Journal of Universal Computer Science, vol. 11, no. 8 (2005), 1411-1425
submitted: 10/3/05, accepted: 5/5/05, appeared: 28/8/05 © *J.UCS*

Online Mining Changes of Items over Continuous Append-only and Dynamic Data Streams

Hua-Fu Li*
Department of Computer Science and Information Engineering
National Chiao-Tung University
1001, Ta Hsueh Road, Hsinchu 300, Taiwan
hfli@csie.nctu.edu.tw

Suh-Yin Lee
Department of Computer Science and Information Engineering
National Chiao-Tung University
1001, Ta Hsueh Road, Hsinchu 300, Taiwan
sylee@csie.nctu.edu.tw

Man-Kwan Shan
Department of Computer Science
National Chengchi University
64, Sec. 2, Zhi-nan Road, Wenshan, Taipei 116, Taiwan
mkshan@cs.nccu.edu.tw

Abstract: Online mining changes over data streams has been recognized to be an important task in data mining. Mining changes over data streams is both compelling and challenging. In this paper, we propose a new, single-pass algorithm, called *MFC-append* (Mining Frequency Changes of append-only data streams), for discovering the frequent frequency-changed items, vibrated frequency changed items, and stable frequency changed items over continuous append-only data streams. A new summary data structure, called *Change-Sketch*, is developed to compute the frequency changes between two continuous data streams as fast as possible. Moreover, a MFC-append-based algorithm, called *MFC-dynamic* (Mining Frequency Changes of dynamic data streams), is proposed to find the frequency changes over dynamic data streams. Theoretical analysis and experimental results show that our algorithms meet the major performance requirements, namely single-pass, bounded space requirement, and real-time computing, in mining data streams.

Keywords: Data streams, change mining, single-pass algorithm
Categories: H.2.8

1 Introduction

In recent years, database and knowledge discovery communities have focused on a new data model, where data arrives in the form of *continuous streams* [Babcock, 02] [Golab, 03]. It is often referred to as *data streams* or *streaming data*. A data stream is a massive, open-ended sequence of data elements continuously generated at a rapid

* Corresponding author.

rate. Many real-world applications generate large amount of data streams in real time, such as sensor data generated from sensor network, transaction flows in retail chains, Web record and click-streams in various Web applications, performance measurement in network monitoring and traffic management, call records in telecommunications, etc.

Online mining of frequent items (or item-sets) over data streams for knowledge discovery has become a novel and rapidly growing research direction [Charilar, 02] [Demaine, 02] [Ganti, 02] [Manku, 02] [Cormode, 03] [Giannella, 03] [Jin, 03] [Karp, 03] [Li, 04] [Li, 05], and has posed new challenges [Babcock, 02] [Golab, 03]: First, each data element over data streams should be examined at most once. Second, the memory usage in the process of mining data streams should be bounded even though new data elements are continuously generated from the stream. Third, each data element in the stream should be processed at fast as possible. Fourth, the analytical results generated by the online algorithms should be instantly available when user requested. Finally, the errors of results should be constricted as small as possible. Consequently, the continuous nature of data streams makes it essential to use the online algorithms which require only *single scan* over the stream for knowledge discovery. The unbounded nature makes it impossible to store all the data into main memory or even secondary storage. This motivates the design of the in-memory *summary* data structure with small footprints that can support both one-time and continuous queries. In other words, single-pass data stream mining algorithms have to sacrifice the correctness of its analytical results by allowing some counting errors. Hence, traditional *multiple-pass* data mining algorithms studied for static datasets cannot be easily used in such a streaming environment.

Change mining on static datasets has been studied in the last ten years [Dong, 99] [Ganti, 99] [Liu, 00] [Dong, 03]. It has become one of the core sub-areas of data mining. Ganti et al. [Ganti, 99] proposed a framework for quantifying the deviation of the induced models, such as two decision tree classifiers, clusters, and frequent itemsets, in the large datasets. The quantify measure is the amount of work required to transform one model into the other. Dong et al. [Dong, 99] proposed an algorithm to find the emerging patterns, and used these patterns to characterize the changes from one dataset to the other. Liu et al. [Liu, 00] proposed a method to discover the changes in the new data with respect to the old data, and the old decision tree models, and generate the exact changes that have occurred to the user. These studies are focused on the effects of data changes on data mining models and algorithms, whereas the paper is focus on the problem of measuring and understanding the changes of data directly rather than measuring the effects on data mining models. Furthermore, the proposed algorithms, *MFC-append* and *MFC-dynamic*, discover the changes of items over continuous data streams, not in static datasets.

The motivation of the problem of online mining changes of items between distributed data streams comes from the context of online transaction flows in large organizations. These companies generate the millions of records every day. For example, Google handles 70-110 millions searches, AT&T produces 250-300 million call records, and WallMart which is composed of thousands of stores, and records 20-40 million transactions in a single day. With the computation model of distributed data streams presented in Figure 1, a data stream processor and the in-memory summary data structure are two major components in the distributed data streaming

environment. The streams in questions are sequences of transaction data which is composed of the records in the form of <*StoreID, Timestamp, TransactionID, Items*>. In other words, a transaction record is a purchasing log generated by a customer in a specific time and store. These transaction flows sent to the server, and we are interested in finding the frequent frequency changes in items between pairs of data streams purchased by the most customers in some period of time. Note that the buffer mechanism can be optionally set for temporary storage of recent transactions from the transaction data streams.

