Resource-oriented Approximation for Frequent Itemset Mining from Bursty Data Streams

Yoshitaka Yamamoto
University of Yamanashi
yyamamoto@yamanashi.ac.jp

Koji Iwanuma
University of Yamanashi
iwanuma@yamanashi.ac.jp

Shoshi Fukuda
University of Yamanashi
fuku@iwlab.org

ABSTRACT

This study considers approximation techniques for frequent itemset mining from data streams (FIM-DS) under resource constraints. In FIM-DS, a challenging problem is handling a huge combinatorial number of entries (i.e., itemsets) to be generated from each streaming transaction and stored in memory. Various types of approximation methods have been proposed for FIM-DS. However, these methods require almost $O(2^L)$ space for the maximal length L of transactions. If some transaction contains sudden and intensive *bursty* events for a short span, they cannot work since memory consumption exponentially increases as L becomes larger. Thus, we present *resource-oriented* approximation algorithms that fix an upper bound for memory consumption to tolerate bursty transactions. The proposed algorithm requires only $O(k)$ space for a resource-specified constant k and processes every transaction in $O(kL)$ time. Consequently, the proposed algorithm can treat any transaction without memory overflow nor fatal response delay, while the output can be guaranteed to be no false negative under some conditions. Moreover, any (even if false negative) output is bounded within the approximation error which is dynamically determined in a resource-oriented manner. From an empirical viewpoint, it is necessary to maintain the error as low as possible. We tackle this problem by dynamically reducing the original stream. Through experimental results, we show that the resource-oriented approach can break the space limitation of previously proposed FIM-DS methods.

Categories and Subject Descriptors

H.2 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms, Theory, Experimentation

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2612171.

Keywords

Itemset mining; Bursty data stream; Approximation

1. INTRODUCTION

A data stream is an unbounded sequence of data arriving at high speed [15]. Frequent itemset mining from data streams (FIM-DS) has been extensively studied with a diversity of applications, such as monitoring surveillance systems, communication networks, and internet traffic as well as mining patterns from on-line transactions in the financial market, retail industry, and electric power grids [2, 4, 6, 7, 13]. FIM-DS must treat the potentially infinite volumes of transactions that are infeasible to store in memory and scan multiple times. Therefore, research interests have been focused on *one-scan approximation* algorithms [8, 9, 10, 12, 14, 15]. These approximation algorithms are divided into two types according to the management of memory consumption. One type of algorithm deletes "unpromising" itemsets that have lower frequency among the current memory, which is triggered by receiving a new transaction. The Lossy Counting (LC) algorithm [12] and the Frequent algorithm [8, 9] are examples of the deletion approach. The other is based on a random sampling approach that stochastically selects transactions to be processed, such as the Sticky Sampling algorithm [12] and the Chernoff-based algorithm [15]. Both approaches provide some guarantee that the resulting itemsets have frequencies with errors bounded by a given parameter; thus, they can return no false negatives under some conditions.

However, these *parameter-oriented* approaches have some crucial drawbacks. The sampling approach must make some statistical assumptions relative to data distribution such as independence. In contrast, the deletion approach does not need to make such an assumption; however it must require $O(2^L)$ space for the maximal length L of transactions in worst case. This may result in a serious "out of memory" crash if the memory consumption goes beyond a system's boundary. Modern FIM-DS methods [8, 10, 12] use compact data structures, such as a prefix tree using a hash table, to manage itemsets in memory efficiently. On the other hand, real data streams often meet sudden and intensive bursty events that are significantly larger than what the system assumes, such as meteorological data associated with large-scale disasters (e.g., earthquakes and hurricanes), web traffic data caused by DOS attacks and web text data on twitter or SNS under fire. These extremely large bursty data streams can cause crucial memory overflow in previously proposed approaches.

Thus, we address the problems in FIM-DS by introducing the following resource constraints.

- *Boundary of memory consumption*:
  We fix the upper bound of the number of entries in order to prevent memory overflow in case bursty events occur, which are beyond the scope of user assumption.

- *Boundary of memory access times*:
  We fix the maximal number of candidate sets from each transaction, which are newly stored in memory. This can prohibit a fatal delay of response time.

The first constraint may be considered similar to *top-k* frequent pattern mining [3, 14, 15]. Note that the parameter k is related to the number of solutions *to be outputted*. Thus, k does not indicate the memory size *to be consumed*. To the best of our knowledge, the first constraint has not yet been introduced in online (top-k) itemset mining, though previous work has examined online top-k item mining [14] and offline top-k itemset mining [3].

In this study, we examine online approximation algorithms for FIM-DS under such constraints. We first propose the *LC-SS* algorithm that is achieved by integrating LC and the Space Saving (SS) algorithms. The original SS algorithm was proposed for *online top-k item mining* [14]. It bounds the memory consumption in such a way that if the memory is full, the most infrequent item is replaced with the newly arrived item. Using the concept of replacement, the LC-SS algorithm ensures that the memory consumption is bounded in $O(k)$ space for some constant k. Note that k denotes the maximal number of entries to be stored in memory. Then, k can be fixed as a resource-specified value.

Metwally [14] shows that the SS algorithm requires at least $min(|I|, \frac{1}{2\epsilon})$ space for an error parameter ϵ, where I is the set of items in a stream. In this paper, we show that parameter-oriented methods require at least $min(|I_s|, \frac{\Sigma(2^{|T_i|}-1)}{\epsilon N})$ space, where I_s is the set of itemsets in a stream consisting of N transactions and T_i is the transaction at time i ($1 \leq i \leq N$). Due to this exponential lower bound of space requirement, it is difficult to prepare appropriate memory for ensuring a fixed parameter ϵ. Alternatively, the LC-SS uses the technique of LC algorithm that monitors the error count of currently stored itemsets to approximate their supports. Thus, the LC-SS algorithm achieves no false-negative behavior for finding frequent itemsets provided the final error count is less than the minimal support. There is another work [10] on bursty data streams, which deals with the ratio explosion problem caused by intensively arriving transactions for a short span. Unlike this problem, we tackle the combinatorial explosion problem caused by rarely arriving bursty transactions that contain a large number of items.

We furthermore propose the *Skip* LC-SS algorithm. The key feature of this algorithm is to *skip over transactions* by terminating the current process when the number of candidate subsets to be newly stored exceeds some upper boundary. Using this feature, the Skip LC-SS algorithm can derive outputs in $O(kLN)$ time, whereas their frequencies are approximated within the same error count as the original one. From an empirical viewpoint, this error count must be kept as low as possible. We address this problem using the so-called *stream reduction* technique. Most items in a bursty stream occur only a few times. Infrequent items can be identified by some online item mining methods. Thus, they should be reduced from the original stream before performing the Skip LC-SS algorithm. The concept of stream reduction has been already introduced in some FIM-DS methods [12, 8]. These methods require $O(n)$ space for n item types, since they exactly count the frequency of each item. In contrast, we apply an approximation algorithm for finding frequent items, and integrate it with the Skip LC-SS algorithm.

We have empirically investigated how the proposed algorithms work in real data streams with bursty events, such as time series data for 1982-2013 earthquake occurrences in Japan. The experimental results show that the Skip LC-SS algorithm can tolerate any bursty transaction without memory overflow or response time fatal delay. The results also show that the stream reduction can inhibit error gain by up to 90%, and can speed up the Skip LC-SS process by a factor of approximately 30.

The remainder of this paper is organized as follows. Section 2 is a preliminary and briefly introduces previously proposed algorithms. We reveal the exponential lower bound of space requirement in Section 3, and propose the LC-SS and Skip LC-SS algorithms in Section 4 and 5, respectively. Next, we embed the stream reduction into the Skip LC-SS algorithm in Section 6. We then conclude in Section 7.

2. PRELIMINARY AND BACKGROUND

Here, we briefly review the notations and terminology used in this paper. Let $I = \{x_1, x_2, \ldots, x_u\}$ be a set of items. An *itemset* is a non-empty subset of I. Transactional data stream $\mathcal{S}$ is a sequence of incoming transactions $\langle T_1, T_2, \ldots, T_N \rangle$, where a *transaction* T_i is an itemset arriving at time i and N is an unknown large number of transactions. *Mining objects* for $\mathcal{S}$ are configured in accordance with various mining tasks: in the context of itemset (*resp.* item) mining, they correspond to itemsets (*resp.* items). Note that $mo(T_i)$ is the set of mining objects generated from a transaction T_i. For a transactional data stream $\mathcal{S}$, we denote by ave and n_{mo} the average size of $mo(T_i)$ and the cardinarity of the set of mining objects in $\mathcal{S}$, respectively. In this study, $mo(T_i)$ consists of $2^{|T_i|} - 1$ itemsets. Accordingly, $ave = \frac{\Sigma(2^{|T_i|}-1)}{N}$ and $n_{mo} = |(2^{T_1} \cup \cdots \cup 2^{T_N}) - \{\emptyset\}|$. Let α be a mining object. The *support* of α, denoted by $sup(\alpha)$, is the number of transactions in $\mathcal{S}$ that contain α. Given a *minimal support* threshold σ such that $\sigma \in (0, 1)$, α is *frequent* if $sup(\alpha) \geq \sigma N$. The task of FIM-DS is finding all frequent objects (itemsets) from $\mathcal{S}$.

