

Supporting Sliding Window Queries for Continuous Data Streams *

Lin Qiao [†] Divyakant Agrawal Amr El Abbadi

Department of Computer Science, University of California, Santa Barbara

Email: {lqiao, agrawal, amr}@cs.ucsb.edu

Abstract

Although traditional databases and data warehouses have been exploited widely to manage persistent data, a large number of applications from sensor network need functional supports for transient data in the continuous data stream. One of the crucial functions is to summarize the data items within a sliding window. A sliding window contains a fixed width span of data elements. The data items are implicitly deleted from the sliding window, when it moves out of the window scope. Several one-dimensional histograms have been proposed to store the succinct time information in a sliding window. Such histograms, however, only handle the data items with attribute values in unary domains. In this paper, we explore the problem of extending the value to a multi-valued domain. A two-dimensional histogram, the hybrid histogram, is proposed to support sliding window queries on a practical multi-valued domain. The basic building block of the hybrid histogram is the exponential histogram. The hybrid histogram is maintained to capture the changes of data distribution. To further compress the exponential histograms, we propose a condensed exponential histogram without losing the error bound. Results of an extensive experimental study are included to evaluate the benefits of the proposed technique.

1 Introduction

In smart environments, small wireless devices will provide accesses to information anytime, anywhere and wireless sensor networks plays a key role in sensing, collecting, and disseminating information about environmental phenomena. There is a wide range of applications for sensor networks with different requirements, such as surveillance, temperature and humidity sampling, metropolitan area traffic study, tornado motion analysis, etc. Traditional databases and data warehouses have been exploited widely for man-

aging and processing information. However, they are generally designed for persistent datasets with finite volume. Hence functional supports are needed for transient data in the form of continuous streams with unbounded length from sensor networks. Because the large volume of incoming data cannot be stored entirely and accessed randomly in bounded storage and time [12], a single pass algorithm to process the streaming data is required.

According to different application requirements, the scope of data streams has been categorized into two different alternatives [10]. The *landmark window* identifies certain starting landmark in the stream and incorporates tuples since that point until the present. The *sliding window* is featured with a fixed length time-span of the data elements. Take lightning prediction as an instance. The ability to predict the onset of lightning in thunderstorms would have utility for those engaged in aviation and recreation industries. The NEXRAD Doppler [1] data which is generated by 158 radar systems continuously is exploited. The temperature and reflectivity within consecutive radar measurements are crucial to the prediction and hence need to be monitored. One possible query for that purpose would be “Within the most recent 100000 measurement intervals, calculate the number of all the radar measurements whose reflection levels exceed 35 dBZ and temperature is -10°C ”. In order to quickly retrieve some approximate answers for this query, it is necessary to build a summary representation to maintain some statistics within such a sliding window.

Histograms, as one of the summarization techniques, have been widely explored in theory and practice for approximating data distribution. They provide good solutions to various problems, such as selectivity estimation for query optimizer [13, 15, 19], and approximation for the queries [3, 5, 6, 21]. There are also some work contributed to build one dimensional histograms for sliding windows [4, 11, 8]. Such histograms can only handle the data items with values in unary domains. However, in practice, many applications need to process multi-valued data, such as temperature and reflection level used for lightening prediction. Hence it is hard for those histograms to answer range queries over such multi-valued data.

*This work is supported in part by NSF awards EIA99-86057, EIA00-80134 AND IIS02-091112.

[†]Supported in part by IBM Corporative Fellowship

In this paper, we overcome these limitations and propose a *two-dimensional* histogram, called the *hybrid histogram* as a summary representation for the sliding window against multi-valued data domains. The hybrid histogram is built based on the *uni-dimensional* and *non-overlapping* exponential histograms. To adapt to the changing data distribution in the sliding window on the fly, the hybrid histogram exploits a dynamic partitioning strategy over the data value domains. To further compress the exponential histogram, condensed exponential histograms are proposed which contain more succinct information without losing the error bound of the original exponential histograms. To the best of our knowledge, this is the first work to dynamically summarize the multi-valued sliding windows using a two-dimensional histogram.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the system model. Section 4 reviews the *exponential histogram* and exploits it to build our new histogram model – the *hybrid histogram*. In section 5, we develop algorithms for adapting our histograms to the data distribution. Section 6 describes the algorithms to compress the exponential histogram. Section 7 presents the experimental evaluation of our proposed histograms. We conclude in Section 8.

