $p$ be a prime path wrt $A(u,v)$ whose tail is $e(u,v)$, and $S(u,v)$ be the set of selectable elements wrt $e(u,v)$.

We clarify the condition that the extended path $(e(x,y)) \circ p$ becomes prime. Let $e(u_p,v_p)$ be the non-empty element in $p$ lastly appeared before $e(u,v)$.

Lemma 2 Let $e(u,v)$ be the tail of the prime path $p$. For every $e(x,y) \in S(u,v)$, $(e(x,y)) \circ p$ is prime wrt $A(x,y)$, provided that the following is satisfied:

Case(1) both $e(x,y)$ and $e(u,v)$ are non-empty;
Case(2) if $e(x,y)$ is empty and $e(u,v)$ is non-empty, then $e(x,v) \not\subseteq e(u,v)$ and $e(u,y) \not\subseteq e(u,v)$;
Case(3) if $e(x,y)$ is non-empty and $e(u,v)$ is empty, then $e(x,v) \not\subseteq e(x,y)$ and $e(u,y) \not\subseteq e(x,y)$;
Case(4) if $e(x,y)$ and $e(u,v)$ are empty, $e(x,v_p) \not\subseteq e(u_p,v_p)$ and $e(u_p,y) \not\subseteq e(u_p,v_p)$. 
Algorithm 1 sketches the procedure for finding the prime paths between two sequences \( \alpha \) and \( \beta \).

**Theorem 3** Let \( \Lambda \) be the area of the cross table of \( \alpha = (s_1 \cdots s_n) \) with \( \beta = (t_1 \cdots t_m) \). The output of Algorithm 1 corresponds to the prime paths wrt \( \Lambda(1,1) \).  

5 RO approximation in FSM

One crucial drawback of incremental intersection lies in the huge memory consumption: \(|TS(i)|\) always increases as the time \( i \) passes over. Besides, \(|int(\alpha, win(i+1))|\) is not necessarily unique: many intersections can be generated. This makes the increase of \(|TS(i)|\) more exhaustive. There are recent works [3, 6, 9] to embed the (counter-based) approximation technique into incremental intersection for FIM. This technique is used to maintain the frequency of a mining object \( \alpha \) at each time \( i \), denoted by \( c(\alpha, i) \), allowing to include some error count \( \Delta \) such that \( c(\alpha, i) - \Delta \leq sup(\alpha, i) \leq c(\alpha, i) \) holds.

Analogous to the previous FIM work, it is considerable to introduce the notion of so-called resource-oriented (RO) approximation [8] to incremental intersection for FSM. In brief, the RO approximation enables to fix the memory resource to be consumed using a size constant \( r \) \((r > 0)\). Only \( r \) counters are used to maintain the frequencies of mining objects. The key idea is simple: \( TS(i) \) is updated with \( win(i+1) \), and if \(|TS(i)| > r\), we delete the “ignoreable” mining objects with lower frequencies so that the reduced \( TS(i) \) stores at most \( r \) objects. In the following, \( TS(i) \) is composed of the data entries each of which is a tuple of form \( (\alpha, c(\alpha, i), \Delta(\alpha, i)) \) where \( \alpha \) is the stored sequence, \( c(\alpha, i) \) is the frequency of \( \alpha \) at time \( i \), and \( \Delta(\alpha, i) \) is the (maximal) error count in \( c(\alpha, i) \). \( \Delta(i) \) denotes the maximal error count in \( TS(i) \). Algorithm 2 describes the update procedure for \( TS(i) \) with the RO approximation. Note that \( buf \) is the buffer size, corresponding to the maximal number of candidate entries to be stored in memory. In case that \(|C|\) is greater than \( buf \), we immediately stop to generate new intersections. In this case, we check if the stored sequence \( \alpha \) is included in the window (Lines 36-42). If not, it is sufficient just to increment both \( c(\alpha, i) \) and \( \Delta(\alpha, i) \), instead of computing \( int(\alpha, win(i+1)) \). We call this intersect-skip.

