Randomized Approximation of the Frequency of Items in a Stream using a Small Space

Heejae Yim[*] Yukiko Yamauchi[*] Shuji Kijima[*] Masafumi Yamashita[*]

Abstract

These days, analyzing large data such as a network traffic becomes more important for its applications such as to proactive response from cyber attacks. However, the size of large data makes it hard to preserve all data or to query on them. Therefore, streaming algorithms which could catch useful information from sequentially coming data on a limited memory space are effective to analyze big data. There are many previous works considering problems in a data stream such as *histograms*, *frequent items*, the *number of distinct items*, and the *frequency moment*.

Motivated by the extension of these problems, this paper deals with the problem of estimating *the frequency distribution* in a data stream. This problem is to find the number of types of items for each frequency in a stream, which is available to examine some statistical properties of the stream. We propose a randomized algorithm to estimate the *frequency distribution*. This algorithm uses $\mathrm{O}(\log^2 m)$ memory space for some streams satisfying simple conditions, where m is the maximum number of distinct items of a stream.

Keywords: Streaming Algorithm, Frequency distribution

1 Introduction

In recent years, computing for large data such as network traffic and sensor data becomes more and more important because of its applications. For example, analyzing data and computing statistics on network traffic are available to use proactive response against cyber-attacks. However, the storage of a large mount of data requires a huge memory space, as well as it takes a long time. Thus, streaming algorithms that can extract valid information from sequentially coming data on a limited memory space is attracting attention.

For processing on a data stream, there are many previous problems such as the *frequent moment*, the *number of distinct elements*, the *heavy hitters*, and the *histogram*. Alon et al.[1] deal with the (k-th) *frequent moment* $F_k = \sum_{e=1}^{m} n_e^k$, where input data stream is $S = e_1, \ldots, e_N (e_i \in \{1, ..., m\})$ and n_e is the frequency of element e in the stream. 0-th *frequent moment*, which is the number of distinct elements, is also well studied problem[2, 4, 5, 6, 12]. One of the practical applications of the *frequent moment* is the entropy of data stream $H(S) = \sum_{e=1}^{m} -\frac{n_e}{N} \log \frac{n_e}{N} = \log N - \frac{1}{N} \sum_{e=1}^{m} n_e \log n_e$. Streaming algorithms estimating the entropy[3, 14] can be applied to detection of network attack under small memory space.

Heavy Hitters problem[16, 19] is to find elements that occur many times in a stream. To find *heavy hitter* under the limited memory space, sampling method or special data structure are used. Another previously studied problem is to find the *heavy distinct hitter*[15, 21], which is an element that appears with a lot of distinct values in a pairwise data stream consist of (element, value).

A *histogram* in a stream data is a data structure which contains frequency information of elements, presented as a graph that x-axis is buckets(or bins) which are subranges of elements and y-axis is the sum

[*]Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan
{heejae.yim,yamauchi,kijima,mak}@inf.kyushu-u.ac.jp

of frequency of elements included in each bucket. Histograms give information of frequency of elements in each bucket. There are some definitions of *histograms* such as *equi-width histograms*[10, 20] and *V-optimal histograms* [7, 9, 11], according to choosing the range of each bucket. *equi-width histogram* is a histogram whose range is divided into buckets that have equal length. In *V-optimal histogram,* the entire range is divided into buckets minimizing the sum of variance of frequency of elements in each bucket.

Our Results

In this paper, we define the *frequency distribution* and deal with algorithms to estimate the *frequency distribution* in small memory space. Let C_j be a set of elements whose frequency is greater than or equal to 2^j and we define the *frequency distribution* is $|C_0|, |C_1|, \ldots, |C_s|$, where N is the length of a stream and $s = \lfloor \log_2 N \rfloor$. Because $|C_j| - |C_{j+1}|$ is the number of elements whose frequency is greater than or equal to 2^j and less than 2^{j+1}, this distribution gives us the number of elements whose frequency is in $[1, 2), [2, 4), \ldots, [2^j, 2^{j+1}), \ldots, [2^s, \infty)$.

Main difference between *histograms* and the *frequency distribution* is that the buckets of *histogram* are continuous. When we make a histogram of the stream in which the range of elements is $[1, m]$ with 3 buckets B_1, B_2 and B_3, buckets are continuous in by definition, so the range of buckets must be $B_1 = [1, a], B_2 = [a + 1, b], B_3 = [b + 1, m]$ for some integers $1 \leq a < b \leq m$. However, the *frequency distribution* outputs size of group depending on only frequency. Because of this difference, the *frequency distribution* gives us more intuitive information rather than *histograms* for catching DDoS attack or cyber attack by network worm.

Meanwhile, to compute the *frequency distribution* exactly in one pass, any deterministic algorithm must must use $\Omega(m)$ bits[1], because $|C_0|$ equals to $|E_S|$ Therefore, we focus on estimating the *frequency distribution* using small memory space. We present a randomized algorithm to estimate the *frequency distribution*. Our algorithm uses $\mathrm{O}(\log^2 m)$ memory space for data streams that satisfy simple conditions.

Organization This paper is organized as follows. In Section 2, we define problem and show an idea of our algorithm.. In Section 3, we present algorithm and analyze memory complexity and accuracy.