2 Related Work

Extensive research in the literature has studied static datasets. As our approach uses a two-dimensional summary representation of a sliding window for data streams and provides approximation answers, it is related to multi-dimensional histograms [18, 5] and sampling [22]. However, because these techniques work in static settings, they can not capture the dynamic stream data distribution without re-computation.

As the need for dynamic datasets arises for an increasing number of applications, much research has been directed to the incremental maintenance of summaries. Babu et al. [2] presented an overview of the problems arising in the field of continuous queries over data streams. Gibbons et al. [7] proposed approaches for the dynamic maintenance of approximate histograms. In [15], techniques were developed to solve the problem of maintaining wavelet-based histograms over changing datasets. Incrementally maintainable quantiles have been proposed in [14, 10]. Gilbert et al. [9] considered small space wavelet to provide approximate answers for aggregation queries over data streams from sketches. Gehrke et al. [6] explored the issue of computing correlated aggregates over data streams. However, those approaches were proposed specifically for landmark windows, not for sliding windows [11].

Guha et al. [11] presented an algorithm to build approximate optimal histograms on time domain for sliding windows, which considers the tradeoffs between accuracy and

speed. Datar et al. [4] proposed a stream statistics, namely the *exponential histogram*, which can be maintained over sliding windows and be used to provide ϵ -approximation answers to the *BasicCounting* query. Gibbons et al. [8] considered aggregate functions over sliding windows from multiple data streams. They proposed so called *waves* to improve the space, processing time and the query time for the worst case for the *BasicCounting* queries. All those proposals summarize the stream in the window based only on the time dimension. Thaper et al. [21] presented a sketch based approach to incrementally track the distribution of a multi-dimensional data stream with approximate quality guarantees. Their proposal is based on explicit arrival/expiry events, while in a sliding window summary, the expiry (delete) event is implicit and can not be associated with individual data elements. In this paper, we propose a two-dimensional histogram incorporating both time and value dimensions for a sliding window.

3 System Model

An input item in a data stream has a value v in the domain D . The domain D can be contiguous or discrete. Each data item associates with a timestamp ts . There can be more than one data item stamped with the same timestamp or no data item with a particular timestamp. Let T_c denote the current timestamp. Given the window size w , the domain space for any data item within a sliding window is $D \times \{T_c - w + 1, \dots, T_c\}$. A data item d in a sliding window is represented by (v, ts) . When T_c increases by 1 at each time tick, the sliding window moves one step forward. This process can be simulated by a number of insertions of d_i and deletions of d_j , where d_i is a data item with timestamp T_c and d_j is a data item with timestamp $T_c - w$.

A histogram H is a function to approximate the frequency distribution in a sliding window, $H : D \times \{T_c - w + 1, \dots, T_c\} \rightarrow N$. A histogram divides the data space into a number of rectangular buckets B_1, B_2, \dots, B_s . Each bucket B_i records its boundaries and frequency. Hence the movement of the sliding window involves a number of insertions of B_i and conditional deletions of B_j , where B_i is the bucket with timestamp T_c and B_j is the bucket with timestamp $T_c - w$. Compared to the sliding window, H moves forward along time in the granularity of buckets instead of data items.

The query Q we consider in this paper is the *aligned window query*. The time predicate specified in Q always aligns with the T_c , represented by $[T_s, T_c]$, where $T_s \in [T_c - w + 1, T_c]$. The value predicates specified in Q can be any range in the attribute value domain, that is $[v_1, v_2]$ for $v_1 < v_2$ and $v_1, v_2 \in D$. The aligned window query is crucial for applications that are only interested in the most recent set of data. Note that, for a unary domain $|D| = 1$,

both the sliding window and query are only on time dimension. In particular, the query Q only considers the time predicate in the form of $[T_s, T_c]$. For simplicity, we only discuss COUNT queries. However, it can be easily extended to answer other aggregation queries, such as SUM or AVG.

