A Simpler and More Efficient Deterministic Scheme for Finding Frequent Items over Sliding Windows

L.K. Lee
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
lklee@cs.hku.hk

H.F. Ting
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
hfting@cs.hku.hk

ABSTRACT

In this paper, we give a simple scheme for identifying ε-approximate frequent items over a sliding window of size n. Our scheme is deterministic and does not make any assumption on the distribution of the item frequencies. It supports $O(1/\varepsilon)$ update and query time, and uses $O(1/\varepsilon)$ space. It is very simple; its main data structures are just a few short queues whose entries store the position of some items in the sliding window. We also extend our scheme for variable-size window. This extended scheme uses $O(1/\varepsilon \log(\varepsilon n))$ space.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Theory

Keywords

Data Mining, Streaming Algorithms, Frequent Items, Network Monitoring

1. INTRODUCTION

A flow in a network can be modeled as a continuous stream of items such as source/destination addresses of the TCP/UDP packets. Identifying frequent items in such stream has found important applications in network monitoring and data mining. Since the items are time sensitive, we are only interested in identifying those frequent items in a sliding window covering the last n items seen so far from the stream. This motivates the following *ε-approximate frequent items problem*, which is formulated by Arasu and Manku [2]: Let θ and ε be user-specified threshold and relative error bound. We are asked to maintain some data structure that allows us to produce, at any time, a set B of items that satisfies the following properties:

- B only contains items that occur at least $(\theta-\varepsilon)n$ times in a sliding window that covers the most recently seen n items of the data stream and
- any item that occurs more than θn times in the window must be in B.

This problem is difficult because of the high-speed traffic of a network. We need to handle a huge volume of items from the stream, usually in the order of gigabytes a second, and as the stream passes we have only a few nanoseconds to react to each item. Thus, any feasible scheme for solving the problem must satisfy the following requirements:

Small memory: Note that there are gigabytes of data in the sliding window and we cannot store them in hard disk because of its long access time. Furthermore, the sensors used in monitoring the networks are relatively cheap and have small main memory. Thus, the memory used by the scheme must be much smaller than the size of the sliding window.

Extremely fast update and query time: Since we only have several nanoseconds to react to an item, we can afford to update a small number of counters or indexed variables.

There are many algorithms for identifying frequent items [3,4,7,8,14,15,17] and other statistics [1,9,12,13] in the entire data stream. Many of them use random sampling; they make assumptions on the distribution of the item frequencies and the quality of their results are only guaranteed probabilistically. Recently, Karp, Shenker and Papadimitriou [15], and independently, Demaine, López-Ortiz and Munro [6], rediscovered a deterministic algorithm of Misra and Gries [18] (the MG algorithm), which can easily be adapted to find ε-approximate frequent items in the entire data stream without making any assumption on the distribution of the item frequencies. The MG algorithm is simple and elegant; it needs $1/\varepsilon$ simple counters to count the items in the stream. The update operation involves only the increment (i.e., $+1$) or decrement (i.e., -1) of some of these counters.

For identifying frequent items over sliding window, Golab, DeHaan, Demaine, López-Ortiz and Munro [10] gave some heuristics for the problem and showed empirically that they worked well. Later, Golab, DeHaan, López-Ortiz and Demaine [11] gave an algorithm for the problem when the item frequencies are multinomially-distributed. Arasu and Manku [2] gave the first deterministic scheme for finding ε-approximate frequent items; it supports $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ query and update time and uses $O(\frac{1}{\varepsilon}\log^2(\frac{1}{\varepsilon}))$ space. Their scheme divides the sliding window into a collection of possibly overlapping sub-windows with different sizes. As the sliding win-

PODS'06, June 26–28, 2006, Chicago, Illinois, USA.

dow shifts, it applies the MG algorithm to each of these sub-windows to find the frequent items in these sub-windows. These sub-windows are organized cleverly into levels so that whenever there is a query on the frequent items, we can traverse these web of sub-windows efficiently to identify the items to be put in the solution.

In this paper, we improve Arasu and Manku's result. We give a deterministic scheme for the ε-approximate frequent items problem that supports $O(\frac{1}{\varepsilon})$ update and query time, and reduces the space requirement from $O(\frac{1}{\varepsilon}\log^2(\frac{1}{\varepsilon}))$ to $O(\frac{1}{\varepsilon})$. More importantly, our scheme is much simpler; it has about twenty lines of codes. It uses only $O(\frac{1}{\varepsilon})$ simple variables and $O(\frac{1}{\varepsilon})$ queues, and the total length of these queues is $O(\frac{1}{\varepsilon})$. To update these data structures when the window slides, we need only to increment/decrement some of the variables for most cases, and we seldom need to insert or delete entries in the queues.

Our scheme is also based on the MG algorithm. However, unlike Arasu and Manku's scheme, which uses the MG algorithm as a black box to build complicated structures for solving the problem, we adapt the MG algorithm directly to solve the ε-approximate frequent items problem. Roughly speaking, we replace the $\frac{1}{\varepsilon}$ simple counters used in the MG algorithm by some "window counters" that count the items over the sliding window. At first glance, our approach does not appear to be promising. ¿From a result of Datar, Gionis, Indyk and Motwani [5], we know that any deterministic or randomized algorithm for counting the number of any particular item over a sliding window needs at least $\Omega(\frac{1}{\varepsilon}\log n)$ space; this suggests that one window counter already needs $\Omega(\frac{1}{\varepsilon}\log n)$ space. Our approach needs $\frac{1}{\varepsilon}$ window counters and hence $\Omega((\frac{1}{\varepsilon})^2\log n)$ space, which is much larger than the $O(\frac{1}{\varepsilon}\log^2(\frac{1}{\varepsilon}))$ space used by Arasu and Manku's scheme. We observe that to solve the problem using our approach, we do not need the window counters to be accurate at all time; we only need them to give good estimates for items with high frequencies in the window. This relaxation of the requirement enables us to design a very simple data structure called λ-counters as the window counters. Because of their simplicity, we can decrement the value of λ-counters easily. (The decrement operation is important to the MG algorithm.) By replacing the simple counters by the λ-counters, and together some simple adaptation, we have a simple deterministic scheme for the ε-approximate frequent items problem.

We note that the λ-counter is just a simple variant of the window counter given in [16], which uses an optimal of $O(\frac{1}{\varepsilon}\log^2\frac{1}{\theta}+\log\varepsilon\theta n)$ bits to solve the following *significant one counting* problem: Given a stream of bits, maintain some data structure that allows us to make, at any time, an estimate $\hat{m}$ of the number m of 1-bits in the sliding window of size n such that if $m \geq \theta n$, we have $|\hat{m}-m| \leq \varepsilon m$. The λ-counter is less efficient in space; it may use $O(\frac{1}{\varepsilon}\log n)$ bits (or $O(\frac{1}{\varepsilon})$ words) in the worst case.