2 Preliminary

2.1 Problem Definition

We consider that an input data stream S is a sequence of elements from $\{1, ..., m\} = [1, m]$, i.e., $S = e_1, \ldots, e_N (e_i \in [1, m])$. e_i is the i-th element of the stream S and N is the length of the stream S. Let E_S be a set that consists of elements observed in the stream S. We denote the frequency of element e by $n_e = |\{i \in 1, \ldots, N : e_i = e\}|$. For example, consider a stream $S_1 = 3, 10, 3, 2, 1, 5, 3, 1, 1, 1$. The length of this stream is 10 and the set of elements observed in this stream is $E_{S_1} = \{3, 10, 2, 1, 5\}$. Each of the frequency of elements are $n_3 = 3, n_{10} = 1, n_2 = 1, n_1 = 4, n_5 = 1$.

Given a stream S, we define that the *frequency distribution* is $|C_0|, |C_1|, \ldots, |C_s|$, where C_j is a set of elements those frequency is greater than or equal to 2^j, i.e., $C_j = \{e \in [1, m] : n_e \geq 2^j\}$ and $s = \lfloor \log_2 N \rfloor$. Note that C_0 equals E_S, which is a set of elements observed in the stream S and $C_0 \supseteq C_1 \supseteq \ldots \supseteq C_{s-1} \supseteq C_s$. For example of the stream S_1, $|C_0| = |\{3, 10, 2, 1, 5\}| = 5$, $|C_1| = |\{3, 1\}| = 2$, $|C_2| = |\{1\}| = 1$, and $|C_3| = 0$.

In this paper, we suppose that we cannot reserve all data from a stream S, because the length of stream S and the range of elements m are very large. Under this condition, our goal is to get the *frequency distribution* from given a data stream S in one pass, which means every element could be read only one time.

2.2 An Idea of Algorithm

In one pass, a naive method to compute the *frequency distribution* is to memorize all elements occurs in a stream and to count each frequency of elements. However, this method uses $\mathrm{O}(|E_S|\log N)$ bits so it is not available for large data stream such as the sequence of IP address whose range of IP address is $m = 2^{32}$.

Meanwhile, to compute the *frequency distribution* exactly is difficult using only small memory space in one pass. Because $|C_0|$ equals to $|E_S|$, which is the number of distinct elements occurs in a stream and any deterministic algorithm that computes the number of distinct elements in one pass must use $\Omega(m)$ bits[1]. Therefore computing $|C_0|, \dots, |C_s|$ exactly is not possible under linear space.

Thus, we focus on estimating the *frequency distribution* using small memory space. An idea of our algorithm is sampling elements for each C_j and using hash function which is bases on Bar-Yossef et al[2]. There are many previous works on estimating the number of distinct elements in a stream[2, 4, 5, 12] including the algorithm by Bar-Yossef et al[2]. By using these algorithm, it is available to estimate $|C_j|$ under $\mathrm{O}(\log |C_j|)$ memory space.

We will show detailed method in section 3. Before going to next section, describe briefly the algorithm by Bar-Yossef et al[2]. This algorithm outputs the estimated value of E_S that is within a relative error of ϵ with probability at least $1 - \delta$, using $\mathrm{O}(\frac{\log \delta^{-1}}{\epsilon^2} \log m)$ memory space and $\mathrm{O}(\log(\epsilon^{-1}))$ update time. The algorithm uses hash function that maps element into $[1, m^3]$ uniformly at random and satisfies pair-wise independence. Reading elements in one pass, the algorithm maintains and updates the list which has $t(= \mathrm{O}(\epsilon^{-2}))$-th smallest hash value. Finally outputs $\frac{tm^3}{v_t}$ as the estimated value of E_S, where v_t is t-th smallest hash value.

3 Algorithms

In this section, we present an algorithm for estimating the *frequency distribution* using small memory space. In subsection 3.1, we describe the sampling method of main algorithm using simple algorithm. In subsection 3.2, we propose main algorithm and discuss its space complexity and accuracy.

3.1 Simple Algorithm

Algorithm 1 Simple Algorithm

```
Require: data stream $S = e_1, \dots, e_N (e_i \in [1, m])$
1: $\widetilde{C}_j \leftarrow \emptyset \ (j = 0, 1, \dots, s)$
2: for $i = 1$ to $N$ do  //for each $i$-th element $e_i$
3:     Flip a fair coin repeatedly until haed appears for the first time or coin is flipped $(s+1)$ times. Let $x$
       be the number of tails.
4:     for $j = 0$ to $x$ do
5:         $\widetilde{C}_j \leftarrow \widetilde{C}_j \cup \{e_i\}$
6:     end for
7: end for
8: return $|\widetilde{C}_0|, \dots, |\widetilde{C}_s|$
```

Algorithm1 called *simple algorithm* outputs $|\widetilde{C}_j|$, which is the estimated value of $|C_j|$, using simple sampling method. For each i-th element e_i, using a fair coin that has the probability of heads is 0.5 and the probability of tails is 0.5, flip the coin repeatedly until head appears for the first time or the coin is flipped $(s+1)$ times. At this time, let x be the number of tails. For $j \in \{0, 1, \dots, x\}$, add e_i to the set $\widetilde{C}_j$ if e_i

