Two Improved Range-Efficient Algorithms for F_0 Estimation*

He Sun[1,2] and Chung Keung Poon[1]

[1] Department of Computer Science, City University of Hong Kong
Hong Kong, China
[2] Department of Computer Science and Engineering, Fudan University
Shanghai, China

Abstract. We present two new algorithms for range-efficient F_0 estimating problem and improve the previously best known result, proposed by Pavan and Tirthapura in [15]. Furthermore, these algorithms presented in our paper also improve the previously best known result for Max-Dominance Norm Problem.

1 Introduction

Problem. Let $S = r_1, \cdots, r_n$ be a sequence of intervals where each interval $r_i = [x_i, y_i] \subseteq [1, U]$ is an interval of integers between x_i and y_i. Let $m_j = ||\{i | j \in r_i\}||$ denote the number of intervals in the sequence S that contains j. Then the *kth-frequency moment* of S is defined as $F_k = \sum_{i=1}^{U} m_i^k$. In practice, the *zeroth-frequency moment* of S is the number of distinct elements in $\cup_{i=1}^{n} r_i$.

In this paper, we consider the problem of estimating F_0 in the above data stream model. Let $\varepsilon, \delta > 0$ be two constants. An algorithm $\mathscr{A}$ is said to (ε, δ)-approximate F_0 if the output Z of the algorithm $\mathscr{A}$ satisfies $\Pr[|F_0 - Z| > \varepsilon F_0] < \delta$. The time and space complexity of algorithm $\mathscr{A}$ are functions of the domain size U, the approximation parameter ε and the confidence parameter δ. In practice, the number of intervals n and the size of the universe U are very large. So we seek for algorithms that run quickly using relatively small space. In particular, the time for processing each interval should be sublinear with the length of the interval. We call such algorithms *range-efficient.*

Motivation. The cardinality of a database or data stream is of great importance in itself. In databases, some operations (such as query optimization) require knowledge of the *cardinality*—the number of distinct items—of a specific column in a database. Since commercial databases are usually very large, we can afford to scan each item only once and use limited space to give a desired approximation of F_0. Another application arises from routing of Internet traffic. In this scenario, the router usually has very limited memory and needs to gather

* The work described in this paper was fully supported by a grant from CityU (SRG 7001969).

various statistical properties of the traffic flow. For instance, the number of distinct destination IP addresses in a specific period is a critical property for the router to analyze the behavior of the Internet users. This motivates the *single item case* of the problem, i.e., the estimation of F_0 in a data stream model where each item of the input is a single integer (instead of intervals).

Bar-Yossef et al. [2] formalize the concept of reductions between algorithms for data streams and motivate the concept of list-efficient streaming algorithms which includes range-efficient F_0 estimation as a special case. Through reductions, a range-efficient F_0 algorithm can solve the problem of estimating the number of distinct triangles in graphs. Pavan and Tirthapura [15] also pointed out the relationship between range-efficient F_k estimation algorithm and the Max-Dominance Norm Problem. Though there are other algorithms for the problem that rely on stable distributions and Nisan's pseudorandom generators [6], the solution based on the problem that we focus on is more elegant and has smaller running time.

Related works. In the past twenty years, most research focuses on the single item case. Flajolet and Martin [9] gave the first algorithm for estimating F_0 for this case. The drawback of their algorithm is that they require a perfect hash function to make the input data uniform and independent. In 1999, Alon et al. [1] gave several algorithms for estimating $F_k, k \geq 0$, and used pairwise independent hash functions to get a constant factor approximation F_0 algorithm with space complexity $O(\log U)$. In 2002, Bar-Yossef et al. [2] gave the first algorithm for estimating the number of distinct elements in a data stream that approximates with arbitrarily small relative error. Since then, several approximation schemes have been proposed such as the `Loglog Counting` algorithm [8,11], algorithm using stable distributions [5], and algorithms based on sampling technique [3].

Unfortunately, applying these algorithms on our problem results in an update time proportional to the product between the length of the interval and the running time for updating one item. To overcome this drawback, Bar-Yossef et al. [2] designed two range-efficient approximation algorithms for F_0 and F_2 estimation, which are, to our best knowledge, the first efficient approximation scheme for this kind of problems. In 2005, Pavan and Tirthapura [15] improved the F_0 estimation algorithm of Bar-Yossef et al. and reduced the processing time per item from $O(\frac{1}{\varepsilon^5} \log^5 U \log \frac{1}{\delta})$ to $O(\log \frac{U}{\varepsilon} \log \frac{1}{\delta})$. However the worst case update time per element could be as much as $O(\frac{\log^2 U}{\varepsilon^2} \log \frac{1}{\delta})$.