Although our scheme is simple, its correctness proof is not. The main difficulty comes from the following two differences between simple counters (which are used in the MG algorithm) and window counters (which are used by our scheme): (i) while simple counters count accurately, window counters can only give some estimates on the 1-bits in the sliding window, and (ii) while we have full control of the simple counters, we have little control of the window counters; their values change when the sliding window shifts. Hence, it is difficult to prove that our scheme still makes good estimates of the item frequencies under these uncertainties.

Our paper also handles the case when the sliding window can change its size. By adapting a technique given in [2], we show that our scheme for fixed window can be extended to identify frequent items in a sliding window whose size can be changed by the user. As pointed out in [2], fixed- and variable-size window capture the essential features of many common types of windows. In particular, given schemes for fixed- and variable-size window, we can extend them to handle time-based window.

The paper is organized as follows. We describe the MG algorithm and explain why it is correct in Section 2. In Section 3, we give an overview of our algorithm. In Sections 4 and 5, we give details on the λ-counters and proves some important properties about them. Section 6 describes our scheme for identifying ε-approximate frequent items and proves its correctness. Finally, in Section 7, we describe how to extend our scheme to solve the problem for sliding window with variable size.

2. THE MG ALGORITHM

Note that the original MG algorithm is only for the special case when the relative error bound ε is equal to the threshold θ. For general ε, we can modify the algorithm as follows:

> We keep a counter for each possible item and initialize them to 0. When a new item is read, we increment its counter, and if after the increment there become more than $\frac{1}{\varepsilon}$ counters with value greater than 0, each of these counters will be decrement once. When all the n items in the data stream is read, we return the set B of items whose counters have value at least $(\theta-\varepsilon)n$.

Note that at any time, there are only $O(\frac{1}{\varepsilon})$ counters with value greater than 0, and since we do not need to keep physically those counters with value 0, the algorithm uses $O(\frac{1}{\varepsilon})$ space. For proof of correctness, we note that for any item $x \in B$, x occurs at least $(\theta-\varepsilon)n$ times in the stream because (i) the counter of x has value at least $(\theta-\varepsilon)n$, and (ii) this counter is initially 0 and we will increment it only when a new x is read. To show that any item x that occurs more than θn times must be in B, note that whenever we decrement the counter of x, we will take away $\frac{1}{\varepsilon}+1$ units because we decrement $\frac{1}{\varepsilon}$ other counters as well. Since there are only n items, we have only n units to be taken away. It follows that we can decrement x's counter at most $n/(\frac{1}{\varepsilon}+1) < \varepsilon n$ times. On the other hand, we will increment the counter of x more than θn times because x occurs more than θn times. Therefore, when the algorithm terminates, the counter of x has value at least $(\theta-\varepsilon)n$ and by design, it is in B.

3. OVERVIEW OF OUR ALGORITHM

Roughly speaking, our algorithm is obtained by replacing all the counters in the algorithm described in Section 2 by some window counters. In other words, we associate each item with a window counter that counts the number of its occurrences in the sliding window. These window counters have an absolute error of $O(\varepsilon n)$ where n is the size of the window. Futhermore, for each item x, we guarantee that at any time, the window counter for x has size $O(\frac{n_x}{\varepsilon n})$ where

position	1	2	3	4	5	6	7	8	9	10	11	12
bit stream	1	0	1	1	1	1	1	1	1	1	0	0
sampled 1-bit				√			√			√		
λ-block	1			2			3			4		
									window W_{23}			

position	13	14	15	16	17	18	19	20	21	22	23
bit stream	0	1	0	0	1	0	1	1	1	1	1
sampled 1-bit							√			√	
λ-block	5			6			7			8	
	window W_{23}										

Figure 1: An example with $\lambda = 3$ and window size $n = 15$.

n_x is the number of occurences of x in the window at that time. Since the size of window is n, we have $\sum_x n_x = n$ and hence these window counters use totally $O(\sum_x \frac{n_x}{\varepsilon n}) = O(\frac{1}{\varepsilon})$ space.

We now give some rough and asymptotical arguments to explain intuitively why our algorithm is correct; we will give a formal and detailed proof in Section 6. Note that by design, the counter for any item $x \in B$ has value at least $(\theta - \varepsilon)n$, and since the counter has absolute error $O(\varepsilon n)$, we conclude that x occurs at least $\Omega((\theta - \varepsilon)n)$ times in the sliding window. To see that every item x that occurs more than $\Omega(\theta n)$ times must be in B, we argue, as in the correctness proof of the MG algorithm, that we will decrement x's counter $O(\varepsilon n)$ times. Since we still take away $\Omega(\frac{1}{\varepsilon})$ units for each decrement of the counter, the key of our argument is to prove that there are at most $O(n)$ units for the decrement operations to take away. This bound of $O(n)$ units is not surprising (though a rigorous proof is not trivial) because (i) we have executed n increment operations for the n items in the sliding windows, and (ii) there are $O(\frac{1}{\varepsilon})$ window counters greater than zero and the total absolute error they can make is $O(\frac{1}{\varepsilon}\varepsilon n) = O(n)$.

4. λ-SNAPSHOT

Let λ be any fixed positive integer that is smaller than the sliding window size n. In this section, we describe a simple sampling technique, the λ-snapshot, for estimating the number of 1-bits in a bit stream over some sliding window. Roughly speaking, λ-snapshot just samples every other λ 1-bits in the stream.

Consider any stream $f = b_1 b_2 b_3 \cdots$ of bits (i.e., $b_i \in \{0, 1\}$). We sample the 1-bits in f as follows: a 1-bit in f is a *sampled* 1-bit if it is the $(i\lambda)$th 1-bit in f for some $i \geq 1$. For example, the (λ)th 1-bit, the (2λ)th 1-bit and the (3λ)th 1-bit are all sampled 1-bits. Note that between any two consecutive sampled 1-bits b_i and b_j $(i < j)$, there are exactly $(\lambda - 1)$ 1-bits. We say that these $(\lambda - 1)$ 1-bits, together with the sampled 1-bit b_j, are *represented* by b_j, and b_j is the *representative* of these λ bits.

Each bit of f is associated with a *position*: the first bit b_1 is at position 1, b_2 position 2, and for any $i \geq 1$, bit b_i is at position i. Given any $1 \leq i \leq j$, we let $[i..j]$ denote the window of positions $i, i + 1, \ldots, j$. Recall that n is the size of the sliding window. For any p, we let W_p denote the window $[(p - n + 1)..p]$, the one that ends at position p. We let $W_p = [1..p]$ if $p \leq n$.

