# Duplicate-insensitive Order Statistics Computation over Data Streams

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

[1] University of New South Wales & 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

**Abstract**—Duplicates in data streams may often be observed by the projection on a subspace and/or multiple recordings of objects. Without the uniqueness assumption on observed data elements, many conventional aggregates computation problems need to be further investigated due to their duplication sensitive nature. In this paper, we present novel, space-efficient, one-scan algorithms to continuously maintain duplicate insensitive order sketches so that rank-based queries can be approximately processed with a relative rank error guarantee $\epsilon$ in the presence of data duplicates. Besides the space efficiency, the proposed algorithms are time-efficient and highly accurate. Moreover, our techniques may be immediately applied to the heavy hitter problem against distinct elements and to the existing fault-tolerant distributed communication techniques. A comprehensive performance study demonstrates that our algorithms can support real-time computation against high speed data streams.

**Index Terms**—Order Statistic, Data Stream, Duplicate Insensitive, Relative Error

## 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 [13], [24], [38] and play very important roles in many real data stream applications [1], [3], [11], [12], [15], [20], [21], [22], [35], [36], 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 [28] 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 *$\phi$-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 [2], [19], [20], [32] that a space-efficient $\epsilon$-approximate quantile sketch can be maintained so that, for a quantile $\phi$, 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 [12], [13], [38], 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 $25$th percentile, or $10$th, or $5$th percentile, etc. among all different types of deals. These two types of rank queries are equivalent [13], [24]; 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 is summarized 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 [12], [14] 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 co-ordinating 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 [14], [25], [31] can provide a duplicate-insensitive approximate quantile solution, with the uniform rank precision $\epsilon n$ 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 $\epsilon r$ 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. To the best of our knowledge, this is the first work regarding such a problem. Our contributions may be summarized as follows:

1) We develop a novel, one-scan theoretical framework with the relative rank error guarantee $\epsilon r$ by $1-\delta$ confidence and $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log m)$ space. This significantly reduces the space requirement in [14], [25], [31] from $O(\frac{1}{\epsilon^3}\log\frac{1}{\delta}\log m)$ to $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log m)$, and also improves rank error precision guarantee from $\epsilon n$ in [14], [25], [31] to $\epsilon r$ for any given rank $r$.
2) To accommodate an on-line processing requirement against high speed data streams, two space- and time- efficient algorithms are also developed following the framework.
3) Finally, we show that our techniques may be immediately applied to computing duplicate-insensitive *heavy hitters* with the space bound the same as or better than those in [14], [31], and to the existing fault-tolerant distributed communication techniques.

A comprehensive performance study demonstrates that our techniques can efficiently compute approximate quantiles over high speed data streams with high accuracy and a small space requirement.

The rest of the paper is organised as follows. Section 2 presents problem definitions and related work. Section 3 presents some necessary preliminaries. In section 4, we present our theoretical framework to continuous maintain space-efficient sketches. Section 5 presents two time-efficient algorithms. In section 6, we report our experiment results. Section 7 shows applications of our techniques to various other problems. Section 8 concludes the paper.

## 2 Background Information

We first state the problem. Then we present the related work. Below in Table 2 we summarize the math notation used throughout the paper.

| Notation | Definition |
|---|---|
| $S$ | a collection of data elements |
| $S\|_{v-}$ | elements in $S$ with values not greater than $v$ |
| $D$ | a set of distinct data elements |
| $N$ | number of elements |
| $m$ | maximal number of distinct elements |
| $n$ | actual number of distinct elements |
| $A$ | estimated number of distinct elements |
| $s$ | a sketch |
| $l$ | number of sketches |
| $k$ | number of elements in a sketch |
| $e$ | a data element |
| $P$ | probability |
| $\delta$ | failure probability |
| $\epsilon$ | precision |

TABLE 1
Math Notation

### 2.1 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 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. We could use the constant-based absolute error metric; that is, enforce $|r'-r| \leq \epsilon$ for a given $\epsilon$. It is immediate that such an absolute error precision guarantee leads to the space requirement $\Omega(m)$ even for an off-line computation. In this paper, we use the relative error metric: $\frac{|r'-r|}{r}$. An answer to a RQ regarding $r$ is *relative $\epsilon$-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 $\epsilon$-approximate may be equivalently stated as follows. An answer $x$ (with value $v$) to RQ regarding $r$ is relative $\epsilon$-approximate iff:

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

**Example 1.** *A data stream consists of the 19 augmented elements:* $(x_1, 15)$, $(x_3, 8)$, $(x_4, 10)$, $(x_5, 9)$, $(x_6, 1)$, $(x_7, 8)$, $(x_8, 10)$, $(x_9, 9)$, $(x_{10}, 6)$, $(x_{11}, 7)$, $(x_{12}, 8)$, $(x_3, 8)$, $(x_{13}, 13)$, $(x_{12}, 8)$, $(x_{14}, 5)$, $(x_{15}, 4)$, $(x_{14}, 5)$, $(x_{16}, 2)$, $(x_2, 3)$. *After removing 3 duplicated data elements, the sorted $D_S$ (over values) consists of the 16 augmented elements:* $(x_6, 1)$, $(x_{16}, 2)$, $(x_2, 3)$, $(x_{15}, 4)$, $(x_{14}, 5)$, $(x_{10}, 6)$, $(x_{11}, 7)$, $(x_{12}, 8)$, $(x_3, 8)$, $(x_7, 8)$, $(x_5, 9)$, $(x_9, 9)$, $(x_8, 10)$, $(x_4, 10)$, $(x_{13}, 13)$, $(x_1, 15)$.

*It may be immediately verified that $r_{min,8} = 8$ and $r_{max,8} = 10$. For a rank 10, any of $x_{12}$, $x_3$, and $x_7$ can be regarded as the exact answer of the RQ query regarding $r = 10$. Clearly, any of $x_5$ and $x_9$ can be treated as a relative $\epsilon$-approximate answer to this RQ query if $\epsilon = 0.1$.*

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

**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 $\epsilon$-approximate answer to a RQ against $D_S$. The aim is to minimize the *maximum memory space required in such a continuous computation*.

### 2.2 Related Work

With recent data-intensive applications in sensor/P2P networks, the FM technique [18] has been first applied in [5], [9], [35] to developing duplicate-insensitive techniques for approximately computing *sum*, *count* (number of sensor nodes), *average* to achieve high communication fault-tolerance. The most related work has been presented in [14], [25], [31].

In [31], Manjhi, Nath, and Gibbons propose an effective adaption paradigm for in-network aggregates computation over stream data with the aim to minimize communication costs and to achieve high fault-tolerance. As indicated, a duplicate-insensitive technique for approximately computing quantiles may be immediately obtained by a combination of their tree-based approximation technique and the existing *distinct counting* technique in [4]. It can be immediately applied to a single site, where a data stream has duplicated elements, with the uniform precision guarantee $|r' - r| \leq \epsilon n$ by confidence $1 - \delta$ and space $O(1/\epsilon^3 \log 1/\delta \log m)$.

In [14], Cormode and Muthukkrishnan present a *DISTINCT RANGE SUMS* technique by applying the FM [18] technique on the top of the *count-min* [10]. The technique can be immediately used to approximately processing RQ with the uniform precision guarantee $|r' - r| \leq \epsilon n$, confidence $1 - \delta$, and space $O(\frac{1}{\epsilon^3} \log \frac{1}{\delta} \log^2 m)$. Independently, Hadjieleftheriou, Byers, and Kollios [25] also developed two novel duplicate-insensitive techniques to approximately compute quantiles in a distributed environment. Applying their techniques to a single site immediately leads the uniform precision guarantee $|r' - r| \leq \epsilon n$ by confidence $1 - \delta$ and space $O(\frac{1}{\epsilon^3} \log \frac{1}{\delta} \log m)$.

Clearly, our results, as stated in section 1, significantly improves these results in both precision guarantee and space usage.

**Other Related Work.** As summarized below, there is great amount of recent work on the problem of conventional quantile computation (i.e., no duplicated elements).

In [2], [20], [19], [30], [32], many space efficient techniques have been developed for whole data streams, sliding windows, and stream data with updates, respectively. Communication-efficient quantile query processing algorithms in sensor networks have also been recently reported in [21], [11], [36]. The *gossip* communication method is proposed in [8], [29] for efficiently computing aggregates, including quantile computation, over networks. In [35], a sampling technique is presented for computing quantiles where transmission duplicates are removed by the nodeID information. Rank queries against multi-dimensional datasets have been recently investigated in [26], [37]. All of them guarantee the uniform precision $|r' - r| \leq \epsilon N$.