Modern FIM-DS algorithms maintain the counter value of each mining object in memory, called the *frequency count*. We call the entity consisting of a mining object α and its frequency count $c(\alpha)$ in memory an *entry*. An *entry table* D is a table storing entries in memory. $|D|$ denotes the number of entries in D. A *minimal entry* is an entry α whose frequency count $c(\alpha)$ is minimal in D. Given a parameter ϵ ($0 < \epsilon \leq \sigma$), an algorithm is ϵ-*accurate* if for every frequent object α, it holds that $c(\alpha) - \epsilon N \leq sup(\alpha) \leq c(\alpha)$, and is ϵ-*honest* if $c(\alpha) \leq \epsilon N$ holds for every α that is not registered in D. We call ϵ an *error parameter*.

2.1 Lossy Counting algorithm

The LC algorithm [12] is a well-known one-scan approximation algorithm. Given a minimal support σ, an error parameter ϵ ($0 < \epsilon \leq \sigma$), and a data stream with N transactions, the LC algorithm outputs every itemset whose sup-

port is greater than or equal to $(\sigma - \epsilon)N$. Thus, the LC algorithm is no false negative. In this algorithm, the entry for an itemset α is represented by the tuple form of $\langle \alpha, f(\alpha), \Delta(t) \rangle$, where $f(\alpha)$ is the number of occurrences of α after the time t when α was lastly stored and $\Delta(t)$ is the *error count* at time t. We often denote by Δ_α the error count $\Delta(t)$ of α. The LC algorithm gives the frequency count $c(\alpha)$ as $f(\alpha) + \Delta_\alpha$. For each time t, the LC algorithm deletes every infrequent itemset α such that $c(\alpha) \leq \epsilon \times t$ (ϵ-deletion). The next error count $\Delta(t+1)$ is updated with $\epsilon \times t$. Thus, the error count is greater than or equal to the frequency count of any itemset that has been previously deleted. Using the notion of the error count, the LC algorithm approximates $sup(\alpha)$ such that $f(\alpha) \leq sup(\alpha) \leq f(\alpha) + \epsilon N$.

The challenging problem in itemset mining lies in the combinatorial explosion of memory consumption. In principle, L type of items can generate 2^L itemsets to be stored or updated in the entry table. The LC algorithm controls memory consumption by ϵ-deletion. However, the LC algorithm must generate (and check) every transaction subset at least once. This may cause memory overflow, especially in data streams with extensive bursty transactions that contain a lot of newly appearing items. In this study, we consider approximation algorithms that can handle combinatorial explosion of memory consumption.

2.2 Space-Saving algorithm

A key idea shown in this paper lies in the SS algorithm, which was proposed for item stream mining [14]. In this algorithm, the maximal table size of entries is fixed as some constant. Given a constant k and an entry table D, the SS algorithm registers a newly arriving item whenever $|D| < k$. However, if the entry table is full (i.e., $|D| = k$), the SS algorithm *replaces* the minimal entry with the new entry. Note that an entry is represented by the form of $\langle \alpha, c(\alpha) \rangle$, where $c(\alpha)$ is the frequency count of an item α. Given $S = \langle e_1, e_2, \ldots, e_N \rangle$ where e_i is an item, the SS algorithm works as follows, for each e_i, do

1. if $\langle e_i, c(e_i) \rangle \in D$, increment $c(e_i)$ by one,
2. else if $|D| < k$, store the new entry $\langle e_i, 1 \rangle$ in D,
3. else, replace the minimal entry $\langle m, c(m) \rangle$ with the new entry $\langle e_i, 1 + c(m) \rangle$.

Now, let ϵ be an error parameter, similar to the LC algorithm. If $k \geq 1/\epsilon$ holds, then the SS algorithm can output every item e for which $sup(e) \geq \epsilon N$. Therefore, given a minimal support σ $(0 < \epsilon < \sigma)$, every frequent item e for which $sup(e) \geq \sigma N$ can also be the output (no false-negative) and $sup(e) \leq c(e) \leq sup(e) + \epsilon N$ holds. Hence, the SS algorithm requires at least $O(1/\epsilon)$ space to be ϵ-accurate.

Example 1. Let the stream $\mathcal{S}$ be $\langle a, b, a, c, b, b \rangle$, the minimal support σ be 0.5, and the error parameter ϵ be 0.5. We fix constant k as 2 (i.e., $1/\epsilon$). Figure 1 shows that two items a and b are stored at the end of time $i = 3$. At time $i = 4$, a new item c arrives; however, the entry table is full. Next, the minimal entry $\langle b, 1 \rangle$, whose frequency count is minimal, is replaced with $\langle c, 2 \rangle$. Finally, c is replaced with b once again at time $i = 5$. Thus, we obtain the frequent item b.

There are several differences between the LC and SS algorithms. The LC (*resp.* SS) algorithm uses the ϵ-deletion (*resp.* replacement) operation to maintain the entry table. Unlike ϵ-deletion scanning the whole table, the replacement

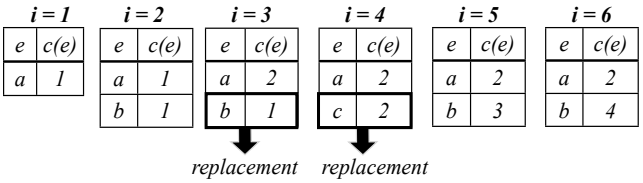

Figure 1: Updating the table during the stream

operation is performed only once for a minimal entry. These two algorithms approximate the support of a mining object α with the frequency count in their own ways. Unlike the SS algorithm, the LC algorithm distinguishes the error part (Δ_α) from the exact part (f_α). Note that monitoring Δ_α enables to check the accuracy of the frequency count on demand, which will be used for ensuring the approximation error in our proposed algorithms.

2.3 Related works

There has been previous work on top-k itemset mining using parameter-oriented algorithms [15]; however, they cannot fix memory consumption. Modern parameter-oriented methods [8, 10, 12] efficiently manage the entry table using their own data structures with prefix or suffix trees in the apriori manner. It is absolutely imperative for practical applications to use these efficient techniques. However, we cannot break the limitation of parameter-oriented approaches only with these techniques, as shown in Theorem 1 later. We may consider that this is caused only in the worst case and can fix relevant ϵ and k using prior knowledge on data streams to be processed, such as the occurrence distribution of items, the maximal length of transactions or frequent itemsets. However, real data streams often meet bursty transactions whose length is significantly larger than what the system assumes, such as meteorological data. In Japan, earthquake data has attracted significant attention. Figure 2 shows the time series transaction data for 1981-

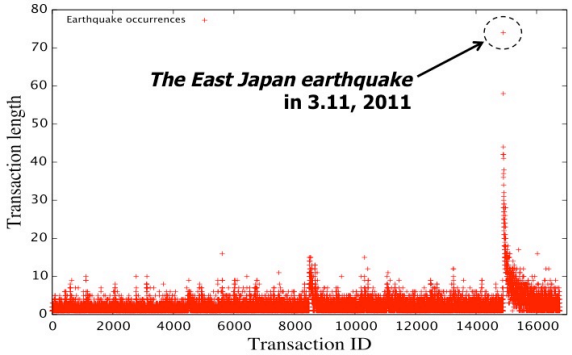

Figure 2: Earthquake occurrences in Japan

2013 earthquakes that have occurred in Japan (16768 transactions). In Fig. 2, the x and y axes represent the ID of each transaction and the number of items contained in the transaction, respectively. Fig. 2 shows that there are bursty transactions that correspond to the 2011 East Japan earthquake. We should emphasize that in real data streams, such "black swan" transactions whose scale is beyond the assumption, such as those evidences from the East Japan earthquake, may arrive at next time. Once it occurs, it may probably

cause parameter-oriented methods serious damage due to intensive memory consumption.

Another study has proposed a method for treating bursty transactions [11], in which the memory comsumption and the processing speed are maintained by dynamically regulating the error parameter ϵ. Their work however focuses on another type of problem in bursty data stream, i.e., the ratio explosion problem of arriving transactions per a unit time. In contrast, this study addresses the combinatorial explosion problem caused by the length of each transaction.

3. PARAMETER-ORIENTED V.S. RESOURCE-ORIENTED

We refer to methods that approximate the output with error as *parameter-oriented* approaches. Such approaches guarantee a user-specified parameter constraint. One drawback of this approach is the difficulty in setting proper parameters because users cannot predetermine what values are relevant for their problems. In this paper, we propose *resource-oriented* approaches which are approximation methods that guarantee a resource-specified constraint, such as memory consumption or data processing time. These approaches are similar to *top-k* mining, which aims to find the most frequent k elements. In top-k mining, the constraint k controls only the number of solutions, not a memory boundary to be consumed. Thus, it should be distinguished from the resource-oriented approach.

In *item* mining, the SS algorithm has proved that both parameter and resource-oriented approaches are compatible using the relationship between the table size k and the error parameter ϵ. However, this property cannot necessarily hold for *itemset* mining. Let ϵ be an error parameter and k be the number of entries. Now, we show a lower bound of k required for any ϵ-accurate and ϵ-honest algorithms.

