Estimation of Non-linear Function of the Frequency in A Pairwise Data Stream

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

Abstract

Analizying large data such as a network traffic becomes more important for its applications to proactive response from cyber attacks. Because of their large data sizes, it is quite difficult to preserve all data or to query on them. Therefore, streaming algorithms are effective to analyze big data, such algorithms which use small memory space and picks up only useful information from data.

This paper is concerned with streams composed of pairwise data (e, v). We present an algorithm which estimates the sum of function values of frequency of pair (e, v) for each element, given a function f and a pairwise data stream S. We also show some theoretical analysis of the algorithm concerning memory space for specific functions $f(n) = n \log n$ and $f(n) = \log n$.

Keywords: Streaming Algorithm, Frequent Moment, Heavy Distinct Hitter.

1 Introduction

These days, data processing for large data such as network traffic and sensor data is increasingly 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, to preserve all data generated rapidly needs to huge memory space and it takes long time. Therefore streaming algorithms, which use small memory space and pick up only useful information from a data stream, is effective to computing on the large data.

This paper is concerned with a streaming algorithm about a pairwise data stream and function values of frequency. There are many pairwise data streams. For example, network traffic is a stream consisting of (Source IP Address, Destination IP Address). One problem about a pairwise data stream is *Heavy Distinct Hitters*[4][5]. This problem is to find element x that appears with a lot of distinct y, given a pairwise stream consisting of (x, y). Streaming algorithms to find *Heavy Distinct Hitters* can be applied to detecting infected machines that try to spread out a network warm, because some Source IP Address that send packets to many distinct Destination IP Address unusually correspond to the infected machines on the network traffic.

The general problem about calculating function values of frequency is the *frequency moments*. Alon et al. [1] presented a streaming algorithm to approximate the *frequency moment* $F_p = \sum_{i=1}^{m} n_i^p$, where input data stream is $x_1, ..., x_N (x_i \in \{1, ..., m\})$, let n_i be the frequency of i on the stream and for any $k \geq 0$. Another work about calculating function values of frequency is to compute the *entropy*[2][3]. The *entropy* of stream $S = x_1, ..., x_N (x_i \in \{1, ..., m\})$ is defined by $H(S) = \sum_{i=1}^{m} -\frac{n_i}{N} \log \frac{n_i}{N} = logN - \frac{1}{N} \sum_{i=1}^{m} n_i \log n_i$. When we compute the *entropy* of the stream S, it is important to calculate $\sum_{i=1}^{m} n_i \log n_i$. Lall et al. propose a streaming algorithm [3] that computes the *entropy* of the stream, which is based on

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

Alon's algorithm. These algorithms deal with a stream consisting of single elements. Our purpose is to extend estimating function values of frequency to a pairwise stream.

Out Results

In this paper, we present a streaming algorithm to estimate $g(e) = \sum_{v:(e,v)\in S} f(n_{e,v})$ which maens the sum of function values of the frequency of (e, v) for each element, given a pairwise stream data S and a function $f : \mathcal{R} \to \mathcal{R}$. This algorithm is based on Alon's algorithm and reservoir sampling. Section 2 briefly reviews Alon's algorithm for *frequency moment*. Section 3 we define our problem. Section 4 our proposed algorithm is given. Finally, we show some theoretical analysis in Section 5.

2 Preliminary

We explain about Alon's algorithm[1] to estimate the *Frequency Moment*. Input data is a stream S of a single element $e_i : S = e_1, ..., e_N (e_i \in E)$. The purpose of this algorithm is to estimate $F_p := \sum_{e\in E_S} n_e^p$, where E_S is a set of element e in the stream S and n_e is the number of occurrences of element e. Alon's algorithm is briefly described in Algorithm 1. Before staring to read stream, the algorithm chooses α from $[1, ..., N]$ uniformly at random. Then, read stream and count r that is the number of elements which have the same value as e_α from αth data to final one in the stream.

Algorithm 1 Alon's algorithm for estimating F_p

```
Input: data stream $S = e_1, ..., e_N (e_i \in E)$
1: choose $\alpha$ from $[1, ..., N]$ uniformly at random.
2: $r \leftarrow |\{e_j : \alpha \le j \le N, e_j = e_\alpha\}|$
3: return $X = N(r^p - (r-1)^p)$
```

Algorithm 1 satisfies as following theorem, so we can get a nice estimated value of $g(e)$ by using multiple Algorithm 1 independently.

Theorem 2.1. *[1] Algorithm 1 outputs X which satisfies* $\mathrm{E}[X] = g(e)$ and $\mathrm{Var}[X] \le pF_1F_{2p-1}$.