In [12], [13], [24], [38], space efficient techniques have been developed for quantile computation with relative error guarantee $|r' - r| \leq \epsilon r$, while a space efficient technique in [23] enforces a finer rank error guarantee $|r' - r| = O(r^{0.5+\epsilon})$.

The techniques cited above do not cover our problem.

## 3 Preliminaries

We present briefly the two sampling algorithms in [18] and [4]. They will be used in our algorithms.

### 3.1 FM Algorithm

Suppose that $S$ is a collection of elements whose domain is $\mathcal{D}$. The FM algorithm [18] 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 [9] 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 $i$th bit with value 1 is $\frac{1}{2^{i+1}}$. In our implementation, we use the public code from Massive Data Analysis Lab [33] 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 $i$th bit of $FM(S)$ takes value 1 iff $\exists x \in S$ such that $h(x)$ assigns the value 1 to the $i$th 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 $\infty$ (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, $\varphi \stackrel{\text{def}}{=} 2^{E(FM_{1,min}(S))}/n_S$,[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 [18], $E(FM_{i,min}(S)) = E(FM_{j,min}(S))$ $(1 \leq i < j \leq l)$. From the insight in Section 3.2 in [15], Theorem 2 in [18], and the *Central Limit Theorem* (pp 229 in [17]), the following lemma can be immediately verified using the independence assumption.

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

An important feature of FM algorithm is that the bitwise-or operator provides an equivalent way to generate a set of FM sketches over $P \cup Q$. The following lemma can be immediately verified.

**Lemma 2.** *Given a set of $l$ hash functions and two collections, $P$ and $Q$, of data points, we have $FM(P \cup Q) = FM(P) \bigvee FM(Q)$.*

### 3.2 BJKST algorithm

In [4], a novel variation of FM algorithm, BJKST algorithm, has been proposed to speed-up the computation, while the accuracy and the space-efficiency can be retained. It proceeds as follows. First, we pick at random a pairwise independent hash function $h$ to hash $\mathcal{D}$ to $[1, m^3]$ where $D$ is the domain of data elements $x$ and $|\mathcal{D}| = m$. The following Lemma has been shown as folklore.

**Lemma 3.** *If $m \geq \delta^{-1}$ then $h$ is injective over $S$ with probability at least $1 - \delta$.*

1. As $E(FM_{1,min}(S))$ cannot be explicitly represented and $n_S$ is unknown, in our implementation we approximately choose $\varphi$ as 0.775351 according to the approximate results in [18].

Based on this, BJKST algorithm always keeps the $k$ smallest elements (i.e. with the $k$ smallest distinct hash values) and uses the following $A_S$ to estimate $n_S$

$$A_S = \frac{k \times m^3}{f_{k\_\min}}. \quad (3)$$

Here, $f_{k\_\min}$ is the $k$th smallest distinct hash value. If there are less than $k$ distinct values, then $A_S = \infty$ (in our implementation, we put $A_S = \frac{k' \times m^3}{f_{k'\_min}}$ in case if there are only $k'$ distinct hash values). To improve the accuracy, BJKST algorithm picks at random $l$ pairwise independent hash functions $h_i$ (hashing $\mathcal{D}$ to $[1, m^3]$), and outputs $A_{i,S}$ for each $h_i$ where $A_{i,S}$ (for $1 \leq i \leq l$) related to $h_i$ is defined in the same way as $A_S$ related to $h$. BJKST algorithm outputs $A_S$ as the medium of these $A_{i,S}$ to estimate $n_S$. BJKST algorithm keeps only $k$ elements with the $k$ smallest distinct hash values. The following Lemma 4 has been proved in [4].

**Lemma 4.** *Suppose that $0 < \epsilon, \delta < 1$. If $m \geq \delta^{-1}$, $k = O(\frac{1}{\epsilon^2})$, $l = O(\log \delta^{-1})$, and $n_S \geq k$, then $P(|A_S - n_S| > \epsilon n_S) < \delta$.*

## 4 Relative Error Sketches

Our technique to construct sketches is based on the following 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 $\epsilon$-approximation. We also present a theoretic analysis towards space complexity, time complexity, and correctness.

### 4.1 The Framework

The following example illustrates the basic idea in our framework based on FM algorithm.

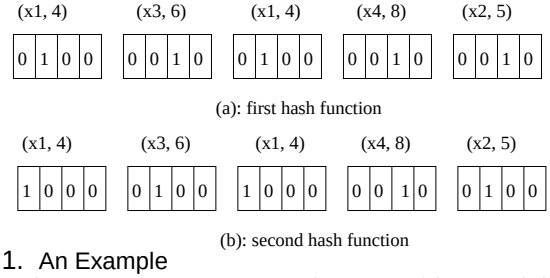



Fig. 1. An Example

As shown in Figure 1, 5 elements (depicted by the augmented form (x, v)) are collected where the first and the third are the same. Suppose that in FM algorithm $l = 2$ and $k = 4$; thus two hash functions $h_1$ and $h_2$ are randomly picked to hash each element, respectively.

A total of 10 bitmaps with length 4 are generated, respectively, by $h_1$ and $h_2$, as depicted in Figure 1(a)-(b).

In our approach, to effectively keep values information we map a bitmap into an array by replacing the bit with value 1 by its corresponding data element value. Figure 2 illustrates the corresponding arrays converted from the bitmaps in Figure 1.

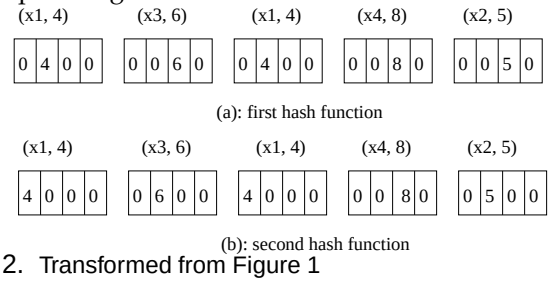



Fig. 2. Transformed from Figure 1

For a value $v$ and a $j$, to estimate $n_{S,v}$ by using FM algorithm we first select the arrays generated $h_j$ such that their corresponding non-zero values not greater than $v$, then find the left-most common element with value 0 and return its subindex as $f_{j,v}$.

**Example 2.** *Let $v = 6$ and $j = 2$. Regarding Figure 2 (b), the 1st array, 2nd array, 3rd array, and 5th array are selected. Then, $f_{2,6} = 2$ (i.e. the subindex of the 3rd element). In this query, the 2nd and 3rd arrays are redundant.*

Clearly, computing $f_{j,v}$, by this way, is equivalent to what have been discussed in the beginning of this section; that is, we do a selection on $S$ to output $S|_{v^-}$; then apply $h_j$ on $S|_{v^-}$ and use FM Algorithm to get $FM_{j,min}(S|_{v^-})$ $(= f_{j,v})$. Moreover, this example also demonstrates that if two arrays have non-zero values allocated in the same position, the one with larger values will never be used in any query (i.e., regarding any $v$); consequently, this redundant array should be removed. Therefore, in the worst case we keep only $k$ arrays where $k$ is the length of bitmaps in the hash functions. Furthermore, after removing redundant arrays the remaining arrays generated by $h_j$ can be merged into one array with non-zero values remain in the same positions, respectively.

**Example 3.** *Regarding the example in Figure 2(a), 2nd, 3rd, and 4th arrays are redundant and thus, are removed. The merged result is depicted in Figure 3(a). For the example in Figure 2(b), 2nd and 3rd arrays are redundant. The merged result is depicted in Figure 3(b).*

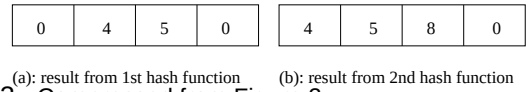



Fig. 3. Compressed from Figure 2

Below, we present our continuous sketch construction and maintenance algorithm in Algorithm 1. We maintain $l$ arrays $\{s_i : 1 \leq i \leq l\}$ 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 $h_i(x)$ $(1 \leq i \leq l)$, $\rho(h_i(x))$ denotes the position (subindex) of the bit, with value 1, in $h_i(x)$. Note that $s_i[\rho]$ is the $\rho$-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 $\{s_i : 1 \leq i \leq l\}$,[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 an RQ.*

**Algorithm 1** Space-Efficient Sketches (**SE**)

**Input:**
$l$, $k$, $L$, a stream $S$ of $(x, v)$.
**Output:**
$\mathcal{L}$: the set of $L$ smallest distinct elements;
$\{s_i : 1 \leq i \leq l\}$: each $s_i$ is an array with $k$ elements.
**Description:**
1: Initialize $\{s_i : 1 \leq i \leq l\}$; $\mathcal{L} \leftarrow \emptyset$; $j \leftarrow 0$;
2: Generate $l$ hash functions $\{h_i() : 1 \leq i \leq l\}$;
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 $\mathcal{L}$ & $\{s_i : 1 \leq i \leq l\}$.