In this paper, we study the problem of online mining frequent frequency changes of items between pairs of continuous, high-volume, open-ended data streams. The summary of our results are described as follows. Three types of frequency change are proposed: *frequent changed-item* (or **FCI** in short), *vibrated frequent changed-item* (or **VFCI** in short), and *stable frequent changed-item* (or **SFCI** in short). A new summary data structure, called *change-sketch*, is developed to store the essential information over the pairs of data streams. Two single-pass algorithms, *MFC-append* and *MFC-dynamic*, are proposed to find the patterns over data streams. The best space bound we achieve for this problem is $\Omega(m\log(n/m))$, where n is the union size of two data streams, and m is the size of the working bucket for frequent changed-items mining. Moreover, the proposed algorithms take $O(\log(n/m))$ time in worst case to process each new data element, but only $O(1)$ amortized time per data element.

The remainder of the paper is organized as follows. The data stream model and problem definition are discussed in Section 2. Algorithms MFC-append and MFC-dynamic are described in Section 3. Performance evaluation is presented in Section 4. Section 5 concludes the study.

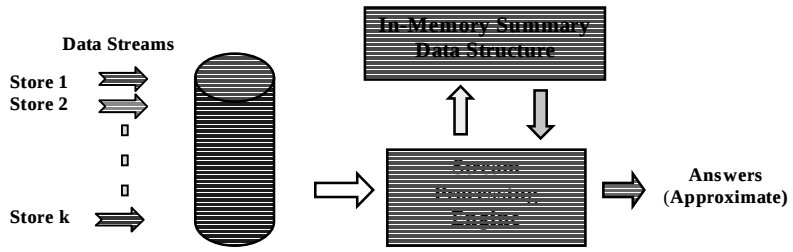

Figure 1: Processing model of distributed data streams

2 Preliminaries

In this section, we discuss the data stream model, the definition of online data mining for changes of items, and the goal of the proposed algorithms.

2.1 Data Stream Model

Let $\Psi = \{x_1, x_2, \ldots, x_m\}$ be a set of literals, called *data items* (or *items* in short). A *data stream* is an infinite sequence of data items, where the items arrive in some order, and may be seen only once. It is also referred to as *item-stream*. In the item-stream model, we focus on two performance issues: *workspace* required in main memory, which is measured as a function of the input union size *n* of two data streams, and the *time* to process an incoming data item over the streams.

In this paper, we assume that the data arrives in the *unordered*[1] form, and the same value can appear multiple times within the streaming data. This is termed the *unordered cash register*, *unordered aggregated model* [Babcock, 02] [Golab, 03].

Definition 1. A data stream is called an *append-only data stream* (or **ADS** in short) if it has no updates and deletions; a data stream is called a *dynamic data stream* (or **DDS** in short) if there are removal as well as addition of data items.

2.2 Problem Statement

Let $\Psi = \{x_1, x_2, \ldots, x_m\}$ be a set of data items. Two parallel item-streams are $P = <p_1, p_2, \ldots, p_i, \ldots>$, and $Q = <q_1, q_2, \ldots, q_j, \ldots>$ with time-varying data rates, where $p_i, q_j \in \Psi$.

Definition 2. The *frequency* of a data item x in an item-stream S over a time period T is the number of items in T in which x occurs, and is denoted as *frequency*(x, S, T). The size of T is n, the total number of data items so far in T.

Definition 3. The *changed support* of a data item x is the difference in frequency between two data streams P and Q divided by the total data items observed in T, and is denoted as *changeSup*(x, T).

Definition 4. The *changed rate* of a data item x is the number of *frequency vibration*[2] divided by the total time-points observed in T, and is denoted as *changeRate*(x, T), where the *time-point* is a basic unit of time over which the system collects data, e.g., second or minute.

Definition 5. A data item x is called a *frequent frequency changed item* (or **FFCI** in short) if *changeSup*(x, T) $\geq$ *mcs*, where *mcs* is a user-defined minimum changed support threshold in the range of [0, 1]; it is a *sub-frequent frequency changed item* (or **SFFCI** in short) if *ase* $\leq$ *changeSup*(x, T) $<$ *mcs*, where *ase* is a user-defined *a*pproximate *s*upport *e*rror threshold in the range of [0, *mcs*]; it is an *infrequent frequency changed item* (or **IFFCI** in short) if *changeSup*(x, T) $<$ *ase*.

---

[1] The streaming data items from various domains arrive in no particular order and without any preprocessing.

[2] Frequency vibration is the ratio of frequency change which exceeds a user-specified threshold, *vibrate rate*. In this paper, we assume that the rate is 100% for simplicity, i.e., frequency vibration is a frequency change from positive one to negative one, or vice versa.