Results. (1). We give two algorithms with different amortized running time and worst case running time updating each interval for approximating F_0, whose space complexity is the same as the algorithm in [15]. The following table summarizes our results and gives the comparison between our results and the previously best known algorithm proposed in [15]. The $\tilde{O}$ notation suppresses $\log \log U$ factors. (2). We improve the previously best known result for Max-Dominance Norm Problem and reduce the worst case update time from $\tilde{O}(\frac{1}{\varepsilon^2} \log a_{i,j} \log \frac{1}{\delta})$ to $\tilde{O}((\frac{1}{\varepsilon^2} + \log a_{i,j}) \log \frac{1}{\delta})$.

2 Preliminaries

Hash functions. A k-universal family of hash functions is a set $\mathscr{H}$ of functions $A \mapsto B$ such that for all distinct $x_1, \cdots, x_k \in A$ and all (not necessarily distinct) $b_1, \cdots, b_k \in B$

$$\Pr_{h \in \mathscr{H}}[h(x_1) = b_1 \wedge \cdots \wedge h(x_k) = b_k] = |B|^{-k}$$

Carter and Wegman's original definition [4] is different from the above, which is what they call *strongly k-universal* hash functions [16].

Here we describe a hash function used in our improved algorithm. First choose a prime number p between U^2 and U^3, and pick a from the set $\{1, \cdots, p-1\}$ and b from $\{0, \cdots, p-1\}$ randomly. Let $h(x) = (a \cdot x + b) \bmod p$. It is well known that $h(x)$ is a pairwise independent hash function. Let $\rho(x)$ be the number of consecutive 0's from the rightmost in x's binary expression. For instance, $\rho(2) = 1$ and $\rho(7) = 0$. In addition, define $\rho(0) = \lceil \log p \rceil$. The following lemma gives the pairwise independence of the hash function $\rho(h(x))$.

Lemma 1. *The random variables $\{\rho((ax + b) \ mod \ p) | a \in \{1, \cdots, p-1\}, b \in \{0, \cdots, p-1\}\}$ are pairwise independent.*

Proof. Since $h(x) = (ax + b) \bmod p$ is a pairwise independent hash function, for any $x \neq y$ and $\alpha, \beta \in \{0, \cdots, p-1\}$, there holds

$$\Pr_{a,b}[h(x) = \alpha \wedge h(y) = \beta] = \Pr_{a,b}[h(x) = \alpha] \cdot \Pr_{a,b}[h(y) = \beta] = \frac{1}{p^2}$$

For all $x \neq y \in \{0, \cdots, p-1\}$ and $i, j \in \{0, \cdots, \lceil \log p \rceil\}$,

$$\begin{aligned}
&\Pr_{a,b}[\rho(h(x)) = i \wedge \rho(h(y)) = j] \\
=&\sum_{\alpha=0}^{p-1} \sum_{\beta=0}^{p-1} \Pr_{a,b}\left[h(x) = \alpha \wedge h(y) = \beta\right] \cdot \Pr\left[\rho(h(x)) = i \wedge \rho(h(y)) = j \middle| h(x) = \alpha \wedge h(y) = \beta\right] \\
=&\sum_{\alpha=0}^{p-1} \sum_{\beta=0}^{p-1} \Pr_{a,b}\left[h(x) = \alpha \wedge h(y) = \beta\right] \cdot \Pr\left[\rho(\alpha) = i \wedge \rho(\beta) = j\right] \\
=&\frac{1}{p^2} \sum_{\alpha=0}^{p-1} \sum_{\beta=0}^{p-1} \Pr[\rho(\alpha) = i] \cdot \Pr[\rho(\beta) = j] \\
=&\frac{1}{p^2} \frac{p}{2^{i+1}} \frac{p}{2^{j+1}} \\
=&\Pr[\rho(h(x)) = i] \cdot \Pr[\rho(h(y)) = j]
\end{aligned} \tag{1}$$

In summary, the random variables $\{\rho((ax + b) \bmod p) | a \in \{1, \cdots, p-1\}, b \in \{0, \cdots, p-1\}\}$ are pairwise independent. □

Table 1. Comparison of time complexity for range-efficient F_0 estimating algorithm