We divide the positions into blocks of λ positions called λ-blocks, and we index them sequentially: the first block $[1..\lambda]$ is a λ-*block* with index 1, the second block $[(\lambda + 1)..2\lambda]$ is a λ-block with index 2, and for any $i \geq 1$, $[((i - 1)\lambda + 1)..i\lambda]$ is a λ-block with index i. We say that a λ-block $B = [((i - 1)\lambda+1)..i\lambda]$ is *significant* for the window $W_p = [(p-n+1)..p]$ if

1. B falls in or overlaps with W_p, i.e., $[(i-1)\lambda+1)..i\lambda] \cap [(p - n + 1)..p] \neq \emptyset$, and
2. B covers one sampled 1-bit.

For example, in Figure 1, the λ-blocks 3, 4 7 and 8 are significant for window W_{23}.

Definition 1. *The λ-snapshot S of bit stream f over window W_p is the pair (Q, ℓ) where Q is the queue of significant λ-block for W_p, and ℓ is the number of 1-bits in W_p that have positions after the last sampled 1-bit in W_p. The value of the λ-snapshot S is defined to be $v(S) = \lambda|Q| + \ell$ where $|Q|$ is the size of Q.*

For example, in Figure 1, the λ-snapshot of the bit stream over window W_{23} is $(Q, \ell) = (\langle 3, 4, 7, 8\rangle, 1)$. The following lemma shows that λ-snapshot gives a good estimate on the number of 1-bits in a window.

Lemma 2. *Let S be the λ-snapshot of f over window W_p. Suppose that f has m 1-bits in W_p. Then, we have $m \leq v(S) \leq m + 2\lambda$.*

Proof. It is obvious that $m \leq v(S)$. To prove $v(S) \leq m + 2\lambda$, we consider two cases. If $|Q| \leq 1$, then $v(S) = \lambda|Q| + \ell < 2\lambda \leq m + 2\lambda$ (because $\ell < \lambda$).

Suppose that $|Q| > 1$. Let B_1 and B_2 be the first two blocks in Q (i.e., B_1 and B_2 have the smallest indices in Q). Let P be the set of blocks in Q that are not equal to B_1 or B_2. Note that for any block $B \in P$, the λ 1-bits represented by the sampled 1-bit in B are all in W_p because they are after the sampled 1-bit of B_2, which must be in W_p. It follows that the $(|Q| - 2)\lambda$ 1-bits represented by the sampled 1-bit in some $B \in P$ are all in W_p. Together with the ℓ 1-bits in W_p that are not represented by any sampled 1-bit in W_p, we conclude $(|Q|-2)\lambda+\ell \leq m$, or equivalently, $v(S) = |Q|\lambda + \ell \leq m + 2\lambda$. The lemma follows. □

5. THE λ-COUNTER

In this section, we describe a simple data structure, the λ-counter C, that estimates the number of 1-bits in the bit stream $f = b_1 b_2 \cdots$ over the sliding window by the value of the corresponding λ-snapshot. It supports the operation *shift*() that allows us to maintain the λ-snapshot of f over the sliding window; when a new bit b_p arrives, we execute $C.shift(b_p)$ to update the λ-snapshot of f over W_{p-1} to the one over W_p.

The main components of C is a queue Q and a variable ℓ for storing a λ-snapshot. To speed up the update time, Q is implemented as a deque, which allows us to remove an entry from both end of Q (using the operation *pop_head* and *pop_tail*), and to append an entry to the end (using the operation *push_tail*). To keep track of the λ-blocks, C also has two auxiliary variables *curblk* and *offset*, where *curblk* stores the index of the current block (i.e., the block having some positions in the sliding window, and some other after the sliding window) and *offset* stores the number of positions in the current block that are after the first position of the current block. We give below the implementation of *shift*(). Initially, the queue Q is empty, and the variables ℓ, *curblk* are equal to 0 while *offset* is equal to $\lambda - 1$.

C.shift(b) :

1: $offset \leftarrow (offset + 1) \mod \lambda$
2: **if** $offset = 0$ **then**
3: $\quad curblk \leftarrow curblk + 1$
4: **if** $head[Q] \times \lambda \leq ((curblk - 1) \times \lambda + offset + 1) - n$ **then** {i.e., the head of Q is expired}
5: $\quad pop\_head(Q)$
6: **if** $b = 1$
7: $\quad \ell \leftarrow (\ell + 1) \mod \lambda$
8: $\quad$ **if** $\ell = 0$ **then** {i.e., bit b is a sampled 1-bit}
9: $\quad\quad push\_tail(Q, curblk)$

(Note that a data stream is potentially infinite. To ensure that the index of a λ-block can be stored in the variable *curblk*, we may store the index as a modulo $2\lceil n/\lambda \rceil$ number without any ambiguity.)

FACT 3. *Suppose that C stores the λ-snapshot of f over window W_{p-1} and the bit of f at position p is b_p. Then, after executing C.shift(b_p), C stores the λ-snapshot of f over window W_p.*

Define the value $v(C)$ of C to be the value of the λ-snapshot that stores in C. The following lemma shows that C can count f over the sliding window with good accuracy.

LEMMA 4. *Let $f = b_1 b_2 b_3 \cdots$ be a bit stream and C be a λ-counter whose value is zero initially. Suppose that we count the bits of f by executing the operations C.shift(b_1), C.shift(b_2), $\cdots$ sequentially. Then, immediately after executing C.shift(b_p) for any $p \geq 1$, we have $m_p \leq v(C) \leq m_p + 2\lambda$ where m_p is the number of 1-bits of f in W_p.*

PROOF. It can be proved easily by mathematical induction using Lemma 2 and Fact 3. □

Besides using λ-counters to count the items in a data stream, our scheme for the ε-approximate frequent items problem also needs to decrement the value of some λ-counters whose values are greater than 0. In the rest of this section, we describe how to implement this operation and prove the invariant maintained by this operation. The idea for implementing the decrement operation is as follows. Suppose that the λ-counter C has executed *C.shift*(b_1), *C.shift*(b_2), $\ldots$, *C.shift*(b_p) to count the bit stream $f = b_1 b_2 \ldots b_p$. Furthermore, suppose that the value of C is not zero. To decrement C, we modify its content such that C becomes storing the λ-snapshot of the stream obtained from f by replacing the last 1-bit of f with a zero. We give below the details of the implementation.