Definition 6. A data item x over a time period T is called a *vibrated frequency changed item* (or **VFCI** in short) if its changed rate and changed support are greater than or equal to a user-defined *min*imum *c*hanged *r*ate (or *mincr* in shot) and *ase*, respectively; it is a stable frequency changed item (or **SFCI** in short) if its changed rate is less than a user-specified *max*imal *c*hanged *r*ate (or *maxcr* in short), and $changeSup(x, T) \geq mcs$, where *mincr* is a real number in the range of [0, 1] and $maxcr > mincr$.

For example, there are *ten* time-points ($T = [t_1: t_{10}]$, where t_1 is the starting time-point and t_{10} is the current time-point) in Figure 2, and we assume that $mincr = 0.1$, and $maxcr = 0.5$. In Figure 2, data item *a* and *b* are VFCIs, where $changeRate(a, T) = 9/10 = 0.9 > 0.5$, and $changeRate(b, T) = 6/10 = 0.6 > 0.5$, and items *c*, *d*, *e* are SFCIs, where $changeRate(c, T) = 0/10 = 0 \leq 0.1$, $changeRate(d, T) = 0/10 = 0 \leq 0.1$, and $changeRate(e, T) = 1/10 = 0.1 \leq 0.1$.

The goal of this paper is to find the changes of items (FFCIs, VFCIs, and SFCIs) over the pairs of data streams (either in ADS or DDS).

2.3 Performance Requirements in Mining Data Streams

The main design issues in mining data streams are:

(1) *Online, one-pass algorithm:* each transaction of a data stream is examined at most once.
(2) *Limited space requirement*: bounded memory requirement for storing crucial, compressed information in the summary data structure.
(3) *Real-time:* per data element must be processed as fast as possible.

Our algorithms have all of these characteristics described above, while none of previously published methods can claim the same [Dong, 99] [Ganti, 99] [Liu, 00] [Dong, 03].

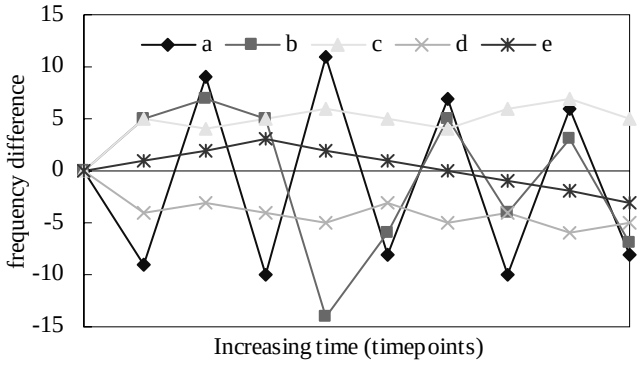

Figure 2: Examples of VFCIs and SFCIs

3 Online Mining Changes of Items over Distributed ADSs

In this section, a new summary data structure, called *Change-Sketch*, is developed to maintain the essential information about the set of all FFCIs, VFCIs, and SFCIs embedded in the data streams. A deterministic single-pass algorithm *MFC-append* (Mining Frequency Changes of append-only data streams) is proposed to find the changes of items over the pairs of data streams. The proposed algorithm uses at most $m\log(n/m)$ space, where n is the union size of the estimated data streams, and m is the size of working bucket.

3.1 A New Summary Data Structure *Change-Sketch*

The proposed in-memory summary data structure, called *Change-Sketch*, is a list of entries of the form (q, $q.count$, $q.w_{id}$, $q.rate$), where q is a data item in the streams, $q.count$ is an integer representing its estimated support, the value of $q.w_{id}$ assigned to a new entry q is the window identifier of current window, and $q.rate$ is the number of frequency vibration of item q. An item q is stored in the current Change-Sketch if $q.count \geq ase \cdot m \cdot (w_{current\text{-}id} - q.w_{id})$, where m is the window size and $m = \lceil 1/ase \rceil$. Note that the parameter *ase* is an acronym of the user-specified approximate support error threshold.

Two operations are used to maintain the structure Change-Sketch:

(1) **Update Change:** For each entry (q, $q.count$, $q.w_{id}$, $q.rate$) $\in$ Change-Sketch, MFC-append increases $q.count$ by computing the frequency changes of q in the current window. If the updated entry q take place a frequency vibration, its $q.rate$ is increased by one. If the changed support of updated entry q is less than the user-specified minimum changed support threshold *mcs*, the entry is deleted from the current Change-Sketch.

(2) **New Change:** If an item $p \notin$ Change-Sketch, and its changed support is larger than or equal to the threshold $ase \cdot m \cdot (w_{current\text{-}id} - p.w_{id})$, a new entry of the form (p, 1, $p.w_{current\text{-}id}$, 0) is created into the current Change-Sketch.

3.2 Algorithm *MFC-append*

Algorithm MFC-append uses the notations and conventions illustrated in Figure 3. In the framework of mining changes of items over data streams, the streaming data is broken into fixed sized buckets B_1, B_2, …, B_i, …, B_N, where B_N is the "*latest*" bucket with bucket identifier N, and B_1 is the "*oldest*" one. Note that each bucket contains k items. The *bucket length* from B_i to B_j is denoted as $B(i, j)$, where $i \geq j$. Let t_1, t_2, …, t_n be the *timepoints* (the smallest unit of time) which group the buckets so far in the streams, where t_n is the most recent timepoint, and t_1 is the oldest one. The form of bucket B_i is (*StreamID*, t_i, *items*), where t_i is the timepoint when the *items* appeared in the stream with identifier *StreamID*.