Table 1: The *frequency distribution* of the stream S_2.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\|C_j\|$	500	480	400	350	345	344	320	200	150	132	105	98	84	62	30	15	10	5	2	2	1

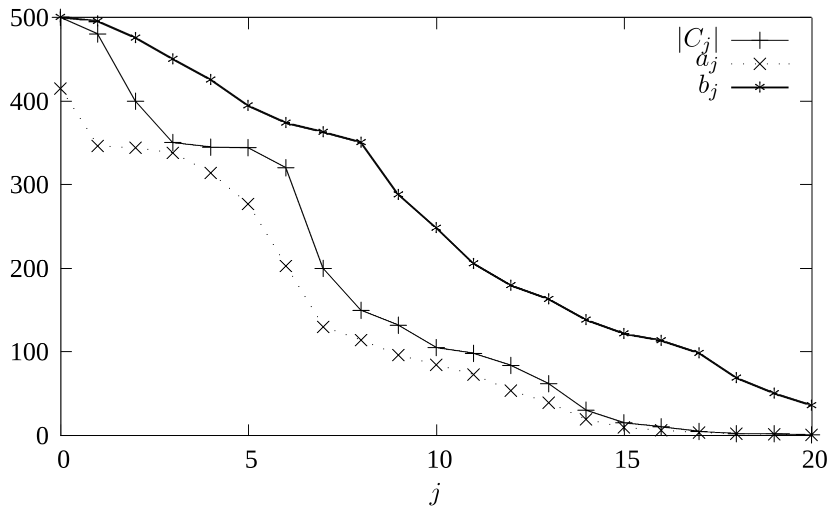

Figure 1: The *frequency distribution* of the stream S_2, the lower bounds a_j and upper bounds b_j of expectation of outputs of Algorithm1.

is not contained in $\widetilde{C}_j$ yet. After reading data stream, algorithm1 outputs $|\widetilde{C}_0|, \dots, |\widetilde{C}_s|$. The theorem 3.1 gives lower and upper bounds on the expectation of $|\widetilde{C}_j|$ in algorithm1.

Theorem 3.1. *For the expectation of $|\widetilde{C}_j|$ that algorithm1 outputs,*

$$a_j \le \mathrm{E}[|\widetilde{C}_j|] < b_j \tag{1}$$

is satisfied, where constant values a_j and b_j are

$$\begin{aligned} a_j &= \max_{0 \le \alpha < (s-j)} \{(1 - \exp(-2^\alpha) \cdot |C_{j+\alpha}|\}, \\ b_j &= \min_{0 \le \alpha \le j} \left\{ \left(1 - 1/4^{2^{-\alpha}}\right) \cdot |C_0| + 1/4^{2^{-\alpha}} |C_{j-\alpha}| \right\}. \end{aligned}$$

Table1 shows the *frequency distribution* of example stream S_2. The *frequency distribution* of stream S_2 is presented as graph in Figure1 with lower bounds a_j and upper bounds on the expectation of output in Algorithm1. Let α be 0 in Theorem 3.1, then the proposition is satisfied.

Proposition 3.2. *For the expectation of $|\widetilde{C}_j|$ that Algorithm1 outputs,*

$$0.63|C_j| \le \mathrm{E}[|\widetilde{C}_j|] < 0.75|C_0| + 0.25|C_j|. \tag{2}$$

3.2 Main Algorithm

Algorithm2 called *main algorithm* is based on algorithm1 and distinct counting algorithm by Bar-Yossef et al[2]. Algorithm1 memorizes all members of $|\widetilde{C}_j|$. To reduce memory space that algorithm1 uses, main algorithm uses the idea of algorithm by Bar-Yossef et al. An input of algorithm2 are a data stream $S =$

Algorithm 2 Main Algorithm

```
Require: data stream $S = e_1, \dots, e_N (e_i \in [1, m])$, parameter $t$
 1: Let $M$ be the smallest prime greater than $m^3$.
 2: Choose $\alpha, \beta$ from $[1, M]$ uniformly at random, respectively.
 3: for $i = 1$ to $N$ do //for each $i$-th element $e_i$
 4:   Flip a fair coin repeatedly until haed appears for the first time or coin is flipped $(s+1)$ times. Let $x$
      be the number of tails.
 5:   for $j = 0$ to $x$ do
 6:     With the hash value of $e_i$, $h(e_i) = (\alpha \cdot e_i + \beta \mod M) + 1$, update the list $L_j$ that maintains $t$
        smallest hash values.
 7:   end for
 8: end for
 9: for $j = 0$ to $s$ do
10:   if $L_j$.size $< t$ then
11:     $|\hat{C}_j| \leftarrow L_j$.size
12:   else
13:     $|\hat{C}_j| \leftarrow \frac{tM}{v_j}$ ($v_j$ is the largest hash value in $L_j$)
14:   end if
15: end for
16: return $|\hat{C}_0|, \dots, |\hat{C}_s|$
```

$e_1, \dots, e_N (e_i \in [1, m])$ and parameter t. We suppose that the length of the stream N and the range of elements m are already known, but these are very large. The theorem3.3 shows the probabilities that output of algorithm2 deviates from $(1-\epsilon)a_j$ and $(1+\epsilon)b_j$ are bounded by $1/\epsilon^2$ and t. In corollary3.4, if we adjust parameter t and run parallel algorithm2, then we get the estimated value of $|C_j|$ which is in $((1-\epsilon)a_j, (1+\epsilon)b_j)$ with high probability.