The following theorem is immediate.

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

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 earlier discussions 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 $\infty$ when we presented FM Algorithm). Let $\Pi$ denote a subset of elements in an array and $I(\Pi)$ denote the set of subindexes of the elements in $\Pi$. Our query algorithm is presented in Algorithm 2.

Similar to Lemma 1, the following Lemma holds for every pair of $A_{S,v}$ and $n_{S,v}$ regardless the value of $L$.

**Lemma 5.** *For a given $v$, $\epsilon$, and $\delta$, $A_{S,v}$ 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 $l = O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ and $k = O(\log m + \log\delta^{-1} + \log\epsilon^{-1})$.*

*Proof:* If $A_{S,v}$ is returned from only counting $\mathcal{L}$, it is the exact answer. The lemma is immediate.

Consider that $A_{S,v}$ is returned from $\{s_i : 1 \leq i \leq l\}$.

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

**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 $A_{S,v}$.

It can be immediately verified that Algorithm 2, in this case, is equivalent to: 1) doing a select on $S$ to output $S|_{v^-}$, and then 2) applying FM algorithm on $S|_{v^-}$. According to Lemma 1, this lemma is also immediate. □

As discussed above, nevertheless, to achieve relative $\epsilon$-approximation ($\forall 0 < \epsilon < 1$), any given RQ query with $r \le \frac{1}{\epsilon}$ will have to be answered exactly; that is, $L \ge \frac{1}{\epsilon}$. In the next subsection, we will show that $L = \frac{1}{\epsilon}$ is enough to guarantee relative $\epsilon$-approximation.

### 4.2 Space VS Accuracy

We first present our rank query algorithm against the sketches generated by Algorithm 1. To retain relative $\epsilon$-approximation, the basic idea is that for a given rank $r$, find the maximal $A_{S,v}$ which is 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 $r \le L$, then we only need to get a data element in $\mathcal{L}$ with the $r$th smallest value. It is the exact solution. Therefore, below we only discuss $r > L$; that is, we only query $\{s_i : 1 \le i \le l\}$.

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

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:* With another constant factor added inside $O$ notation, Lemma 5 also holds for any $\delta/2$.

It is immediate that for an existing element value $v$, $|A_{S,v} - n_{S,v}| > \epsilon_1 n_{S,v}$ with probability less than $\delta/2$. Since

**Algorithm 3** Processing a Rank Query

**Input:**
$r > L$, $0 < \epsilon_1 < 1$, $\{s_i\}$ generated by Algorithm 1;
**Output:**
$x'$;
**Description:**
1: $a \leftarrow \max\{v : A_{S,v} \le r \ \& \ v \in \cup_{i=1}^{l} s_i\}$;
2: get $x'$ such that its value $v'$ is $a$;
3: **if** $|a - r| \le \epsilon_1 r$ **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 $a$;

$n_{S,v} = r_{max,v}$, $|A_{S,v} - r_{max,v}| > \epsilon_1 r_{max,v}$ with probability less than $\delta/2$. There are two cases, either

Case 1: Algorithm 3 returns $x'$ with its value $v'$ such that $|A_{S,v'} - r| \le \epsilon_1 r$, or
Case 2: Algorithm 3 returns $x'$ with its value $v'$ otherwise.

**Case 1 proof.** Since $|A_{S,v'} - r_{max,v'}| \le \epsilon_1 r_{max,v'}$ (with probability at least $1 - \delta/2$) and $|A_{S,v'} - r| \le \epsilon_1 r$, it can be immediately verified that:

$$|r_{max,v'} - r| \le \frac{2\epsilon_1}{1-\epsilon_1} r \le \epsilon r$$

Thus, the theorem holds.

**Case 2 proof.** There are two sub-cases - Case 2a) $r < A_{S,v'} \le (1+\epsilon_1)r$, and Case 2b) $A_{S,v'} > (1+\epsilon_1)r$.

It is immediate that the proof of Case 1 is applicable to Case 2a. Therefore, the theorem holds for Case 2a.

Regarding Case 2b, we have $|A_{S,v'} - r_{max,v'}| \le \epsilon_1 r_{max,v'}$ with probability at least $1 - \delta/2$. This, together with $A_{S,v'} > (1+\epsilon_1)r$, immediately implies that with probability at least $1 - \delta/2$,

$$r_{max,v'} > r. \tag{4}$$

Moreover, suppose that $v''$ is the maximum element value that is smaller than $v'$ and $v'''$ is the value in sketch that is maximum but smaller than $v'$.[3] According to Algorithm 2, $A_{S,v''} = A_{S,v'''}$. According to the monotonic property in Lemma 6 and Algorithm 3, in order to be in Case 2

$$A_{S,v''} = A_{S,v'''} < (1-\epsilon_1)r. \tag{5}$$

We also have

$$r_{min,v'} = r_{max,v''} + 1. \tag{6}$$

Again, $|A_{S,v''} - r_{max,v''}| \le \epsilon_1 r_{max,v''}$ with probability at least $1 - \delta/2$. This, together with (5) and (6), implies $r_{min,v'} < r + 1$ with probability at least $1 - \delta/2$. Since $r > L = 1/\epsilon$, thus with probability at least $1 - \delta/2$,

$$r_{min,v'} < (1+\epsilon)r \tag{7}$$

These imply that one of the inequalities (4) and (7) does not hold with probability less than $\delta$. Thus, the theorem holds. □

3. Note: not every element value appears in the sketch generated by Algorithm 1.

Theorem 2 states that with the set of parameters, the data element returned by Algorithm 3 is $\epsilon$-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 $\epsilon$-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 the 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 $\epsilon$-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 runs 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 Unknown $m$

When the element ID domain is unknown (i.e., no priori knowledge about an upper-bound of $n_S$), we logically divide a data stream into several sub-streams such that each sub-stream corresponds to a different element ID domain and the domain lengths exponentially increase. We start with an initial $m_0$ (say, $m_0 = 64$). If there is an element ID outside $[1, m_0]$, then we create a new substream for the domain $[m_0+1, 2m_0]$. We can continue this process to create $[2m_0+1, 4m_0]$ on demands, $[4m_0+1, 8m_0]$, ... , $[2^i m_0+1, 2^{i+1}m_0]$, and so on. Then, we run Algorithm 1 for each element ID domain, respectively. Once a new element comes, we determine the element ID domain to which the new element belongs and apply Algorithm 1 accordingly to maintaining sketches regarding that element ID domain. It is not necessary to maintain $\mathcal{L}$ for each element ID domain. Instead, we maintain a global $\mathcal{L}$ only.

In each run (i.e. regarding a $[2^i m_0 + 1, 2^{i+1}m_0]$) of Algorithm 1 we maintain $l_i$ arrays each of which has $k_i$ elements. As with what we discussed in section 4.3, to run Algorithm 3 efficiently we sort elements in all sketches according to their values and then scan the sorted list. To guarantee confidence $1-\delta$, we retain failure probability $\delta/2^{i+2}$ with respect to each element ID domain $[2^i m_0 + 1, 2^{i+1}m_0]$ (for $i \geq 0$) and $\delta/2$ with respect to $[1, m_0]$. Similar to the proofs of the lemmas and theorems in section 4.1, the following theorem can be immediately verified.

**Theorem 3.** *Suppose that the element ID domain seen so far is $[1, m]$, $\gamma = \lceil\log_2\frac{m}{m_0}\rceil$, $\delta$, and $\epsilon$. Then, the union of the sketches generated can ensure a relative $\epsilon$-approximate result to a rank query $r$ with confidence at least $1-\delta$ if Algorithm 1 is applied to each domain $[2^i m_0 + 1, 2^{i+1}m_0]$ (for $0 \leq i \leq \gamma - 1$) with $l_i$ arrays and each array has $k_i = O(i + \log m_0 + \log\epsilon^{-1} + \log\delta^{-1})$ elements, and $l_i = O(\frac{1}{\epsilon^2}(i + \log\delta^{-1}))$, and if Algorithm 1 is also applied to $[1, m_0]$ with $l = O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ and $k = O(\log m_0 + \log\epsilon^{-1} + \log\delta^{-1})$.*

With those parameters in Theorem 3, the space is, thus, $O(\frac{1}{\epsilon^2}\log^3 m)$ if $m \geq \delta^{-1}$ and $m \geq \epsilon^{-1}$.