C.decrement():

1: **if** $\ell > 0$ **then**
2: $\quad \ell \leftarrow \ell - 1$
3: **else** {i.e., $\ell = 0$}
4: $\quad pop\_tail(Q)$
5: $\quad \ell \leftarrow \lambda - 1$

FACT 5. *Suppose that the λ-counter C is storing the λ-snapshot of some bit stream f over window W_p and $v(C) > 0$. Then, after executing C.decrement(), we have (i) the value of C is decremented by exactly 1, and (ii) C stores the λ-snapshot of the bit stream f' over W_p where f' is obtained by replacing the last 1-bit of f at or before position p with the bit 0.*

For ease of future referencing, we call the position of this last 1-bit of f the *position of replacement* by *C.decrement*().

Consider any sequence σ of *C.decrement*() and *C.shift*() operations on λ-counter C. Let *C.shift*(b_1) *C.shift*(b_2) $\cdots$ *C.shift*(b_p) be the subsequence of shift operations in σ. We say that σ is associated with the bit stream $f = b_1 b_2 \ldots b_p$. The following lemma describes an invariant that is important to the correctness of our scheme.

LEMMA 6. *Let $\sigma = \sigma_1 \sigma_2 \cdots \sigma_k$ be a sequence of operations on an initially zero λ-counter C. Suppose that σ is associated with the bit stream $f = b_1 b_2 \cdots b_p$. Furthermore, suppose that there is a C.decrement() operation only when $v(C) > 0$. Then, after the last operation σ_k, C is storing the λ-snapshot of some stream $f' = b'_1 b'_2 \cdots b'_p$ over window W_p where $b'_i \leq b_i$ for all $1 \leq i \leq p$.*

PROOF. To prove the lemma by mathematical induction, we note that it is obviously true for $k = 0$. Suppose that it is true for any integer smaller than k and consider the sequence of operations $\sigma = \sigma_1 \sigma_2 \cdots \sigma_k$, which is associated with the bit stream $f = b_1 b_2 \cdots b_p$.

We consider two cases. Suppose that σ_k is a shift operation. Then it must be *C.shift*(b_p), and the sequence $\alpha = \sigma_1 \sigma_2 \cdots \sigma_{k-1}$ is a sequence associated with the stream $g = b_1 b_2 \cdots b_{p-1}$. By the induction hypothesis, C stores the λ-snapshot S of some stream $g' = b'_1 b'_2 \ldots b'_{p-1}$ over window W_{p-1} where $b'_i \leq b_i$ for all $1 \leq i < p - 1$. Then, by Fact 3, the execution of the remaining shift operation *C.shift*(b_p) will transform S to a λ-snapshot of the stream $b'_1 b'_2 \cdots b'_{p-1} b'_p = b'_1 b'_2 \cdots b'_{p-1} b_p$ over window W_p. Obviously $b'_i \leq b_i$ for all $1 \leq i \leq p$.

Suppose that σ_k is the operation *C.decrement*(). Then, the sequence $\alpha = \sigma_1 \sigma_2 \cdots \sigma_{k-1}$ is associated with the stream $f = b_1 b_2 \ldots b_p$, and by the induction hypothesis, after executing σ_{k-1}, C stores the λ-snapshot S of some stream $g = b'_1 b'_2 \cdots b'_p$ over window W_p where $b'_i \leq b_i$ for all $1 \leq i \leq p$. By Fact 5, the execution of the remaining operation σ_k = *C.decrement*() in σ will transform S to the λ-snapshot of stream g' over W_p where $g' = c_1 c_2 \cdots c_p$ is obtained from g by replacing the last 1-bit of g with 0. Obviously, $c_i \leq b'_i \leq b_i$ for all $1 \leq i \leq p$. The lemma follows. □

6. A SCHEME FOR THE ε-APPROXIMATE FREQUENT ITEMS PROBLEM

Recall that the problem ε-approximate frequent items problem asks for a set B of items in a data stream f over a sliding

window of size n such that (i) every item in B must occur at least $(\theta-\varepsilon)n$ times in the window, and (ii) any item that occurs more than θn times in the window must be in B. In this section, we describe a simple and efficient scheme for the problem. For ease of discussion, we first describe a scheme that uses $|\Pi|$ λ-counters where Π is the set of possible items. Then, we explain that most of these counters are unnecessary and our scheme needs at most $\frac{4}{\varepsilon}$ counters. Finally, we prove that our scheme supports $O(\frac{1}{\varepsilon})$ query and update time and uses $O(\frac{1}{\varepsilon})$ space.

In the rest of this section, we let $\lambda = \varepsilon n/8$.[1] Thus, the λ-counters maintain $(\varepsilon n/8)$-snapshot of some bit stream. Let $f = e_1e_2e_3\cdots$ be a stream of items in Π. For any $e \in \Pi$, define $f_e = b_1b_2b_3\cdots$ to be the bit stream where for any $i \geq 1$, $b_i = 1$ if $e_i = e$, and 0 otherwise. For any item $e \in \Pi$, let $n_e(W_p)$ be the number of 1-bit in f_e over W_p. Let $|W_p|$ be the total number of positions in W_p. The next fact follows directly from definitions.

FACT 7. *For any window W_p, $\sum_{e\in\Pi} n_e(W_p) = |W_p|$.*

In our scheme, we have, for every item $e \in \Pi$, a λ-counter C_e for counting the bit stream f_e, or equivalently, for counting the item e in the input stream f. Initially, all these counters have value zero. Below, we give the implementation of the update and query procedure for our scheme.

Update(e) :
1: C_e.*shift*(1)
2: **for** all $x \in \Pi - \{e\}$ **do** C_x.*shift*(0)
3: **if** there are more than $4/\varepsilon$ items x with $v(C_x) > 0$ **then**
4: **for** all $x \in \Pi$ with $v(C_x) > 0$ **do** C_x.*decrement*()

Query()
1: **for** all $e \in \Pi$ **do**
2: **if** $v(C_e) - 2\lambda \geq (\theta-\varepsilon)n$ **then**
3: output $(e, v(C_e) - 2\lambda)$

To count the items in $f = e_1e_2e_3\cdots$, we execute *Update*(e_1), *Update*(e_2), *Update*(e_3), $\cdots$. We execute *Query*() to return the set of items, together with our estimate on their frequencies, requested by the problem. Recall that all the λ-counters have value zero initially. Since (i) each *Update*() operation will increase the value of at most one counter by one and (ii) after each update, we will decrement the counters as soon as there are more than $\frac{4}{\varepsilon}$ counters greater than 0, we have the following fact.

FACT 8. *After any Update operation, there are at most $\frac{4}{\varepsilon}$ counters with value greater than 0.*

When Line 4 of *Update*() is executed, C_x.*decrement*() is executed for $\frac{4}{\varepsilon}+1$ items x. We call these $\frac{4}{\varepsilon}+1$ decrement operations the *batch of decrements* associated with this update operation. The following lemma shows that we cannot make too many batches of decrements.