The *window-id* of t_i is denoted as w_i, and the number of buckets arrived from t_{i-1} to t_i is $|w_i|$, and the number of items (i.e., *size*) in w_i is denoted as $|w_i|$. The size of buckets arrived in T equals $|w_k| + |w_{k+1}| + \cdots + |w_n|$, $\forall k = 1, 2, \ldots, n$. As described above, the goal of this paper is to find the set of all FFCIs, VFCIs, and SFCIs in a

time period $T = t_k \cup t_{k+1} \cup \cdots \cup t_n$, $\forall k = 1, 2, \ldots, n$. Hence, the pair of input data streams P and Q are divided into two sequences of basic windows, i.e., $P = w_1[B_{P_1} + B_{P_2} + \cdots + B_{P_i}] + w_2[B_{P_{i+1}} + B_{P_{i+2}} + \cdots + B_{P_j}] + \cdots + w_m[B_{P_k} + B_{P_{k+1}} + \cdots + B_{P_{currentid-1}}]$, and $Q = w_1[B_{Q_1} + B_{Q_2} + \cdots + B_{Q_i}] + w_2[B_{Q_{i+1}} + B_{Q_{i+2}} + \cdots + B_{Q_j}] + \cdots + w_m[B_{Q_k} + B_{Q_{k+1}} + \cdots + B_{Q_{currentid-1}}]$. The notation $w_i[B_{StreamID_j} + B_{StreamID_{j+1}} + \cdots + B_{StreamID_k}]$ denotes that the buckets of data stream with id *StreamID* arrived at timepoint t_i, and the "latest" bucket id is denoted as $B_{StreamID_{current}}$, whose value is $\lceil n/m \rceil + 1$. For example, there are five buckets in the first window w_1 of Figure 1, in which two buckets ($B_{P_1}$and B_{P_2}) in stream P, and three buckets (B_{Q_1}, B_{Q_2}, and B_{Q_3}) in stream Q.

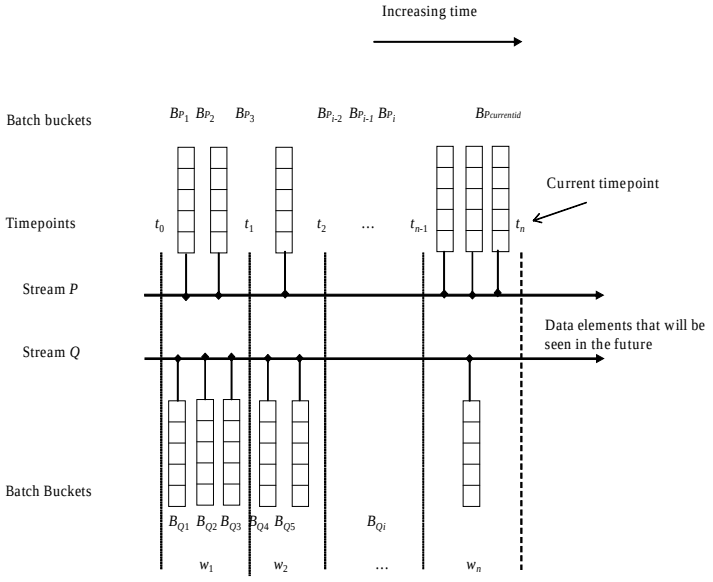

Figure 3: Notations and conventions used in the proposed algorithms

The algorithm description of MFC-append is shown in Figure 4. Four parameters are used in MFC-append algorithm: *mcs*, *ase*, *maxcr*, and *mincr*, where *mcs* is an acronym of the minimum changed support threshold, *ase* is an acronym of the approximate error support threshold, *maxcr* is an acronym of the maximum changed rate, and *mincr* is an acronym of the minimum changed rate. At any moment, a list of FFCIs with their estimated changed supports and changed rates is generated by the proposed algorithm. These approximate answers (i.e., a list of FFCIs) have the following guarantees. First, all items whose changed support exceed $mcs \cdot n$ are output, i.e., *no false negative*. Second, no items whose changed support is less than $(ase–mcs) \cdot n$ are output. Third, estimated changed supports are less the true changed supports by at most $ase \cdot n$. Finally, all items whose changed rate exceed $mcr \cdot n$ or less than $mcr \cdot n$ are output, respectively.

Algorithm *MFC-append*

Input: (1) Two continuous append-only data streams, $P = <p_1, p_2, \ldots, p_n, \ldots>$ and $Q = <q_1, q_2, \ldots, q_n, \ldots>$ with time-varying data rate, (2) A user-defined approximate support error threshold, *ase*, i.e., the window size m is $\lceil 1/ase \rceil$, (3) A user-defined minimum changed support threshold, *mcs*, (4) A user-specified maximum changed rate *maxcr*, (5) A user-specified minimum changed rate *minicr*.

Output: A list of FFCIs, VFCIs, and SFCIs.