## 5 Time and Space Efficient Algorithms

While the framework (Algorithm 1) is space-efficient and guarantees a probabilistic relative $\epsilon$-approximate, each element is hashed into $\Omega(\frac{1}{\epsilon^2}\log\delta^{-1})$ arrays (subsketches). This potentially makes the algorithm less efficient. Our performance study in Section 6 demonstrates it can only handle a medium speed data stream in real time.

Consider that in many recent applications, to support on-line computation of high speed data streams is a crucial requirement. In this section, we propose two time-efficient algorithms following the framework in the last section. One retains the space requirement but there is no theoretical guarantee of accuracy with a high probability. Another retains the accuracy and leads to the average space requirement $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log n)$ though there is no worst case guarantee of the space requirement. Our performance study, nevertheless, indicates the both algorithms are also practically very space-efficient and highly accurate. Moreover, both of them are able to support on-line computation of high-speed data streams. Without loss of generality, we describe them with respect to the situation that $m$ is pre-known.

### 5.1 PCSA-like Algorithm

The first algorithm is an immediate application of the PCSA technique [18] to our algorithm, Algorithm 1. The basic idea is to hash each data element randomly to $\zeta$ arrays (subsketches) instead of the $l$ arrays (subsketches). Algorithm 1 may be modified as follows.

- First, we pick at random another $\zeta$ hash functions: $\{H_i : 1 \leq i \leq \zeta\}$ besides these $l$ hash functions in Algorithm 1, where each $H_i$ hashes $[1, m]$ 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 other parts in Algorithm 1 remain the same.

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

**Example 4.** *Regarding example in Figure 3, suppose* $\zeta = 1$, $H_1(x_1) = H_1(x_3) = 1$ *and* $H_1(x_2) = H_1(x_4) = 2$. *Figure 4 illustrates the merge result.*

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. 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 relatively stable.

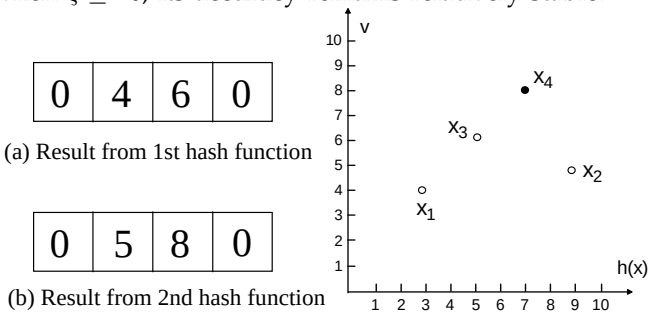



Fig. 4. PCSA-like Algorithm Fig. 5. K-Skyband

### 5.2 K-Skyband Algorithm

Our second algorithm follows the framework in the last section but is based on the BJKST Algorithm. We maintain $l$ subsketches. To estimate a $n_{S,v}$ for each $v$, the $k$ smallest distinct hashed values are required from each subsketch ($s_i$), respectively, with the corresponding element values not greater than $v$.[4] To address the situation that $n_{S,v} \leq k$ (e.g. a query rank not greater than $k$), we globally maintain a $\mathcal{L}$ to store the $k$ distinct data elements with smallest element values. As with Algorithm 1, once a new element comes we first examine if an element in $\mathcal{L}$ needs to be replaced by the new element. Meanwhile, we hash the new element $(x, v)$ into $\{s_i : 1 \leq i \leq l\}$ as follows.

Suppose that $l$ pairwise independent hash functions $\{h_i : 1 \leq i \leq l\}$ are randomly generated, each of which hashes $[1, m]$ to $[1, M]$ ($M = m^3$). The hashed data element $(x, v, h_i(x))$ is added to an $s_i$ (for $1 \leq i \leq l$) if current $s_i$ does not "$k$-dominate" $(x, v, h_i(x))$. An $s_i$ *$k$-dominates* $(x, v, h_i(x))$ iff there are $k$ data elements in $s_i$ with distinct hash values not greater than $h_i(x)$ and their element values not greater than $v$. Note that similar to the observations in Section 4 and in the light of BJKST Algorithm, in each $s_i$ an element, $k$-dominated by $s_i$, does not contribute to the estimation of $n_{S,v_0}$ for any $v_0$. we use $SK(s_i)$ to denote elements in $s_i$ which are not $k$-dominated by $s_i$.

**Example 5.** *Regarding five elements as shown in Figure 1, suppose* $l = 1$, $k = 2$, $h_1(x_1) = 3$, $h_1(x_2) = 9$, $h_1(x_3) = 5$ *and* $h_1(x_4) = 7$, *Figure 5 illustrates that the element* $x_4$ *is* $k$*-dominated.*

**Theorem 4.** *Each current* $SK(s_i)$ *for* $1 \leq i \leq l$ *has the following properties.*

- P1: *If* $s_i$ *currently* $k$*-dominates an element* $e$ ($\in s_i$), $e$ *will never be used by our query algorithm for any* $v$.
- P2: *For each element* $e = (x, v, h_i(x) \in SK(s_i))$, *either*
  - P2a: *there is a* $v_0$ *such that* $h_i(x)$ *is the* $k$*th smallest among the elements in* $D_{s_i, v_0^-}$ *where* $D_{s_i, v_0^-}$ *denotes the set of elements in* $s_i$ *with element values not greater than* $v_0$, *or*
  - P2b: $h_i(x)$ *is one of the* $k-1$ *smallest distinct hash values in* $s_i$.

*Proof:* We prove P1 and P2 as follows.

*Proof of P1.* According to the definition, if $e = (x, v, h_i(x))$ is $k$-dominated by $s_i$ then there are at least $k$ elements in $s_i$ with distinct hashed values smaller than $e$ and element values not greater than $v$. Consequently, our query algorithm will never choose $h_i(x)$. This is because that updates to $SK(s_i)$ retain the property that there are at least $k$ elements in $s_i$ with distinct hash values smaller than $e$ regardless how many new elements come. Thus, P1 holds.

*Proof of P2.* If $(x, v, h_i(x))$ does not belong to category P2b, then there are $\lambda$ ($\lambda > k-1$) elements in $s_i$ with distinct hash values smaller than $h_i(x)$.

Let $v_0$ be the value of the element with the $(k-1)$th smallest element value among these $\lambda$ elements. Since $e$ is in $SK(s_i)$, among these $\lambda$ elements there are only $\lambda_1$ ($\lambda_1 \leq k-1$) elements with element values smaller than $v$. Therefore, $v_0 \leq v$; that is, $e$ belongs to category P2a. □

Note that P1 in Theorem 4 implies that we only need to maintain $SK(s_i)$ instead of $s_i$. Clearly, an element in the category P2a will be used in an approximate query with value $v_0$. Moreover, any element with one of the $k-1$ smallest distinct hashed values (category P2b) may be used in query processing for the whole stream once future elements have hashed values smaller than the current $k-1$ smallest values; thus, it needs to be kept. Therefore, Theorem 4 implies that $SK(s_i)$ is the minimum number of elements we should keep.

To speed-up the computation, the $k$-dominance is not examined for hashing in each $(x, v)$. Instead, we initially give a space upper-limit $\Gamma$ (in terms of the number of elements). We add a $(x, v, h_i(x))$ to each $s_i$ (for $1 \leq i \leq l$), respectively, till $\sum_{i=1}^{l} |s_i|$ reaches the limit $\Gamma$. Then, we do space compression in each $s_i$, respectively, by probing the $k$-dominance relationship once the upper-limit is reached. After the space compression, if the total number of tuples left in $\sum_{i=1}^{l} s_i$ is greater than $\Gamma/2$, then we increase $\Gamma$ to $2\Gamma$. These describe our sketch construction algorithm, Algorithm K-Skyband.

4. To minimize the number of elements in each subsketch, we introduce the "$k$-dominance" relationship later so that only necessary information is kept.

In the compression phase, in each $s_i$ we remove all elements that are $k$-dominated by $s_i$; that is, we only keep the elements in $s_i$, which are not $k$-dominated by $s_i$. To do this efficiently, we first build two sorted lists $eV$, and $eF$ on element values and hash values, respectively. Then, we scan the two lists once to remove the elements $k$-dominated by others. Here, $eV$ is a sorted linked list (pointing to each element, respectively) decreasingly based on the lexicographical order of $(v, h_i(x))$, while $eF$ is a sorted linked list increasingly based on hash values $h_i(x)$. Moreover, to enforce the distinct hash value condition, in the element set $s_i$ if there are several elements with the same feature value, we keep only one element - the element with the minimum value. We present our compression technique in Algorithm 4.