The RO-approximation appears in Lines 29-34. In case that \(|TS(i)| \) is greater than the size constant \( r \), we remove the entry with the minimal frequency from \( TS(i) \) one by one. Then, the frequency of the removed entry is maintained as \( \Delta(i) \). Consequently, Algorithm 2 achieves the completeness for finding the frequent sequences, described as follows:

**Theorem 4** Let \( S = (t_1 \cdots t_n) \) be the stream, \( \sigma \) the minimal support threshold and \( F(n) \) the frequent sequences over \( S \) wrt \( \sigma \). If \( \Delta(n) \geq \sigma n \), for every \( \alpha \in F(n) \), \( TS(n) \) stores an entry for \( \beta \) such that \( \beta \supseteq \alpha \) and \( \sup(\beta, n) \leq \sup(\alpha, n) \leq \sup(\beta, n) + \Delta(\beta, n) \).

6 Preliminary experiment

We have implemented the proposed method by C and evaluated its performance using two kinds of real datasets (Kosarak in FIMI and Weblog in [8]). Figures 5 and 6 show the total execution time (sec) and total error count in accordance with varying the sequence width \( k \) and the size constant \( r \), respectively.

7 Concluding remarks

This paper studies frequent sequence mining from transaction stream (FSM-TS). We apply the incremental intersect to FSM-TS, which is then integrated into the resource-oriented approximation. By Theorem 4, it is possible to restore the frequent sequences from the output, allowing the approximation error count (\( \Delta(n) \) in maximal). Hence, we succeed in lossy (but accuracy-preserved) compression for the frequent sequences. It is required to furthermore investigate the scalability the width \( k \) and the size constant \( r \).
We also intend to apply the method to real text data to detect causalities in it.

参考文献


Algorithm 2 Incremental intersection

Input: $TS(i), win(i+1), \Delta(i), r, buf$

Output: $TS(i+1)$

1: initialize $C$  // the candidate entries to be stored
2: $e := get(win(i+1), TS(i))$
3: if $e$ is null then  // no entry for win(i+1)
4:    add $\langle win(i+1), \Delta(i), \Delta(i) \rangle$ to $TS(i)$
5: end if
6: for each sequence $\alpha$ stored in $TS(i)$ do
7:    if $|C| > buf$ then  // $buf$ is the buffer size
8:      skip($\alpha, win(i+1)$)  // intersect-skip
9:    else
10:       compute int($\alpha, win(i+1)$)  // Algorithm 1
11:       for each $\beta \in int(\alpha, win(i+1))$ do
12:          $e_\beta := get(\beta, C)$  // the entry $e_\beta$ for $\beta$
13:          if $e_\beta$ is null then
14:              add $\langle \beta, C, \alpha, i + 1, \Delta(\alpha, i) \rangle$ to $C$
15:          else if $c(\beta, i) < c(\alpha, i) + 1$ then
16:              $c(\beta, i) := c(\alpha, i) + 1$
17:              $\Delta(\beta, i) := \Delta(\alpha, i)$
18:          end if
19:       end for
20:    end if
21: end for
22: for each sequence $\gamma$ stored in $C$ do
23:    $e_\gamma := get(\gamma, TS(i))$  // the entry $e_\gamma$ for $\gamma$
24:    if $e_\gamma$ is null then  // no entry for $\gamma$ in $TS(i)$
25:        add the entry for $\gamma$ to $TS(i)$
26:    else
27:        replace $e_\gamma$ with the entry for $\gamma$
28:    end if
29: end for
30: while $|TS(i)| > r$ do
31:    $m := getMin(TS(i))$
32:    // $m$ is a sequence with the minimal frequency
33:    $\Delta(i) := e(m, i)$  // updating $\Delta(i)$
34:    delete($TS(i), m$)  // deleting the entry for $m$
35: end while
36: return $TS(i)$  // corresponding to $TS(i+1)$

37: function $\text{SKIP}(\alpha, \beta)$
38: if $\alpha \subseteq \beta$ then
39:    increment $c(\alpha, i)$ by one
40: else
41:    increment both $c(\alpha, i)$ and $\Delta(\alpha, i)$ by one
42: end if
43: end function