Algorithm	Worst case update time	Amortized update time
Algorithm in [15]	$O(\frac{1}{\varepsilon^2}\log^2 U \log\frac{1}{\delta})$	$O(\log\frac{U}{\varepsilon}\log\frac{1}{\delta})$
Our algorithm	$\tilde{O}((\frac{1}{\varepsilon^2}+\log U)\log\frac{1}{\delta})$	$\tilde{O}(\log\frac{U}{\varepsilon}\log\frac{1}{\delta})$
Our algorithm[revised]	$O(\log\frac{1}{\varepsilon}\log U\log\frac{1}{\delta})$	$O(\log\frac{1}{\varepsilon}\log U\log\frac{1}{\delta})$

3 Algorithm for Range-Efficient F_0 Estimation

We first give the high level overview of our approach. Like [3,15], our algorithm maintains a current sampling level ℓ. Initially, $\ell = 0$. We use a set S, whose size is $\alpha = \Theta(\frac{1}{\varepsilon^2})$, to store the sampled intervals. When a new interval r comes, the algorithm checks whether or not r intersects with any existing interval r' in S. If there exists such interval r', let $r \leftarrow r' \cup r$. Then we calculate $M(r)$ and $G(r)$, where $M(r) := \max_{x\in r}\rho(h(x))$ and $G(r) := ||\{x \in r|\rho(h(x)) = M(r)\}||$. In other words, $M(r)$ is the highest level achieved by the elements in the interval r and $G(r)$ is the number of elements attaining this level.

If $M(r) \geq \ell$, then we put the interval r into S. When the number of intervals in S exceeds α, the level ℓ increases and the algorithm deletes the intervals whose $M(\cdot)$-value is less than ℓ. Finally, the estimated value of F_0 is

$$Z = \Big(\sum_{i=\ell}^{\lfloor\log p\rfloor} X_i \cdot 2^{i+1}\Big)\cdot 2^{\ell} \tag{2}$$

where

$$X_i = \sum_{r\in S\wedge M(r)=i} G(r) \tag{3}$$

Calculating $M(r)$ and $G(r)$. For the given interval $r = [x, y]$ and hash function $h(x) = (a \cdot x + b) \bmod p$, where p is a prime number, we design an efficient algorithm to calculate $M(r)$ and $G(r)$.

For this problem, a naive solution to get $M(r)$ is to calculate $\rho(h(z))$ for each $z \in [x, y]$, and get the maximum value of them. The time complexity is $O(y - x + 1)$, which could be as much as $\Theta(U)$. In this paper, we reduce the processing time per interval to $O(\log U \log\log U)$.

Consider the following problem: Given the sequence $u, (u+d) \bmod p, \cdots, (u+t\cdot d) \bmod p$, we want to find the maximum integer i, $i \in \{0, \cdots, \lceil\log p\rceil\}$, such that there exists an integer $x \in \{0, \cdots, t\}$ satisfying the following equation

$$(u + x\cdot d) \bmod p \equiv 0 \pmod{2^i} \tag{4}$$

It is obvious to see the equivalence of calculating $M(\cdot)$ and the above problem by putting $u = h(x)$, $d = a$ and $t = y - x$.

For any fixed i, Equation (4) is equivalent to $u+x\cdot d \equiv v\cdot 2^i \pmod p$, for some $v \in \{0, \cdots, p-1\}$. Therefore $d\cdot x \equiv v\cdot 2^i - u \pmod p$. Since p is a prime number, $(d, p) = 1$ and the solution of the congruence equation $d\cdot x \equiv v\cdot 2^i - u \pmod p$ is

$$\begin{aligned} x &\equiv d^{\phi(p)-1} \cdot (v \cdot 2^i - u) \bmod p \\ &= d^{-1} \cdot (v \cdot 2^i - u) \bmod p \end{aligned} \tag{5}$$

where $\phi(\cdot)$ is the Euler function.

We can express x as

$$x \equiv (-u \cdot d^{-1} + v \cdot 2^i \cdot d^{-1}) \bmod p \tag{6}$$

Thus we can use the procedure Hits, described in [15], to determine the size of the intersection of the set $\{0, \cdots, t\}$ and the sequence

$$u' \bmod p, (u' + d') \bmod p, \cdots, (u' + (p-1) \cdot d') \bmod p$$

where $u' = -u \cdot d^{-1}$ and $d' = 2^i \cdot d^{-1}$.

