Summarizing Order Statistics over Data Streams with Duplicates

Ying Zhang[1] Xuemin Lin[1] Yidong Yuan[1]
Masaru Kitsuregawa[2] Xiaofang Zhou[3] Jeffrey Xu Yu[4]

[1]The University of New South Wale & NICTA
{yingz, lxue, yyidong}@cse.unsw.edu.au

[2]University of Tokyo
kitsure@tkl.iis.u-tokyo.ac.jp

[3]University of Queensland
zxf@itee.uq.edu.au

[4]Chinese University of Hong Kong
yu@se.cuhk.edu.hk

1 Introduction

A rank query is essentially to find a data element with a given rank against a monotonic order specified on data elements. Rank queries have several equivalent variations [6, 13, 22] and play very important roles in many real data stream applications [2, 4, 5, 12, 11, 20, 21], including monitoring high speed networks, trends and fleeting opportunities detection in the stock market, sensor data analysis, Web ranking aggregation and log mining, and summarizing data distributions via *equal-depth histograms*. It has been shown in [16] that an exact computation of rank queries requires memory size linearly proportional to the size of a dataset by any one-scan technique; this may be impractical in on-line data stream computation where streams are massive in size and fast in arrival speed.

Approximately computing rank queries over data streams has been investigated in the form of *quantile computation*. A *ϕ-quantile* ($\phi \in (0, 1]$) of a collection of N data elements is the element with rank $\lceil \phi N \rceil$ against a monotonic order specified on data elements. The main paradigm is to continuously and efficiently maintain a small space data structure (sketch/summary) over data elements to be on-line queried. It has been shown in [1, 10, 12, 18] that a space-efficient ϵ-approximate quantile sketch can be maintained so that, for a quantile ϕ, it is always possible to find an element at rank r' with the *uniform* precision guarantee $|r' - r| \leq \epsilon N$ ($r = \lceil \phi N \rceil$). Observe that many real datasets often exhibit skew towards heads (or tails depending on a given monotonic order). Relative rank error (or biased) quantile computation techniques have been recently developed in [5, 6, 22], which aim to give finer rank error guarantees towards heads; that is, enforce the precision $|r' - r| \leq \epsilon r$ instead of a uniform precision guarantee $|r' - r| \leq \epsilon N$ for each rank r.

In many data stream applications, duplicates may often occur due to the projection on a subspace if elements have multiple attributes. For example, in the stock market a deal with respect to a particular stock is recorded by the transaction ID (TID), volume (vol), and average price (av) per share. To study purchase trends, it is important to estimate the number of different types of deals (i.e. deals with the same vol and the same av are regarded as the same type of deal) with their total prices (i.e. vol*av) higher (or lower) than a given value. It is also interesting to know the total price (of a deal) ranked as a median, or 25th percentile, or 10th, or 5th percentile, etc. among all different types of deals. These two types of rank queries are equivalent [6, 13]; we focus on the later form in this paper. To accommodate processing such queries, each deal transaction (TID, vol, av) is projected on (vol, av) and then summarize the distribution of **distinct** (vol, av)s according to a decreasing (or increasing) order of vol*av; that is, (TID, vol, av) is mapped to (vol, av). Clearly, any generated duplicates (vol, av) must be removed while processing such rank queries. Moreover, relative (or biased) rank error metrics need to be used to provide more accurate results towards heads (or tails depending on which monotonic order is adopted). Note that the generality of rank queries (quantiles) remains unchanged in this application since two different types of deals (i.e., (vol, av)s) may also have the same value vol*av. The unique challenge is to detect and remove the effect of duplicated elements without keeping every element.

Duplicates may also occur when data elements are observed and recorded multiple times at different data sites. For instance, as pointed out in [5, 7] the same packet may be seen at many tap points within an IP network depending on how the packet is routed; thus it is important to discount those duplicates while summarizing data distributions by rank queries (quantiles). Moreover, to deal with possible communication loss TCP retransmits lost packets and leads to the same packet being seen even at a given monitor more than once. In such applications, continuously maintaining order sketches for processing rank queries may be conducted either centrally at one site or at a set of coordinating sites depending on the computing environment and the availability of software and hardware devices. Nevertheless, in either situation a crucial issue is to efficiently and continuously maintain a small space sketch with a precision guarantee, at a single site, by discounting duplicates.