LEMMA 1. *Let D be an entry table with size k, $\mathcal{S}$ be a stream with N transactions, and α be a frequent mining object from $\mathcal{S}$. There exist $\mathcal{S}$ and α such that $sup(\alpha) = \frac{N \times ave}{k+1}$ and α is not registered in D.*

PROOF. Let $\mathcal{S}$ be a transactional data stream, where n_{mo} is $k+1$ and each $mo(T_i)$ has the same number of objects: ave. There are $N \times ave$ occurrences of $k+1$ mining objects in total. Suppose that every object O_i $(1 \leq i \leq k+1)$ uniformly occurs through N transactions. Then, the support of each O_i is $\frac{N \times ave}{k+1}$. For any minimal support σ, we can set the value of ave so as to satisfy that $\sigma \leq \frac{ave}{k+1} < 1$. Then, every O_i is frequent wrt σ. Hence, there is a frequent object α from $\mathcal{S}$ such that $sup(\alpha) = \frac{N \times ave}{k+1}$ and α is not registered. □

THEOREM 1. *k is at least $min(n_{mo}, \frac{ave}{\epsilon} - 1)$ for every ϵ-accurate and ϵ-honest algorithm.*

PROOF. Since n_{mo} entries are enough to maintain the exact frequency count of every mining object, it must be a lower bound of k. By Lemma 1, there are $\mathcal{S}$ and α such that $sup(\alpha) = \frac{N \times ave}{k+1}$ and α is not registered in D. Every ϵ-honest algorithm must give the frequency count $c(\alpha)$ of α such that $c(\alpha) \leq \epsilon N$. If it is also ϵ-accurate, it must hold that $sup(\alpha) \leq c(\alpha)$. Then, we have $\frac{N \times ave}{k+1} \leq \epsilon N$. Thus, we obtain $\frac{ave}{\epsilon} - 1 \leq k$. Hence, a lower bound of k is described as $min(n_{mo}, \frac{ave}{\epsilon} - 1)$. □

Theorem 1 is achieved using the notion of *ϵ-honesty* implicitly described in [1], which characterizes the prior FIM-DS algorithms. Metwally [14] shows that a lower bound of k is $min(n_{mo}, \frac{1}{2\epsilon})$ in item stream mining. In contrast, Theorem 1 derives a new lower bound that is $min(n_{mo}, \frac{1}{\epsilon} - 1)$, since the case $ave = 1$ corresponds to the item stream mining, for the ϵ-accurate and ϵ-honest algorithms.

Theorem 1 also shows a crucial limitation of the parameter-oriented approaches in itemset mining. Given an error parameter ϵ, we require at least $\min(n_{io}, \frac{ave}{\epsilon} - 1)$ entries; however, $ave = \frac{\Sigma(2^{|T_i|}-1)}{N}$ increases exponentially in accordance with each transaction length T_i. This indicates that any parameter-oriented method may fail due to lack of resources especially for itemset mining with bursty transactions.

4. INTEGRATING LC AND SS ALGORITHMS

Now, we propose a resource-oriented algorithm to solve this problem, which is obtained by integrating the LC and SS algorithms. Unlike the previously proposed parameter-oriented methods, the proposed algorithm, called *LC-SS*, only requires $O(k)$ space for some constant k. The original SS algorithm uses only one newly arriving item for replacement when the entry table is full. In contrast, itemset mining must handle the possibility that $2^{|T_i|} - 1$ subsets may be replaced for each transaction T_i. For this task, we should consider how those itemsets are maintained with the SS algorithm. The $2^{|T_i|} - 1$ subsets can be classified into two groups: one is for sets that have been already registered in the table and the other is for sets that have not been registered. For the former group, it is sufficient to increment their frequency counters by one. Conversely, the latter group, referred to as the *candidate sets*, should be newly registered in some relevant manner.

Based on the SS algorithm, we can insert candidate sets unless the entry table is full. Otherwise, we can replace minimal entries with candidate sets. However, such SS-based algorithms require at least $min(n_{io}, \frac{ave}{\epsilon} - 1)$ space according to Theorem 1. Thus, it is not straightforward to ensure the approximation error in the context of itemset mining only with the original SS framework. In addition, we use the notion of the error count $\Delta(i)$ in the LC algorithm, and dynamically monitors the approximation error in the currently stored entries. Algorithm 1 describes the LC-SS algorithm with the above concepts of replacement and error count.

We represent each entry as a tuple $\langle \alpha, f(\alpha), \Delta(i) \rangle$, similar to the LC algorithm, where $f(\alpha)$ is the number of occurrences of α from the time i when α was lastly registered and $\Delta(i)$ is the error count at time i. Then, the LC-SS algorithm gives the frequency count $c(\alpha)$ of α as $f(\alpha) + \Delta(i)$. Hence, a minimal entry in the LC-SS algorithm means the one such that $f(\alpha) + \Delta(i)$ is minimal in the table. Note that $\Delta(i)$ means an upper bound of frequency counts of itemsets that are *deleted* at time i. Analogously, the LC-SS algorithm updates $\Delta(i)$ with the maximal frequency count of itemsets that are *replaced* at time i by way of Line 21 in Algorithm 1.

Example 2. Here, the stream $\mathcal{S}$ consists of four transactions $\{a\}$, $\{a, b\}$, $\{b, c\}$, and $\{a, b, c, d, e\}$. Let the minimal support σ be 0.6 and the maximal table size k be 4. Figure 3 illustrates how the entry table is updated by processing each transaction using Algorithm 1. At time $i = 3$, the table becomes full, and a minimal entry is replaced with one candidate set from T_3. $\Delta(4)$ is updated to the frequency count

Algorithm 1 Baseline (LC-SS) algorithm

Input: maximal table size k, minimal support σ, data stream $\mathcal{S} = \langle T_1, T_2, \ldots, T_N \rangle$.
Output: an approximated set of frequent itemsets from $\mathcal{S}$.

```
1: set i as 1 (i := 1)                          ▷ i is the current time
2: initialize D, Δ(1) and θ (D := ∅, Δ(1) := 0, θ := 0)
3: while i ≤ N do
4:     read T_i
5:     for each α ⊆ T_i s.t. α has been registered in D do
6:         f(α)++
7:     end for
8:     for each α ⊆ T_i s.t. α is not registered in D do
9:         if |D| < k then
10:            insert the new entry ⟨α, 1, Δ(i)⟩
11:        else
12:            select a minimal entry ME = ⟨β, f(β), Δ_β⟩
13:            replace ME with ⟨α, 1, Δ(i)⟩
14:            if θ < f(β) + Δ_β then
15:                θ := f(β) + Δ_β
16:                    ▷ θ will be used for updating Δ(i) later
17:            end if
18:        end if
19:    end for
20:    if Δ(i) < θ then
21:        Δ(i + 1) := θ and θ := 0          ▷ updating Δ and θ
22:    else
23:        Δ(i + 1) := Δ(i)
24:    end if
25:    i++                                   ▷ updating the time i
26: end while
27: for α ∈ D s.t. f(α) + Δ_α ≥ σN do
28:    output α ▷ α can be a frequent itemset wrt σ and N
29: end for
```

of the replaced entry at time $i = 3$ (i.e., $\Delta(4) = 1$). At time $i = 4$, every registered entry is a subset of T_4; thus, every f is incremented by one. Next, the two minimal entries are replaced with candidate sets. As there are 27 candidate sets $(2^5 - 4 - 1)$ from T_4, the replacement operation is performed 27 times. Finally, the error count Δ is updated to 2, which is the maximal frequency count in those 27 replaced entries. The baseline algorithm outputs every entry whose frequency count (i.e., $f + \Delta$) is greater than or equal to $\sigma N = 0.6 \times 4$. Thus, we obtain the two frequent itemsets $\{a\}$ and $\{b\}$.

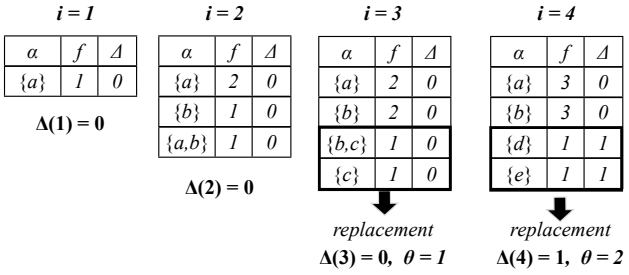

Figure 3: Updating the table from T_1 through T_4

From Example 2, it can be seen that the baseline algorithm processes any transaction using the fixed size of the entry table. Thus, it requires only $O(k)$ space for the table size k. In the following, we discuss about the validity in the baseline algorithm.