We now describe our algorithm, MG, for computing $M(\cdot)$ and $G(\cdot)$. For the given hash function $h(\cdot)$, integers d, p and interval $r = [x, y]$, set $u' \leftarrow -h(x) \cdot d^{-1} (\bmod p)$ first. Then the algorithm uses the binary search to determine the maximum $i \in \{0, \cdots, \lceil \log p \rceil\}$ such that $v := \texttt{Hits}(p, 2^i \cdot d^{-1}, u', p-1, [0, y-x]) > 0$. Finally, the algorithm outputs i and v as the value of $M(r)$ and $G(r)$.

The formal description of the procedure Hits, which calculates the size of the intersection between a given interval and an arithmetic progression over $\mathbb{Z}_p$, can be found in [15].

Theorem 1. *The time complexity of algorithm* MG *is* $O(\log U \log\log U)$ *and the space complexity is* $O(\log U)$.

Proof. Since the maximum value of i is $\lceil \log p \rceil$ and we use binary search to determine the required i, we call the procedure Hits at most $O(\log\log U)$ times to get the maximum i. By the analysis of [15], the time complexity of Hits is $O(\log U)$, and the space complexity is $O(\log U)$. Therefore, the time complexity of the algorithm MG is $O(\log U \log\log U)$, and the required space is $O(\log U)$. $\square$

Algorithm and complexity analysis. In the initialization step, the algorithm picks a prime number p between U^2 and U^3, and chooses two numbers a from $\{1, \cdots, p-1\}$ and b from $\{0, \cdots, p-1\}$ at random. Let ℓ be the current level the algorithm stays in and $\ell \leftarrow 0$ initially. In addition, let the sample set S be empty and $\alpha \leftarrow \frac{c}{\varepsilon^2}$ where c is a constant determined by the following analysis. We store an interval r in S as a triple (r, d, w) where $d = M(r)$ and $w = G(r)$.

When a new interval $r_i = [x_i, y_i]$ arrives, the algorithm executes the following operations:

```
1. If $\exists (r, d, w) \in S$ such that $r_i \cap r \neq \emptyset$:
    (a) While $\exists (r, d, w) \in S$ such that $r \cap r_i \neq \emptyset$
            $S \leftarrow S - \{(r, d, w)\}$, $r_i \leftarrow r \cup r_i$, $X_d \leftarrow X_d - w \cdot 2^{d+1}$,
            $Z \leftarrow Z - w \cdot 2^{d+1}$.
    (b) $d_i \leftarrow M(r_i)$, $w_i \leftarrow G(r_i)$.
    (c) $S \leftarrow S \cup \{(r_i, d_i, w_i)\}$, $X_{d_i} \leftarrow X_{d_i} + w_i \cdot 2^{d_i+1}$, $Z \leftarrow Z + w_i \cdot 2^{d_i+1}$.
```

```
2. Else If $M(r_i) \geq \ell$ then
    (a) $d_i \leftarrow M(r_i)$, $w_i \leftarrow G(r_i)$.
    (b) $S \leftarrow S \cup \{(r_i, d_i, w_i)\}$, $X_{d_i} \leftarrow X_{d_i} + w_i \cdot 2^{d_i+1}$, $Z \leftarrow Z + w_i \cdot 2^{d_i+1}$.
    (c) If $||S|| > \alpha$ then
        i. $Z \leftarrow Z - X_\ell$; $S \leftarrow \{(r, d, w)|d > \ell\}$; $\ell \leftarrow \min_{(r,d,w)\in S} d$.
        ii. If $\ell > \lfloor \log p \rfloor$ then return;
```

When an estimate for F_0 is asked for, the algorithm returns $Z \cdot 2^\ell$.

To boost up the probability of achieving the desired approximation value, we run in parallel $O(\log \frac{1}{\delta})$ copies of the algorithm above and take the median of the resulting approximations as the final estimated value.

Theorem 2. *The space complexity of the algorithm above is* $O(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$.