While most existing quantile approximate computation techniques are duplicate-sensitive (i.e. cannot discount duplicates appropriately), the techniques in [7, 14, 17] can provide a duplicate-insensitive approximate quantile solution, with the uniform rank precision ϵn and confidence $1 - \delta$, by space $O(\frac{1}{\epsilon^3} \log \frac{1}{\delta} \log m)$. Here, n is the number of distinct elements and m is the maximal possible number of distinct elements. Nevertheless, the techniques do not provide relative rank error guarantee ϵr unless linear space $O(n)$ is used.

Motivated by this, in this paper we present novel, space-efficient algorithms to continuously maintain order sketches over data streams, in the presence of arbitrary data duplicates, with relative rank error guarantee ϵr. They require space $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} \log m)$. They significantly reduce the space requirement in [7, 14, 17] from $O(\frac{1}{\epsilon^3} \log \frac{1}{\delta} \log m)$ to $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} \log m)$, while also improves rank error precision guarantee from ϵn in [7, 14, 17] to ϵr for any given rank r. To the best of our knowledge, this is the first work regarding such a problem.

The rest of the paper is organised as follows. Section 2 presents problem definitions and related work. Section 3 presents preliminaries. In Section 4, we present our algorithms. This is followed by conclusions and remarks.

2 Problem Statement

In our problem setting, an element x may be either an original element in data streams or the "image" of a projection on an original element (e.g. (vol, av) in the example in section 1). Each element x is augmented to (x, v) in our computation where $v = f(x)$ (called "value") is to rank elements according to a monotonic order of v, and f is a pre-defined function; for instance f could be specified as $vol * av$ (or just av) regarding the example in section 1. Without loss of generality, we assume $v > 0$ and a monotonic order is always an increasing order.

In a collection S of elements, there may be many *duplicated elements*; D_S denotes the set of distinct data elements in S. In this paper, we study the following rank query over a data stream S.

Rank Query (RQ) : Given a rank r, find the the rank r element in D_S.

We investigate the problem of processing RQ queries with ranks to be approximated where ranks are obtained from D_S rather than S. Suppose that r is the given rank in a RQ query, and r' is the rank of an approximate solution. In this paper, we enforce the relative error metric: $\frac{|r'-r|}{r}$. An answer to a RQ regarding r is *relative ϵ-approximate* if its rank r' has the precision $|r' - r| \leq \epsilon r$.

In D_S, there are no duplicates; however, many different elements may happen to have the same values. With the presence of duplicated element values, the rank of an element against its value is not well defined; it can take any rank in $[r_{min,v}, r_{max,v}]$. Here, $r_{min,v}$ and $r_{max,v}$ denote the minimum rank and the maximum rank of an element in D_S with value v, respectively, against a monotonic order (the increasing order as assumed above). Consequently, the definition of relative ϵ-approximate may be equivalently stated as follows. An answer x (with value v) to RQ regarding r is relative ϵ-approximate iff:

$$[r_{min,v}, r_{max,v}] \cap [(1-\epsilon)r, (1+\epsilon)r] \neq \emptyset \quad (1)$$

Quantile Computation VS RQ. Without loss of generality, we assume that a ϕ-quantile is an element with rank ϕn against n distinct elements. Although n is not pre-known in a data stream, our techniques can always guarantee an ϵ-approximate estimation A of n; that is, $|A - n| \leq \epsilon n$ ($\forall \epsilon > 0$). Consequently, we use ϕA in the corresponding rank query instead of ϕn. Immediately, we can verify that a relative ϵ-approximate answer (with rank r') to RQ regarding ϕA leads to a ϕ' ($\phi' = r'/n$) such that $\frac{|\phi - \phi'|}{\phi} \leq 2.5\epsilon$ if $\epsilon \leq \frac{2}{9}$; that is, ϕ' is relative 2.5ϵ-approximate to ϕ.

Problem Description. We investigate the problem of continuously maintaining a sketch (consisting of several sub-sketches) over a data stream S such that at any time, the sketch can be used to return a relative ϵ-approximate answer to a RQ against D_S. The aim is to minimize the *maximum memory space required in such a continuous computation*.

3 FM Algorithm

Suppose that S is a collection of elements whose domain is $\mathcal{D}$. FM algorithm [9] proceeds as follows.