LEMMA 2. *Let $\Delta(t)$ be the value in Algorithm 1 at time t. $\Delta(t)$ is an upper bound of the supports of itemsets in the period $[1, t)$, each of which has been replaced before t.*

PROOF. We prove Lemma 2 by mathematical induction.
Step 1: When $t = 1$, $\Delta(1) = 0$ holds. Since there is no itemset that is replaced before time $t = 1$, Lemma 2 holds.
Step 2: Assume that when $t \leq i$, Lemma 2 also holds. When $t \leq i+1$, we divide every itemset β replaced in period $[1, t)$ into two cases:
Case (1) β is replaced when $t < i$. By the assumption in Step 2, the support of β in period $[1, t)$ is less than or equal to $\Delta(t)$.
Case (2) β is replaced when $t = i$. $f(\beta) + \Delta_\beta$ is an upper bound of the support of β in period $[1, i+1)$. From Line 21 of Algorithm 1, $\Delta(i+1)$ is updated to $f(\beta)+\Delta_\beta$ if $f(\beta)+\Delta_\beta \geq \Delta(i)$. Thus, $\Delta(i+1)$ is an upper bound of $sup(\beta)$. Therefore, when $t \leq i + 1$, Lemma 2 also holds. □

LEMMA 3. *Let $\mathcal{S}$ be a stream consisting of N transactions. For every itemset α, if $sup(\alpha) > \Delta(N + 1)$, then α remains in the table.*

PROOF. Suppose there is an itemset α such that $sup(\alpha) > \Delta(N + 1)$ and α does not remain in the table. Since α has been replaced in period $[1, N]$, the true support of α is less than or equal to $\Delta(N + 1)$ according to Lemma 2. However this contradicts the condition of α. □

Using Lemma 3, we have the following theorem that guarantees the validity (i.e., completeness and accuracy) of the outputs by Algorithm 1.

THEOREM 2. *Let $\mathcal{S}$ be a stream consisting of N transactions and σ be a minimal support. If $\Delta(N + 1) < \sigma N$, then every frequent itemset wrt $\mathcal{S}$ and σ is output by Algorithm 1. Besides, every output α satisfies $\sigma N - \Delta(N + 1) \leq sup(\alpha)$.*

PROOF. Since $\Delta(N + 1) < \sigma N$, the support of every frequent itemset wrt $\mathcal{S}$ and σ is greater than $\Delta(N + 1)$. By Lemma 3, every itemset whose true support is greater than $\Delta(N + 1)$ remains in the table. Thus, from Line 28 of Algorithm 1, every frequent itemset is output. Every output α satisfies $\sigma N \leq f(\alpha) + \Delta_\alpha$. Since $f(\alpha) \leq sup(\alpha)$ and $\Delta_\alpha \leq \Delta(N+1)$, it holds that $\sigma N \leq sup(\alpha)+\Delta(N+1)$. □

The error count $\Delta(t)$ changes according to the maximal table size k. In general, as k becomes smaller, the time series gain of $\Delta(t)$ increases. By Theorem 2, the LC-SS algorithm returns no false-negative and also ensures the accuracy of output supports provided that $\Delta(N + 1) < \sigma N$. Sometimes $\Delta(N+1)$ can exceed the threshold σN if k is too small. Even for such a case, the baseline algorithm can still guarantee the quality of outputs, as is described below.

THEOREM 3. *Let $\mathcal{S}$ be a stream consisting of N transactions and support σ_Δ be $\frac{\Delta(N+1)+1}{N}$. Every frequent itemset wrt $\mathcal{S}$ and σ_Δ can be output by Algorithm 1.*

PROOF. Every itemset α such that $sup(\alpha) \geq \Delta(N+1)+1$ remains in the table according to Lemma 3. Then, α can be output by Algorithm 1. In addition, $sup(\alpha) \geq \sigma_\Delta N$ holds. Hence, α is a frequent itemset wrt $\mathcal{S}$ and σ_Δ. □

According to Theorem 3, users can dynamically monitor a realistic minimal support (i.e., σ_Δ) associated with a fixed table size k. If the current σ_Δ is insufficient for users, they can manage it by dynamically reallocating memory resource.

5. SKIP LC-SS ALGORITHM

The baseline algorithm requires $2^{|T_i|} - 1$ updating or replacing operations for any transaction T_i. If a bursty transaction arrives, such as T_4 in Example 2, the access time to the entry table will increase exponentially. Thus, although the baseline algorithm can prevent memory overflow, it may cause a fatal delay in response time when bursty transactions arrive. Therefore, we propose another algorithm to address this problem. The key idea is to *skip* over the transactions by terminating replacement operations.

First, we recall Example 2. At time $i = 4$, the bursty transaction yields 27 candidate sets. Since the table is full, each candidate set is replaced by some registered entry with the minimal estimated support. Every candidate set is registered with the same support as $\Delta(4)+1$. Accordingly, once every minimal entry has been replaced, only *some candidate sets are replaced with the other candidate sets*. Note that the order by which each candidate set is registered is arbitrary. Thus, any candidate set can remain in the table at the end of the replacement. In Fig. 3, although the two candidate sets $\{d\}$ and $\{e\}$ remain in the table, we can freely select any two candidate sets for replacing minimal entries (For example, selecting $\{a, c\}$ and $\{b, e\}$ is possible). Thus, it is sufficient to replace only the two minimal entries (i.e., $\{b, c\}$ and $\{c\}$). Hence, we do not need to operate all 27 candidate sets in this example.

Definition 1. We call the entry for the empty set by the *empty entry*. In the following, we suppose that the initial entry table consists of k empty entries for the maximal table size k. Then, if the table contains empty entries, they are regarded as the minimal entries whose frequency count is zero. We denote by $cs(i)$, $me(i)$ and $c_{min}(i)$ the number of candidate sets, the number of minimal entries and the frequency count of minimal entries when T_i arrives, respectively.

Using Definition 1, the replacement operation of the LC-SS algorithm can be summarized into the following cases.

Case A: $me(i) > cs(i)$. After replacing $cs(i)$ minimal entries with the candidate sets, prior minimal entries remain in the table. Note that every newly registered entry has the same frequency count $\Delta(i) + 1$. Thus, we can describe $c_{min}(i+1)$ and $\Delta(i+1)$ at the next process $i+1$ as follows:

$$\begin{aligned} c_{min}(i+1) &= min(c_{min}(i), \Delta(i) + 1), \\ \Delta(i+1) &= c_{min}(i). \end{aligned}$$

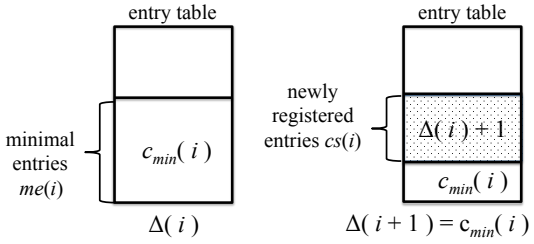

Figure 4: Case A: $me(i) > cs(i)$

Case B: $me(i) = cs(i)$. Consequently, all minimal entries are replaced with candidate sets whose frequency count is $\Delta(i)+1$. In addition, the next $c_{min}(i+1)$ is either $\Delta(i)+1$ or the second lowest frequency count (i.e., $c_{min}(i) + r$ for some r $(r \geq 1)$) at time i. Thus, we obtain the following:

$$\begin{aligned} c_{min}(i+1) &= min(\Delta(i) + 1, c_{min}(i) + r), \\ \Delta(i+1) &= c_{min}(i). \end{aligned}$$

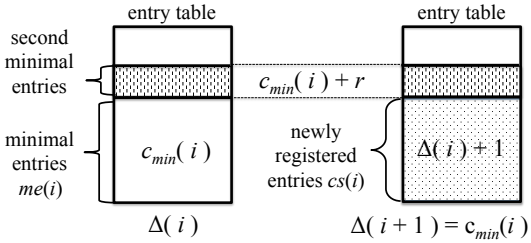

Figure 5: Case B: $me(i) = cs(i)$

Case C: $me(i) < cs(i)$. Initially, each minimal entry is replaced with some candidate set. Then, previously registered candidate sets are replaced with the other candidate sets, whereas their frequency count is given as $\Delta(i) + 1$. Note

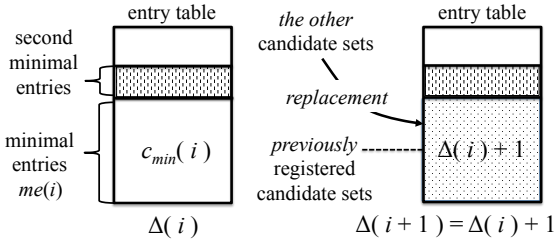

Figure 6: Case C: $me(i) < cs(i)$

that $\Delta(i+1)$ is updated to θ (See Line 21, Algorithm 1) that is the maximal frequency count in the replaced entries. When $me(i) < cs(i)$, θ must be $\Delta(i) + 1$. We thus obtain

$$\begin{aligned} c_{min}(i+1) &= min(c_{min}(i) + r, \Delta(i) + 1), \\ \Delta(i+1) &= \Delta(i) + 1. \end{aligned}$$

From the above cases, we obtain the following relationship between $\Delta(t)$ and $c_{min}(t)$. This proof is simply achieved by the mathematical induction for time t in each case.

THEOREM 4. $\Delta(t) \leq c_{min}(t) \leq \Delta(t) + 1$ *for each time* t.

By Theorem 4, we can obtain the following recurrence formula for $c_{min}(t)$ and $\Delta(t)$ (See Figures 4, 5, and 6):

$$\begin{cases} \text{Case A}: & me(i) > cs(i) & c_{min}(i+1) = c_{min}(i), \\ & & \Delta(i+1) = c_{min}(i). \\ \text{Case B}: & me(i) = cs(i) & c_{min}(i+1) = \Delta(i) + 1, \\ & & \Delta(i+1) = c_{min}(i). \\ \text{Case C}: & me(i) < cs(i) & c_{min}(i+1) = \Delta(i) + 1, \\ & & \Delta(i+1) = \Delta(i) + 1. \end{cases} \quad (1)$$

Equation (1) allows us to compute the next c_{min} and Δ without performing all replacement operations ($cs(i)$ total times). Especially for Case C, we do not need to register more candidate sets, once every minimal entry at time t has been replaced. In other words, we *skip the irredundant replacements with such residue candidate sets*. Algorithm 2 describes the proposal method, i.e., *Skip LC-SS algorithm*, which embeds the skip operation into the baseline algorithm.