3 Problem Definition

In this section, we define our problem. Let E and V be a finite set of items, respectively. An input stream S consists of a pair of element–value respectively in E and V, i.e., $S = (e_1, v_1), (e_2, v_2), ..., (e_N, v_N)$. N is the length of stream S. We define E_S be a set consist of elements that observed in the stream S and its size is called by k, i.e., $k := |E_S|$. Let m be the number of distinct pairs in the stream. We denote the frequency of a pair of element–value (e, v) by $n_{e,v} := |\{i \in \{1, .., N\} : (e, v) = (e_i, v_i)\}|$ and the frequency of element e by $n_e := |\{i \in \{1, ..., N\} : e \in E_S, e_i = e\}|$. Note that $N = \sum_{(e,v)\in S} n_{e,v} = \sum_{e\in E_S} n_e$.

For example, consider a stream $S_1 = (A, b), (B, b), (C, a), (A, b), (B, b), (A, c), (A, b)$. The length of this stream is $N = 7$ and the set of elements observed in this stream is $E_{S_1} = \{A, B, C\}$. In this stream, there are four distinct pairs, $m = 4$, and each of the frequency of them are $n_{A,b} = 3, n_{A,c} = 1, n_{B,b} = 2, n_{C,a}$=1.

In this paper, our goal is to output a set $\Pi := \{(e, g(e)) : e \in E_S\}$, given a pair-wise Stream S and a function f. A function $g : E \to \mathcal{R}$ is defined by $g(e) = \sum_{v:(e,v)\in S} f(n_{e,v})$, which means that $g(e)$ is the sum of function values of the frequency of (e, v) pairs in the stream S. For our example stream S_1, element A appears with b and c, so $g(A)$ is calculated by $f(n_{A,b}) + f(n_{A,c}) = f(3) + f(1)$. If we define f as follows:

$$f(n) = \begin{cases} 1 & (n \ge 1) \\ 0 & (\text{else}), \end{cases} \tag{1}$$

then $g(e)$ is the count how many distinct values v appear with e. Under this definition of f, to compute a set Π is the same with to find *Heavy Distinct Hitters*.

One simple method to calculate $g(e)$ exactly for every element e is to memorize each frequency of the pair(e, v) in the stream. However it needs $O(m \log N)$bit memory space, where m is the number of distinct pair (e, v). We suppose that we cannot reserve all frequency of pairs, because m is very large. Under this condition, we define the following problem.

Definition 3.1. Estimation Function value of Frequency in a Pairwise stream Problem. Given a pair-wise stream S, a function $f : \mathcal{R} \to \mathcal{R}$ and parameters $\epsilon, \delta \in (0, 1)$, output a set L of pairs $(e, \hat{g}(e))$ for which it holds that

(i) Every element e in E_S is included in $\mathcal{L}$.

(ii) For every $(e, \hat{g}(e)) \in \mathcal{L}$, the probability that $\hat{g}(e)$ deviates from $g(e)$ by more than $\epsilon g(e)$ is at most δ, i.e., $\Pr[|\hat{g}(e) - g(e)| \geq \epsilon g(e)] \leq \delta$.

We called this problem as *EFFP* problem. Condition (ii) is called (ϵ, δ)-approximation.

4 Algorithm

Algorithm 2 An algorithm to solve *EFFP* problem.

```
Input: data stream $S = (e_i, v_i)_{i=1}^N \in (E \times V)^N$ and function $f : \mathcal{R} \to \mathcal{R}$.
1: initialize $\mathcal{W} \leftarrow \emptyset$, and $i \leftarrow 1$
2: for $i = 1$ to $N$ do
3:     increment a counter of $e_i$ by one in $\mathcal{W}$
4:     $keepCount(e_i, v_i)$
5: end for
6: For every $e$ in $\mathcal{W}$, $\hat{g}(e) \leftarrow getEstimate(e)$
7: return $\{(e, \hat{g}(e)) | e \in \mathcal{W}\}$
```

Algorithm 3 $keepCount(e, v)$

```
1: for $i = 1$ to $s$ do
2:     for $j = 1$ to $z$ do
3:         choose $\alpha \in [0, 1)$ uniformly
4:         if $\alpha \leq \frac{1}{\mathcal{W}(e)}$ then
5:             $\mathcal{D}_{ij}(e) \leftarrow (v, 1)$
6:         else
7:             if $\mathcal{D}_{ij}(e).v = v$ then
8:                 $\mathcal{D}_{ij}(e).r \leftarrow \mathcal{D}_{ij}(e).r + 1$
9:             end if
10:        end if
11:    end for
12: end for
```

Now, we present a simple algorithm to solve *EFFP* problem. This algorithm is based on Alon's algorithm for the *frequency moment*[1] and Reservoir Sampling. Our algorithm is described in Algorithm 2.