Proof. The space required by calculating $M(\cdot)$ and $G(\cdot)$ is $O(\log U)$. For estimation algorithm, the sample S consists of $\alpha = \Theta(\frac{1}{\varepsilon^2})$ elements, each of whom needs $O(\log U)$ space. In addition, the algorithm needs $\min\{\frac{c}{\varepsilon^2}, \lfloor \log p \rfloor\} \cdot \log U$ space to store the value of $X_0, \cdots, X_{\lfloor \log p \rfloor}$. Therefore the total space is $O(\frac{1}{\varepsilon^2} \log U)$. Since we execute the algorithm $O(\log \frac{1}{\delta})$ times in parallel, the space complexity of this algorithm is $O(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$. □

Theorem 3. *The amortized time to process an interval* $r = [x, y]$ *for the algorithm is* $\tilde{O}(\log \frac{U}{\varepsilon} \log \frac{1}{\delta})$, *and the worst case running time to process an interval* $r = [x, y]$ *is* $\tilde{O}((\frac{1}{\varepsilon^2} + \log U) \log \frac{1}{\delta})$.

Proof. The running time to process an interval consists of three parts: 1. Check whether or not there exists an interval $r' \in S$, such that $r \cap r' \neq \emptyset$; 2. Time for calculating $M(r)$ and $G(r)$; 3. Time for handling an overflow in the sample.

We use a balanced binary search tree T to store the elements in S. So we can use $O(\log \frac{1}{\varepsilon})$ time to check if r intersects with any interval in S in the first part. As Theorem 1 mentioned, we need $O(\log U \log \log U)$ time to calculate $M(r)$ and $G(r)$. Now we analyze the running time of the third part. When the size of S exceeds α, the algorithm uses $O(\frac{1}{\varepsilon^2})$ time to calculate the current level $\ell' \leftarrow \min_{(r,d,w)\in S \wedge d > \ell} d$ and discards the intervals whose $M(\cdot)$'s value is less than ℓ'. This step need scan each element $(r, d, w) \in S$ once, which requires $O(\frac{1}{\varepsilon^2})$ time. Therefore the worst case time complexity of the algorithm is $\tilde{O}((\frac{1}{\varepsilon^2} + \log U) \log \frac{1}{\delta})$.

As for the amortized time, we follow the approach of Pavan and Tirthapura and argue that the total time for handling overflow in the sample (i.e., part 3) over the whole data stream is not more than $\tilde{O}(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$ since the maximum number of level changes is $O(\log U)$. Therefore, the amortized time for inserting an interval for this part is $O(1)$ if the number of input intervals in the data stream is large. Consequently, the amortized time is dominated by the time for part 1 and 2 which is $\tilde{O}(\log \frac{U}{\varepsilon} \log \frac{1}{\delta})$ in total. □

Revised algorithm implementation. The algorithm above uses a balanced binary tree to store the intervals in S. In the streaming algorithms, some researchers (such as [2]) use the maximum number of steps the algorithm spent on a single item as the measure of time complexity. In order to improve the worst case running time for updating per element, we revise our algorithm proposed above. We use a list of balanced binary trees $T_0, T_1, \cdots, T_u$, $u = \lfloor \log p \rfloor$, to store the intervals in the sample S. The number of trees is not more than $\min\{\lfloor \log p \rfloor, \frac{c}{\varepsilon^2}\}$. When we need to store an interval r in S, the algorithm calculates $M(r)$ and $G(r)$ first of all, and stores r in $T_{M(r)}$ if $M(r)$ is not less than the current level ℓ. At the same time, the algorithm updates the estimator Z, and $X_{M(r)}$, whose value is defined by Equation (2) and (3).

Theorem 4. *The space complexity of the revised algorithm is* $O(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$.

Proof. The space used by the algorithm is the space required for the procedure `MG` plus the space for storing the list of trees $T_0, \ldots, T_u$. By Theorem 1, the space complexity for calculating $M(r)$ and $G(r)$ is $O(\log U)$. For the list of binary trees, we store at most $O(\frac{1}{\varepsilon^2})$ items, each of which consists of an interval $r_i = [x_i, y_i]$. In addition, we need $O(\min\{\lfloor \log p \rfloor, \frac{c}{\varepsilon^2}\} \cdot \log U)$ space to store X_i for each tree T_i and $O(\log U)$ space to store Z. Therefore the total space is $O(\frac{1}{\varepsilon^2} \log U)$. Since we run $O(\log \frac{1}{\delta})$ copies of the algorithm in parallel, the total space required by the algorithm is $O(\frac{1}{\varepsilon^2} \log U \log \frac{1}{\delta})$. □

Theorem 5. *The amortized time to process an interval* $r = [x, y]$ *for the revised algorithm is* $O(\log \frac{1}{\varepsilon} \log U \log \frac{1}{\delta})$, *and the worst case running time to process an interval* $r = [x, y]$ *is* $O(\log \frac{1}{\varepsilon} \log U \log \frac{1}{\delta})$.