Algorithm 2 Skip LC-SS based algorithm

Input: the maximal table size k, the minimal support σ, the data stream $\mathcal{S} = \langle T_1,\ T_2,\ \ldots,\ T_N \rangle$.
Output: an approximate set of frequent itemsets in $\mathcal{S}$ wrt σ at the current time N.

1: set i as 1 ($i := 1$) ▷ i is the current time
2: D consists of k empty sets, $\Delta(1) := 0$
3: $r(1) := 0$ ▷ $r(i)$: the num. of entries included in T_i
4: $c_{min}(1) := 0$, $mn(1) := k$, $cs(1) := 0$ ▷ 4 new variables
5: **while** $i \leq N$ **do**
6: read T_i
7: **for** each $\alpha \subseteq T_i$ s.t. α has been registered in D **do**
8: $f(\alpha)$++ and $r(i)$++
9: **end for**
10: update $c_{min}(i)$ and $me(i)$ by checking D
11: update $cs(i)$ by $r(i)$ (i.e., $cs(i) := 2^{|T_i|} - r(i) - 1$)
12: **if** $cs(i) < me(i)$ **then**
13: select all ($cs(i)$) candidate sets from T_i
14: **else**
15: randomly select $me(i)$ candidate sets from T_i
16: **end if**
17: replace minimal entries with the selected sets in D
18: whose frequency counts are set as $\Delta(i) + 1$
19: update $c_{min}(i+1)$ and $\Delta(i+1)$ by Equation (1)
20: i++ ▷ updating the time i
21: $r(i) := 0$ ▷ resetting $r(i)$
22: **end while**
23: **for** $\alpha \in D$ s.t. $f(\alpha) + \Delta_\alpha \geq \sigma N$ **do**
24: output α ▷ α can be a frequent itemset wrt σ and N
25: **end for**

In Algorithm 2, $r(i)$ denotes the number of entries of which itemsets are included in the transaction T_i. First, we compute $r(i)$ by way of Lines 7-8, which is the same process in the baseline algorithm. It holds that $cs(i) = 2^{|T_i|} - r(i) - 1$; thus, $cs(i)$ is obtained directly from $r(i)$. In Lines 12-16, we select candidate sets to be replaced with the minimal entries. Note that the number of replacements is now at most $me(i)$ rather than $cs(i)$. Finally, we update $c_{min}(i+1)$ and $\Delta(i+1)$ by Equation (1) according to the condition of $cs(i)$ and $me(i)$ for Cases A, B, or C.

Example 3. Recall Example 2. Let a stream S consisting of five transactions $\langle \{a\}, \{a, b\}, \{b, c\}, \{a, b, c, d, e\},$ *and* $\{a, c\} \rangle$, the minimal support σ be 0.6 and the maximal table size k be 4. Figure 7 illustrates how table S and the set var of 5 variables $r(t)$, $cs(t)$, $c_{min}(t)$, $me(t)$ and $\Delta(t)$ are updated in the period from $i = 1$ to 5 by Algorithm 2. In Fig. 7, each entry in the table S consists of an itemset α and its frequent count $c(\alpha)$ (i.e., $f(\alpha) + \Delta_\alpha$). Note also that each variable in Fig. 7 corresponds to the value at the end of the process at each time. At $i = 4$, every registered entry is a subset of T_4; thus we have $r(4) = 4$. In turn, $c_{min}(4)$ and $mn(4)$ are also updated to 2 in Line 10 of Algorithm 2. Then, the number of candidate sets at $i = 4$ is calculated in Line 11 (i.e., $c(4) = 2^5 - 1 - r(4) = 27$). Since $cs(4) > mn(4)$, we update $c_{min}(5)$ and $\Delta(5)$ as $c_{min}(5) = \Delta(4) + 1 = 2$ and $\Delta(5) = \Delta(4) + 1 = 2$ according to Equation (1).

At $i = 5$, only the entry for $\{a\}$ is a subset of T_5; thus $r(5) = 1$ holds. Next, we get $c(5) = 2$ (i.e., $2^2 - 1 - 1$) and $min(5) = mn(5) = 2$. Since $c(5) = mn(5)$, Equation (1) gives $min(6) = \Delta(5) + 1 = 3$ and $\Delta(6) = min(5) = 2$. The algorithm outputs all the entries because their frequency counts are greater than or equal to $\sigma N = 0.6 \times 5 = 3$, which contain the three frequent itemsets $\{a\}$, $\{b\}$ and $\{c\}$.

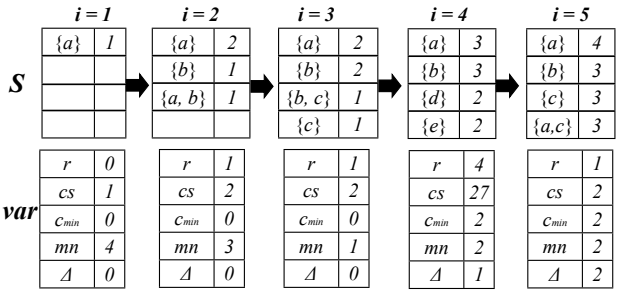

Figure 7: Updating the table from T_1 through T_5

The Skip LC-SS algorithm ensures the validity of the outputs, which are owned by the baseline algorithm.

THEOREM 5. *If $\Delta(N+1) < \sigma N$, then every frequent itemset wrt $\mathcal{S}$ and σ is outputted by Algorithm 2. Besides, every output α satisfies $\sigma N - \Delta(N+1) \leq sup(\alpha)$.*

The error counts in both Algorithms 1 and 2 have the same value by way of Line 19 in Algorithm 2 and Equation (1). Theorem 5 then holds using the same proof of Theorem 2.

In addition to validity, Algorithm 2 can refer to computational cost as follows:

THEOREM 6. *Let T be a transaction with length L and k be the table size. Algorithm 2 processes T in $O(kL)$ steps.*

PROOF. In Algorithm 2, there are two expensive processes: updating the table at Lines 7-9 and selecting at most $me(i)$ candidate sets from T_i at Lines 12-16. For the updating process, we require at most $k \times O(L)$ steps for a transaction with length L because this requires at most $O(L)$ steps to check if the itemset of each entry is included in T. For selecting the candidate sets, we require at most $2k \times O(L)$ steps. Note that every subset of T is either a candidate set or a registered one. The number of registered sets is k; thus, if we create at most $2k$ subsets of T, those $2k$ subsets contain at least k candidate sets. Since $me(i) \leq k$, k candidate sets are sufficient for completing the replacement operation. Thus, both updating and selecting processes can be performed in $O(kL)$ steps for T. □

Consequently, the Skip LC-SS algorithm can derive the outputs within $O(kLN)$ steps, whereas it ensures the same validness of the baseline algorithm by Theorems 5 and 6.

5.1 Performance of Skip LC-SS algorithm

Here, we investigate the performance of the Skip LC-SS algorithm using real transactional stream data. We use the online data for earthquake occurrences from 1981 to 2013 in Japan (Figure 2). Each earthquake data, which is provided from the Japan Meteorological Agency, is identified with one item in accordance with its focus location and magnitude and so on. Each transaction consists of such earthquake items that occur in every half a day. Consequently, the stream consists of 16769 transactions with 1229 items.

Figure 8 shows the logarithmic number of replacement operations in the baseline and Skip LC-SS algorithms, for each transaction ID. Note that the maximal table size k is

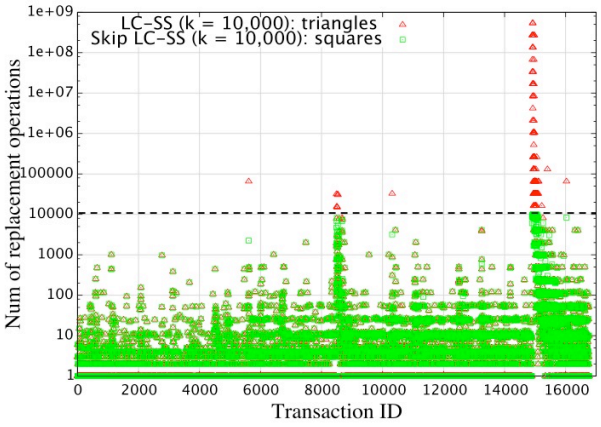

Figure 8: Number of replacement operations

set to 10000. The baseline algorithm, an integration of LC and SS algorithms, must register all subsets in each transaction at least once. Then, it requires an intractable number of replacement operations when bursty transactions arrive (see △ plots). In contrast, it is sufficient for Skip LC-SS to replace at most k entries in any transaction (see □ plots).