LEMMA 9. *Consider any window $W_p = [p-n+1..p]$. Suppose that during the execution of sequence of update operations *Update*(e_{p-n+1}), *Update*(e_{p-n+2}), $\cdots$, *Update*(e_p), we have executed d batches of decrements. Then, we have $d < 3\varepsilon n/4$.*

PROOF. Note that after counting the first $p-n$ items in f, we have executed the operations *Update*(e_1), *Update*(e_2), $\cdots$, *Update*(e_{p-n}). Correspondingly, we have executed, for each item e, a sequence σ_e of *shift* and *decrement* operations on the counter C_e, and this sequence is associated with the bit stream $f_e = b_1b_2\cdots b_{p-n}$ where $b_i = 1$ if $e_i = e$, and 0 otherwise. By Lemma 6, C_e stores the λ-snapshot of the bit stream $f'_e = b'_1b'_2\cdots b'_{p-n}$ over W_{p-n}, where $b'_i \leq b_i$ for each $1 \leq i \leq p-n$. Let $n'_e(W_{p-n})$ be the number of 1-bits of f'_e over W_{p-n}. Together with Lemma 2, we conclude that after executing σ_e, we have $v(C_e) \leq n'_e(W_{p-n}) + 2\lambda \leq n_e(W_{p-n}) + 2\lambda$. Therefore,

$$\begin{aligned}\textstyle\sum_{e\in\Pi} v(C_e) &= \textstyle\sum_{e\in\Pi, v(C_e)>0} v(C_e)\\ &\leq \textstyle\sum_{e\in\Pi, v(C_e)>0}(n_e(W_{p-n}) + 2\lambda),\end{aligned}$$

and together with Facts 7 and 8, we conclude that just after executing *Update*(e_{p-n}), the total value of the counters is

$$\textstyle\sum_{e\in\Pi} v(C_e) \leq |W_{p-n}| + (\frac{4}{\varepsilon})2\lambda = |W_{p-n}| + (\frac{4}{\varepsilon})\frac{\varepsilon n}{4} \leq 2n. \quad (1)$$

Note that for each $p-n+1 \leq i \leq p$, *Update*(e_i) will execute C_e.*shift*(1) for exactly one item e, and C_e.*shift*(0) for the remaining items. Since C_e.*shift*(1) will increase the value of C_e by at most 1 (sometimes it may even decrease its value), and C_e.*shift*(0) will not increase its value, the n operations *Update*(e_{p-n+1}), *Update*(e_{p-n+2}), $\ldots$, *Update*(e_p) will increase the total value of the counters by at most n. Together with (1), and the fact that one batch of decrements will decrease $1+4/\varepsilon$ units from the counters and all counters are always non-negative, we conclude $d(4/\varepsilon+1) \leq 2n+n = 3n$ or equivalently, $d < 3\varepsilon n/4$. The lemma follows. □

We are now ready to show the value of C_e is a good estimate of the number of items e in the data stream f over the sliding window.

LEMMA 10. *Suppose that we have executed the sequence of operations $\sigma = \langle$*Update*(e_1), *Update*(e_2), $\ldots$, *Update*$(e_p)\rangle$. Then, for any item $e \in \Pi$, the number $n_e(W_p)$ of items e in the stream $f = e_1e_2\cdots e_p$ over W_p is related to the value of C_e as follows: $n_e(W_p) - \varepsilon n < v(C_e) - 2\lambda \leq n_e(W_p)$.*

PROOF. Let $f_e = b_1b_2\cdots b_p$. By Lemma 6, after executing σ, C_e is storing the λ-snapshot of some stream $f'_e = b'_1b'_2\cdots b'_p$, and by Lemma 2 we conclude

$$n'_e(W_p) \leq v(C_e) \leq n'_e(W_p) + 2\lambda,$$

where $n'_e(W_p)$ is the number of 1-bits of f'_e over the window W_p. Lemma 6 asserts that $b'_i \leq b_i$ and this implies that $n'_e(W_p) \leq n_e(W_p)$. Note that the C_e.*decrement* operations resulted from *Update*(e_1), $\cdots$, *Update*(e_{p-n}) have positions of replacements before position $p-n+1$, and by Lemma 9, there are $d < 3\varepsilon n/4$ operations C_e.*decrement*() resulted from *Update*(e_{p-n+1}), $\ldots$, *Update*(e_p). There are thus at most d positions in $[p-n+1..p]$ such that the corresponding 1-bit of f_e will be replaced by 0 by some C_e.*decrement*() operations. This follows that $n'_e(W_p) \geq n_e(W_p) - d$. Putting everything together, we have

$$n_e(W_p) - d \leq n'_e(W_p) \leq v(C_e) \leq n'_e(W_p) + 2\lambda \leq n_e(W_p) + 2\lambda,$$

and since $d + 2\lambda < 3\varepsilon n/4 + \varepsilon n/4 = \varepsilon n$, the lemma follows. □

[1] To simplify notation, we assume that $\varepsilon n/8$ and $4/\varepsilon$ are integers. In case they are not integers, our scheme can be easily changed to adapt the case. Also, we assume that $\varepsilon n \geq 8$; otherwise, we can do exact counting by storing all the n items in the sliding window and using only $n < 8/\varepsilon$ counters. This approach uses $O(1/\varepsilon)$ space.

The following lemma asserts that *Query*() will always return the correct set of items for the ε-approximate frequent items problem.

THEOREM 11. *Suppose that we have executed* $Update(e_1)$, $Update(e_2), \ldots, Update(e_p)$. *Then,* $Query()$ *will return a set* B *of items such that (i) every item in* B *must occur at least* $(\theta - \varepsilon)n$ *times in* W_p, *and (ii) any item that occurs more than* θn *times in* W_p *must be in* B.

