WUJNS
Wuhan University Journal of Natural Sciences

Vol. 11 No. 1 2006 283-288

Article ID: 1007-1202(2006)01-0283-06

Dynamically Computing Approximate Frequency Counts in Sliding Window over Data Stream

□ NIE Guo-liang, LU Zheng-ding†
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China

Abstract: This paper presents two one-pass algorithms for dynamically computing frequency counts in sliding window over a data stream-computing frequency counts exceeding user-specified threshold ε. The first algorithm constructs sub-windows and deletes expired sub-windows periodically in sliding window, and each sub-window maintains a summary data structure. The first algorithm outputs at most $1/\varepsilon + 1$ elements for frequency queries over the most recent N elements. The second algorithm adapts multiple levels method to deal with data stream. Once the sketch of the most recent N elements has been constructed, the second algorithm can provides the answers to the frequency queries over the most recent $n (n \leq N)$ elements. The second algorithm outputs at most $1/\varepsilon + 2$ elements. The analytical and experimental results show that our algorithms are accurate and effective.

Key words: data stream; sliding window; approximation algorithms; frequency counts
CLC number: TP 311

Received date: 2005-04-20
Foundation item: Supported by the National Natural Science Foundation of China (60403027)
Biography: NIE Guo-liang(1975-), male, Ph.D. candidate, research direction: distributed computing, the theory of data stream. E-mail: nie_guoliang@163.com
† To whom correspondence should be addressed. E-mail: zdlu4409@public.wh.hb.cn

0 Introduction

In several emerging applications including networking, telecommunications, financial services, e-commerce, and sensor networks, data takes the form of data streams as opposed to finite stored datasets. Data stream is modeled as an infinite sequence of finite lists elements, and differs from traditional data in three primary aspects: ①continuity, ②unknown or unbounded length, and ③inability to backtrack over previously arrived items.

Approximate frequency counts over data streams play an important role in many data streams' decision support systems, but traditional count algorithms are not suitable for approximate frequency counts over data streams. Hence, it is necessary and significant to study count algorithms over data stream.

Until now, approximate frequency counts over data streams have received considerable attention and many research breakthroughs have been made. Misra and Gries[1] firstly presented deterministic algorithm for εapproximate counting over data streams. This algorithm had been improved by Demaine *et al*[2] and Karp *et al*[3]. Manku and Motwani[4] presented sticky sampling counting algorithm and lossy counting algorithm.

But these algorithms ignore the fact that the recent elements of a stream are more important than those arrived a long time ago. Using a sliding window[5], Arasu and Manku[6] brought forward an algorithm for εapproximate counts over data streams. The query time is quite long. Further, the out-

put includes too many false positives[7].

The task we will tackle in this paper is defined as follows:

Given N and ε, we have designed a one-pass algorithm to compute the ε approximate frequency of frequent elements of the most recent N elements using as little main memory as possible. Frequent elements of the most recent N elements are items whose actual frequencies exceed εN.

1 Related Work

1.1 Frequent Elements Algorithm

Fisher and Salzberg[7] brought forward majority algorithm for finding an item whose frequency is more than half of the total number. By making one pass over dataset, the algorithm finds one element that is guaranteed to be the majority element if any such element exists.

A natural generalization of this method to find frequent items whose frequencies exceed N/k was given by Misra and Gries[1]. The same algorithm has been rediscovered recently by Demaine *et al*[2] and Karp *et al*[3], who reduced the processing time to $O(1)$ in the worst case.

Manku and Motwani[4] presented lossy counting algorithm, a deterministic algorithm.

Above all, lossy counting algorithm is well suitable to the skewed data stream. Further, the algorithm can be adapted to compute association rules[8] over data streams.

Though above algorithms are suitable for data streams, they do not have obvious counterparts in the sliding window model.

Golab *et al*[9] proposed an algorithm that uses the basic window model[10] to find frequent elements in sliding window. This algorithm uses limited memory, requires constant processing time per element, and makes only one pass over the data. But in the worst case, the memory cost is very large.