Proof. The running time to process an interval r consists of three parts: 1. Check whether or not there exists an interval $r' \in T_j$, $0 \le j \le \lfloor \log p \rfloor$, such that $r \cap r' \neq \emptyset$; 2. Time for calculating $M(r)$ and $G(r)$; 3. Time for handling an overflow in the sample.

For the first part, let n_i denote the number of intervals in T_i. Since all the intervals in each tree are disjoint, we can use a balanced binary search tree to store the intervals. Therefore for each tree T_i, we can use $O(\log n_i)$ time to check if r intersects with any interval in T_i. The total time for this part is not more than

$$\begin{aligned} \sum_{i=0}^{\lfloor \log p \rfloor} \log n_i &= \log \prod_{i=0}^{\lfloor \log p \rfloor} n_i \\ &\le \log \Big(\frac{\alpha}{\log p}\Big)^{\log p} \\ &= O\big(\log p \big(\log \frac{1}{\varepsilon} - \log \log p\big)\big) \\ &= O\big(\log U \log \frac{1}{\varepsilon} - \log U \log \log U\big) \end{aligned} \tag{7}$$

By Theorem 1, the required time for the second part is $O(\log U \log \log U)$. For part 3, when the size of S exceeds α, the algorithm finds the minimum $\ell^{'}$, $\ell^{'} > \ell$, such that $T_{\ell'}$ is not an empty tree. The algorithm discards tree T_ℓ, and lets $\ell \leftarrow \ell^{'}$. We can use a linked list to store the root of each (non-empty) tree and the running time for finding ℓ' is $O(1)$. Since the maximum number of level changes is $O(\log U)$, the total time taken by level changes over the whole data stream is not more than $O(\log U \log \frac{1}{\delta})$, and the amortized time updating per element for this part is $O(1)$ if the number of intervals in the data stream is large.

Combined with the three parts, both the amortized and worst case update time to process each interval are $O(\log \frac{1}{\varepsilon} \log U \log \frac{1}{\delta})$. □

Correctness proof. Let the sample $S = \cup_{i=0}^{\lfloor \log p \rfloor} T_i$, where $T_i = \{r | M(r) = i\}$. Let NT_i be the number of distinct elements in set T_i. In addition, let $W(x, i)$ be the indicator random variable whose value is 1 if and only if $\rho(h(x)) = i$.

Define

$$Z_\ell = \frac{Z}{2^\ell} = \sum_{i=\ell}^{\lfloor \log p \rfloor} X_i \cdot 2^{i+1} \tag{8}$$

Lemma 2. $\mathrm{E}[Z_\ell] = F_0 \cdot \frac{1}{2^\ell}, \mathrm{Var}[Z_\ell] = F_0 \frac{1}{2^\ell}(1 - \frac{1}{2^\ell})$.

Proof. Let $D(I)$ denote the set of distinct elements in $I = \{r_1, \cdots, r_n\}$. We want to estimate $F_0 = ||D(I)||$.

Since $\mathrm{E}[W(x, \ell)] = \frac{1}{2^{\ell+1}}$, we get

$$\mathrm{E}[X_i] = \sum_{r \in T_i} \sum_{x \in r} \mathrm{E}[W(x, i)] = \mathrm{E}[NT_i] \cdot \frac{1}{2^{i+1}}$$

Assume that the current level is ℓ, so we get

$$\mathrm{E}[Z_\ell] = \mathrm{E}\Big[\sum_{i=\ell}^{\lfloor \log p \rfloor} X_i \cdot 2^{i+1}\Big] = \sum_{i=\ell}^{\lfloor \log p \rfloor} 2^{i+1}\mathrm{E}[NT_i] \cdot \frac{1}{2^{i+1}} = \sum_{i=\ell}^{\lfloor \log p \rfloor} \mathrm{E}[NT_i] = F_0 \cdot \frac{1}{2^\ell}$$

By Lemma 1, the random variables $\{W(x, i) | x \in D(I)\}$ are all pairwise independent, thus the variance of Z_ℓ is $F_0 \frac{1}{2^\ell}(1 - \frac{1}{2^\ell})$. □

Theorem 6. $\Pr\big\{Z \in [(1 - \varepsilon)F_0, (1 + \varepsilon)F_0]\big\} \geq \frac{2}{3}$.