Theorem 3.3. *For every $0 < \epsilon < 1$, it holds that algorithm2 outputs $|\hat{C}_j|$ satisfying*

$$\Pr[|\hat{C}_j| > (1+\epsilon)b_j] < \frac{4}{\epsilon^2 t}, \tag{3}$$

$$\Pr[|\hat{C}_j| < (1-\epsilon)a_j] < \frac{2b_j}{\epsilon^2 t a_j} \tag{4}$$

, where a_j and b_j are

$$\begin{aligned} a_j &= \max_{0 \le \alpha < (s-j)} \{(1 - \exp(-2^{\alpha})) \cdot |C_{j+\alpha}|\}, \\ b_j &= \min_{0 \le \alpha \le j} \left\{ \left(1 - 1/4^{2^{-\alpha}}\right) \cdot |C_0| + 1/4^{2^{-\alpha}} |C_{j-\alpha}| \right\}. \end{aligned}$$

Corollary 3.4. *If set parameter $t = \frac{12}{\epsilon^2} \max(2, \frac{b_j}{a_j})$ in algorithm2, then*

$$\Pr[(1-\epsilon)a_j < |\hat{C}_j| < (1+\epsilon)b_j] \ge \frac{2}{3}. \tag{5}$$

If we implement algorithm2 $\lceil 15 \log_2(\delta^{-1}) \rceil$ times independently and let $|\widetilde{C}_j|$ be the median of outputs, then

$$\Pr[(1-\epsilon)a_j < |\hat{C}_j| < (1+\epsilon)b_j] \ge 1 - \delta. \tag{6}$$

We discuss space complexity. Algorithm2 uses $\mathrm{O}(\log m)$ bits memory space to memorize $\alpha, \beta \in [1, m^3]$ for hash function. In addition, this algorithm uses $s (= \mathrm{O}(\log N))$ lists that each list maintains t hash values. Hash value could be memorized in $\mathrm{O}(\log m)$ bits. Since we supposed $N < m$, the space complexity of algorithm2 is $\mathrm{O}(t \log^2 m)$. Assume that b_j/a_j is constant value, then implementing algorithm2 $\lceil 15 \log_2(\delta^{-1}) \rceil$ times uses $\mathrm{O}(\epsilon^{-2} \log \delta^{-1} \cdot \log^2 m)$ bits memory space, which holds (6).

4 Conclusions

In this paper, we proposed the algorithm to estimate the *frequency distribution* in small space and one pass. Furthermore, we presented theoretical analysis on this algorithm. An improvement of the algorithm for more precise estimation is a future work. Applying the *frequency distribution* to estimate the frequency moment is another work.

References

[1] N. Alon, Y. Matias and M. Szegedy, The space complexity of approximating the frequency moments, *Proceedings of the 28th annual ACM Symposium on Theory of Computing*(STOC 1996), pp.20–29.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, Counting distinct elements in a data stream, *Randomization and Approximation Techniques in Computer Science*(RANDOM 2002), pp.1–10.

[3] A. Chakrabarti, G. Cormode and A. Mcgregor, A near–optimal algorithm for estimating the entropy of a stream, *ACM Transactions on Algorithm*(TALG 2010), 6(3), 51.

[4] P. Flajolet and G. N. Martin, Probabilistic counting algorithms for data base applications, *Journal of computer and system sciences*(1985), 31(2), pp.182–209.

[5] P. B. Gibbons, Distinct sampling for highly-accurate answers to distinct values queries and event reports, *Proceedings of the 27th International Conference Very Large Data Bases*(VLDB 2001), pp.541–550.

[6] P. B. Gibbons, Distinct-values estimation over data streams, *Data Stream Management: Processing High-Speed Data Streams, (M. Garofalakis, J. Gehrke, and R. Rastogi, Eds.)*(2007), Springer.

[7] S. Guha, N. Koudas, and K. Shim, Approximation and streaming algorithms for histogram construction problems, *ACM Transactions on Database Systems*(TODS 2006), 31(1), pp.396–438.

[8] G. Grimmett and D. Stirzaker, Probability and random processes, *Oxford University Press*, 2001.

[9] P. Indyk, R. Levi, and R. Rubinfeld, Approximating and testing k-histogram distributions in sub-linear time, *Proceedings of the 31st symposium on Principles of Database Systems*(PODS 2012), pp.15–22.

[10] Y. Ioannidis, The history of histograms(abridged), *Proceedings of the 29th International Conference Very Large Data Bases*(VLDB 2003), 29, pp.19–30.

[11] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel, Optimal histograms with quality guarantees, *Proceedings of the 24th International Conference on Very Large Data Bases*(VLDB 1998), pp.275–286.

[12] D. M. Kane, J. Nelson, and D. P. Woodruff, An optimal algorithm for the distinct elements problem, *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*(PODS 2010), pp.41–52.

[13] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff, Fast moment estimation in data streams in optimal space, *Proceedings of the 43d annual ACM symposium on Theory of computing*(STOC 2011), pp.745–754.