Arasu and Manku presented various deterministic and randomized algorithms for ε approximate counts and quantiles[6]. These algorithms are suitable for both fixed-size sliding window and variable-size sliding window. But the large number of false positives the result includes degrades severely value in use.

1.2 Sliding Window Synopsis Data Structures

The sliding window model is useful for discounting stale data in data stream applications.

In this model, data elements arrive continually and old items must be simultaneously evicted from the window so that only the most recent N elements are used when answering queries.

Zhu and Shasha introduced basic windows to incrementally compute simple windowed aggregates in sliding window[10]. The window is divided into equally-sized basic windows and only a synopsis and a timestamp are stored for each basic window. This method does not require the storage of the entire sliding window, but results are refreshed only after the stream fills the current basic window. If the available memory is small, the number of synopses that may be stored is small and the refresh interval is large.

Exponential histogram (EH) was introduced by Datar *et al*[5] and recently expanded by Qiao *et al*[11] to provide approximate answers to simple window aggregates at all times. The idea is to build basic windows with various sizes and maintain a bound on the error caused by counting those elements in the oldest basic window which may be expired.

Gibbons and Tirthapura[12] improved the results from Datar *et al*[5] for computing counts and sums over sliding windows. They present a new data structure called waves that have a worst-case update time of $O(1)$ compared to $O(\log N)$ for the EH data structure.

2 The First Algorithm

2.1 The Description of the First Algorithm

In this section, we bring forward one algorithm for frequency counts in sliding window over data stream. The answers produced by the algorithm have the following guarantees:

1) $(f_e - \varepsilon N) < \tilde{f}_e \leq f_e$. $\tilde{f}_e$ represents the approximate frequency of element e in current sliding window with size N; f_e represents the true frequency of e in current sliding window with size N.

2) All elements whose true frequencies exceed εN are output. There are no false negatives.

We assume that $1/\varepsilon$ and εN are integers to avoid floors and ceilings in expressions. For arbitrary N and ε, we identify N_0 and ε_0 such that:

a) N_0, $1/\varepsilon_0$ and $\varepsilon_0 N_0$ are integers;

b) $N \leq N_0 < N + 1/\varepsilon_0$;

c) $\varepsilon_0 N_0 \leq \varepsilon N$. We then use N_0 and ε_0 in the place

of N and ε.

We employ the notion of timestamp which corresponds to the position of an active data element in the current sliding window. We timestamp the active data elements from right to left, with the most recent element 's timestamp being 1. Clearly, the timestamps change with every new arrival. We adopt the solution provided by Datar *et al*[5] to avoid making explicit updates of timestamps.

In this paper, we employ sub-windows that are unequally-sized windows. Moreover, there are overlaps among sub-windows.

For one sub-window, we keep a synopsis and a timestamp which is identical to the timestamp of the oldest element in this window. When its timestamp becomes $N + \varepsilon N$, the sub-window expires and is dropped. For the sliding window with size N, we create one new sub-window per εN elements. Hence, there are at most $1/\varepsilon + 1$ active sub-windows in current sliding window. The size of sub-window is not fixed, which increases gradually with every new arrival until the sub-window expires. It is obvious that the size of sub-window equals its timestamp. Once the sub-window is created, MG algorithm[1] always runs over this window and updates the synopsis with every new arrival until the window expires.

The detailed description of the first algorithm is as follows:

When a new data element e arrives,

1) Increase variable COUNT by 1, which is initialized to 0.

2) If COUNT = $N + \varepsilon N$, delete the window with the oldest timestamp, and assign N to COUNT.

3) If COUNT mod $\varepsilon N = 1$, create a new sub-window with timestamp 1, and execute MG algorithm with parameter $\varepsilon/(\varepsilon+1)$ over this new window until this window is deleted.

4) Append element e to all active sub-windows.

When answering frequency queries, we just find the oldest active sub-window QW in current sliding window. Obviously, the synopsis of QW provides the correct answers we need.

2.2 The Analysis of the First Algorithm

Theorem 1 Algorithm described in section 2.1 allows ε-approximate frequency counts to be computed over the most recent N elements. Using $O(1/\varepsilon^2)$ space, it provides at most $1/\varepsilon + 1$ candidate elements and does not make false negatives. Further, it guarantees that $(f_e - \varepsilon N) < \tilde{f}_e \leq f_e$.