Let B be a bitmap of length k with subindexes $[0, k-1]$. Suppose that $h()$ is a randomly generated hash function $\mathcal{D} \rightarrow B$, such that $\forall x \in \mathcal{D}$, 1) for each bit, $h(x)$ has the equal opportunity to have 0 or 1, 2) $h(x)$ is enforced to have one and only one bit with value 1, and 3) $h(x)$ assigns the last bit (the bit with subindex $k-1$) with value 1 iff the first $k-1$ bits (from left) take value 0. To enforce property 2), $h(x)$ may be interpreted as a serial binary hash functions that start from the first bit and terminate once the current bit is assigned by value 1. It can be immediately shown [3] that on average, $h()$ runs in time $O(1)$ (two calls of a binary hash function) per data element and the probability of having the ith bit with value 1 is $\frac{1}{2^{i+1}}$. In our implementation, we use the public code from Massive Data Analysis Lab [19] to randomly generate such hash functions.

A FM sketch on S is defined as $FM(S) = \bigvee_{x \in S} h(x)$, where $FM(S)$ is a bitmap with length k and the ith bit of $FM(S)$ takes value 1 iff $\exists x \in S$ such that $h(x)$ assigns the value 1 to the ith bit. We define $FM_{min}(S)$ as follows:

- If i is the least bit (from left) with value 0, $FM_{min(S)}$ is defined as i.
- Otherwise, $FM_{min}(S)$ is defined as ∞ (in our implementation, we define $FM_{min}(S)$ as k).

To improve the accuracy of FM algorithm, multiple copies (say, l) of FM sketches are constructed. Therefore, each data element is hashed into l FM sketches, $FM_1(S)$, $FM_2(S)$, ... , $FM_l(S)$, respectively. The number n_S of distinct elements in S is estimated by:

$$A_S = \frac{1}{\varphi} 2^{\sum_{i=1}^{l} FM_{i,min}(S)/l}. \quad (2)$$

Here, $\varphi \overset{\text{def}}{=} 2^{E(FM_{1,min}(S))}/n_S$,[1] and each $FM_{i,min}(S)$ related to $FM_i(S)$ is defined in the same way as $FM_{min}(S)$ related to $FM(S)$. As shown in [9], $E(FM_{i,min}(S)) = E(FM_{j,min}(S))$ ($1 \leq i < j \leq l$). From Theorem 2 in [9] and the *Central Limit Theorem* (pp 229 in [8]), the following lemma can be immediately verified using the independence assumption.

[1]As $E(FM_{1,min}(S))$ cannot be explicitly represented and n_S is unknown, in our implementation we approximately choose φ as 0.775351 according to the approximate results in [9].

Lemma 1. *Suppose that A_S is returned by FM algorithm as shown in (2). Then, the probability $P(|A_S - n_S| > \epsilon n_S)$ is smaller than δ, for any given $0 < \delta < 1$, $0 < \epsilon < 1$, and $L = \frac{1}{\epsilon}$, if $k = O(\log m + \log \epsilon^{-1} + \log \delta^{-1})$ and $l = O(\frac{1}{\epsilon^2} \log \delta^{-1})$, where $m = |\mathcal{D}|$.*

4 Relative Error Sketches

Below is a key observation. For a dataset S, if we first select the data elements from S with element values not greater than a given v (the result is denoted by $S|_{v-}$) and apply FM Algorithm on $S|_{v-}$, then the obtained estimation $A_{S,v}$ of the number $n_{S,v}$ of distinct data elements in $S|_{v-}$ follows Lemma 1. Recall that $r_{max,v}$ is the maximum rank of the data element with value v in D_S against the non-decreasing order of v. Consequently, $r_{max,v} = n_{s,v}$.

Intuitively, we can get a good approximate solution if for each v, $n_{S,v}$ may be estimated accurately. Note that maintaining sketches with the presence of every value v is not only expensive in space but also expensive in running time in case that the total number of distinct values is $\Omega(|D_S|)$. Below, we present a novel, space-efficient data structure (sketch) to be continuously maintained to achieve a relative ϵ-approximation. We also present a theoretic analysis towards space complexity, time complexity, and correctness.