4 Histograms

4.1 The Exponential Histogram

A number of results on estimating functions were presented by Datar et al. [4] for a sliding window over data streams. The essential problem they explored is the *BasicCounting* problem. “Given a stream of data elements, consisting of 0s and 1s, the algorithm for BasicCounting maintains at every time tick the count of the number of 1s in the last N elements” [4]. The *exponential histogram* [4], referred to as H^e , is defined to provide the approximate answer with a predefined ϵ relative error bound. It partitions the sliding window into l contiguous time intervals. We call these time intervals as the *mini-buckets*. Each mini-bucket records the number of 1s and the timestamp of the latest 1. In the rest of the paper, we use frequency and depth interchangeably as the number of 1s in a mini-bucket. Let F_i denote the frequency of the i^{th} mini-bucket. The mini-bucket depths of H^e , i.e. F_1, F_2, \dots, F_l , are non-decreasing and are constrained to be an element in $\{1, 2, 4, \dots, l'\}$, where $l' \leq l < \log_{\frac{w}{2\epsilon}} + 1$. It uses $O(\frac{1}{\epsilon} \log^2 N)$ bits of memory to store the mini-bucket sizes and timestamps.

Cases	Description	LB	UB
1	$i = 1$ and $m > 1$	$2\lceil \frac{1}{2\epsilon} \rceil - 1$	$2\lceil \frac{1}{2\epsilon} \rceil$
2	$i = 1$ and $m = 1$	1	$2\lceil \frac{1}{2\epsilon} \rceil$
3	$i > 1$ and $i < m$	$\lceil \frac{1}{2\epsilon} \rceil$	$\lceil \frac{1}{2\epsilon} \rceil + 1$
4	$i > 1$ and $i = m$	1	$\lceil \frac{1}{2\epsilon} \rceil + 1$

Figure 1. The Mapping Table

The definition of the *exponential histogram* is shown in [4] in Definition 4.1¹. Here we use \mathcal{S}_i^e to denote a set of consecutive mini-buckets with depth 2^{i-1} in the exponential histogram. Let $|\mathcal{S}_i^e|$ be the number of mini-buckets in \mathcal{S}_i^e . In the following paragraphs, we use $LB(i, m)$ or $UB(i, m)$ to denote the lower bound or the upper bound where i and m are defined in Definition 4.1. For convenience, Figure 1 depicts the mapping from the cases (1 to 4) to the lower bound (LB) and the upper bound (UB) of $|\mathcal{S}_i^e|$.

Definition 4.1 H^e consists of a set of neighboring mini-buckets along the time dimension, $\mathcal{S}_1^e, \mathcal{S}_2^e, \dots, \mathcal{S}_m^e$, for $m \geq$

¹We slightly modify the requirement on the number of mini-buckets of the same depth shown in [4] to guarantee any ϵ error bound.

1. Given ϵ , for each \mathcal{S}_i^e , $1 \leq i \leq m$, $|\mathcal{S}_i^e| \in [LB(i, m), UB(i, m)]$.

In Figure 2(a), we give an example of H^e with a total of fourteen 1s. Each rectangle represents a mini-bucket. The timestamps are marked above each mini-bucket and the depth of a mini-bucket is noted within the rectangle. Since $\epsilon = 0.3$, $\lceil \frac{1}{2\epsilon} \rceil = 2$ and $m = 3$, according to Definition 4.1, the number of mini-buckets in \mathcal{S}_1^e can be either 3 or 4. Similarly, $|\mathcal{S}_2^e|$ should be 2 or 3 and $|\mathcal{S}_3^e|$ is 1, 2 or 3.

An exponential histogram is maintained by *cascading merge* [4]. When a new data item arrives, the last mini-bucket is checked. If its timestamp indicates expiry, delete it. If the new element is 1, create a new mini-bucket with depth 1 and the current timestamp. Traverse mini-buckets from the most recent to the oldest. If the number of mini-buckets with the depth 2^{i-1} exceeds the upper bound of $|\mathcal{S}_i^e|$, merge the oldest two mini-buckets into a single bucket with doubled-size and keep the more recent timestamp. Figure 2(b) shows an example of H^e maintenance. The first figure depicts the maintenance result after a new 1 has arrived with timestamp 33. The number of mini-buckets with depth 1 in Figure 2(a) becomes 5, which is greater than the upper bound of $|\mathcal{S}_1^e|$. The oldest two mini-buckets with depth 1 are merged into one and timestamp 22 is deleted. Similarly, two mini-buckets with depth 2 merged into one mini-bucket. In the second figure, the input has value 1 with timestamp 35. Now the window boundary becomes $[9, 35]$. The last mini-bucket is thus out of the window and deleted.