Proof For the sliding window with size N, we create one sub-window per εN elements. When its timestamp becomes $N + \varepsilon N$, the sub-window expires and is dropped. Hence, there are at most $1/\varepsilon + 1$ sub-windows in current sliding window.

In addition, the space complexity of MG algorithm is $O(1/\varepsilon)$. So, algorithm described in section 2.1 utilizes $O(1/\varepsilon^2)$ space.

Because when answering frequency queries, we just find the oldest active sub-window QW with timestamp T which is larger than or equal to N to report the candidate elements.

In addition, the timestamp of QW is smaller than $N + \varepsilon N$, otherwise, the sub-window is dropped. For QW, the MG algorithm guarantees that all elements whose frequencies exceed $T\varepsilon/(\varepsilon+1)$ must be stored. As long as we guarantee that εN is larger than or equal to $T\varepsilon/(\varepsilon+1)$, algorithm described in section 2.1 does not make false negatives.

Clearly, $T\varepsilon/(\varepsilon+1) < (N + \varepsilon N)\,\varepsilon/(\varepsilon+1) = \varepsilon N$

According to MG algorithm, $(f_e - \varepsilon N) < \tilde{f}_e \leq f_e$.

Further, the synopsis created by MG with parameter $\varepsilon/(\varepsilon+1)$ contains at most $1/\varepsilon + 1$ elements. Hence, we output at most $1/\varepsilon + 1$ elements.

MG algorithm requires $O(1)$ amortized processing time per element, and algorithm described in section 2.1 has $1/\varepsilon + 1$ sub-windows in current sliding window. Hence, algorithm described in section 2.1 requires $O(1/\varepsilon)$ amortized processing time per element.

The query time is $O(1)$.

3 The Second Algorithm

3.1 The Description of the Second Algorithm

Once the sketch has been constructed, algorithm described in section 2.1 allows ε-approximate frequency counts to be computed over the most recent N elements, but it is of no effect for answering frequency queries over the most recent n $(n < N)$ elements.

In this section, we present an algorithm to deal with this question.

We conceptually make $L + 1$ copies of the stream, where $L = \lceil \log(\varepsilon N) \rceil$ We say that these copies are at different levels, which are numbered sequentially as 0, 1, $\cdots, L$. We maintain at most $1/\varepsilon + 1$ active sub-windows for each level. We treat the size of sliding window in kth level as $1/\varepsilon^k$ except that the size of sliding window in

last level is N. Within kth level ($k < L$), we create one new sub-window per 2^k elements, and delete it when its timestamp becomes $(1 + 1/\varepsilon)2^k$. Within last Level, we create one new sub-window per εN elements, and delete it when its timestamp becomes $N + \varepsilon N$. Once a sub-window is created, MG algorithm with parameter $\varepsilon/(2\varepsilon+1)$ always runs over this window and updates the synopsis with every new arrival until the window expires.

When a new data element arrives, we insert it into all levels. In each level, the newest element is processed in the light of algorithm described in section 2.1.

When answering the frequency queries for $n \leq N$, first, we calculate $k = \lceil \log(\varepsilon n) \rceil$. Then, within kth level, we find the sub-window QW whose timestamp exceeds or equals n and is closest to n. Last, we output the synopsis of QW.

3.2 The Analysis of the Second Algorithm

Theorem 2 Once the sketch of the most recent N elements has been constructed, algorithm described in section 3.1 allows ε approximate frequency counts to be computed over the most recent $n (n \leq N)$ elements. Using $O(1/\varepsilon^2 \log \varepsilon N)$ space, it provides at most $1/\varepsilon + 2$ candidate elements and does not make false negatives. Further, it guarantees that $(f_e - \varepsilon n) < \tilde{f}_e \leq f_e$.

Proof The sketch is maintained in $\lceil \log(\varepsilon N) \rceil + 1$ levels, each of which includes $1/\varepsilon + 1$ sub-windows. Further, each sub-window needs $O(1/\varepsilon)$ space. So, we need $O(1/\varepsilon^2 \log \varepsilon N)$ space for storing sketch.