4.1 Algorithm

In our approach, we follow the framework of FM algorithm. To effectively keep values information, we map a bitmap into an array by replacing the bit with hashed value 1 by its corresponding data element value. At each element of such an array, we keep only the smallest data value if multiple data elements have been hashed into this element.

Below, we present our continuous sketch construction and maintenance algorithm in Algorithm 1. We maintain l arrays $\{s_i : 1 \le i \le l\}$ each of which is generated, as described above, by a randomly picked hash function h_i, and has k elements with subindexes from 0 to $k-1$. Recall that without loss of generality, we assumed each element takes positive values. Thus, each array s_i can be initialized to $(0, 0, ..., 0)$. For every $h_i(x)$ $(1 \le i \le l)$, $\rho(h_i(x))$ denotes the position (subindex) of the bit, with value 1, in $h_i(x)$. Note that $s_i[\rho]$ is the ρ-th element in s_i. Moreover, to ensure relative rank errors for a give rank $r < \frac{1}{\epsilon}$ precise answers are the only possibility; consequently, we always keep the L smallest distinct elements (i.e., L distinct elements with the smallest element values) in $\mathcal{L}$ in addition to $\{s_i : 1 \le i \le l\}$,[2] so that RQ with ranks smaller than L can be answered exactly. We use v_{max} to denote the maximal data element value in $\mathcal{L}$ and x_{max} is the element with maximal value. Note that in $\mathcal{L}$ we keep each element x in its augmented form - (x, v). *In each s_i, we link every non-zero value to the corresponding data element so that we can return a data element by a RQ.*

The following theorem is immediate.

Theorem 1. *Algorithm 1 requires the space of $L + l \times k$ elements.*

[2]All duplicates for the elements in L are removed according to the algorithm.

Algorithm 1 RQ-FM Sketches (**RQ-FM**)

Input: l, k, L, a stream S of (x, v).
Output: $\mathcal{L}$: the set of L smallest distinct elements; $\{s_i : 1 \le i \le l\}$: each s_i is an array with k elements.
Description:

```
1:  Initialize $\{s_i : 1 \le i \le l\}$; $\mathcal{L} \leftarrow \emptyset$; $j \leftarrow 0$;
2:  Generate $l$ hash functions $\{h_i() : 1 \le i \le l\}$;
3:  for each new $x$ with value $v$ do
4:      if $(x, v) \notin \mathcal{L}$ then
5:          if $j < L$ then
6:              $\mathcal{L} \leftarrow \mathcal{L} \cup \{(x, v)\}$; $j \leftarrow j + 1$
7:          else if $v < v_{max}$ then
8:              replace $(x_{max}, v_{max})$ in $\mathcal{L}$ by $(x, v)$;
9:      for i=1 to l do
10:         $\rho \leftarrow \rho(h_i(x))$;
11:         if $s_i[\rho] > v$ or $s_i[\rho] = 0$ then
12:             $s_i[\rho] \leftarrow v$;
13: Return $\mathcal{L}$ & $\{s_i : 1 \le i \le l\}$.
```

To estimate $n_{S,v}$ for a given v, our query algorithm proceeds as follows. If $v < v_{max}$ then we only query $\mathcal{L}$. Otherwise, in the light of key observation we first select the elements in s_i with positive values (corresponding to data elements in D_S) but not greater than v; the result is denoted by $s_i|_{v-}$. Then, we return the location of the left-most element in s_i that is not included in $s_i|_{v-}$. If such a left-most element does not exist, we return k (corresponding to the situation ∞ when we presented FM Algorithm). Let Π denote a subset of elements in an array and $I(\Pi)$ denote the set of subindexes of the elements in Π. Our query algorithm is presented in Algorithm 2.

Algorithm 2 Approximating $n_{S,v}$

Input: v, $\mathcal{L}$, $\{s_i : 1 \le i \le l\}$ generated by Algorithm 1;
Output: $A_{S,v}$;
Description:

```
1:  get $v_{max}$ from $\mathcal{L}$;
2:  if $v_{max} > v$ then
3:      $A_{S,v} \leftarrow |\mathcal{L}|_{v-}|$;
4:  else
5:      for $i = 1$ to $l$ do
6:          if $[0, k-1] - I(s_i|_{v-}) \neq \emptyset$ then
7:              $f_{i,v} \leftarrow \min\{j : j \in [0, k-1] - I(s_i|_{v-})\}$;
8:          else
9:              $f_{i,v} = k$;
10:     $A_{S,v} \leftarrow \frac{1}{\varphi} 2^{\sum_{i=1}^{l} f_{i,v}/l}$;
11: Return $A_{S,v}$.
```

From Lemma 1, the following Lemma immediately holds.