Next, we confirm the performance of the Skip LC-SS algorithm for a table size k. Note that we have implemented the algorithms using C and ran it on our machine (Mac Pro, Mac OS 10.6, 3.33GHz, 16GB). All experiments were performed using the same environment.

k	Δ	Ave. of me	Num. of outputs	Recall (%)	Prec. (%)	Time (sec.)
10K	168	4824	10000	100	N/A	1.3
30K	125	17300	95	100	77	3.4
50K	105	31322	84	100	88	4.7
70K	99	50518	84	100	88	6.7
90K	92	61363	80	100	92	8.5
100K	91	62814	80	100	92	9.5

Table 1: Performance of Skip LC-SS

For each k (from 10K to 100K), Table 1 shows the error count Δ, the average number of minimal entries me, the number of outputs, the recall and precision of outputs with respect to the minimal support $\sigma = 0.01$ and CPU time.

Note that there is a trade-off between k and Δ. As the maximal table size k decreases, more the maximal error count Δ also increases. This can also be interpreted with the number of minimal entries me: increasing me can suppress the gain of Δ. According to Equation (1), if me is less than the number of candidate sets, the error count must increment (i.e., Case C). For every k, recall is maintained at 100%. This is consistent with Theorem 5 because $\Delta(N+1) < \epsilon N$ holds (i.e., $\epsilon N = 167.7$). Conversely, the outputs can contain non-frequent itemsets because their frequency counts contain some error counts. The precision value shows the ratio of the frequent itemsets in the outputs. For $k = 10000$, we notice that the precision value is almost zero (N/A means the value is less than 1%). Indeed, most outputs (9926 itemsets) are not frequent, except for 74 itemsets. That is because Δ is almost same as σN (i.e., $\Delta = 168$ and $\sigma N = 167.7$). According to Theorem 5, the Skip LC-SS algorithm ensures that $sup(\alpha) \geq \sigma N - \Delta(N+1)$ for every output α. However, in case that $\Delta(N+1)$ is similar to σN, the Skip LC-SS algorithm cannot ensure a sufficient accuracy to the output. In fact, the number of outputs decreases only to 95 outputs (Prec.: 77%) for the case $k = 20000$ ($\Delta = 125$). In terms of CPU time, it increases linearly for the size k, which is consistent with Theorem 6.

Figures 9 and 10 show the time series change of me and Δ, respectively. When the number of minimal entries me corresponds to Case A in Equation (1), me monotonically decreases in Figure 9. In turn, Case B and C cause the rapid increase and decrease of me. Indeed, we may notice the chattering phenomenon around the transaction IDs 8000 and 15000. This indicates that some bursty transactions arrive around them. At those two points, the error count rapidly increases too in Figure 10. Note that we put the line indicating the minimal support σt for each time t. If the error count exceeds this minimal support (see about the transaction ID 15000) at some time, the entry table can temporary contain false-negatives according to Theorem 5. However, as the system processes transactions, the minimal support gradually increases and then the accuracy (recall and precision) of outputs can also recover. This feature shows robustness of the Skip LC-SS algorithm. Note also that we have empirically evaluated how many entries are registered in this dataset using the Stream Mining [8], which is one of the state-of-art parameter-oriented mining algorithms. Our experimental result[1] shows that the table size rapidly increases around the transaction ID 15000, and then at least one million entries have been temporally registered. Whereas the

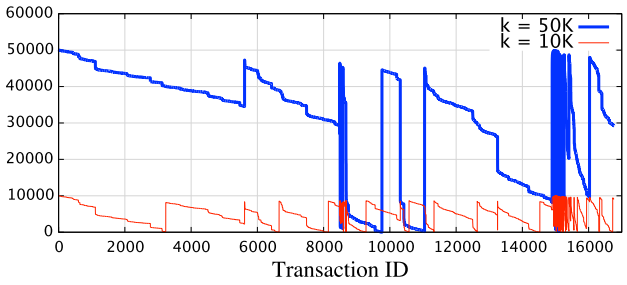

Figure 9: Time series minimal entries num. $me(t)$

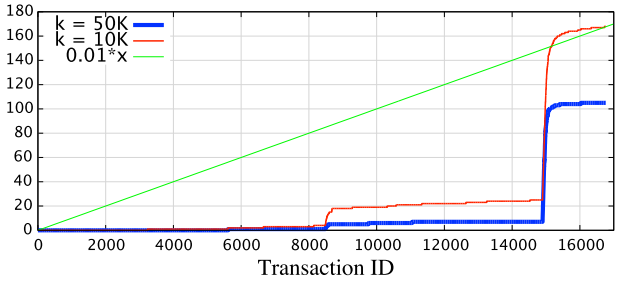

Figure 10: Time series error count $\Delta(t)$

[1]We implemented the skeleton of the algorithm in [8] and ran it by setting the two parameters as $\sigma = 0.01$ and $\epsilon = 0.2 \times \sigma$ and also by assuming that the maximal length of frequent itemsets is less than or equal to five.

Stream Mining can face the intensive memory consumption and response delay, the Skip LC-SS algorithm derives all the frequent itemsets only using 10000 entries. This is achieved by skipping the serious process involved in bursty transactions. On the other hand, the skip operation must increase the error count Δ. Indeed, we can observe that Δ intensively increases when bursty transactions arrive around the transaction ID 15000 in Fig. 10.

5.2 Improvement of Skip LC-SS algorithm

Now, we consider two variations of the skip operation to improve the performance of the Skip LC-SS algorithm. There are two bottleneck operations in the algorithm: updating entries and replacing them. The update operation corresponds to Lines 7-9 in Algorithm 2, where each entry is checked if it is contained in the current transaction. Then, it requires $O(k)$ steps for the table size k. On the other hand, the replacement operation corresponds to Lines 17-18 in Algorithm 2, where at most $me(i)$ candidate sets are generated and then replaced with minimal entries. As shown in the proof of Theorem 6, it requires $O(2k)$ steps. Figure 11 plots the CPU times for update (x-axes) and replacement (y-axes) operations for each transaction in the earthquake stream. Fig. 11 shows that both executing times increase as

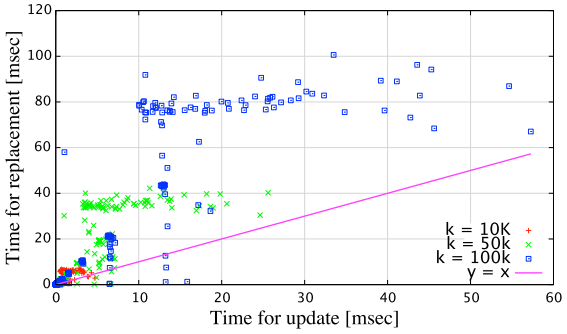

Figure 11: Replacing v.s. Updating times

the size k becomes large, whereas the replacement operation tends to be more expensive than the update one. Thus, we first consider improving the replacement operation.

In Case C of Equation (1), the error count $\Delta(i)$ must increment by one whether the replacement operations have been done or not. In other words, we can *skip all replacements in Case C*. This is easily realized by checking if $me(i) > cs(i)$ before Line 17 in Algorithm 2. We call by *r-skip* the above skip operation to distinguish it with the original one. When we apply the r-skip operation, the minimal entries, which must be originally replaced, remain in the table. Hence, the result can be different from the original one.

k		10K	30K	50K	70K	90K	100K
Δ	skip	168	125	105	99	92	91
	r-skip	168	125	105	99	92	91
Time	skip	1.3	3.4	4.7	6.7	8.5	9.5
(sec.)	r-skip	0.6	1.1	1.7	2.4	2.9	3.3

Table 2: skip v.s. r-skip

Table 2 shows the error count Δ and CPU time obtained by applying the Skip LC-SS algorithm with r-skip to the earthquake stream. In this experiment, r-skip could speed up the original process about 3 times, whereas Δ and the precision did not change.

Next, we extend r-skip based on the following lemma:

LEMMA 4. *Let T_i be a transaction at time i and k be the table size. If $2^{|T_i|-1} > k$, then $me(i) < cs(i)$ holds.*

PROOF. By the definition, we have $cs(i) = 2^{|T_i|} - r(i) - 1$. Since $k \geq r(i)$, $cs(i) \geq 2^{|T_i|} - k - 1$ holds. Now, we suppose $2^{|T_i|-1} > k$. Then, $2^{|T_i|} \geq 2k + 2$ holds. Accordingly, we have $cs(i) \geq k+1$. Since $k \geq me(i)$, $cs(i) > me(i)$ holds. □

If we meet such transaction T_i that satisfies the condition $2^{|T_i|-1} > k$, we can realize in advance that the error count $\Delta(i)$ must increment by one before updating the entries, because T_i causes the Case C by Lemma 4 and Equation (1). Based on this notion, we modify the original skip operation so as to *skip* processing each transaction if its length is more than $log(k) + 1$. In other words, we skip both the replacement and update operations for such transactions. We call the above skip operation by *t-skip*.

In t-skip, the frequency count of an entry can be lower than its true support because some update operations can be skipped in the previous time. In other words. t-skip can produce false-negatives even if $\Delta(N + 1) < \sigma N$. For this problem, we consider the following two approaches.

Approach 1: Relaxing the condition of outputs. We prepare a new counter X for registering the number of t-skip operations. Then, we check if $f(\alpha)+\Delta_\alpha+X > \sigma N$ for every entry α to be the output at the end of process. We call this approach by *t_1-skip*.

Approach 2: Incrementing the frequency count of each entry whenever t-skip is performed. Unlike t_1-skip, we do not change the condition of outputs. We call this approach by *t_2-skip*.