When answering frequency queries over the most recent $n (n \leq N)$ elements, we find sub-window QW with timestamp T which is larger than or equal to n and is closest to n in level $k = \lceil \log(\varepsilon n) \rceil$. So, $n > T - 2^k$ and $n \leq T$. Obviously, $(1/\varepsilon)2^{k-1} < n$. According to MG algorithm, we cause no false negatives if we guarantee that $\varepsilon/(2\varepsilon+1)\, T \leq \varepsilon n$.

When $T = n$, there are no false negatives.

When $T > n$, without loss of generality, we assume $T = u2^{k-1} + v$, $v < 2^{k-1}$. u must be larger than or equal to $1/\varepsilon$; otherwise, $(1/\varepsilon)2^{k-1} > n$. For u, there are two cases: $u < 1/\varepsilon + 2$; $u \geq 1/\varepsilon + 2$.

If u is smaller than $1/\varepsilon + 2$, $\varepsilon/(2\varepsilon+1)\, T < \varepsilon/(2\varepsilon+1)\,((1/\varepsilon)2^{k-1} + 2^k) = 2^{k-1}$. Obviously, $\varepsilon n > 2^{k-1}$. Hence, $\varepsilon/(2\varepsilon+1)\, T < \varepsilon n$. Consequently, when T is smaller than $(1/\varepsilon)2^{k-1} + 2^k$, algorithm described in section 3.1 does not generate false negatives.

If u is larger than or equal to $1/\varepsilon + 2$, $(1/\varepsilon)2^{k-1} + 2^k \leq T$, then we get $0 \leq 2\varepsilon T - 2\varepsilon 2^k - 2^k$. Consequently, $\varepsilon/(2\varepsilon+1)\, T \leq \varepsilon(T - 2^k)$. In addition, $n > (T - 2^k)$. So, when T is larger than or equal to $(1/\varepsilon)2^{k-1} + 2^k$, no false negatives are generated by algorithm described in section 3.1.

In conclusion, algorithm described in section 3.1 does not generate false negatives.

Because the parameter of MG algorithm over sub-window is set to $\varepsilon/(2\varepsilon+1)$, so there are at most $1/\varepsilon + 2$ candidate elements to be output.

According to the MG algorithm, $(f_e - \varepsilon n) < \tilde{f}_e \leq f_e$.

When a new element arrives, it will be inserted into all sub-windows. Hence, algorithm described in section 3.1 requires $O(1/\varepsilon \log \varepsilon N)$ amortized processing time per element.

The query time is $O(\log(1/\varepsilon))$.

4 Experiments

The figures are shown in Figs. 1-6.

We experiments with data stream that follows Zipf distribution. In our experiments, the Zipf parameter z is set to 0.9. All experiments are carried out on a 1.6 GHz

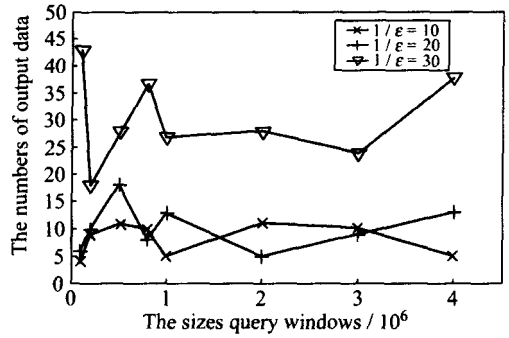

Fig. 1 The changes of the number of elements output by algorithm described in section 2.1 while both N and ε change

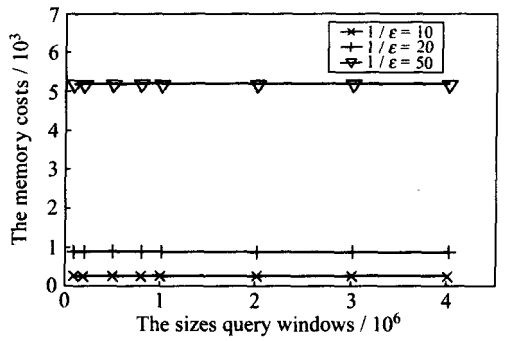

Fig. 2 The changes of the memory cost of algorithm described in section 2.1 while both N and ε change