Lemma 2. *For a given v, ϵ, and δ, $A_{S,v}$ returned by Algorithm 2 against the output of Algorithm 1 has the property that $P(|A_{S,v} - n_{S,v}| > \epsilon n_{S,v}) < \delta$ if $L = \frac{1}{\epsilon}$, $l = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ and $k = O(\log m + \log \delta^{-1} + \log \epsilon^{-1})$.*

4.2 Space VS Accuracy

We first present our rank query algorithm against the sketches generated by Algorithm 1. To retain relative ϵ-approximation, the basic idea is that for a given rank r, find the maximal $A_{S,v}$ but not greater than r by invoking Algorithm 2 multiple times. If $|A_{S,v} - r| < \epsilon_1 r$ ($\epsilon_1 = \epsilon/3$ for $0 < \epsilon < 1$), then return x with value v otherwise return x' with value v' where v' is the value in the sketch immediately greater than v.

Remark 1: Clearly, if $r \leq L$, then we only need to get a data element in $\mathcal{L}$ with the rth smallest value. It is the exact solution. Therefore, below we only discuss $r > L$; that is, we only query $\{s_i : 1 \leq i \leq l\}$.

Our query algorithm is presented in Algorithm 3. It is based on the following monotonic property that can be immediately verified according to Algorithm 2.

Lemma 3. *Applying algorithm 2 to $\{s_i : 1 \leq i \leq l\}$ (generated by Algorithm 1), $A_{S,v_1} \leq A_{S,v_2}$ for any $v_1 < v_2$.*

Algorithm 3 Processing a Rank Query

```
Input:  $r > L$, $0 < \epsilon_1 < 1$, $\{s_i\}$ generated by Algorithm 1;
Output:  $x'$;
Description:
 1: $a \leftarrow \max\{v : A_{S,v} \leq r \ \&\ v \in \cup_{i=1}^{l} s_i\}$;
 2: get $x'$ such that its value $v'$ is $a$;
 3: if $|a - r| \leq \epsilon_1 r$ then
 4:     Return $x'$;
 5: else
 6:     if $a$ is the maximum value in $\cup_{i=1}^{l} s_i$ then
 7:         Return $r > n_S$; (outside solution range)
 8:     else
 9:         $a \leftarrow \min\{v : A_{S,v} > r \ \&\ v \in \cup_{i=1}^{l} s_i\}$;
10:         Return $x$ such that its value $v'$ is $a$;
```

Now, we show the precision guarantee of Algorithm 3.

Theorem 2. *For any $0 < \delta < 1$, $0 < \epsilon < 1$ and $r > L$, suppose that the element x' is returned by Algorithm 3 with value v'. Then,*

$$P([r_{min,v'}, r_{max,v'}] \cap [(1-\epsilon)r, (1+\epsilon)r] = \emptyset) < \delta$$

if $l = O(\frac{1}{\epsilon_1^2} \log \frac{1}{\delta})$, $k = O(\log m + \log \delta^{-1} + \log \epsilon_1^{-1})$, $L = \frac{1}{\epsilon}$, and $\epsilon_1 = \frac{\epsilon}{3}$.

Proof. The proof is quite lengthy and we omit it here due to the space limit. The basic idea is to prove that for two "consecutive" values, v_1 and v_2, occurred in the sketch, the difference of their corresponding n_{S,v_1} and n_{S,v_2} is within $\frac{\epsilon}{3} \max\{n_{S,v_1}, n_{S,v_2}\}$. □

Theorem 2 states that with the set of parameters, the data element returned by Algorithm 3 is ϵ-approximate with probability at least $1 - \delta$. It can be immediately verified that another output, "$r > n_S$", has the probability at least $1 - \delta$ to be correct with this set of parameters. Theorems 2 and 1 immediately imply that to ensure the relative ϵ-approximate property for rank queries against distinct elements in a data stream, the space requirement is $O(\frac{1}{\epsilon^2} \log \delta^{-1} \log m)$ if $m \geq \epsilon^{-1}$ and $m \geq \delta^{-1}$.