[14] A. Lall, V. Sekar, M. Ogihara, J. Xu and H. Zhang, Data streaming algorithms for estimating entropy of network traffic, *Proceedings of the joint international conference on Measurement and modeling of computer systems*(SIGMETRICS 2006), pp.145–156.

[15] T. Locher, Finding heavy distinct hitters in data streams, *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures*(SPAA 2011), pp.299-308.

[16] G. S. Manku and R. Motwani, Approximate frequent counts over data streams, *Proceedings of the 28th International Conference on Very Large Data Bases*(VLDB 2002), pp.346–357.

[17] M. Mitzenmacher and E. Upfal, Probability and Computing, *Cambridge University Press*, 2005.

[18] R. Motwani and P. Raghavan, Randomized algorithm, *Chapman & Hall/CRC*, 2010.

[19] M. Ogata, Y. Yamauchi, S. kijima, and M. Yamashita, A randomized algorithm for finding frequent elements in streams using O(log log N) space, *Proceeding of the 22nd International Symposium on Algorithms and Computation*(ISAAC 2011), pp.514–523.

[20] R. Sebastiao, J. Gama, and T. Mendonca, Constructing fading histograms from data streams, *Progress in Artificial Intelligence*(2014) 3(1), pp15–28.

[21] S. Venkatamaran, D. Song, P.B. Gibbons, and A. Blum, New streaming algorithms for fast detection of superspreaders, *Proceedings of the 12th ISOC Symposium on Network and Distributed Systems Security*(NDSS 2005), pp.149-166.

A Supplemental Proofs

In this section, we prove theorem3.1, theorem3.3 and Corollary3.4.

A.1 Proof of theorem3.1

Firstly define some random variables on algorithm1. Next show lemmaA.1 and proof theorem3.1 using lemmaA.1.

Let $X_{j,e}$ be the indicator random variable whose value is 1 if and only if $\widetilde{C}_j$ contains e after reading all data stream in algorithm1. Since the probability that i-th element e_i arrives in $\widetilde{C}_j$ is $1/2^j$, the probability that element e whose frequency in the stream is n_e hasn't arrived in $\widetilde{C}_j$ is $(1-1/2^j)^{n_e}$. We denote the probability that $\widetilde{C}_j$ contains e by $p_{j,e}$. Then,

$$p_{j,e} = \Pr[X_{j,e} = 1] = 1 - (1 - 1/2^j)^{n_e}. \tag{7}$$

The expectation and variance of $X_{j,e}$ are

$$\mathrm{E}[X_{j,e}] = 1 \cdot p_{j,e} + 0 \cdot (1 - p_{j,e}) = p_{j,e},$$

$$\mathrm{Var}[X_{j,e}] = \mathrm{E}[X_{j,e}^2] - \mathrm{E}[X_{j,e}]^2 = p_{j,e} \cdot (1 - p_{j,e}).$$

Let a random variable X_j be the sum of $X_{j,e}$ for every $e \in [1, m]$, i.e, $X_j = \sum_{e=1}^{m} X_{j,e}$. Then X_j implies the number of distinct elements in $\widetilde{C}_j$. For every element $e \in [1, m]$, $X_{j,e}$ are independent. Thus the expectation and variance of X_j are

$$\mathrm{E}[X_j] = \sum_{e=1}^{m} \mathrm{E}[X_{j,e}] = \sum_{e=1}^{m} p_{j,e}, \tag{8}$$

$$\mathrm{Var}[X_j] = \sum_{e=1}^{m} \mathrm{Var}[X_{j,e}] = \sum_{e=1}^{m} p_{j,e} \cdot (1 - p_{j,e}). \tag{9}$$

Lemma A.1. *Let* e *be the base of the natural logarithm, integer* $j \geq 1$ *and* $\alpha \geq 0$*. For element* e *whose frequency is greater than or equal to* $2^{j+\alpha}$*, i.e,* $e \in C_{j+\alpha}$*,*

$$\begin{aligned} \Pr[X_{j,e} = 1] &\geq 1 - \exp(-2^{\alpha}), & (10)\\ \Pr[X_{j,e} = 0] &\leq \exp(-2^{\alpha}). & (11) \end{aligned}$$

For element e *whose frequency is less than* $2^{j-\alpha}$*, i.e,* $e \in E_S \setminus C_{j-\alpha}$*,*

$$\begin{aligned} \Pr[X_{j,e} = 1] &< 1 - 1/4^{2^{-\alpha}}, & (12)\\ \Pr[X_{j,e} = 0] &> 1/4^{2^{-\alpha}}. & (13) \end{aligned}$$

C_j is a set of elements whose frequency are greater than or equal to 2^j. Therefore we hope that $\widetilde{C}_j$ which is the estimated set of C_j doesn't contain some elements whose frequency are less than 2^j. Lemma A.1 implies the probability that the element e is contained $|\widetilde{C}_j|$ is getting higher as the frequency of the element e is greater. On the contrary, the element whose frequency is quite less than $2^{j-\alpha}$ doesn't be contained in $\widetilde{C}_j$ with high probability.