Each approach has its own drawback. In terms of the executing time, Approach 2 needs to scan all entries when t_2-skip is performed. On the other hand, Approach 1 tends to decrease the approximation accuracy and thus increase the number of non-frequent itemsets in the outputs. That is because Approach 1 must add error X to every entry α even if α should not be affected by previous t-skip operations.

k	Δ / num. of t-skips		Prec. (%)		Time (sec.)	
	t_1	t_2	t_1	t_2	t_1	t_2
10K	168/111	168/111	N/A	N/A	0.6	0.6
30K	126/88	126/88	N/A	40	1.4	1.6
50K	105/72	106/72	N/A	53	1.7	2.5
70K	99/64	99/64	8	60	2.4	3.7
90K	92/64	92/64	20	61	2.8	4.1
100K	91/64	92/64	22	61	3.0	4.5

Table 3: t_1-skip v.s. t_2-skip

Table 3 shows the error count Δ, the number of t-skip operations, the precision of outputs and the CPU times, obtained by applying the Skip LC-SS algorithm with t_1-skip and t_2-skip to data stream for earthquake occurrences. Note that the minimal support σ is set to 0.01 (same as the previous experiments). Note also that recall is maintained at 100%, since Δ is less than σN (167.7) in every

case. The right value of Δ corresponds to the number of the t-skip operations. This value may be regarded as the number of bursty transactions in the stream. Table 3 shows that whereas t_1-skip is slightly faster than t_2-skip, t_1-skip makes the precision of outputs much lower than t_2-skip, as mentioned in the above.

We can perform both t-skip and r-skip operations. For every transaction T_i satisfying $2^{|T_i|-1} > k$, we apply t-skip for skipping the update and replacement operations to T_i. Otherwise, we first update the support of each entry and then apply r-skip if $me(i) < cs(i)$ (i.e., Case C). We call by t_1&r-skip (*resp.* t_2&r-skip) the combination of r-skip and t_1-skip (*resp.* t_2-skip). Compared with Table 1 and Table 2,

k	Δ / num. of t-skips		Prec. (%)		Time (sec.)	
	t_1&r	t_2&r	t_1&r	t_2&r	t_1&r	t_2&r
10K	168/111	168/111	N/A	N/A	0.5	0.6
30K	126/88	126/88	N/A	40	0.8	1.2
50K	105/72	106/72	N/A	53	1.3	2.1
70K	99/64	99/64	8	60	1.8	3.0
90K	92/64	92/64	20	61	2.1	3.3
100K	91/64	92/64	22	61	2.3	3.7

Table 4: t_1&r-skip v.s. t_2&r-skip

both t_1&r-skip and t_2&r-skip cannot significantly improve the execution time of r-skip, whereas they must decrease their precision values (We recall that every precision value of r-skip is same as the one of its original skip operation). This fact may indicate that the operations scoped by r-skip and t-skip are similar with each other. Indeed, the speed-up gain by the combination of r-skip and t-skip does not reach the product of their synergetic gains. Table 5 summarizes the average performance of 6 variations of skip operation in the experiment, where r-skip gives the best performance.

variations	original	r	t_1	t_2	t_1&r	t_2&r
Recall (%)	100	**100**	100	100	100	100
Prec. (%)	77.0	**77.0**	6.2	48.2	6.2	48.2
Time (sec.)	5.4	**1.9**	1.9	2.8	1.4	2.3

Table 5: Performances of various skip operations

6. FURTHERMORE IMPROVEMENTS

Here, we furthermore improve the performance of Skip LC-SS algorithm using the *stream reduction* to dynamically repress each transaction. The length L of a transaction is a key factor for determining the performance of the Skip LC-SS algorithm, according to Theorem 6 and Lemma 4. The idea of stream reduction lies in the fact that *most items in bursty transactions are non-frequent*. By the principle of non-monotonicity, every itemset with any non-frequent item is no longer frequent. Thus, we can eliminate such non-frequent items from each transaction.

6.1 Item mining from transactional streams

The concept of stream reduction has been already introduced in modern FIM-DS methods [8, 12]. In this paper, we consider the resource-oriented approximation approach for stream reduction. This is achieved with frequent *item mining from (bursty) transactional streams*. There are several approximation methods for this task. One simple way is to use our LC-SS algorithm by restricting the length of each candidate set as one. However, this simple variation cannot estimate the error count in advance. Indeed, the error count should change in accordance with the fixed size of the entry table k_{im}, whereas the error count can negatively affect the accuracy of the reduced itemset. Thus, we consider another type of approximation item mining methods which can guarantee the maximal error count in terms of k_{im}. In

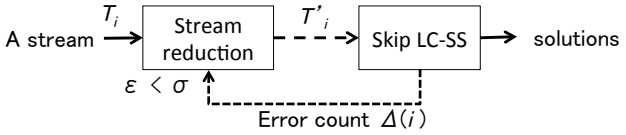

Figure 12: Embedding stream reduction

this paper, we use a recently proposed algorithm, called the *SS-ST* algorithm, which is obtained by simply extending the SS algorithm into item mining from a transactional stream [5]. Using the SS-ST algorithm, we compute the frequent count $c(e)$ for each item e in the transaction T_i at time i. Let $\Delta(i)$ be the error count at that time in the Skip LC-SS algorithm. Then, if $c(e) \leq \Delta(i)$ holds, e should be removed from T_i, because for every subset α_e of T_i that contains e, $c(\alpha_e) \leq \Delta(i)$ must hold. We should not store those subsets with lower frequency counts than $\Delta(i)$. Consequently, we perform the stream reduction as described in Algorithm 3.

Algorithm 3 Stream reduction with the SS-ST algorithm

Input: initial table size k_{im}, error parameter ϵ, data stream $\mathcal{S} = \langle T_1, \ldots, T_N \rangle$, error counts $\Delta(1), \ldots, \Delta(N)$.
Output: reduced transactions $T'_1, T'_2, \ldots, T'_N$.

```
1: i := 1, D := ∅, C := 0, L_ave := 0
2: while i ≤ N do
3:     read T_i and Δ(i)
4:     C := C + |T_i|
5:     L_ave := C/i        ▷ L_ave is the average length over T_i s
6:     if k_im < L_ave/ε then
7:         k_im := ⌈L_ave/ε⌉
8:     end if
9:     for each item e ∈ T_i do
10:        if ⟨e, c(e)⟩ ∈ D then
11:            c(e) + +
12:        else if |D| < k_im then
13:            insert the new entry ⟨e, 1⟩ in D
14:        else
15:            replace the minimal entry with ⟨e, 1 + c(m)⟩
16:        end if
17:    end for
18:    remove from T_i every item e such that c(e) ≤ Δ(i)
19:    output the reduced transaction T'_i
20:    i + +
21: end while
```

The completeness of the SS-ST algorithm has been shown in [5], which is briefly described as follows:

THEOREM 7. *[5] If $k_{im} \geq \frac{L_{ave}}{\epsilon}$, every frequent item e such that $sup(e) \geq \epsilon N$ is stored in the table D.*

Theorem 7 was proved in the similar way as Theorem 1 in the literature [14]. Let the minimal support be σ. Then,

it is sufficient for stream reduction to extract every item e such that $sup(e) \geq \sigma N$ at the end. Algorithm 3 ensures it by monitoring if $k_{im} \geq \frac{L_{ave}}{\epsilon}$ in Lines 6-8. Hence, it is sufficient to give an error parameter ϵ such that $\epsilon < \sigma$. In the following, the initial table size k_{im} is set to the same value as the entry table size of the Skip LC-SS algorithm.

6.2 Experimental results

We first show the result obtained using the earthquake data. By the stream reduction, the maximal (*resp.* average) length decreases to 57 (*resp.* 2.17) from 74 (*resp.* 2.48). In

k	Δ/num. of t-skips		Prec. (%)		Time (sec.)	
	$t_2\&r$	+red	$t_2\&r$	+red	$t_2\&r$	+red
10K	168 (111)	88 (48)	N/A	75	0.6	2.4
30K	126 (88)	70 (45)	40	77	1.2	2.7
50K	106 (72)	64 (44)	53	78	2.1	2.8
70K	99 (64)	61 (41)	61	81	3.0	3.2

Table 6: Improvement in earthquake transactions

Table 6, two rows named "$t_2\&r$ and "+red" correspond to the performances of $t_2\&r$ skip only and it with the stream reduction, respectively. The CPU time contains the executing times of both the stream reduction and Skip LC-SS. In this experiment, the stream reduction succeeds in about 50% decreases of the error counter (See Figure 13).

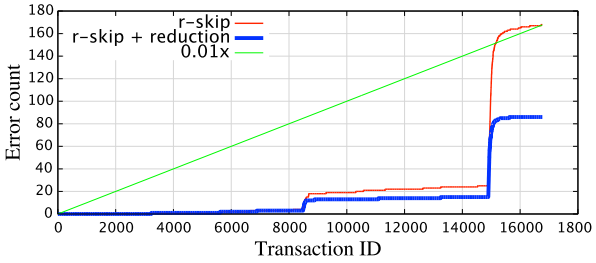

Figure 13: Comparison of $\Delta(t)$ in earthquake data

We have evaluated the performance by using two transactional streams: web-log data and retail data[2]. The web-log data stream consists of 19466 transactions with 9961 items. Using stream reduction, the maximal (*resp.* average) length decreases by 29 (*resp.* 9.81) from 106 (*resp.* 18.31).