Proof. Let s be the level in which the algorithm stops, and $t^\star$ is the lowest level such that $\mathrm{E}[Z_{t^\star}] < \frac{\alpha}{C}$, where C is the constant number determined by the following analysis. Let the size of the sample S be $\alpha = \frac{c}{\varepsilon^2}$. Then the probability that the algorithm fails to give a desired estimation is

$$\begin{aligned}
\Pr\big\{|Z - F_0| > \varepsilon F_0\big\} &= \Pr\Big\{\Big|\frac{Z}{2^s} - \frac{F_0}{2^s}\Big| > \varepsilon \frac{F_0}{2^s}\Big\} \\
&= \sum_{i=0}^{\lfloor \log p \rfloor} \Pr\Big\{\big|Z_i - \frac{F_0}{2^i}\big| > \varepsilon \frac{F_0}{2^i}\Big| s = i\Big\} \cdot \Pr\{s = i\} \\
&= \sum_{i=0}^{\lfloor \log p \rfloor} \Pr\Big\{\big|Z_i - \mathrm{E}[Z_i]\big| > \varepsilon \mathrm{E}[Z_i]\Big| s = i\Big\} \cdot \Pr\{s = i\} \\
&\leq \sum_{i=0}^{t^\star} \Pr\Big\{\big|Z_i - \mathrm{E}[Z_i]\big| > \varepsilon \mathrm{E}[Z_i]\Big\} + \sum_{i=t^\star+1}^{\lfloor \log p \rfloor} \Pr\{s = i\}
\end{aligned} \tag{9}$$

By Chebyshev's inequality, we know that for all $i \in \{0, \cdots, t^\star\}$, there holds

$$\Pr\left\{\left|Z_i - \mathrm{E}[Z_i]\right| > \varepsilon \mathrm{E}[Z_i]\right\} \leq \frac{\mathrm{Var}[Z_i]}{\varepsilon^2 \mathrm{E}^2[Z_i]}$$

On the other hand, if the algorithm stops in the level $\ell' > t^\star$, it implies that there are at least α disjoint intervals in S in level $t^\star$, each of whom contributes at least one to the corresponding $X_j, t^\star \leq j \leq \lfloor \log p \rfloor$. So we get $Z_{t^\star} \geq \alpha$, and

$$\begin{aligned}
\Pr\left\{|Z - F_0| > \varepsilon F_0\right\} &\leq \sum_{i=0}^{t^\star} \frac{\mathrm{Var}[Z_i]}{\varepsilon^2 \mathrm{E}^2[Z_i]} + \Pr\{Z_{t^\star} \geq \alpha\} \\
&< \sum_{i=0}^{t^\star} \frac{2^i}{\varepsilon^2 F_0} + \Pr\left\{Z_{t^\star} - \mathrm{E}[Z_{t^\star}] \geq \alpha - \frac{\alpha}{C}\right\} \\
&< \frac{1}{\varepsilon^2 F_0} \cdot 2^{t^\star+1} + \frac{1}{C\alpha(1-1/C)^2} \\
&< \frac{4}{\varepsilon^2 \mathrm{E}[Z_{t^\star-1}]} + \frac{1}{C\alpha(1-1/C)^2} \qquad (10)\\
&\leq \frac{4C}{\varepsilon^2 \alpha} + \frac{1}{C\alpha(1-1/C)^2} \\
&= \frac{4C}{c} + \frac{\varepsilon^2}{Cc(1-1/C)^2} \\
&< \frac{4C}{c} + \frac{1}{Cc(1-1/C)^2} \\
&< \frac{1}{3}
\end{aligned}$$

by using $C = 3$ and $c = 50$. □

So the probability can be amplified to $1 - \delta$ by running in parallel $O(\log \frac{1}{\delta})$ copies of the algorithm and outputting the median of the returning $O(\log \frac{1}{\delta})$ approximating values.

4 Extension: Max-Dominance Norm Problem

Let the input consist of k streams of m integers, where each integer $1 \leq a_{i,j} \leq U$, $i = 1, \cdots, k, j = 1, \cdots, m$, represents the jth element of the ith stream. The max-dominance norm is defined as $\sum_{j=1}^{m} \max_{1 \leq i \leq k} a_{i,j}$.

Employing stable distributions, Cormode and Muthukrishnan [6] designed an (ε, δ)-approximation algorithm of this problem. Pavan and Tirthapura showed the relationship between this problem and range-efficient F_0 estimation [15]. In the same paper, they gave an approximation algorithm for Max-Dominance Norm Problem, whose space complexity is $O(\frac{1}{\varepsilon^2}(\log m + \log U) \log \frac{1}{\delta})$, with amortized update time $O(\log \frac{a_{i,j}}{\varepsilon} \log \frac{1}{\delta})$ and worst case update time $\tilde{O}(\frac{1}{\varepsilon^2} \log a_{i,j} \log \frac{1}{\delta})$.