**Algorithm 4** Compression

**Input:**
$k$, two sorted linked lists $eF$ and $eV$ pointing to the elements in $s_i = \{(x, v, h_i(x))\}$, respectively.
**Output:**
all elements in $s_i$, which are not $k$-dominated by $s_i$.
**Description:**
1: $ef \leftarrow k$th element $\in eF$; $ev \leftarrow$ 1st element $\in eV$;
2: **while** $ef \neq null$ **do**
3: $a \leftarrow$ the element in $s_i$ pointed by $ef$;
4: $b \leftarrow$ the element in $s_i$ pointed by $ev$;
5: **if** $a.h < b.h$ **then**
6: $s_i \leftarrow s_i - \{b\}$;
7: $ev \leftarrow ev.next$;
8: **else**
9: $ev \leftarrow ev.next$;
10: $ef \leftarrow ef.next$;
11: return $s_i$;

Here, $a.h$ and $b.h$ are hash values of element $a$ and $b$ respectively. And $ev.next$ and $ef.next$ represent the next elements in $eV$ and $eF$, respectively. In Algorithm 4, if an element $e$ is removed from $s_i$ then the corresponding elements in $eV$ and $eF$ pointing to $e$ are also removed. It can be immediately verified that Algorithm 4 is correct; that is, it always outputs all elements in $s_i$, which are not $k$-dominated by $s_i$.

**Remark 1.** *In [34], it proposes to use k-skyband to answer top-k queries over sliding windows. The technique is to simply increase the dominance count of the elments which are dominated by the new incoming elements. It is efficient when k is small (a typical situation in top-k queries); nevertheless, it is inefficient when k is large - a typical situation in our problem to guarantee $\epsilon$-approximation for a small $\epsilon$. Thus, the technique in [34] are not applicable to our problem setting where we need to process data streams in real time.*

To estimate a $n_{S,v}$, Algorithm 2 may be modified by changing lines 5-10 as follows.

- Firstly, for each sketch $s_i$ $(1 \leq i \leq l)$ select all elements whose hash values are distinct and not greater than $v$.
- Secondly, use the query technique in Section 3.2 to query these $l$ selected results - $s_i|_{v^-}$ $(1 \leq i \leq l)$.

This, combining with Algorithm 3, returns an answer to a rank query. Based on Lemma 4, using similar proof techniques to those in section 4.1 it can be immediately verified that if $m > \delta^{-1}$, $k = O(\frac{1}{\epsilon^2})$, $L = k$, and $l = O(\log \delta^{-1})$, then an element returned by Algorithm 3 against the subsketches by Algorithm K-Skyband (as described above) is relative $\epsilon$-approximate with confidence $1 - \delta$.

Note that in Algorithm $k$-Skyband, we may have to keep all distinct elements in the worst case; nevertheless, using similar arguments to that in [6] the following theorem is immediate.

**Theorem 5.** *In a 2-d set $s = \{(x_i, y_i) : 1 \leq i \leq n\}$ with $n$ elements, assume all $x$ and $y$ values are unique, $x$ and $y$ are independent, each $x$ follows the same distribution, and each $y$ also follows the same distribution. Then, the k-skyband $SK(s)$ has the expected number of elements $O(k \ln(\frac{n}{k}))$ where $x_i$ corresponds to a hashed value and $y_i$ corresponds to an element value.*

*Proof:* Without loss of generality, we assume that $y_i < y_j$ if $i < j$. For $1 \leq i \leq n$, let the random variable $X_i = 1$ if $(x_i, y_i)$ is a $k$-skyband element, otherwise, $X_i = 0$. The expected number of $k$-skyband elements is $E(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} E(X_i) = \sum_{i=1}^{n} P(X_i = 1)$ where $P$ denotes the probability.

Clearly, the value (0 or 1) of each $X_i$ (for $1 \leq i \leq n$) depends on $\{(x_j, y_j) : 1 \leq j \leq i-1\}$ as $y_j$ is increasingly ordered and any element $(x_j, y_j)$ for $j > i$ does not dominate $(x_i, y_i)$. Note that every element $(x_i, y_i)$ belongs to $SK(s)$ when $i \leq k$ ; thus, the probability of $X_i = 1$ for $i \leq k$ is 1.

For $i > k$, $(x_i, y_i)$ is a $k$-skyband element iff $x_i$ is one of the $k$ smallest values in $\{y_j : 1 \leq j \leq i\}$. Note that each $y_j$ has the same probability to fall into the $k$ smallest values as each $y_j$ follows the same distribution, and we assume the independence among all $y_j$ and between $x$ and $y$. Thus, $P(X_i = 1) = \frac{k}{i}$.

It can be immediately verified

$$\begin{aligned} E(\sum_{i=1}^{n} X_i) &= k + \sum_{i=k+1}^{n} P(X_i = 1) \\ &= k \times (1 + H_{1,n} - H_{1,k}). \end{aligned}$$

Here, $H_{1,n} = \ln(n)$, the Theorem immediately follows. □

To ensure relative $\epsilon$-approximate with $1 - \delta$ confidence, $k$ is chosen to $O(\frac{1}{\epsilon^2})$ and $\log \frac{1}{\delta}$ subsketches are maintained. Consequently, the number of elements maintained in $\log \frac{1}{\delta}$ subsketches is $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} \log \epsilon^2 n)$ on average if all element values are unique, object ID and element value are independent, object ID of each element follows the same distribution, and the value of each element follows the same distribution.

Our experiment demonstrates that in practice, this algorithm requires less space than the FM-based techniques in Section 3. In fact, we can immediately show that $\Gamma$ is at most 4 times of the actual space requirement in terms of the number of tuples. Due to the batch compression technique, Algorithm K-Skyband now runs in time $O(\log \Gamma)$ per data element on average.[5] More-

5. To amortize the computation of the space compression, in our implementation we maintain an upper-limit $\Gamma_i$ for each subsketch $s_i$ to determine when an $s_i$ needs to be compressed and $\Gamma_i$ needs to be doubled.

over, our query algorithm runs in time $O(\Gamma \log \Gamma)$ if subsketches are not compressed (thus, not pre-sorted); however if the rank falls into $\mathcal{L}$ then it takes $O(\log L)$ time to answer a query.

## 6 Performance Evaluation

In this section, we only present the evaluation results of our techniques. This is because our techniques significantly improve the existing duplicate-insensitive quantile techniques [14], [25], [31] in both precision guarantee and space requirement. These existing techniques only guarantee uniform rank error $\epsilon n$; such a rank guarantee does not provide much useful approximation information for small ranks (i.e., no constant bound for relative rank errors). In fact, our implementation demonstrates that these existing techniques return the same data element for RQ queries with $r < c\epsilon n$ ($c$ is different regarding different algorithms, datasets, and executions); this is consistent with the above observation. Moreover, they require an order of $\frac{1}{\epsilon}$ more space than our technique even though ours guarantee the relative rank error $\epsilon r$ instead of the uniform rank error $\epsilon r$. Below is a list of our techniques to be evaluated.

SE. Algorithm 1: the space-efficient sketch construction algorithm in Section 4.
SE-PCSA. The technique in Section 5.1.
KSKY. Algorithm $k$-Skyband in Section 5.2: sketch construction technique.

We evaluate their space and time efficiency, as well as accuracy in terms of the relative errors. The corresponding query algorithms are also implemented.

In our experiments, two synthetic datasets are generated, *Random* and *Semisort*. In a Random dataset, each data element (object) value is randomly generated following a uniform distribution. Data elements in a Semisort dataset are partitioned into groups with the average group size $5K$ such that values in later groups are greater than those in earlier groups, while the values within each group are randomly generated. We use the *duplication ratio*, $\frac{N-n}{N}$, to control the total number of duplicated data elements. For each synthetic dataset, we first generated $n$ distinct elements so that element IDs are picked at random in the element domain and values follow, accordingly, the Random model or the Semisort model. The remaining $N - n$ data elements randomly duplicate the exiting data elements.

A real dataset *WCH* (World Cup 98's HTTP request data) is downloaded from the Internet Traffic Archive [27] and is used in our experiments. It consists of 17 million records of requests made to the 1998 World Cup Web site between April 30, 1998 and July 26, 1998. Each record contains time stamp, clientID, URLID, and package size (PSIZE). In the dataset, we use ⟨clientID, URLID, PSIZE⟩ as the element ID to identify a record and rank data elements according to their package size (PSIZE). 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. Table 2 below lists the parameters that potentially have an impact on our performance study. In our experiments, all parameters use default values unless otherwise specified.