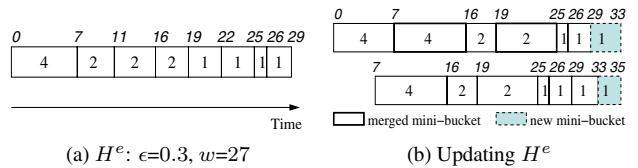


Figure 2. The Exponential Histogram

4.2 The Challenges

Several techniques have been proposed in the literature to construct histograms [13, 15, 17, 19] for *uni-dimensional* data set. However, partitioning multidimensional spaces is challenging. It can be shown [16] that even for static two-dimensional data sets, building the *V-optimal(v,f)* [13] histogram with arbitrary rectangle is NP-Hard. Constructing multi-dimensional histograms over dynamically evolving data sets is even more challenging, since the histograms need to be incrementally maintained to capture the changing data distribution.

The exponential histogram only handles incoming data elements with unary value and the timestamp in a sliding window. As pointed out by Datar et al. [4], it can be easily

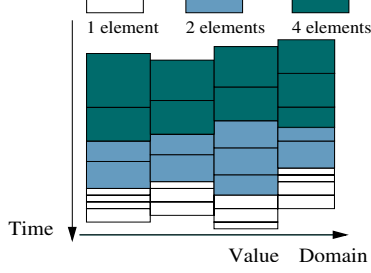


Figure 3. Equi-width histogram with embedded exponential histogram

extended to process input data elements with positive integer value by viewing the data value as a positive number of 1s arriving at the same time. However, H^e by itself is not sufficient to answer queries on arbitrary value domains because it consists of time information only. To maintain a summary which represents the data distribution against both time and value domains, a set of H^e s can tile along attribute value domain with *a priori* partitioning boundaries. A natural choice is to partition the attribute value domain into equal ranges. As shown in Figure 3, each exponential histogram serves as a bucket in an equi-width histogram. The problem with these approaches is that, after initialization, the attribute value range of an H^e cannot be changed. Without re-computation, it is difficult to adapt the overall summary to capture the variation of the data distribution.

In this paper, we will propose a histogram technique which exploits exponential histograms as building blocks. It summarizes arbitrary value domains and captures the dynamic distribution of data within a sliding window.

4.3 The Hybrid Histogram

We discussed in Section 4.2 that the straightforward extension of H^e to real value domain runs into problems. The challenge is how to partition the two-dimensional window dynamically according to evolving data set. Rather than partitioning the spaces *one dimension at a time*, we propose the *hybrid histogram*, referred to as H^h , with disjointed exponential histograms tiling on both dimensions. As introduced in Section 4.1, each exponential histogram will further be partitioned into mini-buckets based on time domain. Consequently, each exponential histogram is associated with a value range as well as a time range. We use H^e and *bucket* interchangeably in the following paragraphs.

Figure 4 depicts a snapshot of a hybrid histogram at current time T_c . The current time is at the origin of the time axis. Eight big rectangular blocks with bold boundaries, labelled from H_1^e to H_8^e , are the buckets of the hybrid histogram. The degree of grey indicates the number of data items contained in a mini-bucket.