Remark 2: In Algorithm 3, the output $r > n_S$ (i.e. the answer is outside solution range) implies the condition $r > \frac{A_S}{1-\epsilon}$ where A_S is an estimation of n_S by Algorithm 2. According to the discussions above, such an answer (output) is correct with probability at least $1 - \delta$. Similarly, in our other techniques presented in the paper this property also holds. Therefore, without loss of generality we assume, thereafter, that in a rank query r, $1 \leq r \leq \frac{A_S}{1-\epsilon}$ where A_S is an estimation of n_S by the corresponding query algorithm to estimate A_S. Consequently, we no longer need to handle the situation that no element is returned.

Remark 3: To accommodate t quantile queries, it is immediate that $O(\frac{1}{\epsilon^2} \log \frac{t}{\delta} \log m)$ space is required to ensure relative ϵ-approximate with the confidence $1 - \delta$.

4.3 Time Complexity

In Algorithm 1, it runs in time $O(\log \frac{1}{\epsilon})$ per element to dynamically maintain $\mathcal{L}$ if we maintain a search tree on $\mathcal{L}$. As discussed earlier, each $h_j()$ ($1 \leq j \leq l$) takes constant time on average to hash a data element. Thus, Algorithm 1 runs in time $O(\frac{1}{\epsilon^2} \log \delta^{-1})$ on average per data element, given there are $O(\frac{1}{\epsilon^2} \log \delta^{-1})$ such arrays.

Algorithm 3 can be implemented as follows. We sort $\cup_{i=1}^{l} s_i$ on element values, and then scan the sorted list, by calling Algorithm 2 iteratively, till find such v'. Note that in each iteration, we do not run Algorithm 2 from scratch; instead we incrementally update the result from last iteration. Clearly, the dominant costs appear in the sorting process; consequently Algorithm 3 run in time $O(K \log K)$ where $K = O(\frac{1}{\epsilon^2} \log \delta^{-1} \log m)$ (assuming $m \geq \epsilon^{-1}$ and $m \geq \delta^{-1}$) if subsketches have not been pre-sorted.

4.4 PCSA-based

Note that in Algorithm 1, each element is hashed into $\Omega(\frac{1}{\epsilon^2} \log \delta^{-1})$ arrays (subsketches). This potentially makes the algorithm less efficient. Our experiment demonstrates it can only handle 300-400 elements per second.

In this section, we modify the algorithm based on the PCSA technique [9] to our algorithm, Algorithm 1. The basic idea is to hash each data element randomly to ζ arrays (subsketches) among the l arrays (subsketches). Algorithm 1 may be modified as follows.

- First, we pick at random another ζ hash functions: $\{H_i : 1 \leq i \leq \zeta\}$ besides the l hash functions in Algorithm 1, where each H_i hashes the element domain $\mathcal{D}$ to $[1, l]$.
- Then, in Algorithm 1 instead of the iteration (in line 9) from $i = 1$ to l, we do the iteration for each $i \in \{H_1(x), H_2(x), ..., H_\zeta(x)\}$. The others in Algorithm 1 remain the same.

We call such a modified Algorithm 1 "Algorithm RQ-PCSA". Suppose that all the parameters are selected as those in Theorem 2. It is immediate Algorithm RQ-PCSA runs in time $O(\log \frac{1}{\epsilon} + \zeta)$ for each data element.

In the light of PCSA technique, Algorithm 2 is modified accordingly as follows to estimate a $n_{S,v}$. We change line 10 in Algorithm 2 to $A_{S,v} \leftarrow \frac{l}{\zeta\varphi} 2^{\sum_{j=1}^{l} f_{j,v}/l}$. Then, Algorithm 3 remains the same to answer a rank query but calls the modified version of Algorithm 2. It can be implemented in the same way as what we described in Section 4.3 with the same time complexity. These, together with the facts in [9], immediately imply that the expected accuracy of Algorithm RQ-PCSA is relative ϵ-approximate. Note that in our implementation, we use pairwise independent hash function for H_i and our performance study indicates that when $\zeta \geq 10$, its accuracy remains quite stable.