proof of lemma A.1. The probability that $\widetilde{C}_j$ contains element e whose frequency is n_e is $1 - (1 - 1/2^j)^{n_e}$ by equation (7). We have for element e whose frequency is greater than or equal to $2^{j+\alpha}$, i.e, $n_e \geq 2^{j+\alpha}$, that

$$\begin{aligned} \Pr[X_{j,e} = 1] &= 1 - \big(1 - \frac{1}{2^j}\big)^{n_e} \\ &\geq 1 - \big(1 - \frac{1}{2^j}\big)^{2^{j+\alpha}} \\ &= 1 - \big(\big(1 - \frac{1}{2^j}\big)^{2^j}\big)^{2^{\alpha}} \\ &\geq 1 - \mathrm{e}^{-2^{\alpha}} = 1 - \exp(-2^{\alpha}). & (14) \end{aligned}$$

Since $(1 - \frac{1}{2^j})^{2^j} \leq \mathrm{e}^{-1}$ for $j \geq 1$, (14) is obtained. By $\Pr[X_{j,e} = 0] = 1 - \Pr[X_{j,e} = 1]$, the probability that element e satisfying $n_e \geq 2^{j+\alpha}$ is not contained in $|\widetilde{C}_j|$ is less than or equal to $\exp(-2^{\alpha})$.

In the same way, we have for element whose frequency is less than $2^{j-\alpha}$ that

$$\begin{aligned} \Pr[X_{j,e} = 1] &= 1 - \big(1 - \frac{1}{2^j}\big)^{n_e} \\ &< 1 - \big(1 - \frac{1}{2^j}\big)^{2^{j-\alpha}} \\ &= 1 - \big(\big(1 - \frac{1}{2^j}\big)^{2^j}\big)^{2^{-\alpha}} \\ &< 1 - 1/4^{2^{-\alpha}}, & (15) \end{aligned}$$

where (15) is obtained from $(1-\frac{1}{2^j})^{2^j} \geq 1/4$ for $j \geq 1$. $\square$

Now, we will proof theorem 3.1 using lemma A.1.

proof of theorem 3.1. Let X_j be value that algorithm1 outputs. The idea of proof is to calculate expectation of X_j, $\mathrm{E}[X_j] = \sum_{e\in E_S} p_{j,e}$, by dividing into elements concluded in $C_{j+\alpha}$ and other elements.

$$\begin{aligned}
\mathrm{E}[X_j] &= \sum_{e\in E_S\setminus C_{j+\alpha}} p_{j,e} + \sum_{e\in C_{j+\alpha}} p_{j,e} \\
&\geq \sum_{e\in E_S\setminus C_{j+\alpha}} p_{j,e} + (1-\mathrm{e}^{-(\alpha+1)})\cdot|C_{j+\alpha}| && (16)\\
&\geq (1-\exp(-2^\alpha))\cdot|C_{j+\alpha}| && (17)
\end{aligned}$$

(16) is obtained by lemmaA.1 and (17) is obtained by $p_{j,e} \geq 0$. Since this inequality is satisfied for every integer $0 \leq \alpha < (s-j)$, $\mathrm{E}[X_j]$ is lower bounded by $\max_{0\leq\alpha<(s-j)}\{(1-\exp(-2^\alpha))\cdot|C_j|\}$.

In the same way, we have upper bound on $\mathrm{E}[X_j]$ that

$$\begin{aligned}
\mathrm{E}[X_j] &= \sum_{e\in E_S\setminus C_{j-\alpha}} p_{j,e} + \sum_{e\in C_{j-\alpha}} p_{j,e} \\
&< \left(1-1/4^{2^{-\alpha}}\right)\cdot(|C_0|-|C_{j-\alpha}|) + \sum_{e\in C_{j-\alpha}} p_{j,e} && (18)\\
&\leq \left(1-1/4^{2^{-\alpha}}\right)\cdot(|C_0|-|C_{j-\alpha}|) + |C_{j-\alpha}| \\
&= \left(1-1/4^{2^{-\alpha}}\right)\cdot|C_0| + 1/4^{2^{-\alpha}}|C_{j-\alpha}|, && (19)
\end{aligned}$$

where (18) follows from lemma A.1. We get minimum upper bound using (19) from $0 \leq \alpha \leq j$,

$$\mathrm{E}[X_j] < \min_{0\leq\alpha\leq j}\left\{\left(1-1/4^{2^{-\alpha}}\right)\cdot|C_0| + 1/4^{2^{-\alpha}}|C_{j-\alpha}|\right\}. \quad (20)$$

Furthermore, (20) could be expressed as follows,

$$\mathrm{E}[|C_0|-X_j] \geq \max_{0\leq\alpha\leq j}\left\{(|C_0|-|C_{j-\alpha}|)\cdot\left(\frac{1}{4}\right)^{2^{-\alpha}}\right\}. \quad (21)$$

$\square$

A.2 Proof of theorem3.3 and corollary 3.4

First, explain the detail of algorithm 2. Next, proof theorem3.3 and corollary 3.4.

The sampling method of algorithm2 is same with algorithm1's one. However, in order to use less memory space, algorithm2 estimates $|C_j|$ using the hash function h and the list L that maintains t smallest hash values. In other words, algorithm2 outputs $|\hat{C}_j|$, which is estimated values of $|\widetilde{C}_j|$, for estimating $|C_j|$.