Combining with Pavan and Tirthapura's technique and our algorithm presented in this paper, it is not hard to show the following theorem.

Theorem 7. *There exists an (ε, δ)-approximation algorithm for Max-Dominance Norm Problem, whose space complexity is $O(\frac{1}{\varepsilon^2}(\log m + \log U)\log\frac{1}{\delta})$, with amortized update time $\tilde{O}(\log\frac{a_{i,j}}{\varepsilon}\log\frac{1}{\delta})$ and worst case update time $\tilde{O}((\frac{1}{\varepsilon^2} + \log a_{i,j})\log\frac{1}{\delta})$.*

5 Further Work

We consider a more general range-efficient F_0 estimation problem — range-efficient F_0 estimation under the *turnstile model* [13] where there can be both insertions and deletions of intervals. Let the multiset S be empty initially. When the intervals arrive, we can not only insert some intervals into S but also delete the intervals from S. When an estimate is requested, the algorithm need to give a desired approximation value of $||S||$.

Some algorithms, such as [5,10], focus on single item case and are suitable for this turnstile model. However, all these known algorithms cannot be easily generalized to the range-efficient case for the following reasons: (1) It is proven in [5] that stable distributions with small stability parameter can be used to approximate F_0 norm. The difficulty of generalizing this method to range-efficient case is the lack of general range-summable p-stable random variables. Though strong range-summability results are known for F_1 and F_2, for general $0 < p \leq 2$, there is no any known p-stable range-summable random variable construction algorithm, which was also listed in [7]. (2) Ganguly et al. gave another algorithm to estimate the cardinality of the multiset S [10], but this algorithm required the use of $\Theta(\log\frac{1}{\delta})$-wise independent hash function. Let h be such kind of hash functions. The algorithm presented in [10] need to calculate $\rho(h(x))$. Though there exist some k-wise range-summable hash function construction algorithms for general k, it is not clear how to calculate $||\{x \in r | \rho(h(x)) = t\}||$ effectively, for the given interval r and parameter t. We leave this more general range-efficient F_0 estimation problem for further work.

Acknowledgements. We thank Piotr Indyk, Graham Cormode and Omer Reingold for some helpful discussions.

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. Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 623-632, 2002.
3. Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proceedings of 6th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1-10, 2002.
4. J. L. Carter, M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143-154, 1979.

5. G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (How to zero in). In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 335-345, 2002.
6. G. Cormode, S. Muthukrishnan. Estimating dominance norms of multiple data streams. In *Proceedings of the 11th European Symposium on Algorithms*, pages 148-160, 2003.
7. G. Cormode. Stable distributions for stream computations: it's as easy as 0,1,2. In *Workshop on Management and Processing of Massive Data Streams*, at FCRC, 2003.
8. M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *Proceedings of the European Symposium on Algorithms*, pages 605-617, 2003.
9. P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31:182-209, 1985.
10. S. Ganguly, M. Garofalakis, and R. Rastogi. Tracking set-expression cardinalities over continuous update streams. *The International Journal on Very Large Data Bases*, 13:354-369, 2004.
11. F. Giroire. Order statistics and estimating cardinalities of massive data sets. *Discrete Mathematics and Theoretical Computer Science*, Vol. AD, pages 157-166, 2005.
12. P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, pages 189-197, 2000.
13. S. Muthukrishnan. Data streams: algorithms and applications. Invited talk at 14th ACM-SIAM Symposium on Discrete Algorithms. Available from `http://athos.rutgers.edu/~muthu/stream-1-1.ps`
14. N. Nisan. Pseudorandom generators for space-bounded computation, In *Proceedings of the 22nd Symposium on Theory of Computation*, pages 204-212, 1990.
15. A. Pavan, S. Tirthapura. Range-efficient computation of F_0 over massive data stream. In *Proceedings of the 21st International Conference on Data Engineering*, pages 32-43, 2005.
16. M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Science*, 22:265-279, 1981.
17. R. Weron. On the Chambers-Mallows-Stuck method for simulating skewed stable random variables. Technical report, Hugo Steinhaus Center for Stochastic Methods, Wrocław, 1996.