Begin

Change-Sketch( )←{ };

Repeat:

for each bucket from the data streams (*P* and *Q*) **do**

for each item q in $w_i(C, B_i)$ **do** /* $i = 1, 2, \ldots, \lceil n/m \rceil + 1$ */

Change-Sketch(q, $q.count$++, $q.w_{id}$, $q.rate$);

for each item q in $w_i(D, B_i)$ **do**

Change-Sketch(q, $q.count$--, $q.w_{id}$, $q.rate$);

while *Change-Sketch*(q, $q.count$, $q.w_{id}$, $q.rate$) $\neq \varnothing$ **then**

if $|q.count| \geq mcs \cdot m \cdot (w_{current} - q.w_{id})$ **then**

item q is a frequent frequency change pattern in *Change-Sketch*;

else if $|q.q.count| \geq ase \cdot m \cdot (w_{current} - q.w_i)$ **then**

preserve q in *Change-Sketch*;

else remove q from *Change-Sketch*;

if $q.w_i$ change its symbol (either from positive frequency to negative one or from negative one to positive one)

then $q.rate$++;

End

Figure 4 : Algorithm description of MFC-append

The maintenance process of Change-Sketch is described as follows. Let the window identifier of current window be k. Initially, Change-Sketch is empty. For each item q in the current window of item-stream P, MFC-append first checks Change-Sketch to see whether an entry with id q already exists or not. If the entry exists in the current Change-Sketch, the frequency of q (i.e., $q.count$) is increased by one. Otherwise, a new entry of the form (q, 1, k, 0) is created in the current Change-Sketch. After processing all items in w_k of stream P, MFC-append computes all the items in w_k of another stream Q to maintain the changed information in Change-Sketch. The computation first checks Change-Sketch to see whether an entry q already exists or not in the Change-Sketch. If the search succeeds, the proposed algorithm updates the entry with id q by decreasing its frequency $q.count$ by one. Otherwise, a new entry of the form (q, -1, k, 0) is created in the current Change-Sketch. Now, if the updated entry q take place frequency vibration, $q.rate$ is increased by one, i.e., from zero to one.

In order to bound the memory usage in mining changes of items over data streams, a pruning mechanism of Change-Sketch is proposed. The technique deletes some entries of Change-Sketch before MFC-append computes the next working window with window-id k+1. It is a trade-off between the accuracy of the outputs and the memory requirement of Change-Sketch. The pruning is described as follows. An entry of the form (q, $q.count$, $q.w_i$, $q.rate$) is deleted, if $|q.count| < ase \cdot m \cdot (w_{current\text{-}id} - q.w_{id})$. After the pruning, MFC-append computes the next working windows with window-id w_{k+1} of data streams P and Q in the same way as described above.

When a user requests the results of the set of all FFCIs, VFCIs, and SFCIs embedded in the data streams, MFC-append algorithm outputs the entries whose $|q.count| \geq mcs \cdot m \cdot (w_{current\text{-}id} - q.w_{id})$, $|q.rate| \geq mincr \cdot m \cdot (w_{current\text{-}id} - q.w_{id})$, and $|q.rate| \geq maxcr \cdot m \cdot (w_{current\text{-}id} - q.w_{id})$, respectively, by one scan of the current Change-Sketch.

3.3 Space Analysis of Change-Sketch

In this section, we prove that MFC-append algorithm uses at most $O(m\log(n/m))$ space, where n denotes the current length of the estimated data streams, and $m = \lceil 1/ase \rceil$ is the size of working bucket.

Theorem 1: *The space requirement of MFC-append algorithm is* $O(m\log(n/m))$.

Proof: Let $w_{current\text{-}id}$ be the current window-id, i.e., $w_{current\text{-}id} = \lceil n/m \rceil$, where m is the size of working bucket. Let c_i denote the number of items in *Change-Sketch*, whose window id is $w_{current\text{-}id} - i + 1$. Since the size of each working bucket is m, we get the following constraints:

$$\sum_{i=1}^{k} ic_i \leq km \qquad \text{for } k = 1, 2, \ldots, w_{current\text{-}id}. \quad (1)$$

We claim that

$$\sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} \frac{m}{i} \qquad \text{for } k = 1, 2, \ldots, w_{current\text{-}id}. \quad (2)$$

We prove Inequality (2) by induction on k. If $k = 1$, then the claim is true because $c_1 \leq m$, i.e., we prove it from Inequality (1) directly. We now assume that Inequality (2) is true for $k = 1, 2, \ldots, j\text{-}1$, and prove that this assumption implies that it is true for $k = j$. We now add Inequality (1) for $k = j$ to j-1 instances of Inequality (2) and we have

$$\sum_{i=1}^{j} ic_i + \sum_{i=1}^{1} c_i + \sum_{i=1}^{2} c_i + \cdots + \sum_{i=1}^{j-1} c_i \leq jm + \sum_{i=1}^{1} \frac{m}{i} + \sum_{i=1}^{2} \frac{m}{i} + \cdots + \sum_{i=1}^{j-1} \frac{m}{i}.$$

$$\Rightarrow c_1 + 2c_2 + \cdots + (j\text{-}1)c_{j-1} + jc_j + [c_1 + (c_1 + c_2) + \cdots + (c_1 + c_2 + \cdots + c_{j-1})] \leq jm + [m + (m + m/2) + \cdots + (m + m/2 + \cdots + m/(j-1))].$$

$$\Rightarrow jc_1 + jc_2 + \cdots + jc_{j-1} + jc_j \leq jm + [(j-1)m + (j\text{-}2)m/2 + \cdots + m/(j-1)]$$

$$\Rightarrow j \sum_{i=1}^{j} c_i \leq jm + \sum_{i=1}^{j-1} \frac{(j-i)m}{i}.$$