We explain the hash function and the list. Hash function h is randomly chosen from the following hash family $\mathcal{H}$, where M is the smallest prime number greater than m^3.

$$\mathcal{H} = \{h_{\alpha,\beta}(e) : (\alpha,\beta)\in[1,M]\times[1,M], h_{\alpha,\beta}(e) = (\alpha\cdot e+\beta \mod M)+1\}$$

By choosing α and β from $[1,M]$ independently, hash function h maps element e in a stream to $[1,m]$ uniform at random. The hash family $\mathcal{H}$ satisfies pairwise independence. We use the list L_j to estimate $|\widetilde{C}_j|$. This list can memorize t hash values. We define $L_j.size$ is the number of hash values in the list.

In algorithm2, for each i-th element e_i, the algorithm updates(or dismisses) the t smallest hash value of e_i over the stream in L_j. In other words, if the hash value of e_i, $h(e_i)$, is smaller than the largest hash value in L_j, then remove the largest hash value in L_j and add $h(e_i)$ into L_j. Note that do not any process if same hash value is already existed in L_j. The case that different elements have same hash value occurs error. However we assume that we could dismiss this case because the probability that different elements have same hash values is $M^{-2}(\simeq m^{-6})$ abnormally small. After reading all data, the algorithm calculates $|\hat{C}_j|$ according to the size of L_j in step 9-15.

proof of theorem 3.3. First, show $\Pr[|\hat{C}_j| > (1+\epsilon)b_j] < \frac{4}{\epsilon^2 t}$.

$$\Pr[|\hat{C}_j| > (1+\epsilon)b_j] = \Pr[v_j < \frac{tm^3}{(1+\epsilon)b_j}] \tag{22}$$

$$< \Pr[v_j < (1-\frac{\epsilon}{2})\frac{tm^3}{b_j}] \tag{23}$$

, where (22) is obtained from definition of $|\hat{C}_j| = \frac{tm^3}{v_j}$, (23) is satisfied by $1/(1+\epsilon) < 1-\epsilon/2$ for $0 < \epsilon < 1$. Meanwhile, v_j is the t smallest value arrived in $\widetilde{C}_j$. The event v_j is smaller than some value α equals to the event more than t elements whose hash value are smaller than α have arrived at $\widetilde{C}_j$. Define a random variable $Y_{j,e}$ as

$$Y_{j,e} = \begin{cases} 1 & \text{if } (e \in \widetilde{C}_j \text{ and } h(e) < (1-\epsilon/2)\frac{tm^3}{b_j}), \\ 0 & \text{otherwise.} \end{cases}$$

Also define $Y_j = \sum_{e \in E_S} Y_{j,e}$. Then (23) is same with $\Pr[Y_j > t]$.

Since the event that element e arrives at $\widetilde{C}_j$ and the event that the hash value of element e is smaller than $(1-\epsilon/2)\frac{tm^3}{b_j}$ are independent, $\Pr[Y_{j,e} = 1]$ is calculated by $\Pr[e \in \widetilde{C}_j] \cdot \Pr[h(e) < (1-\epsilon/2)\frac{tm^3}{b}]$. The probability that $h(e)$ is less than $(1-\epsilon/2)\frac{tm^3}{b_j}$ is $(1-\epsilon/2)\frac{t}{b_j}$ by the definition of hash function. Since (8) and (20), $\sum_{e \in E_S} p_{j,e} < b_j$. For the expectation of Y_j,

$$\begin{aligned} \mathrm{E}[Y_j] &= \sum_{e \in E_S} \mathrm{E}[Y_{j,e}] = \sum_{e \in E_S} \Pr[Y_{j,e} = 1] \\ &= \sum_{e \in E_S} \Pr[e \in \widetilde{C}_j] \cdot \Pr[h(e) < (1-\epsilon/2)\frac{tm^3}{b_j})] \\ &= \sum_{e \in E_S} p_{j,e}(1-\epsilon/2)\frac{t}{b_j} \\ &= (1-\epsilon/2)\frac{t}{b_j} \sum_{e \in E_S} p_{j,e} \\ &< (1-\epsilon/2)t. \end{aligned} \tag{24}$$

For the variance of Y_j, by pairwise independence of hash function family,

$$\begin{aligned}
\mathrm{Var}[Y_j] &= \sum_{e\in E_S} \mathrm{Var}[Y_{j,e}] = \sum_{e\in E_S} \Pr[Y_{j,e}=1]\cdot(1-\Pr[Y_{j,e}=1]) \\
&= \sum_{e\in E_S} \big(p_{j,e}(1-\epsilon/2)\frac{t}{b_j}\big)\cdot\big(1-p_{j,e}(1-\epsilon/2)\frac{t}{b_j}\big) \\
&\le \sum_{e\in E_S} \big(p_{j,e}(1-\epsilon/2)\frac{t}{b_j}\big) \\
&= (1-\epsilon/2)\frac{t}{b_j}\sum_{e\in E_S} p_{j,e} \\
&\le (1-\epsilon/2)t \\
&< t. \qquad (25)
\end{aligned}$$

Since (24), $t-\mathrm{E}[Y_j] > \frac{\epsilon t}{2}$,

$$\begin{aligned}
\Pr[Y_i > t] &= \Pr[Y_j - E[Y_j] > t - E[Y_j]] \\
&\le \Pr[Y_j - E[Y_j] > \frac{\epsilon t}{2}] \qquad (26) \\
&\le \Pr[|Y_j - E[Y_j]| > \frac{\epsilon t}{2}] \\
&\le \frac{4\mathrm{Var}[Y_j]}{\epsilon^2 t^2} \\
&< \frac{4t}{\epsilon^2 t^2} \qquad (27) \\
&= \frac{4}{\epsilon^2 t}
\end{aligned}$$

where (27) is obtained by Chebyshev's inequality and (25). From the above, (3) is shown in theorem3.3.