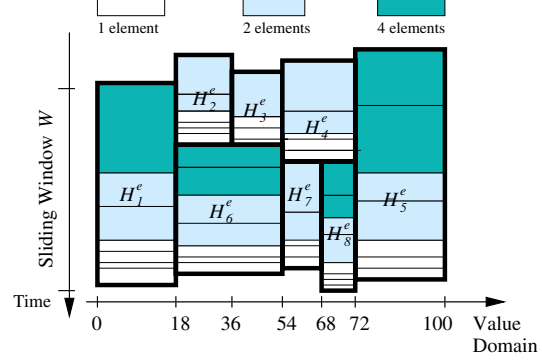


Figure 4. Hybrid Histogram

For each exponential histogram in the hybrid histogram, we store the following sets of values:

- The value boundaries v_l, v_r .
- A set of timestamps $\{TS_1, TS_2, \dots, TS_{last}, TS_{end}\}$, where TS_i ($i \in \{1, \dots, last\}$) is the latest timestamp of the data item in the i^{th} H^e mini-bucket. TS_1 is the most recent timestamp of H^e and TS_{end} is the ending timestamp.
- *TTL*: the total number of data items in the H^e .

We further classify these exponential histograms into two classes. The first class is the H^e s that continuously incorporate the incoming data items, denoted as *active exponential histograms*. The second class is the H^e s that are no longer fed with any input data due to the non-overlapping property of hybrid histogram, denoted as *inactive exponential histograms*. For example, in Figure 4, there are five active exponential histograms, namely $H_1^e, H_5^e, H_6^e, H_7^e$ and H_8^e and three inactive exponential histograms, namely H_2^e, H_3^e and H_4^e . The new data item cannot be put into H_2^e or that would result in the overlapping between H_2^e and H_6^e .

4.4 Answer the Aligned Window Query

Recall that the aligned window query Q has time range $[T_s, T_c]$ and value range $[v_1, v_2]$. To answer Q , we start from each H_j^e in H^h intersects with Q , which contains partial answer. We approximate the query result based on a uniform distribution assumption on both time and value domain. First, we approximate on the time range. For each mini-bucket B_i in H_j^e with frequency F_i and time range $[T_a, T_b]$, the partial answer is $F_i \times \frac{|[T_a, T_b] \cap [T_s, T_c]|}{T_b - T_a + 1}$, where $|[T_a, T_b]|$ denotes $T_b - T_a + 1$. Hence the answer on the time range is the sum of the partial answers from all its mini-buckets B_i , i.e., $(\sum F_i \times \frac{|[T_a, T_b] \cap [T_s, T_c]|}{T_b - T_a + 1})$. Second, we further refine this query result on the value range of Q . The final query result from H_j^e is $(\sum F_i \times \frac{|[T_a, T_b] \cap [T_s, T_c]|}{T_b - T_a + 1}) \times$

$\frac{|H_j^e \cap [v1, v2]|}{|H_j^e|}$. Here $H_j^e \cap [v1, v2)$ denotes the intersecting value interval between $[H_j^e.v_l, H_j^e.v_r)$ and $[v1, v2)$, and $|H_j^e|$ denotes $H_j^e.v_r - H_j^e.v_l$.

To get a complete answer for Q , we simply sum up the answers from all H_j^e s. H^h no longer preserves the ϵ error bound although its building block H^e does. Actually no existing histogram technique can guarantee any relative error bound for ad hoc value range query. However, since its building block H^e keeps the error bound, the accuracy of H^h can take benefit from that as we will show in the experiment in the Section 7.2.

5 Dynamic Maintenance Algorithm

5.1 Motivation

We observe the following behaviors of H^e s: 1) The incoming data items with values falling between the value range of an active exponential histogram will generate new mini-buckets. The *most recent* time boundaries of active exponential histograms will then be set to the timestamp of the new incoming data item. While the *oldest* time boundaries of active exponential histograms are fixed till they hit the sliding window boundary. An active exponential histogram expands in this way. 2) The inactive exponential histograms have fixed *most recent* time boundaries since they no longer accept new data items. Hence after hitting the sliding window boundary, they keep shrinking and will disappear eventually.

Per these observations, active exponential histograms will potentially dominate the partition granularity on the value domain. Hence if current partition on value domain cannot effectively capture the data distribution, we can adjust the partition by creating new active exponential histograms which result in a better partition. In the following sections, we develop such dynamic partitioning algorithms for a hybrid histogram.

5.2 Overview of Algorithms

A hybrid histogram, H^h , is initially constructed from a training dataset. Given a set of initial data items $\{(v_1, ts_1), \dots, (v_n, ts_n)\}$ of a data stream and the current timestamp T_c , we initialize an H^h as follows. First we choose only the data items within current window. Then we sort them by their values and partition them into equi-depth buckets. For each equi-depth bucket, we sort its data items by their timestamps and build an exponential histogram on that. Note that all the H^e s are initially active.