Upon rearrangement, we get $j \sum_{i=1}^{j} c_i \leq jm + \sum_{i=1}^{j-1} \frac{(j-i)m}{i}$, which then easily simplifies to Inequality (2) for $k = j$, then we can complete the induction.

Since $|Change\text{-}Sketch| = \sum_{i=1}^{w_{current}} c_i$, from Inequality (2), we get $|Change\text{-}Sketch| \leq \sum_{i=1}^{w_{current}} \frac{m}{i} \leq m \log(w_{current\text{-}id}) = m \log(n/m)$.

□

Note that, if $ase \leq (1/m)$, the space is effectively $\Omega(m \log(n/m))$. If we set $ase = (d/m)$ for some small d, then it requires time at worst $O(m \log(n/m))$, but this occurs only every $1/m$ items, and so the total time is $O(n \log(n/m))$.

3.4 MFC-dynamic: Online Mining Changes of Items over Distributed DDSs

In this section, a MFC-append based-algorithm, called *MFC-dynamic* (Mining Frequency Changes of dynamic data streams), is proposed to mine the set of all FFCIs, VFCIs, and SFCIs over dynamic data streams. Note that a data stream is called a *dynamic data stream* (or **DDS** in short) if there are removal as well as addition of data items.

An effective encoding method is used in the proposed algorithm to distinguish the inserted items and deleted items over DDSs, and described as follows. If an item q is an inserted item, MFC-dynamic encodes it to be a "*positive*" item, and denotes it as $+q$. Otherwise, the proposed algorithm encodes it to be a "*negative*" item, and denotes it as $-q$. After processing the encoding, MFC-append algorithm is used to find the set of all FFCIs, VFCIs, and SFCIs over dynamic data streams. Figure 5 gives the description of MFC-dynamic algorithm. From the interpretation of MFC-dynamic, a space usage guarantee, which is similar to Theorem 1, is given as follows.

Claim 1. Whenever the deletions of item p occurs, $frequency(p)_{Deleted} \leq frequency(p)$, where $frequency(p)_{Deleted}$ is the number of item p needed to be drop.

Claim 2. If an item $q \notin$ Change-Sketch, if and only if $|q.count| < ase \cdot m \cdot (w_{current\text{-}id} - q.w_{id})$

Algorithm *MFC-dynamic*
Input: (1) Two dynamic data streams, $C=\{c_1, c_2, \ldots, c_n, \ldots\}$ and $D=\{d_1, d_2, \ldots, d_n, \ldots\}$ with time-varying data rate, (2) A minimum change support threshold, *mcs*, (3) An approximation support error threshold, *ase*, (4) A maximum change rate threshold, *maxcr*, (5) A minimum change rate threshold, *minicr*.
Output: A list of change patterns $\{ q_i, \ldots, q_j \}$ over dynamic data streams.
Begin
Dynamic_Encode_Streamming_Items(*C*, *D*);
MFC-append(*C*, *D*, *mcs*, *ase*, *maxcr*, *minicr*) ;
End
Procedure *Dynamic_Encode_Streamming_Items*(*C*, *D*);
Begin
for each bucket w_{C_i} of stream *C* and bucket w_{D_i} of stream *D*
if the item *q* is an inserted item **then**
Set it to be a positive (+*q*) item;
else
Set it to be a negative (-*q*) item;
end
endfor
End

Figure 5 : Algorithm MFC-dynamic

Theorem 2. *The space requirement of MFC-dynamic algorithm is O*($m\log(n/m)$).

Proof: According to the pruning rule, only items with frequency *f* or larger within the last updated *f* windows age are not pruned. Thus, at most *m*/*f* items could have been survived from that window which gives $m\sum_{i=1}^{n/m}\frac{1}{i}$ as the upper-bound on the number of items we are keeping track of. Now, using the well know inequality $\sum_{i=1}^{p}\frac{1}{i} \le \log(p)$, the result follows directly.

□

4 Performance Evaluation

4.1 Synthetic Data Generation

In the experiments of *MFC-append*, we generated three datasets |*D*| of 10,000, 100,000, and 1,000,000 transactions of single-item, and searched for frequent frequency changes while varying the Zipf parameter from 0 (uniform) to 3 (highly skewed), and the *ase* from 1% to 0.001%.

In order to evaluate algorithm *MFC-dynamic*, the generation approach of synthetic data was modified from [Cormode, 03]. The generated data consists of three parts: first, a sequence of insertions distributed uniformly over a small range; next, a sequence of inserts was drawn from a Zipf distribution with varying parameter (from 0 to 3); lastly, a sequence of deletes was distributed uniformly over the same range as

the starting sequence. We examine *MFC-dynamic* in the fourth dataset of 1,000,000 transactions of single-item, Zipf parameter from 0 to 3, and *ase* from 1% to 0.001%. Table 1 summarizes the meaning of various parameters used in our experiments.