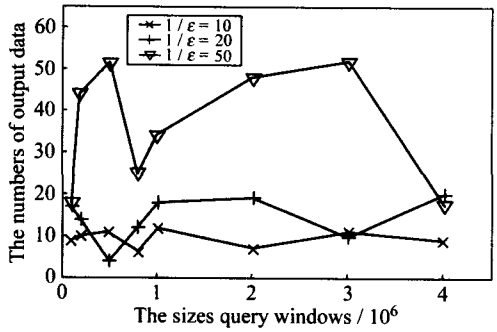

Fig. 3 The changes of the number of elements output by algorithm described in section 3.1 while both N and ε change

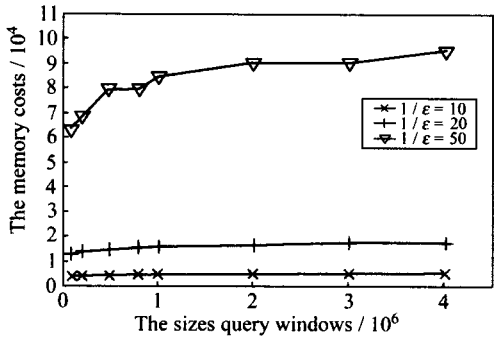

Fig. 4 The changes of the memory cost of algorithm described in section 3.1 while both N and ε change

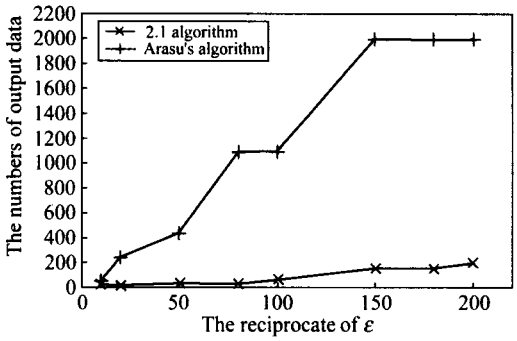

Fig. 5 The comparison of the number of output data between algorithm described in section 2.1 and Arasu's algorithm while N is 2000000

Pentium V processor running Windows Server.

Firstly, we carry out experiments to survey the performances of Algorithm described in section 2.1 and Algorithm described in section 3.1.

According to Fig. 1 and Fig. 3, we can draw a conclusion that the numbers of elements output by our algorithms are independent of the size of sliding window. The number of output data in algorithm described in section 2.1 fluctuates between 0 and $1/\varepsilon + 1$ while the number of elements output by algorithm described in section 3.1 fluctuates between 0 and $1/\varepsilon + 2$. Moreover, our algo-

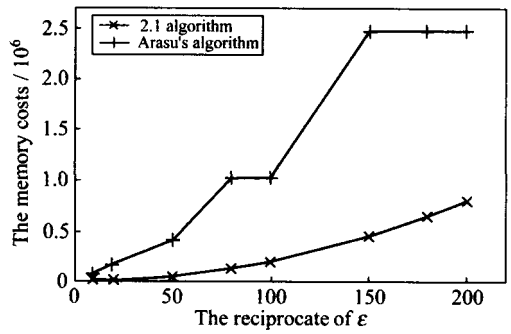

Fig. 6 The comparison of memory cost between Algorithm described in section 2.1 and Arasu's algorithm while N is 4 000 000

rithms are more likely to output more elements as ε decreases. From Fig. 2 and Fig. 4, we can conclude that the memory cost of algorithm described in section 2.1 keeps steady in despite of the increment of sliding window's size while the memory cost of algorithm described in section 3.1 keeps increasing trend as the sliding window's size increases. It is obvious that the memory costs of algorithm described in section 2.1 and algorithm described in section 3.1 are influenced by ε. Further, the smaller ε is, the larger the memory costs of algorithm described in section 2.1 and algorithm described in section 3.1 are.

Last, we compare the performances of algorithm described in section 2.1 and Arasu's algorithm[6].