Now we describe how to dynamically maintain the hybrid histogram. When a new data item arrives, all H^e s should be updated (Section 6.2). In particular, the data item

value will be mapped into 0 or 1 as the input to each exponential histogram. At the same time, all H^e s are checked whether their last mini-bucket expires or not. As mentioned in Section 5.1, it is necessary to dynamically adjust the partition on value domain of the hybrid histogram to better capture the data distribution. We estimate the number of data items in the value range of each active exponential histogram (Section 5.3), and use it to measure whether current active exponential histograms represent for appropriate partitions on value domain or not. Depending on the result from the last step and the usage of the memory space, we identify the active exponential histograms as the targets to be refined and create new active exponential histograms as the result of refinement (Section 5.4). Finally, in order to reclaim memory, some neighboring active exponential histograms will be consolidated by generating new active exponential histogram covering the union of their value ranges (Section 5.5).

The high level algorithm is presented below.

```

Build_Maintain
  ( $H : H^h, td : training\ dataset, ds : datastream$ )
  Initialize  $H$  from  $td$ ;
  For each data item  $d_i$  from  $ds$ 
    Update  $H^e(d_i)$  for all the  $H^e$ s in  $H$ ;
    /*section 6.2*/
  Calculate statistics from  $H$ ;
  /*section 5.3*/
  Identify candidate active  $H^e$ s for refinement;
  Refine active  $H^e$ s;
  /*section 5.4*/
  Consolidate  $H$  by combining superfluous
  active  $H^e$ s; /*section 5.5*/

```

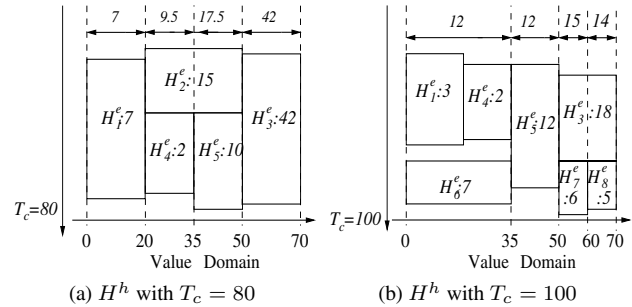


Figure 5. An Example of H^h Maintenance

5.3 Calculate a Measurement

As we discussed in Section 5.1, the value ranges of the active exponential histograms represent the current partitions on the value domain. To measure whether current partitions are appropriate or not, we calculate the total number

of data items with the values between the value ranges of active exponential histograms. Other than active H^e s, inactive H^e s also contribute to the number of data items in a value range. For instance in Figure 5(a), the attribute value range of inactive H_2^e intersects with both that of active H_4^e and active H_5^e . We approximate the number of data items in the value range of an active H_i^e , denoted by $Cnt(H_i^e)$, as follows,

$$Cnt(H_k^e) = \sum_{H_i^e \cap H_k^e \neq \emptyset} H_i^e.TTL \times \frac{|H_i^e \cap H_k^e|}{|H_i^e|}$$

$|H_i^e \cap H_k^e|$ denotes the intersected range of $[H_i^e.v_l, H_i^e.v_r)$ and $[H_k^e.v_l, H_k^e.v_r)$, and $|H_i^e|$ denotes $H_i^e.v_r - H_i^e.v_l$. We compute the average count \overline{Cnt} from all the $Cnts$ of active exponential histograms. Figure 5 depicts the number of data items in each H^e and Cnt in the value range of each active H^e . For instance, in Figure 5(a), $Cnt(H_4^e)$ is approximated as $H_2^e.TTL \times \frac{35-20}{50-20} + H_4^e.TTL \times \frac{35-20}{35-20} = 15 \times \frac{15}{30} + 2 \times \frac{15}{15} = 9.5$. Similarly we calculate $Cnt(H_1^e) = 7$, $Cnt(H_5^e) = 17.5$ and $Cnt(H_3^e) = 42$. Hence the average count \overline{Cnt} is 19. In figure 5(b), $Cnt(H_6^e)$ is a summation of $H_1^e.TTL$, $H_4^e.TTL$ and $H_6^e.TTL$, which equals to 12. The value range $[35, 50)$ only contains H_5^e , so $Cnt(H_5^e)$ equals to $H_5^e.TTL = 12$. Similarly we compute $Cnt(H_7^e) = 15$ and $Cnt(H_8^e) = 14$. The average count $\overline{Cnt} = 13.25$.