PROOF. Note that an item e returned by $Query()$ must have $v(C_e) - 2\lambda \geq (\theta - \varepsilon)n$, and by Lemma 10, we have $v(C_e) - 2\lambda \leq n_e(W_p)$. Thus, $n_e(W_p) \geq (\theta - \varepsilon)n$, or equivalently, it occurs at least $(\theta - \varepsilon)n$ times in W_p. On the other hand, if an item e is not in B, then $v(C_e) - 2\lambda < (\theta - \varepsilon)n$, and by Lemma 10, we have $n_e(W_p) - \varepsilon n \leq v(C_e) - 2\lambda$. It follows that $n_e(W_p) < \theta n$. Therefore, any item that occurs more than θn times in W_p must be in B. □

In the rest of this section, we analyze the query and update time of our scheme, and the space used by our scheme. In our simple implementation, we maintain $|\Pi|$ λ-counters. Note that for any counter C with value 0, the queue $C.Q$ must be empty and $C.\ell = 0$. Furthermore, the variables $C.curblk$ and $C.offset$ are for maintaining the λ-blocks and can be shared by all counters. Also, we will never decrement a counter with value 0. We conclude that we do not need to store physically any counter with value 0. By Fact 8, there are at most $\frac{4}{\varepsilon}$ counters greater than 0 after any $Update$ operations. Thus, we only need to keep $\frac{4}{\varepsilon}$ λ-counters.[2]

THEOREM 12. *Our scheme needs* $O(\frac{1}{\varepsilon})$ *time for update and query, and uses* $O(\frac{1}{\varepsilon})$ *space.*

PROOF. For $Update()$, there are $\frac{4}{\varepsilon}$ counters to be updated, and updating each counter takes constant time, while for $Query()$, we need to compute the value of $\frac{4}{\varepsilon}$ counters, and each value can be computed at constant time. Thus, our scheme needs $O(\frac{1}{\varepsilon})$ time for update and query.

To estimate the space requirement, suppose that we have executed $Update(e_1)$, $Update(e_2), \ldots, Update(e_p)$, and the sliding window is currently at W_p. Suppose that there are now k counters $C_1, C_2, \ldots, C_k$ with value greater than zero. Note that $k \leq \frac{4}{\varepsilon}$ and each of these counters has only three variables. Hence, to prove the total space is $O(\frac{1}{\varepsilon})$, it suffices to show that total size of the queues in these k variables is $O(\frac{1}{\varepsilon})$.

For each $1 \leq i \leq k$, let $Q_i = C_i.Q$ be the queue maintained by the counter C_i, and let f_i be the bit stream associated with C_i. By Lemma 6, C_i is storing the λ-snapshot of some stream f_i' over W_p. ¿From the proof of Lemma 2, we know that at least $\lambda(|Q_i| - 2)$ bits of f_i' are in W_p. Hence,

$$\begin{aligned} \textstyle\sum_{1 \leq i \leq k} \lambda(|Q_i| - 2) &\leq \textstyle\sum_{1 \leq i \leq k} n_i'(W_p) \\ &\leq \textstyle\sum_{1 \leq i \leq k} n_i(W_p) \leq |W_p| = n, \end{aligned}$$

where $n_i(W_p)$ and $n_i'(W_p)$ are the number of bits of f_i and f_i' in W_p, respectively. Therefore, we have $\sum_{1 \leq i \leq k} |Q_i| = O(n/\lambda) = O(1/\varepsilon)$. □

[2]With this modification, the procedure $Update(e)$ should be changed as follows: just before line 1, if item e does not have a λ-counter in all the $\frac{4}{\varepsilon}$ λ-counters and there is some λ-counter with 0 value, assign the λ-counter with 0 value to item e. Also, lines 3 to 4 are changed to: If item e does not have a λ-counter in all the $\frac{4}{\varepsilon}$ λ-counters, for all item x that has a λ-counter, we execute $C_x.decrement()$.

7. EXTENDED SCHEME FOR VARIABLE-SIZE SLIDING WINDOW

In the model of variable-size sliding window, the window size n is a variable. n is increased by one when there is a new item inserted into the window, while n is decreased by one when the least recent item in the window is deleted from the window, i.e., the item is expired.

Let h be the positive integer such that $2^{h-1} < n \leq 2^h$. In our algorithm, we keep $h+1$ levels (level 0 to level h) of instances of our scheme for fixed-size sliding window, where level i $(0 \leq i \leq h-1)$ is of window size 2^i and the level h is of window size n. In the level i, if $\varepsilon 2^i < 16$, we do exact counting in that level, i.e., storing the whole sliding window of the stream and keeping exact counts for each item in the sliding window. Otherwise, we use the scheme for sliding window of fixed size 2^i, which has $\lambda = \varepsilon 2^i/16$ and performs the batch of decrements in $Update(e)$ if there are more than $16/\varepsilon$ λ-counters with positive values.[3]

Note that if $\varepsilon 2^h \geq 16$, the sliding window for level h is of size n but we use the scheme for sliding window of fixed size 2^h in level h. Therefore, for each λ-counter C in level h, when we remove expired item in the head of the deque, i.e., lines 4 and 5 in $C.shift(b)$, the window size used is n instead of 2^h. Also, when the window size n is decreased, we have to additionally check and remove expired items in the head of the deque in all λ-counters in the level h.

The following is our extended scheme. Initially, since the window size n is 0, there is not any level, i.e., $h = -1$.

For query, we return the estimate of the scheme instance in the highest level, i.e., the level h. (If there is not any level, we return 0 as the estimate of each item.)

When the window size n is decreased by one to n', we update as follows. If $n' = 2^{h-1}$, level h is removed so that level $h-1$ is the highest level; otherwise, if level h is doing exact counting, we remove the expired item in the stored stream and update the count of the corresponding item, else we remove the expired head of the deque in all λ-counters in level h.

When the window size n is increased by one to n', there is a new item e' inserted to the sliding window. If $n' \leq 2^h$, all the $h+1$ levels are updated according to the scheme for fixed size window. If $n' > 2^h$, we create the level $h+1$ as follows. If $\varepsilon 2^{h+1} < 16$, then exact counting on the frequency of all items in the sliding window is done in level $h+1$ by copying the stream and exact counts stored in level h. (If $h = -1$, the sliding window stored in level $h+1$ is simply the new item e' and its count is 1.) Otherwise, we first create the algorithm instance of level $h+1$ which has sliding window size $n = 2^h$ by using information from level h as follows and then update all the $h+2$ levels for inserting the new item e', i.e., increasing the window size by one to n':

Case 1. If level h is doing exact counting, the whole sliding window is stored, it is straightforward to create all the λ-counters in level $h+1$ from the sliding window stored in level h. Since level h is doing exact counting, we have $\varepsilon 2^h < 16$ and there are at most $2^h < \frac{16}{\varepsilon}$ items with positive counts, i.e., at most $\frac{16}{\varepsilon}$ λ-counters in level $h+1$ have positive values.