According to Fig. 5, which depicts the comparison of the number of output data between algorithm described in section 2.1 and Arasu's algorithm, it is obvious that Arasu's algorithm outputs more elements than $1/\varepsilon$ while algorithm described in section 2.1 outputs at most $1 + 1/\varepsilon$ elements. Figure 6 reveals the comparison of memory cost between algorithm described in section 2.1 and Arasu's algorithm. Arasu's algorithm utilizes more memory than algorithm described in section 2.1 for fixed ε. Further, as ε decreases, the memory cost of algorithm described in section 2.1 increases while the memory cost of Arasu's algorithm keeps non-decreasing trend. The non-decreasing trend is caused by the fact that in Arasu's algorithm arbitrary ε is transformed to ε' such that $1/\varepsilon'$ is the power of 2 and ε' does not exceed ε.

5 Conclusion

In this paper, we present two algorithms for computing frequency counts in sliding window over data stream. The first algorithm outputs at most $1/\varepsilon + 1$ elements using $O(1/\varepsilon^2)$ space. Above all, the space complexity

keeps invariable despite the increment of sliding window's size. Once the sketch is constructed for sliding window with size N, the second algorithm outputs at most $1/\varepsilon + 2$ elements for the frequency queries over the most recent $n(n \leq N)$ elements.

Though the dataset output by our algorithm includes all frequent items in sliding window, it includes many elements that do not belong to frequent items. In the future, we will analysis the probability of false positives. Another work is that depending on the distribution of data stream, we will adjust our algorithm dynamically to generate more accurate dataset. Relating to our work, the top-k query in sliding window is an open problem. The studying of data stream is in its infancy, many work need to be done.

References

[1] Misra J, Gries D. Finding Repeated Elements. *Sci Comput Programming*, 1982, **2**(2):143-152.

[2] Demaine E D, Lopez-Ortiz A, Munro J I. Frequency Estimation of Internet Packet Streams with Limited Space. *Proc of the 10th Annual European Symp on Algorithms*. Rome, Springer-Verlag, 2002. 348-360.

[3] Karp R M, Shenker S, Papadimitriou C H. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *ACM Trans on Database Systems*, 2003, **28**(1):51-55.

[4] Manku G S, Motwani R. Approximate Frequency Counts over Data Streams. *Proc of the 28th Intl Conf on very Large Data Bases*. Hong Kong, Morgan Kaufmann, 2002. 346-357.

[5] Datar M, Gionis A, Indyk P, *et al*. Maintaining Stream Statistics over Sliding Windows. *Proc 13th SIAM-ACM Symp on Discrete Algorithms*. San Francisco, ACM/SIAM, 2002. 635-644.

[6] Arasu A, Manku G S. Approximate Counts and Quantiles over Sliding Window. *Proc of ACM Symp on Principles of Database Systems*. Paris, ACM, 2004. 286-296.

[7] Fischer M, Salzberg S. Finding a Majority among n Votes: Solution to Problem 81-5. *Journal of Algorithms*, 1982, **3**(4):376-379.

[8] Agrawal R, Imielinski T, Swami A N. Mining Association Rules Between Sets of Items in Large Databases. *Proc of the 1993 ACM SIGMOD Intl Conf on Management of Data*. Washington, ACM, 1993. 207-216.

[9] Golab L, Dehaan D, Demaine E, *et al*. Identifying Frequent Items in Sliding Windows over On-Line Packet Streams. *SIGCOMM Internet Measurement Conference*. Miami, ACM, 2003. 173-178.

[10] Zhu Y, Shasha D. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. *Proc 28th Int Conf on Very Large Data Bases*. Hong Kong, Morgan Kaufmann, 2002. 358-369.

[11] Qiao L, Agrawal D, Abbadi A E. Supporting Sliding Window Queries for Continuous Data Streams. *Proc 15th Int Conf on Scientific and Statistical Database Management*. Massachusetts: IEEE Computer Society, 2003. 85-94.

[12] Gibbons P B, Tirthapura S. Distributed Streams Algorithms for Sliding Windows. *Proc 14th ACM Symposium on Parallel Algorithms and Architectures*. Winnipeg, ACM Press, 2002. 63-72.

□