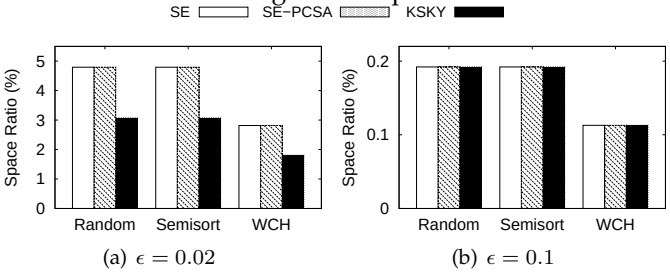
| Notation | Definition (Default Values) |
|---|---|
| $N$ (synth. data) | Dataset Size ($10M$) |
| $\alpha$ (synth. data) | Duplication Ratio (0.2) |
| $\zeta$ (SE-PCSA) | Number of times to hash an item (10) |
| $\epsilon$ | Guaranteed Precision (0.02) |
| $1-\delta$ | Confidence (0.95) |

TABLE 2
System Parameters

*To "discount" O notation in space requirements of SE, SE-PCSA, and KSKY, respectively, we adopt the same constant factor 2. That is, $l = \frac{2}{\epsilon^2}\log\delta^{-1}$ in SE and SE-PCSA, and $k = \frac{2}{\epsilon^2}$ in KSKY. In SE and SE-PCSA, we choose $k = 32$ because we use the public code from Massive Data Analysis Lab to generate hash functions [33] and $2^{32}$ is large enough to accommodate massive number of distinct data elements; we also choose $L = \frac{1}{\epsilon}$. In KSKY, we choose $L = k$, $l = \log\frac{1}{\delta}$. We also modify the code in [33] to generate hash functions in KSKY.*

### 6.1 Space Efficiency

We record the maximal space size (i.e. the maximal number of elements) of sketch, by each algorithm, during the continuous processing of a dataset. The ratio of such sketch size to the total number of elements processed is called *space ratio*. Note that the space requirements in SE and SE-PCSA are the same and fixed for given $m$, $\epsilon$, and $\delta$, while the space ratio changes when stream sizes change. The space required in KSKY is "opportunistic" as it is not fixed during the computation.




(a) $\epsilon = 0.02$ (b) $\epsilon = 0.1$

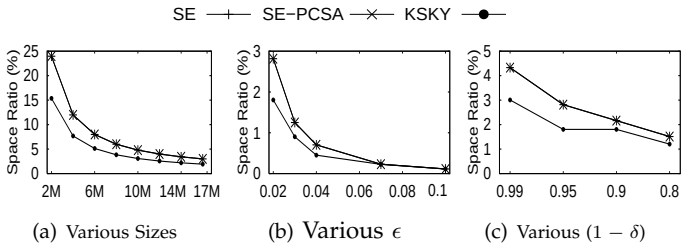
Fig. 6. Space Efficiency against Different Dataset

The first experiment results are presented in Figure 6. They demonstrate that KSKY requires the smallest space especially when $\epsilon$ is small. The second experiment evaluate the possible impacts from data sizes, $\epsilon$, and $\delta$. The evaluation results against the real dataset (WCH) are presented in Figure 7 where the experiments regarding Figures 7(b) and 7(c) are against the whole dataset. Again, they demonstrate that KSKY requires the smallest space.

### 6.2 Evaluating Accuracy

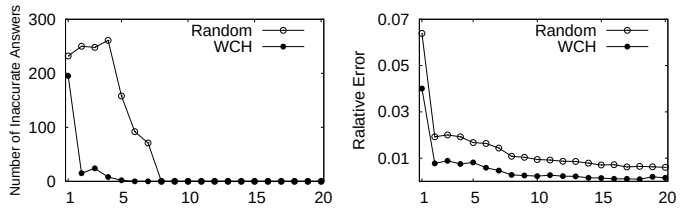
To evaluate accuracy, we randomly generated 1K rank queries each, respectively, for synthetic dataset Random and the real dataset WCH, to span the corresponding whole domain of feasible ranks. We record the average relative error.

The results of the first experiment, against the real dataset WCH, are reported in Figure 8. We study an




(a) Various Sizes (b) Various $\epsilon$ (c) Various $(1-\delta)$

Fig. 7. Impact of Sizes, $\epsilon$ & $\delta$

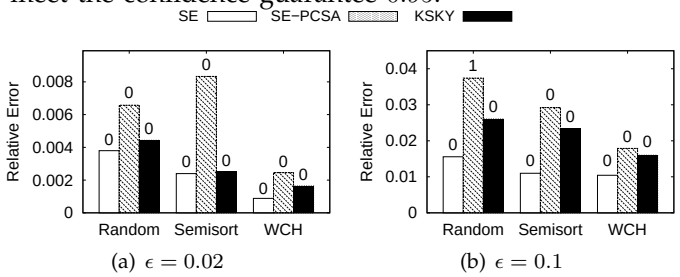



(a) "Bad" Answers (b) Relative Error

Fig. 8. Accuracy of SE-PCSA Variants

impact of different values of $\zeta$ (i.e., the number of subsketches an element will be hashed in SE-PCSA).[6] As demonstrated by Figure 8(a), when $\zeta = 10$ the number of query results exceeding the relative error guarantee is $0$, and an improvements of relative errors becomes less significant after $\zeta \geq 10$.

The second experiment is conducted against the $3$ different datasets and is reported in Figure 9. It shows that SE provides the highest accuracy, while KSKY is the second. The numbers ($0$ or $1$) above those "bar figures" are the number of answers exceeding the designated relative error guarantee $\epsilon = 0.02$ even though they all meet the confidence guarantee $0.95$.




(a) $\epsilon = 0.02$ (b) $\epsilon = 0.1$

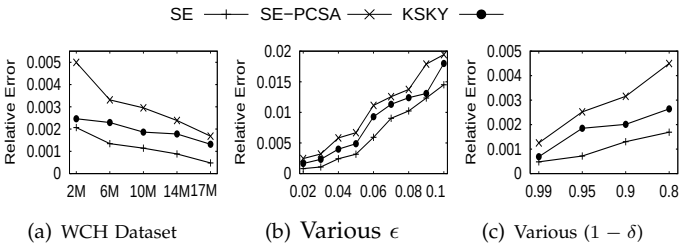
Fig. 9. Accuracy against Different Dataset

The third experiment evaluates possible impacts from data sizes, $\epsilon$, and $(1-\delta)$. The experiment is conducted against real dataset WCH and is reported in Figure 10. It also shows that SE always provides the highest accuracy and KSKY is the second accurate. We report that all answers obtained against the sketches by SE or KSKY or SE-PCSA satisfy the corresponding probabilistic error guarantees though SE-PCSA leads to $4$ answers exceeding a designated relative error guarantee $\epsilon$ for the setting - $\epsilon = 0.02$ and $(1-\delta) = 0.8$.

### 6.3 Time Efficiency

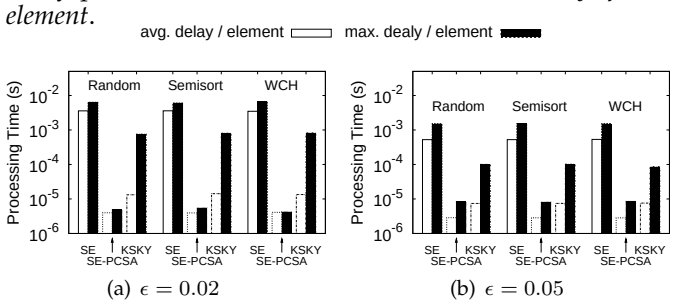
The cost of processing one data element may be too small to be recorded accurately (especially for SE-PCSA and KSKY), we record the average time for processing every batch of 1K elements as *the delay of one element*. In addition, we also record the maximum value of such delay per data element time as *the maximal delay of each element*.

6. In SE-PCSA, different values of $\zeta$ will not make any difference in space requirement if the other parameters are the same.




(a) WCH Dataset (b) Various $\epsilon$ (c) Various $(1-\delta)$

Fig. 10. Effect of Sizes, $\epsilon$, & $(1-\delta)$

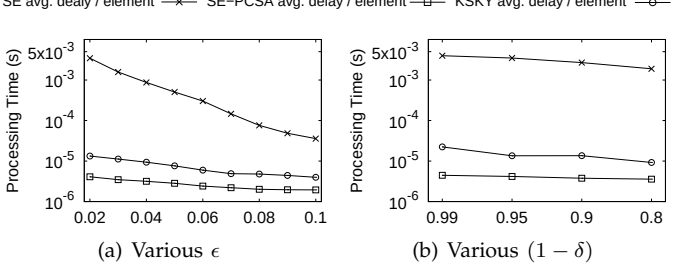



(a) $\epsilon = 0.02$ (b) $\epsilon = 0.05$

Fig. 11. Time Evaluation over Different Datasets

The first experiment is conducted against the $3$ datasets Random, Semisort, WCH. The experiment results are reported in Figure 11. They indicate that SE can only process a medium speed data stream online - $300$-$400$ elements per second when $\epsilon = 0.02$ and about $2500$ elements per second when $\epsilon = 0.05$. However, both KSKY and SE-PCSA can process high speed data streams. They can process at least $75,000$ data elements per second even with $\epsilon = 0.02$.