Algorithm 4 $getEstimate(e)$: return the estimated value of $g(e)$

```
1: for $i = 1$ to $s$ do
2:     for $j = 1$ to $z$ do
3:         $X_{ij}(e) \leftarrow n_e\big(f(D_{ij}(e).r) - f(D_{ij}(e).r - 1)\big)$
4:     end for
5:     $X_i(e) \leftarrow \frac{1}{z}\sum_{j=1}^{z} X_{ij}(e)$
6: end for
7: return $median(X_1(e), ..., X_s(e))$
```

Given a stream S and a function f, Algorithm 2 outputs a set of elements in stream and the estimated value of function $g(e)$ for each of the elements, where the real value $g(e)$ is defined by $g(e) := \sum_{v:(e,v)\in S} f(n_{e,v})$. The difference between the real value $g(e)$ and the estimated value $\hat{g}(e)$ is constrained stochastically by modifying parameters s and r, which are related to memory space. In other word, we need to adjust s and z in order to satisfy $\Pr[|\hat{g}(e) - g(e)| \geq \epsilon g(e)] \leq \delta$ for every element in stream.

We will describe data structures, $\mathcal{W}$ and $\mathcal{D}_{ij}(e)$. Let $\mathcal{W}$ be memory space to preserve elements that appear in the stream and its frequency. When $\mathcal{W}(e)$ is called in the algorithm, it returns the frequency of the element e at the time. After finishing to read the stream, the value of $\mathcal{W}(e)$ is the same as n_e, which is frequency of element e over the stream.

Denote $\mathcal{D}_{ij}(e)$ be memory space to record (v, r) for each element e. This v corresponds to the value of e_α chosen by uniformly at random in Alon's algorithm. r is counter to record the number of occurrences of (e, v) from the stream when (e, v) is chosen to end of the stream. When $\mathcal{D}_{ij}(e).v$ and $\mathcal{D}_{ij}(e).r$ is called in the algorithm, it returns the value v and counter r respectively. $1 \leq i \leq s$ and $1 \leq j \leq z$ are index numbers, so the number of $\mathcal{D}_{ij}(e)$ used in Algorithm 2 is $s \cdot z$ for each element.

Algorithm 2 reads (e, v) one by one, and records element e and updates its frequency on $\mathcal{W}$. Next, the algorithm implements $keepCount(e, v)$, which renews (v, r) in $\mathcal{D}_{ij}(e)$. Step 3-5 in $keepCount(e, v)$ shows process to reset (e, v) by probability $1/\mathcal{W}(e)$, which is Reservoir Sampling. After reading all stream, the algorithm computes $\hat{g}(e)$ by using $getEstimate(e)$.

5 Analysis of Our Algorithm

Algorithm 2 satisfies as follows.

Theorem 5.1. *If a function f satisfies $f(0) = 0$ and $\forall x, y \in \mathcal{R}, x \leq y \rightarrow f(x) \leq f(y)$, then for every $e \in E_S$,* $\mathrm{E}[X_{ij}(e)] = g(e)$, $\mathrm{Var}[X_{ij}(e)] \leq n_e \sum_{v:(e,v)\in S} f(n_{e,v})^2$.

Proof of Theorem 5.1. $X_{ij}(e)$ is defined by $X_{ij}(e) := n_e(f(D_{ij}(e).r) - f(D_{ij}(e).r - 1))$. We abbreviate $D_{ij}(e).r$ to r for convenience.

$$\begin{aligned} E[X_{ij}(e)] &= E[n_e(f(r) - f(r-1))] \\ &= n_e E[f(r) - f(r-1)] \\ &= \frac{n_e}{n_e} \sum_{v:(e,v)\in S} \sum_{p=1}^{n_{e,v}} (f(p) - f(p-1)) \\ &= \sum_{v:(e,v)\in S} f(n_{e,v}) = g(e) \end{aligned}$$

$$\begin{aligned}
Var[X_{ij}(e)] \leq E[X_{ij}(e)^2] &= E[n_e^2(f(r)-f(r-1))^2] \\
&= n_e^2 E[(f(r)-f(r-1))^2] \\
&= \frac{n_e^2}{n_e} \sum_{v:(e,v)\in S} \sum_{p=1}^{n_{e,v}} (f(p)-f(p-1))^2 \\
&\leq n_e \sum_{v:(e,v)\in S} f(n_{e,v})^2
\end{aligned}$$