$\|D\|$	Number of transactions of single item in data streams.
ase	Approximate error support.
mcs	Minimum changed support.
maxcr	Maximum change rate.
minicr	Minimum change rate.
Zipf	From 0 (uniform) to around 3 (highly skewed).

Table 1: Meanings of various parameters

4.2 Experimental Results

In this following experimental testing (results as Figure 6 and Figure 9), we use threshold *mcs* = 0.01, and *ase* = 0.1·*mcs*. First, we computed recall and precision for *MFC-append*, with the results shown in Figure 6. In this Figure, we can see that *MFC-append* algorithm has excellent precision (0.90-1) and recall (0.6-0.81) on the synthetic data $|D|$=10,000 transactions, and the recall decreases as the parameter *ase* increases, while the precision increases as the *ase* decreases. An important observation is that the Zipf parameter (from 0 to 3) does not affect the recall and precision of *MFC-append*.

In Figure 7, we can see that *MFC-append* has precision (0.93-1) and recall (0.57-0.76) on the synthetic data $|D|$=100,000 transactions. In Figure 8, we can see that *MFC-append* has precision (0.92-1) and recall (0.51-0.71) on the synthetic data $|D|$=1,000,000 transactions.

In Figure 9, we can see that the *MFC-dynamic* has the similar experimental results as algorithm *MFC-append*. The recall increases as the *ase* decreases while the precision decrease as the *ase* increases, and the various Zipf parameters do not influence the recall and precision of *MFC-dynamic*.

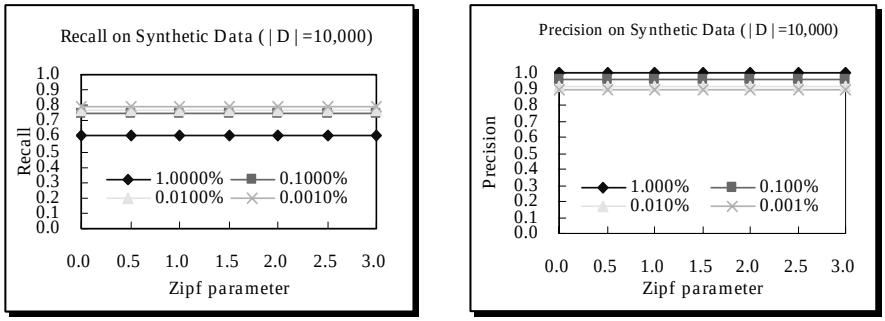

Figure 6: Experiments on synthetic data (10^4 transactions) for *MFC-append*. Left: testing recall (proportion of the frequent change patterns reported). Right: testing precision (proportion of the output frequency change patterns which are frequent)

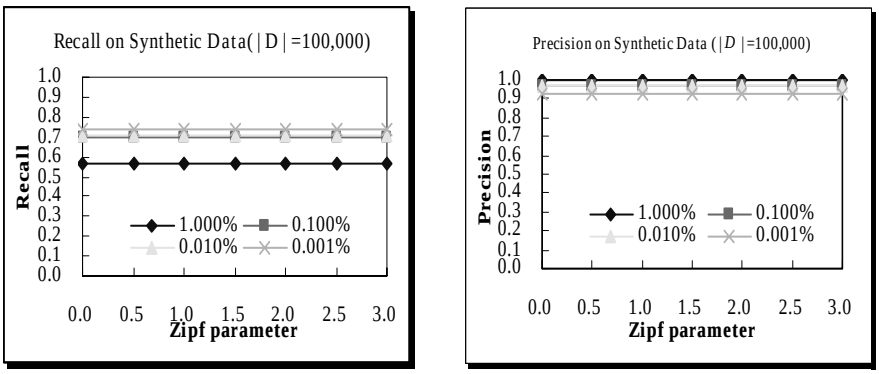

Figure 7: Experiments on synthetic data (10^5 transactions) for *MFC-append*. Left: testing recall. Right: testing precision

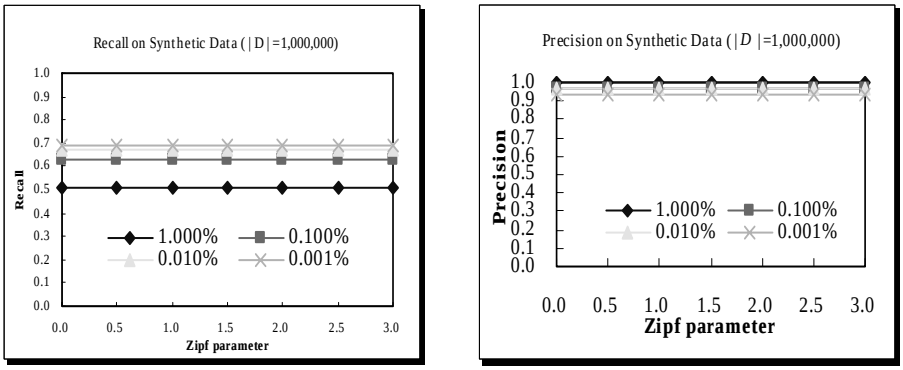

Figure 8: Experiments on synthetic data (10^6 transactions) for *MFC-append*. Left: testing recall. Right: testing precision