Next, we also examine possible impacts of $\epsilon$ and $(1-\delta)$. We conduct experiments on the real dataset WCH. We vary $\epsilon$ with fixed $(1-\delta) = 0.95$, as well as vary $(1-\delta)$ with fixed $\epsilon = 0.02$. We record the average delay per data element. The experiment results are reported in Figure 12. They demonstrate that the processing costs of SE increase dramatically as $\epsilon$ decreases due to the factor of $1/\epsilon^2$ in the time complexity, while varying $\delta$ does not change the processing costs so dramatically.




(a) Various $\epsilon$ (b) Various $(1-\delta)$

Fig. 12. Time Evaluation over Different $\epsilon$ and $\delta$

The third experiment set evaluates possible impact of duplication ratios. As we cannot change duplication ratios in real dataset, the dataset Random is used for this purpose. Figure 13 shows the experiment results where average delay per data element is used. They demonstrate that our techniques are insensitive to different duplication ratios.

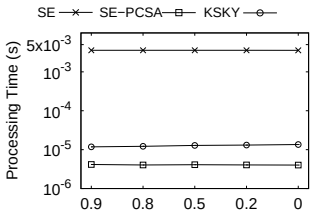



Fig. 13. Various $\alpha$

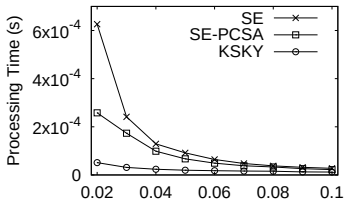



Fig. 14. Query Time

In the last experiment, we evaluate the 3 (rank) query processing algorithms against the real dataset WCH. We vary $\epsilon$ from 0.02 to 0.1. The average response time of the $1K$ queries for each algorithm, after pre-sorting the subsketches, is reported in Figure 14. It shows that querying sketches by SE-PCSA or KSKY is significantly more efficient than those of SE. This is because that the sketch size by KSKY is significantly lower that by SE. Although SE and SE-PCSA have the same space, the number of non-zero values in the sketch by SE-PCSA is significantly lower than that in SE.

### 6.4 Summary

As a short summary, our performance evaluation demonstrates that the proposed techniques are not only space and time efficient but also of highly accurate. Among these three algorithms, SE is the most accurate; SE-PCSA has the fastest processing speed; and KSKY takes the smallest space. With the requirement of processing high speed data streams, SE-PCSA or KSKY is a better choice than SE.

## 7 Applications

The techniques we developed in this paper may be applied into the following problems. We state them based on our techniques in section 4 with pre-knowledge about an upper bound $m$ of $n_S$; the other techniques can also be applied in a similar way.

### 7.1 Sliding Window Computation

In some applications, users might be more interested in statistics of recent data rather than that of the entire history. Regarding the stock market example presented in section 1, the purchase trends of the most recent week or last one million deals might be preferred by some buyers. This is referred as "sliding window" model and has been extensively studied in many recent works. In this subsection, we will investigate the problem of duplicate-insensitive order statistics computation over the count-based sliding window. That is, given rank $r$ and $t \leq W$, find the element with rank $r$ in $D_{S,t^+}$ where $D_{S,t^+}$ represents distinct elements in the last $t$ elements of the data set $S$ and $W$ is the maximal window size.

A simple solution is to keep the most recent $W$ elements. However it is infeasible when $W$ is very large which is common in many applications. We can extend our FM sketch based technique to support the sliding window queries. Besides the value and id information, we also need to keep the timestamp for each incoming elements $e$, denoted by $e.ts$. Without loss of generality, we assume the recent elements have larger $ts$ values. In the SE algorithm, for each position of the subsketches, instead of keeping the minimal element value hashed to it so far we maintain a set of elements. Then for given $t \leq W$, we can find out the minimal value of the last $t$ elements. Let $E$ denote a set of elements which are hashed to a particular position in a subsketch. For two elements $e1$, $e2 \in E$, $e1$ *dominates* $e2$ if $e1.v \leq e2.v$ and $e1.ts \geq e2.ts$. It is immediate that $e2$ is redundant as it does not contribute to above query for any $t$ because of the existence of $e1$. Consequently, we only need to keep the elements which are not dominated by any other elements in $E$. They are 2-dimensional skyline points[7] of $E$ with expected size $O(\ln |E|)$ if element values are independent with the arriving orders. For each FM subsketch, the probability of an element being mapped to $i$-th position $(0 \leq i < k)$ is $\frac{1}{2^{i+1}}$, so the expected number of elements kept in each subsketch is $O(\ln \frac{w}{2} + \ln \frac{w}{2^2} + \ldots + \ln \frac{w}{2^k}) = O(\ln k \ln w)$ where $w$ is the total number of distinct elements within latest $W$ elements. Together with space $k$ used in SE algorithm, the expected space complexity of each subsketch is $O(\ln k \ln w + k)$. As to the $\mathcal{L}$ in the SE algorithm, we also need to keep more elements such that for any $t \leq W$ the $\mathcal{L}$ can return the smallest $\frac{1}{\epsilon}$ distinct elements within $t$ latest elements. In the spirit of Theorem 4, it is easy to verify that keeping $\frac{1}{\epsilon}$-skyband of the most recent $W$ elements will suffice. According to Theorem 5, the expected size of $\mathcal{L}$ is $O(\frac{1}{\epsilon} \ln \epsilon w)$. Then we have the following theorem.

**Theorem 6.** *If $m \geq \epsilon^{-1}$, $m \geq \delta^{-1}$, $0 < \delta < 1$ and $0 < \epsilon < 1$, with expected space of $O(\frac{1}{\epsilon^2} \log \delta^{-1} (\ln w \ln \ln m + \log m))$, we can find the relative $\epsilon-$ approximate answer for given rank $r$ with probability at least $1 - \delta$ regarding to arbitrary window size $t \leq W$ where $w$ is the number of distinct elements in the most recent $W$ elements.*

### 7.2 Value-based Rank Queries.

It is immediate that the sketches generated by Algorithm 1, with $O(\frac{1}{\epsilon^2} \log \delta^{-1} \log m)$ space, can guarantee the relative $\epsilon$-approximation (with confidence $1-\delta$) for counting the number of distinct elements with values smaller (or not greater) than a given $v$.

### 7.3 Distinct Heavy Hitters

Suppose each data element is represented by $(id, i)$ where $id$ is the element ID and $i$ means the $i$th item hit by the element. In Algorithm 1, we keep $i$ instead of $v$. Let $c(i)$ denote the number of hitters of item $i$. Let $A_{S,i}$, obtained by Algorithm 2, be the estimation of the number of distinct elements hitting the items from 1 to $i$. It is immediate that if the parameters used in Theorem 2 are modified accordingly to $\epsilon/2$ and $\delta/2$ instead of $\epsilon$ and $\delta$, then $c(i) - \epsilon n \leq A_{S,i} - A_{S,i-1} \leq c(i) + \epsilon n$ with confidence $1-\delta$. Moreover, the items, which (whose subindexes) do not appear in the sketch, have the number of hitters not greater than $\epsilon n$ with the confidence $1-\delta$. This means our techniques can be used to get an $\epsilon$-approximate solution, with confidence $1-\delta$ and space $O(\frac{1}{\epsilon^2} \log \delta \log m)$, for the heavy hitter problem over data streams by discounting duplicated hitters. It improves the space requirement $O(\frac{1}{\epsilon^3} \log \frac{1}{\delta} \log^2 m)$ in [31].

Viewing $i$ as a graph nodeID, the above technique can be immediately used to get an $\epsilon$-approximate solution for heavy hitters in multi-graphs. The space requirement, as given above, is similar to [14].

### 7.4 Counting Inversions

Counting the number of inversions in massive datasets has many applications including ranking aggregation [16]. Following is an example of counting the number of inversions.

**Example 6.** *Suppose four elements* $(x_1, 4)$, $(x_2, 6)$, $(x_3, 5)$, $(x_4, 8)$ *arrive in sequence order, then the number of inversions is* 1.