Case 2. If level h is not doing exact counting, we create level $h+1$ from level h as follows. Let $Q_{i,e}$, $\ell_{i,e}$, $curblk_i$, $offset_i$ and λ_i be the deque $C_e.Q$ and variables $C_e.\ell$, $curblk$,

[3]Again, to simplify notation, we assume that $\varepsilon 2^i/16$ and $16/\varepsilon$ are integers.

position	1	2	3	4	5	6	7	8	9
bit stream f'_e	1	1	1	1	1	1	1	1	1
sampled 1-bit for λ_h-block	√			√			√		
λ_h-block	1			2			3		
sampled 1-bit for λ_{h+1}-block				√					
λ_{h+1}-block	1						2		
		window W_9							

Figure 2: An example of the bit stream f'_e with $\lambda_h = 3, \lambda_{h+1} = 6$ and window size $n = 8$.

offset and λ of the scheme instance in level i. Consider the sliding window $W_p = [p - n + 1..p]$, for any item e, $f_e = b_1 b_2 \cdots b_p$ where $b_i = 1$ if $e_i = e$; and 0 otherwise. In the scheme instance of level h, recall that each λ_h-counter C_e for item e stores the λ_h-snapshot of the bit stream $f'_e = b'_1 b'_2 \cdots b'_p$ over W_p, where $b'_i \leq b_i$ for each $1 \leq i \leq p$. Conceptually, we create the λ_{h+1}-counters in level $h + 1$ such that the λ_{h+1}-counter for item e stores the λ_{h+1}-snapshot of the bit stream $f'_e = b'_1 b'_2 \cdots b'_p$ over W_p: (1) Since $\lambda_{h+1} = \varepsilon 2^{h+1}/16 = 2\lambda_h$, the size of the λ-block in level $h + 1$ is double of that in level h. We set $curblk_{h+1} = \lceil \frac{curblk_h}{2} \rceil$. If $curblk_h \mod 2 \neq 0$, we set $offset_{h+1} = offset_h$; otherwise, we set $offset_{h+1} = \lambda_h + offset_h$. (2) For each item e with a positive value of its λ_h-counter in level h, we create a corresponding λ_{h+1}-counter in level $h + 1$: All the items in $Q_{h,e}$ with their deque indices (which starts from 1) divisible by 2 are copied to $Q_{h+1,e}$. If the λ_h-block index is q, the λ_{h+1}-block index stored in $Q_{h+1,e}$ is $\lceil q/2 \rceil$. If there are even number of items in the deque $Q_{h,e}$, we set $\ell_{h+1,e} = \ell_{h,e}$; otherwise, we set $\ell_{h+1,e} = \lambda_h + \ell_{h,e}$. Since there are at most $\frac{16}{\varepsilon}$ items with positive counts in level h, there are also at most $\frac{16}{\varepsilon}$ λ-counters in level $h + 1$ with positive values.

Figure 2 shows an example bit stream f'_e where $\lambda_h = 3$ and window size $n = 8$. Suppose just before the new item e' arrives, the current position is 9. Note that in Figure 2, position 1 is not the actual start of the stream and thus there is a sampled 1-bit in position 1. In level h, we have $curblk_h = 3$, $offset_h = 2$, $Q_{h,e} = \langle 1, 2, 3 \rangle$ and $\ell_{h,e} = 2$. Then in level $h + 1$, we set $curblk_{h+1} = \lceil 3/2 \rceil = 2$, $offset_{h+1} = 2$, $Q_{h+1,e} = \langle 1 \rangle$ and $\ell_{h+1,e} = \lambda_h + 2 = 5$.

In the rest of this section, we show that for any item e, the estimate of its frequency obtained from the highest level h is a good estimate of the number of items e in the data stream $f = b_1 b_2 \cdots b_p$ over the sliding window $W_p = [p - n + 1..p]$ of variable-size n.

In level h, each λ_h-counter C_e for item e stores the λ_h-snapshot of the bit stream $f'_{e,h} = b'_1 b'_2 \cdots b'_p$ over W_p, where $b'_i \leq b_i$ for $1 \leq i \leq p$. Only batch of decrements that occurs when the current position is p' where $p - n + 1 \leq p' \leq p$ can change some b'_i $(p - n + 1 \leq i \leq p)$ from 1 to 0. Note that for exact counting, we have the bit stream $f'_{e,h} = f_e$. The following lemma bounds the number of such batches of decrements that occurs when the current position is p' where $p - n + 1 \leq p' \leq p$.

LEMMA 13. *For any item e, consider the bit stream $f'_{e,i} = b'_{i,1} b'_{i,2} \cdots b'_{i,p}$ of a λ_i-counter in level i where $0 \leq i \leq h$. For any interval of size $k2^i$ bits in the sliding window W_p where $0 < k \leq 1$, let d denotes the number of batches of decrements that occurs when the current position p' is in that interval and change some $b'_{i,q}$ $(p - n + 1 \leq q \leq p)$ from 1 to 0. We have $d < (\frac{5+k}{16})(\varepsilon 2^i)$.*

PROOF. We prove it by induction on i. When $i = 0$, we have $\varepsilon 2^0 \leq 1 < 16$, so we do exact counting on level i. Therefore, the number d of batches of decrements in that interval is $0 < (\frac{5+k}{16})(\varepsilon 2^0)$.

Assume that in the bit stream $f'_{e,j}$ for λ_j-counter of item e in level j, the number of batches of decrements is less than $(\frac{5+k}{16})(\varepsilon 2^j)$ for any interval of size $k2^j$ in the sliding window.

Consider the bit stream $f'_{e,j+1}$ for an item e in level $j+1$. If $\varepsilon 2^{j+1} < 16$, we do exact counting on level $j + 1$ and thus the number d of decrement operations in that interval is $0 < (\frac{5+k}{16})(\varepsilon 2^{j+1})$; otherwise, i.e., $\varepsilon 2^{j+1} \geq 16$, at the time that level $j + 1$ is created, the bit stream $f'_{e,j+1}$ of item e in level $j + 1$ is equal to the one $f'_{e,j}$ of item e in level j. Therefore, consider any interval in $f'_{e,j+1}$, there are two intervals A and B, in which interval A is copied from level j and interval B is produced after the level $j + 1$ is created, i.e., by the scheme instance of level $j + 1$. There are two possible cases for an interval I of size $k2^{j+1}$ in $f'_{e,j+1}$:

Case 1. The first $k_1 2^{j+1}$ bits of the interval I is in interval A while the remaining $k_2 2^{j+1}$ bits of the interval I is in interval B, where $k_1 + k_2 = k$, $k_1 > 0$ and $k_2 \geq 0$. Let d_A be the number of batches of decrements in interval A for interval I, and d_B be the number of batches of decrements in interval B for interval I. Since $k_1 2^{j+1} = 2k_1 2^j$, by induction hypothesis, we have $d_A = (\frac{5+(2k_1)}{16})(\varepsilon 2^j)$. For analysis of d_B, we use arguments similar to that in Lemma 9. Let $n_{e,i}(W_{p-n})$ be the number of 1-bits of f_e over W_{p-n} and $n'_{e,i}(W_{p-n})$ be the number of 1-bits of $f'_{e,i}$ over W_{p-n}. When $f'_{e,j}$ is copied to level $j + 1$, there are at most 2^j items in the sliding window in all the $\frac{16}{\varepsilon}$ λ_j-counters in level j. Let bit t be the first bit in interval B. Therefore, in level $j + 1$, just before executing $Update(t)$, there are at most 2^j items in the sliding window in all the $\frac{16}{\varepsilon}$ λ_{j+1}-counters in level $j + 1$. By similar arguments used in the proof of Lemma 9, we have $v(C_e) \leq n'_e(W_p) + 2\lambda_{j+1} = n_e(W_p) + \varepsilon 2^{j+1}/8$ and hence

$$d_B < \frac{2^j + \frac{16}{\varepsilon}(\frac{\varepsilon 2^{j+1}}{8}) + k_2 2^{j+1}}{\frac{16}{\varepsilon}} = \frac{(5 + 2k_2)\varepsilon 2^j}{16}$$

Therefore, the number d of batches of decrements in I is

$$\begin{aligned} d &= d_A + d_B < \frac{(5 + 2k_1)\varepsilon 2^j}{16} + \frac{(5 + 2k_2)\varepsilon 2^j}{16} \\ &= \frac{(10 + 2k)\varepsilon 2^j}{16} = \frac{(5 + k)\varepsilon 2^{j+1}}{16} \end{aligned}$$

Case 2. The interval I of size $k2^{j+1}$ is in interval B only. Just before executing *Update* on the first bit in interval I, there are at most 2^{j+1} items in the sliding window in all $\frac{16}{\varepsilon}$ window counters. By similar arguments in the proof in Lemma 9, the number d of batches of decrements in interval I is smaller than $(2^{j+1} + \frac{16}{\varepsilon}(\frac{\varepsilon 2^{j+1}}{8}) + k2^{j+1})/\frac{16}{\varepsilon} = \frac{(3+k)\varepsilon 2^{j+1}}{16} < \frac{(5+k)\varepsilon 2^{j+1}}{16}$. □

THEOREM 14. *Suppose W_p is the sliding window. The above scheme will return a set B of items such that every item in B must occur at least $(\theta-\varepsilon)n$ times in W_p, and any item that occurs more than θn times in W_p must be in B.*

PROOF. When query, the answer is from the highest level, i.e., level h. If $h = 0$, since $\varepsilon 2^0 < 16$, the answer is from exact counting and is correct; otherwise, the sliding window is of size $n = (1+k)2^{h-1} = (\frac{1+k}{2})2^h$ where $0 < k \leq 1$. Let n_e be the number of 1-bits of f_e over W_p. For any item e, by Lemma 13 and arguments similar to that in Lemma 10, we have $v(C_e) - 2\lambda_h \leq n_e$ and

$$\begin{aligned} v(C_e) - 2\lambda_h &\geq n_e - d - \frac{\varepsilon 2^h}{8} > n_e - \frac{(5+\frac{1+k}{2})\varepsilon 2^h}{16} - \frac{\varepsilon 2^h}{8} \\ &= n_e - \frac{(15+k)\varepsilon 2^h}{32} > n_e - \varepsilon n \end{aligned}$$

Therefore, if an item e is in the set B, we have $(\theta-\varepsilon)n \leq v(C_e) - 2\lambda_h \leq n_e$. Thus, every item in B must occur at least $(\theta-\varepsilon)n$ times in W_p. On the other hand, if an item e is not in B, we have $(\theta-\varepsilon)n > v(C_e) - 2\lambda_h > n_e - \varepsilon n$, i.e., $n_e < \theta n$. Therefore, any item that occurs more than θn times in W_p must be in B. □

THEOREM 15. *The above scheme uses $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ space.*

PROOF. There are totally $h+1$ levels, where $2^{h-1} < n \leq 2^h$. Since $\varepsilon 2^0 < 16$, we do exact counting in level 0. Suppose we are doing exact counting from level 0 to level j where $1 \leq j \leq 2^h$ and we use the scheme instances for fixed-size sliding window from level $j+1$ to level h. Since level j is doing exact counting, we have $\varepsilon 2^j < 16$, i.e., $2^j < 16/\varepsilon$. The space required for all levels that do exact counting is $O(\sum_{0\leq k\leq j} 2^k) = O(2^j \sum_{0\leq k\leq j} 1/2^k) = O(2^{j+1}) = O(\frac{1}{\varepsilon})$.

Consider the number m of level using the scheme instances for fixed-size sliding window. If $j = h$, we have $m = 0$; otherwise, level $j+1$ exists and we have $\varepsilon 2^{j+1} \geq 16$, so that $m = h - j = \log(2^h/2^j) < \log(2n/(8/\varepsilon)) = \log(\varepsilon n/4)$. Since each scheme instance requires $O(\frac{1}{\varepsilon})$ space, the total space required is $O(\frac{1}{\varepsilon} + \frac{m}{\varepsilon}) = O(\frac{1}{\varepsilon}\log(\varepsilon n))$. □

Acknowledgement. We are grateful to the anonymous referees for many helpful suggestions on improving the presentation of the paper.

8. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.

[2] A. Arasu and G. Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems*, pages 286–296, 2004.

[3] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming*, pages 693–703, 2002.

[4] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 296–306, 2003.

[5] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[6] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proceeding of the 10th Annual European Symposium on Algorithms*, pages 348–360, 2002.

[7] C. Estan and G. Varghese. New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice. *ACM Transactions on Computer System*, 21(3):270–313, 2003.

[8] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 331–342, 1998.

[9] A. C. Gilbert, Y. Kotidis, and M. J. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 79–88, 2001.

[10] L. Golab, D. DeHaan, E. D. Demaine, A. López-Ortiz, and J. I. Munro. Identifying frequent items in sliding windows over on-line packet streams. In *Proceedings of the Internet Measurement Conference*, pages 173–178, 2003.

[11] L. Golab, D. DeHaan, A. López-Ortiz, and E. D. Demaine. Finding frequent items in sliding windows with multinomially-distributed item frequencies. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, pages 425–426, 2004.

[12] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 471–475, 2001.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

[14] 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 International Conference on Information and Knowledge Management*, pages 287–294, 2003.

[15] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems*, 28:51–55, 2003.

[16] L. K. Lee and H. F. Ting. Maintaining significant stream statistics over sliding windows. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 724–732, 2006.

[17] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *Proceedings of the 21st International Conference on Data Engineering*, pages 767–778, 2005.

[18] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982.