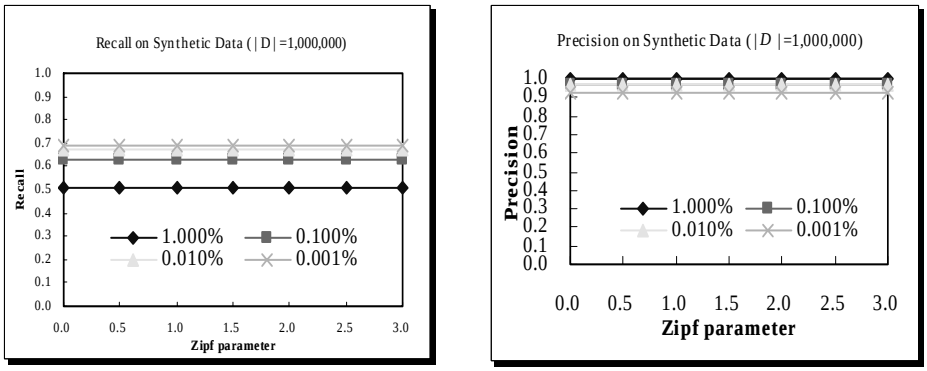

Figure 9: Experiments on synthetic data (10^6 transactions) for *MFC-dynamic*. Left: testing recall. Right: testing precision

5 Conclusions

In this paper, we propose two single-pass algorithms, called *MFC-append* and *MFC-dynamic*, for mining frequent frequency changed items, vibrated frequency changed items, and stable frequency changed items over continuous append-only and dynamic data streams, respectively. A new summary data structure, called *Change-Sketch*, is developed to store the essential changed patterns of data streams. The space complexity of Change-Sketch is $O(m\log(n/m))$, and the proposed algorithms take $O(\log(n/m))$ time in worst case to compute each new arrived item, but only $O(1)$ amortized time per item. The experimental results show that our algorithms have linear scalability and high accuracy in the analytical outputs.

Acknowledgements

The authors thank the reviewers' precious comments for improving the quality of the paper. The research is supported by National Science Council of R.O.C. under grant no. NSC93-2213-E-009-043.

References

[Babcock, 02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems", In *PODS'02*, Madison, WI, June 2002.

[Charilar, 02] M. Charilar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the International Colloquium on Automata, Languages, and Programming* (*ICALP*), 2002, pp. 693-703.

[Cormode, 03] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *PODS'03*, June 2003.

[Demaine, 02] E. Demaine, A. López-Ortiz, and J. I. Munro. Frequent estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, 2002, pp. 348-360.

[Dong, 03] G. Dong, J. Han, L.V.S. Lakshmanan, J. Pei, H. Wang and P.S. Yu. Online mining of changes from data streams: Research problems and preliminary results. In *Proceedings of the 2003 ACM SIGMOD Workshop on Management and Processing of Data Streams*. In cooperation with *the 2003 ACM-SIGMOD International Conference on Management of Data* (*SIGMOD*'03), San Diego, CA, June 8, 2003.

[Dong, 99] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining* (*KDD'99*), Aug. 1999, pp. 43-52.

[Ganti, 99] V. Ganti., J. Gehrke, and R. Ramakrishnan. A framework for measuring changes in data characteristics. In *PODS'99*, 1999, pp. 126-137.

[Ganti, 02] V. Ganti., J. Gehrke, and R. Ramakrishnan. Mining data streams under block evolution. *SIGKDD Explorations*, 3(2), 2002, pp. 1-10.

[Giannella, 03] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu, Mining frequent patterns in data streams at multiple time granularities, in H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining, AAAI/MIT, 2003.

[Golab, 03] L. Golab and M. Tamer Ozsu. Issues in data stream management. In *SIGMOD Record*, Volume 32, Number 2, June 2003, pp. 5--14.

[Jin, 03] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *Proceedings of the 12th ACM Conference on Information and Knowledge Management*.

[Karp, 03] R. Karp, C. Paradimitriou, and S. Shenker. A simple algorithm for finding elements in sets and bags. *ACM Transactions on Database Systems*, 2003.

[Li, 04] H.-F. Li, S.-Y. Lee, and M.-K. Shan. An efficient algorithm for mining frequent itemsets over the entire history of data streams. In *Proceedings of First International Workshop on Knowledge Discovery in Data Streams*, 2004.

[Li, 05] H.-F. Li, S.-Y. Lee, and M.-K. Shan. Mining (recently) maximal frequent itemsets over data streams. In *Proceedings of 15th International Workshop on Research Issues on Data Engineering: Stream Data Mining and Applications*, April 3-4, 2005.

[Liu, 00] B. Liu, W. Hsu, H.-S. Han, and Y. Xia. Mining changes for real-life applications. In *the 2nd International Conference on Data Warehousing and Knowledge Discovery*, Sept. 2000.

[Manku, 02] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.