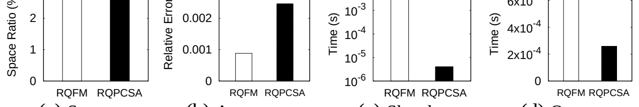
5 Conclusions and Remarks

In this paper, we investigated the problem of approximately processing rank queries against distinct data elements in a data stream with the presence of duplicated data elements. Novel space and time efficient techniques are developed for continuously maintaining order statistics so that rank queries can be answered with a relative error guarantee. This is the first work providing the space and time efficient data stream techniques to process approximate rank queries with *relative error* guarantees against *distinct* data elements.

We have also done a thorough performance evaluation of our sketch techniques and the corresponding query algorithms. Due to the space limit, we only present the following results regarding *space ratio* (i.e., the number of tuples in sketches over that in a data stream), *accuracy* (i.e., the relative error), and efficiency. They are based on a real dataset WCH (World Cup 98's HTTP request data) downloaded from the Internet Traffic Archive [15]. It consists of 17 million records of requests made to the 1998 World Cup Web site between April 30, 1998 and July 26, 1998. There are total more than 1.53M duplicated data elements and the maximum duplication number of an element is 235.

All experiments have been carried out on a PC with Intel P4 2.8GHz CPU and 1G memory. In our experiment, we choose $\epsilon = 0.02$, $\delta = 0.05$, $l = \frac{2}{\epsilon^2}\log\delta^{-1}$, and $L = \frac{1}{\epsilon}$ in both RQFM and RQPCSA. We also set $k = 32$ in both algorithm because we use the public code from Massive Data Analysis Lab to generate hash functions [19] and 2^{32} is large enough to accommodate massive number of distinct data elements. We assign $\zeta = 10$ in RQPCSA.

Figure 1(a) shows the space ratio; note that both algorithms always have a same pre-defined sample size if other parameters are the same. Figure 1(b) shows the accuracy where we report the average relative error. It shows that the actual average error is much smaller than the designated error 0.02; in fact, in the experiment we have no query result with the relative error larger than 0.02. Figure 1(c) reports the average time of processing each element in continuous maintaining sketches. Our experiment shows that RQFM can only process 300-400 elements per second while RQPCSA can process about 75K elements per second. Finally, Figure 1(d) reports the average query processing time of processing a batch of 1000 quantile queries randomly generated.



(a) Space (b) Accuracy (c) Sketch (d) Query

Figure 1. Experiment Results

The experiment demonstrates that besides proven accuracy and space guarantees, both algorithms are very space efficient and highly accurate in practice. RQPCSA is efficient enough to support on-line computation of very high speed data streams with an element arrival rate up to 75K/second. We also report that our performance evaluation against various synthetic datasets have very similar trends.

Acknowledgement. The work was partially supported by an ARC discovery grant (DP0666428) and UNSW FRG (RGP, PS08709). The work was partially done when the 2nd author visited Tokyo University as a JSPS fellow and visited the Chinese University of Hong Kong.

References

[1] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS04*.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS'02*.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE'04*.

[4] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD'05*.

[5] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Effective computation of biased quantiles over data streams. In *ICDE'05*.

[6] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In *PODS'06*, 2006.

[7] G. Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *PODS'05*.

[8] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, Inc., 1966.

[9] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[10] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB2002*.

[11] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS'04*.

[12] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD'01*.

[13] A. Gupta and F. Zane. Counting inversions in lists. In *SODA'03*.

[14] M. Hadjieleftheriou, J. W. Byers, and G. Kollios. Robust sketching and aggregation of distributed data streams. Technical report, Boston University, 2005.

[15] Internet Traffic Archive. `http://ita.ee.lbl.gov`.

[16] J.I.Munro and M.S.Paterson. Selection and sorting with limited storage. In *TCS12*, 1980.

[17] A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *SIGMOD'05*.

[18] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *SIGMOD'99*.

[19] Massive Data Analysis Lab. `http://www.cs.rutgers.edu/~muthu/massdal.html`.

[20] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys'04*.

[21] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *SenSys'04*, pages 239–249, 2004.

[22] Y. Zhang, X. Lin, J. Xu, F. Korn, and W. Wang. Space-efficient relative relative error order sketch over data streams. In *ICDE'06*.