Next, show (4),$\Pr[|\hat{C}_j| < (1-\epsilon)a_j] < \frac{2b_j}{\epsilon^2 t a_j}$, in theorem3.3.

$$\begin{aligned}
\Pr[|\hat{C}_j| < (1-\epsilon)a_j] &= \Pr[v_j > \frac{tm^3}{(1-\epsilon)a_j}] \\
&\le \Pr[v_j > (1+\epsilon)\frac{tm^3}{a_j}] \qquad (28)
\end{aligned}$$

For $0<\epsilon<1$, $\frac{1}{1-\epsilon} > (1+\epsilon)$, so (28) is satisfied.

The event that the t smallest hash value v_j greater than $(1+\epsilon)\frac{tm^3}{a_j}$ equals to the event that the number of hash value less than $(1+\epsilon)\frac{tm^3}{a_j}$ and arrives at $\widetilde{C}_j$ is less than t. Define a random variable $Z_{j,e}$ as follows,

$$Z_{j,e} = \begin{cases} 1 & \text{if } (e\in\widetilde{C}_j \text{ and } h(e) < (1+\epsilon)\frac{tm^3}{a_j}), \\ 0 & \text{otherwise}, \end{cases}$$

Also define $Z_j = \sum_{e\in E_S} Z_{j,e}$. Then,

$$\Pr[v_j > (1+\epsilon)\frac{tm^3}{a_j}] = \Pr[Z_j < t]. \qquad (29)$$

The expectation of Z_j is calculated by

$$\mathrm{E}[Z_j] = \sum_{e \in E_S} \mathrm{E}[Z_{j,e}] = \sum_{e \in E_S} \Pr[Z_{j,e} = 1].$$

Since $\Pr[Z_{j,e} = 1] = \Pr[e \in \widetilde{C}_j] \cdot \Pr[h(e) < (1+\epsilon)\frac{tm^3}{a_j}] = p_{j,e} \cdot \frac{(1+\epsilon)t}{a_j}$ と $a_j \leq \sum_{e \in E_S} p_{j,e} < b_j$, for the expectation of Z_j,

$$(1+\epsilon)t \leq \mathrm{E}[Z_j] < (1+\epsilon)\frac{tb_j}{a_j} \tag{30}$$

is satisfied. By (30),

$$\mathrm{E}[Z_j] - t \geq \epsilon t. \tag{31}$$

For the variance,

$$\begin{aligned}
\mathrm{Var}[Z_j] &= \mathrm{E}[Z_j^2] - \mathrm{E}[Z_j]^2 \\
&\leq \mathrm{E}[Z_j^2] \\
&= \mathrm{E}[Z_j] \\
&< (1+\epsilon)\frac{tb_j}{a_j} \\
&< \frac{2tb_j}{a_j}. \qquad (32)
\end{aligned}$$

Finally, we have that, with (31), (32), and Chebyshev's inequality,

$$\begin{aligned}
\Pr[Z_i < t] &\leq \Pr[|Z_j - \mathrm{E}[Z_j]| > \mathrm{E}[Z_j] - t] \\
&\leq \Pr[|Z_j - \mathrm{E}[Z_j]| > \epsilon t] \\
&\leq \frac{\mathrm{Var}[Z_j]}{\epsilon^2 t^2} \\
&\leq \frac{2tb_j}{\epsilon^2 t^2 a_j} \\
&= \frac{2b_j}{\epsilon^2 t a_j}. \qquad (33)
\end{aligned}$$

□

proof of corollary 3.4. If set $t = \frac{12}{\epsilon^2}\max(2, \frac{b_j}{a_j})$,

$$\begin{aligned}
\Pr[|\hat{C}_j| > (1+\epsilon)b_j] &< \frac{4}{\epsilon^2 t} < \frac{1}{6} \qquad (34) \\
\Pr[|\hat{C}_j| < (1-\epsilon)a_j] &< \frac{2b_j}{\epsilon^2 t a_j} < \frac{1}{6} \qquad (35)
\end{aligned}$$

are obtained by (3),(4).

Therefore, $\Pr[(1-\epsilon)a_j < |\hat{C}_j| < (1+\epsilon)b_j] \geq 2/3$. Let r be $\lceil 15\log_2(1/\delta) \rceil$. Implement algorithm2 r times and get the median of outputs, then $\Pr[(1-\epsilon)a < |\hat{C}_j| < (1+\epsilon)b] \geq 1-\delta$ is satisfied by Chernoff bounds. □