□

Corollary 5.2. *For each element $e \in E_S$, if $s := \lceil 2\log(\frac{1}{\delta})\rceil, z := \lceil \frac{8n_e \sum_{v:(e,v)\in S} f(n_{e,v})^2}{(\epsilon g(e))^2}\rceil$, then Algorithm 2 holds that $\Pr[|g(e)-\hat{g}(e)| \geq \epsilon g(e)] \leq \delta$.*

A sketch of proof of Corollary 5.2. The algorithm gets $\hat{g}(e)$ by $getEstimate(e)$, $\hat{g}(e) := median(X_1(e), ..., X_s(e))$ and each $X_i(e)$ is calculated by $\frac{1}{z}\sum_{j=1}^{z} X_{ij}(e)$. According to Theorem 5.1, $\mathrm{E}[X_i(e)] = g(e)$ and $\mathrm{Var}[X_i(e)] \leq \frac{n_e}{s}\sum_{v:(e,v)\in S} f(n_{e,v})^2$. If we set $z := \lceil \frac{8n_e \sum_{v:(e,v)\in S} f(n_{e,v})^2}{(\epsilon g(e))^2}\rceil$, $\Pr[|X_i(e)-g(e)| \geq \epsilon g(e)] \leq \frac{1}{8}$ is satisfied by Chebyshev's inequality. Next, using $s := \lceil 2\log(\frac{1}{\delta})\rceil$ and Chernoff bound, we can prove $\Pr[|g(e)-\hat{g}(e)| \geq \epsilon g(e)] \leq \delta$. □

Corollary 5.2 shows the required value of s and r to (ϵ,δ)-approximation for each element e. If we assume that we can memorize labels of element, value and frequency by using only constant bits, space complexity of Algorithm 2 refers to (size of $\mathcal{W}$) + (the number of all $D_{ij}(e)$). The following propositions show space complexity when *EFFP* problem for specific functions $f(n) = n\log n$ and $f(n) = \log n$ are solved by Algorithm 2.

Proposition 5.3. *Algorithm 2 solves EFFP problem for $f(n) = n\log n$ using $O(k\log N)$ memory space.*

A sketch of proof Proposition 5.3. When $f(n)$ is $n\log n$, it is satisfied that $\mathrm{E}[X_{ij}(e)] = g(e)$ and $\mathrm{Var}[X_{ij}(e)] \leq 4n_e \log n_e g(e)$. By the same method with Corollary 5.2, if we choose $z := \lceil\frac{32\log N}{\epsilon^2}\rceil$ and $s := \lceil 2\log(\frac{1}{\delta})\rceil$, Algorithm 2 holds (ϵ,δ)-approximation for every element in stream. At this time, the algorithm uses memory $O(k)$ for $\mathcal{W}$ and $O(k\cdot s\cdot z) = O(\frac{\log\delta^{-1}}{\epsilon^2}k\log N)$ for all $D_{ij}(e)$. Therefore memory complexity of the algorithm is $O(k\log N)$. □

Proposition 5.4. *Algorithm 2 solves EFFP problem for $f(n) = \log n$ using $O(k\max\limits_{e\in E_S}\frac{n_e}{g(e)})$ memory space.*

A sketch of proof Proposition 5.4. When $f(n)$ is $\log n$, $\mathrm{E}[X_{ij}(e)]$ is the same as $g(e)$, but $\mathrm{Var}[X_{ij}(e)] \leq n_e g(e)$ is different from the equality at $f(n) = n\log n$. In order to satisfy (ϵ,δ)-approximation, set $z := \lceil\frac{8}{\epsilon^2}\max\limits_{e\in E_S}\frac{n_e}{g(e)}\rceil$, which is shown in a similar way to Proposition 5.3. □

6 Conclusions

We proposed the algorithm to estimate $g(e)$, which is the sum of function values of frequency for each element e as a pairwise data stream and function f are given. We presented theoretical analysis on this problem. The proposed algorithm can be applied to find cyber attack such as network warms and DDoS attack from the network traffic. Our future work is to find some element e whose value of $g(e)$ is greater than a threshold value.

References

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

[2] A.Chakrabarti, G.Cormode and A.Mcgregor, A near–optimal algorithm for estimating the entropy of a stream, ACM Transactions on Algorithm, 6(2010), No51.

[3] 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.

[4] 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.

[5] 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.