In [24], a randomized algorithm is proposed to achieve the relative $\epsilon$-approximation with space $O(\frac{1}{\epsilon^3}\log^2 N)$ and confidence at least $1 - 1/N$, where $N$ is a pre-known upper-bound of the number of elements and no duplicated elements are allowed. With the same assumption that no duplicates are allowed and $N$ is pre-known, using the above ranking value technique immediately implies that we can apply Algorithm 2 per new data element $e$ to rank the value of $e$ against the sketches maintained by Algorithm 1. Then, adding such counts together can guarantee the relative $\epsilon$-approximation for the inversion counting problem with confidence at least $1 - 1/N$ if space used is $O(\frac{1}{\epsilon^2}\log^2 N)$ and $N \geq 1/\epsilon$. To ensure the global failure probability less than $1/N$, we enforce the failure probability for ranking each new value less than $\frac{1}{N^2}$. Since no duplicated objects, $m = N$. Clearly, our technique improves the technique in [24] by a factor of $\frac{1}{\epsilon}$.

### 7.5 Quantile Queries against Distinct Values

In some applications, one may want to know a distribution over distinct values. Our techniques can immediately support such a requirement with a relative error guarantee. In Algorithm 1, instead of hashing each object ID it hashes each object value.

With such a modification, Algorithms 1 & 2 & 3 can immediately guarantee relative $\epsilon$-approximate for rank queries (thus, quantile queries) with space $O(\frac{1}{\epsilon^2}\log\delta^{-1}\log m)$ if $m \geq \epsilon^{-1}$ and $m \geq \delta^{-1}$, and confidence at least $1 - \delta$.

### 7.6 Fault-tolerant Distributed Quantiles

To address a high fault rate over a P2P and/or sensor network, a multi-path based routing approach is widely employed in many distributed statistic computation applications. Since more than one copy of a particular subsketch at a node are sent to the target node via different paths, the problem of over-counting arises. Recently, several novel techniques are proposed in [35], [9], [31] to resolve the over-counting issue for aggregates such as count, sum, heavy hitter, and quantiles (with precision guarantee $\epsilon N$) over a network.

Algorithm 1 has the property that two local sketches can be merged if they are created with the same $k$ and set of $l$ hash functions, while the precision guarantee may be retained. Then, all of the local sketches can be merged into a global sketch at the target node via multi-paths. Then we present the following claim:

**Theorem 7.** *There is a distributed algorithm, with transmission load* $O(\frac{\Upsilon}{\epsilon^2}\log\delta^{-1}\log m)$ *at each node and probability at least* $1 - \delta$*, that guarantees a relative* $\epsilon$*-approximate quantile query result. Here,* $\Upsilon$ *is the maximal number of sketch copies sent from each node.*

*Proof:* According to Lemma 2, at each node we can simply merge the FM sketches from its children and the local FM sketch. Based on Theorem 2, with probability at least $1-\delta$, we can get the relative $\epsilon$-approximate quantile query result against the elements from the subtree if the size of each FM sketch is $O(\frac{1}{\epsilon^2}\log\delta^{-1}\log m)$, Since there are at most $\Upsilon$ copies of FM sketches being sent from a particular node, the maximal transmission load is at most $O(\frac{\Upsilon}{\epsilon^2}\log\delta^{-1}\log m)$ for each node. □

## 8 Conclusions

In this paper, we investigate 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. Besides proven accuracy and space guarantees, our algorithms are also efficient enough to support on-line computation of very high speed data streams with an element arrival rate up to 75K/second. Moreover, we show that our techniques may be extend to sliding window model and other problems, such as finding distinct heavy hitters, counting inversions, fault-tolerant distributed quantiles computation, etc.

## References


[1] M. Ajtai, I. S. Jayram, R. Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *STOC 2002*.

[2] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS*, 2004.

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

[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *RANDOM*, 2002.

[5] M. Bawa, H. G. Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network. Technical report, Stanford University, 2003.

[6] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J.ACM*, 25(4):536–543, 1978.

[7] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[8] J.-Y. Chen, G. Pandurangan, and D. Xu. Robust computation of aggregates in wireless sensor networks: distributed randomized algorithms and analysis. In *IPSN*.

[9] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, page 449, 2004.

[10] C. Cormode and S. Muthukrishnan. An improved data stream: The count-min sketch and its applications. In *Latin American Informatics*, 2004.

[11] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In *SIGMOD*, pages 25–36, 2005.

[12] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Effective computation of biased quantiles over data streams. In *ICDE*, 2005.

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

[14] G. Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *PODS*, 2005.

[15] G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In *ICDE*.

[16] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, 2001.

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

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

[19] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB*, 2002.

[20] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, 2001.

[21] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS*, 2004.

[22] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *STOC 2001*.

[23] S. Guha and A. McGregor. Approximate quantile and the order of the stream. In *PODS*, 2006.

[24] A. Gupta and F. Zane. Counting inversions in lists. In *SODA*, 2003.

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

[26] J. Hershberger, N. Shrivastava, S. Suri, and C. Toth. Adaptive spatial partitioning for multidimensional data streams. In *ISAAC*, 2004.

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

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

[29] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS*, 2003.

[30] X. Lin, H. Lu, J. Xu, and J. X. Yu. Continuously maintaining quantile summaries of the most recent n elements over a data stream. In *ICDE*, 2004.

[31] A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *SIGMOD*, 2005.

[32] 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*, 1999.

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

[34] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*, pages 635–646, 2006.

[35] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys*, pages 250–262, 2004.

[36] 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.

[37] M. Yiu, N. Marmoulis, and Y. Tao. Efficient quantile retrieval on multi-dimensional data. In *EDBT*.

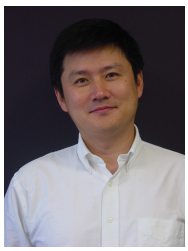
[38] Y. Zhang, X. Lin, J. Xu, F. Korn, and W. Wang. Space-efficient relative relative error order sketch over data streams. In *ICDE*.


**Ying Zhang** is a Research Fellow in the School of Computer Science and Engineering, the University of New South Wales. He received his BSc and MSc degrees in Computer Science from Peking University, and PhD in Computer Science from the University of New South Wales. His reserch interests include query processing on data stream, uncertain data and graphs.

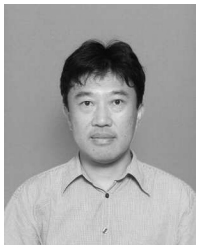
**Xuemin Lin** is a Professor in the School of Computer Science and Engineering, the University of New South Wales. He has been the head of database research group at UNSW since 2002. Before joining UNSW, Xuemin held various academic positions at the University of Queensland and the University of Western Australia. Dr. Lin got his PhD in Computer Science from the University of Queensland in 1992 and his BSc in Applied Math from Fudan University in 1984. During 1984-1988, he studied for PhD in Applied Math at Fudan University. He currently is an associate editor of ACM Transactions on Database Systems. His current research interests lie in data streams, approximate query processing, spatial data analysis, and graph visualization.

**Yidong Yuan** received a Ph.D. degree in Computing Science from the University of New South Wales, Australia, in 2007. Yidong also received his B.S. and M.S. degree in Computer Science from Shanghai Jiao Tong University, P.R. China, in 1998 and 2001.

**Masaru Kitsuregawa** received a Ph.D. degree from the University of Tokyo in 1983. He is currently a professor and a director of the Center for Information Fusion at the Institute of Industrial Science of the University of Tokyo. His current research interests include database engineering, web mining, parallel database processing/data mining, advanced storage system architecture, digital earth etc. He is serving a science advisor of Ministry of Education(MEXT) and also a principal investigator of one the largest IT project, 'Cyber Infrastructure for Information Explosion Era(2005-2010)' funded by MEXT where more than 200 researchers join. He also serves an advisor of 'Grand Information Voyage Project (2007-2009)' by METI. He is a fellow and a vice president of Information Processing Society of Japan and a fellow of Institute of Electronic, Information, and Communication Engineers Japan. He is serving Asian Coordinator of IEEE TCDE.



**Xiaofang Zhou** is a Professor of Computer Science at the University of Queensland, Australia, leading the Data and Knowledge Engineering Research Group at UQ. Xiaofang is the Convener and Director of Australia Research Council (ARC) Research Network in Enterprise Information Infrastructure (EII), and a Chief Investigator of ARC Centre of Excellence in Bioinformatics. He received his BSc and MSc degrees in Computer Science from Nanjing University, China, and PhD in Computer Science from the University of Queensland. His research interests include spatial and multimedia databases, data quality, high performance query processing, Web information systems and bioinformatics.



**Jeffrey Xu Yu** received his B.E., M.E., and Ph.D in computer science, from the University of Tsukuba, Japan, in 1985, 1987 and 1990, respectively. Currently, Dr. Yu is a Professor in the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong. His current main research interests include keywords search in relational databases, graph mining, graph query processing, graph pattern matching, uncertainty query processing, and data stream processing. Dr. Yu has published over 190 papers including papers published in reputed journals and major international conferences, and served/serves in over 150 organization committees and program committees in international conferences/workshops.