Figures 14 shows the original (left) and reduced (right) lengths of transactions. Tables 7 shows improvements by embedding the stream reduction. We set the table size k from 500000 to 700000 and the minimal support σ to 0.05. In Table 8, we compare the stream reduction with the original Skip LC-SS. Note that the columns named "skip" correspond to the performance of the Skip LC-SS. This result shows that stream reduction succeeds in about 75% decreases of Δ (See Figure 15 for $k = 500000$), compared with the original one. It also succeeds in speeding up about 10 times.

Finally, we use the retail data consists of 88162 transactions with 16470 items. Using the stream reduction, the maximal (*resp.* average) length decreases by 58 (*resp.* 4.84) from 76 (*resp.* 10.31). Figures 16 shows the original (left) and reduced (right) lengths of transactions. In Table 9, we compare the performance (especially on Δ and CPU time)

[2]Available from http://fimi.ua.ac.be/

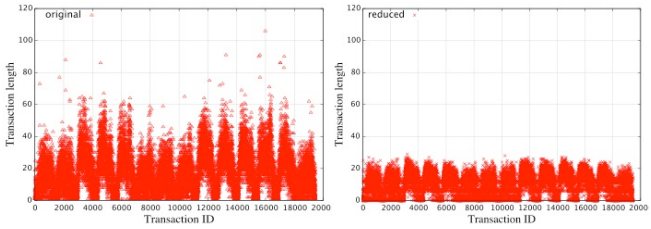

Figure 14: Reduction of the web-log transactions

k	Δ/num. of t-skips		Prec. (%)		Time (sec.)	
	$t_2\&r$	+red	$t_2\&r$	+red	$t_2\&r$	+red
500K	9003 /8343	2351 /1391	6	100	600	501
600K	8602 /7828	2284 /1025	6	100	891	762
700K	8526 /7828	2237 /1136	6	100	1017	856

Table 7: Improvement in web-log data (1)

by the reduction stream with the original Skip LC-SS for each table size k. This result shows that the stream reduction succeeds in about 90% decreases of error counter Δ (see the time series Δ for $k = 500000$ in Figure 17), and it also succeeds in speeding up about 30 times.

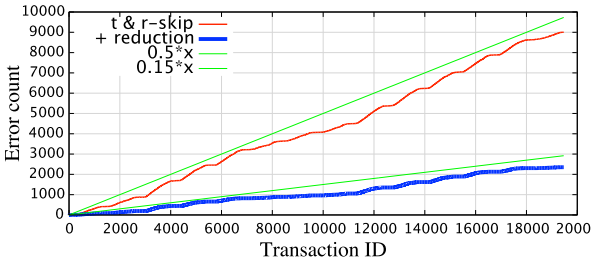

Figure 15: Comparison of $\Delta(t)$ in web-log data

7. CONCLUSION AND FUTURE WORK

This paper has proposed novel FIM-DS methods based on resource-oriented approximation. The difficulty of FIM-DS lies in how to handle the combinatorial explosion of candidate sets to be stored. Although there are several well-known methods based on parameter-oriented approximation, every method has its own drawback that causes the memory overflow especially in real data streams with bursty events. In this paper, we first show an exponential lower bound of space requirement of prior parameter-oriented methods, and then propose a resource-oriented algorithm in $O(k)$ space, which is achieved by integrating the Lossy Counting and Space-Saving algorithms in the context of itemset mining. We next improve the proposed method LC-SS algorithm so as to compute the solutions in $O(kLN)$ time where L and N are the maximal length and number of transactions. We call this algorithm by the Skip LC-SS algorithm. We emphasize that the Skip LC-SS algorithm can derive the outputs without memory overflow or fatal delay of response time provided that it ensures the same

Parameters		k = 500K	k = 600K	700K
Δ	skip	9000	8598	8523
	t_2&r skip	9003	8602	8526
	+ reduction	**2351**	**2284**	**2237**
Time (sec)	skip	5850	6912	8029
	t_2&r skip	600	891	1017
	+ reduction	**501**	**762**	**856**

Table 8: Improvement in web-log data (2)

Parameters		k = 500K	k = 600K	700K
Δ	skip	12699	11888	11571
	t_2&r skip	12699	11889	11571
	+ reduction	**960**	**960**	**960**
Time (sec)	skip	7853	9294	10041
	t_2&r skip	1400	2195	2408
	+ reduction	**249**	**353**	**393**

Table 9: Improvement in retail data

validity as the baseline algorithm. We also propose several variations of the skip operation and the novel technique dynamically reducing the transaction in a resource-oriented manner. Consequently, we have shown that the Skip LC-SS algorithm can partially break the space limitation of previously proposed FIM-DS methods through several real data streams with bursty transactions.

It is an important future work to introduce efficient data structures for the Skip LC-SS algorithm. Especially, the selection and replacement of candidate sets still require expensive computational costs. We also intend to investigate the adaptive approach using the Skip LC-SS algorithm that can fit the relevant resource in the context of FIM-DS.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number 25730133 and 25330256, and also supported by The Telecommunications Advancement Foundation. The authors are grateful to Hidetomo Nabeshima and Shuji Ito for their inspiration and fruitful comments on this work. We would also like to thank the anonymous reviewers for giving us useful and constructive comments.

8. REFERENCES

[1] P. Bose, E. Kranakis, P. Morin and Y. Tang: Bounds for frequency estimation of packet streams, *Proc. of the 10th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 33-42 (2003).

[2] J. Cheng, Y. Ke and W. Ng: A survey on algorithms for mining frequent itemsets over data streams, *J. of Knowledge and Information Systems*, 16(1), 1–27 (2008).

[3] K.-T. Chuang, J.-L. Huang and M.-S. Chen: Mining top-k frequent patterns in the presence of the memory constraint, *Int. J. on Very Large Data Bases*, 17(5), 1321–1344 (2008).

[4] G. Cormode and M. Hadjieleftheriou: Methods for finding frequent items in data streams, *Int. J. on Very Large Data Bases*, 19(1), 3–20 (2010).

[5] S. Fukuda, K. Iwanuma and Y. Yamamoto: Online frequent item mining on a stream consisting of variable-length transactions, *SIG-FPAI-B303-01* (in Japanese), 1–6 (2014).

[6] M. M. Gaber, A. Zaslavsky and S. Krishnaswamy: Mining data streams, *SIGMOD Record*, 34(2), 18–26 (2005).

[7] J. Han, H. Cheng, D. Xin and X. Yan: Frequent pattern mining: current status and future directions, *J. of Data Mining and Knowledge Discovery*, 15, 55–86 (2007).

[8] R. Jin and G. Agrawal: An algorithm for in-core frequent itemset mining on streaming data, *Proc. of Int. Conf. on Data Mining (ICDM)*, 210–217 (2005).

[9] R. M. Karp and S. Shenker: A simple algorithm for finding frequent elements in streams and bags, *ACM Trans. on Database Systems*, 28(1), 51–55 (2003).

[10] H.-F. Li, M.-K. Shan and S.-Y. Lee: DSM-FI: an efficient algorithm for mining frequent itemsets in data streams, *J. of Knowledge and Information Systems*, 17, 79–97 (2008).

[11] B. Lin, W.-S. Ho, B. Kao and C.-K. Chui: Adaptive frequency counting over bursty data streams, *Proc. of the 2007 IEEE Symp. on Comp. Intell. and Data Mining (CIDM)*, 516–523 (2007).

[12] G. S. Manku and R. Motwani: Approximate frequent counts over data streams, *Proc. of the 28th Int. Conf. on Very Large Data Bases (VLDB)*, 346–357 (2002).

[13] A. Mala and F. R. Dhanaseelan: Data stream mining algorithms: a review of issues and existing approaches, *Int. J. of Comp. and Eng.*, 3(7), 2726–2732 (2011).

[14] A. Metwally, D. Agrawal and A. E. Abbadi: Efficient computation of frequent and top-k elements in data streams, *Proc. of the 10th Int. Conf. on Database Theory (ICDT)*, 398–412 (2005).

[15] R. C. Wong and A. W. Fu: Mining top-k frequent itemsets from data streams, *J. of Data Mining and Knowledge Discovery*, 13, 192–217 (2006).

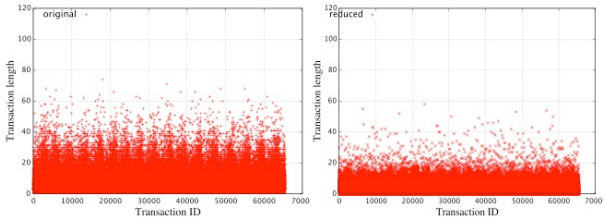

Figure 16: Reduction of the retail transactions

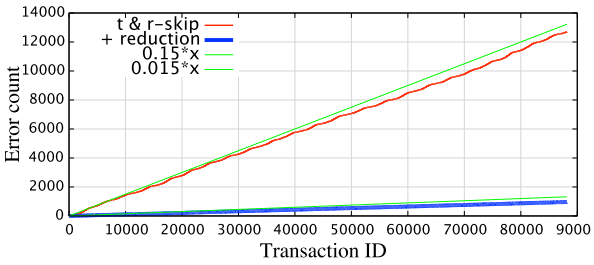

Figure 17: Comparison of $\Delta(t)$ in retail data