5.4 Refine an Active H^e

If the value range of the H^e contains too many data items compared to other active exponential histograms, meaning the estimation error could be high, it is identified as the target to be refined. According to the approximate histogram techniques proposed by Gibbons et al. [7], $2\overline{Cnt}$ is a reasonable threshold for all the $Cnts$. Per these concerns, we perform refinement on active H_i^e when $Cnt(H_i^e) \geq 2\overline{Cnt}$.

To refine an active H^e , we divide its value range $[H^e.v_l, H^e.v_r)$ into two, i.e. $[H^e.v_l, v)$ and $[v, H^e.v_r)$. Two exponential histograms are created respectively on the new value ranges. v is calculated so that after refinement, the $Cnts$ for the new H^e s are half of the Cnt of the original H^e . Note that both newly generated exponential histograms are active. The previous active exponential histogram becomes inactive after refinement. The incoming data items no longer feed into it although its value range covers the values of the data items.

Figure 5 provides an concrete example for H^e refinement. In Figure 5(a), consider H_3^e , $Cnt_4 = 42 > 2\overline{Cnt} = 2 \times 19 = 38$. Hence the value range of H_3^e , which is $[50, 70)$, divides into $[50, 60)$ and $[60, 70)$. Figure 5(b) depicts the hybrid histogram after the refinement and several data inputs. Two new exponential histograms, namely H_7^e and H_8^e , are created. They are active and incorporate the newly coming data items. H_3^e becomes inactive.

5.5 Combine Active H^e s

Let H_i^e and H_{i+1}^e be the two neighboring exponential histograms. Their value ranges are respectively $[H_i^e.v_l, H_i^e.v_r)$ and $[H_{i+1}^e.v_l, H_{i+1}^e.v_r)$, where $H_i^e.v_r = H_{i+1}^e.v_l$. After combining H_i^e and H_{i+1}^e , a new exponential histogram is built on the value range $[H_i^e.v_l, H_{i+1}^e.v_r)$. H_i^e and H_{i+1}^e become inactive.

Now we describe how to find the most suitable active exponential histogram pair for combination. First some candidates are chosen from all the active exponential histograms. Different from refinement, the conditions here are that the $Cnts$ of the active exponential histograms are smaller than \overline{Cnt} .

Given all the candidates, we need to find the best neighboring pair for combination. The inaccuracy based on histograms arises because the boundaries of queries may not completely align with the boundaries of the buckets. In such cases the query result is extrapolated usually by making the *uniform spread assumption* [19] of data distribution within each bucket. To lower the error produced from extrapolation, the data density is considered for each H^e in the pair. If the density of the incoming data in one value range is closest to the other, the penalty for combination will be minimized. We use the data incoming rate per data value unit interval as the metric M for the density of the incoming data. Due to the space limit, we do not further discuss in this paper. Please refer to our full paper [20] for the details. Finally, we note that the combination operation results in a small overhead by creating a new H^e . However, the small overhead can be quickly offset by the benefit from it.

6 Compress the Inactive Exponential Histogram

6.1 The condensed exponential histogram

Recall that there are two classes of exponential histograms existing in a hybrid histogram, namely the active and inactive exponential histograms. Unlike active exponential histograms, the inactive exponential histograms are not fed with newly arrived data, hence their mini-buckets will not be merged. However, the sliding window size may be fairly large resulting in large number of mini-buckets of inactive exponential histograms. In this section, we propose a strategy to reduce the number of such mini-buckets to save the resources without losing error bound ϵ for each exponential histogram. We call the resulting histogram *condensed exponential histogram*.

The intuition behind the condensed exponential histogram is similar to the cascading merge, while with no newly arriving data item as input. A count CT is recorded for each inactive exponential histogram as the number of data items

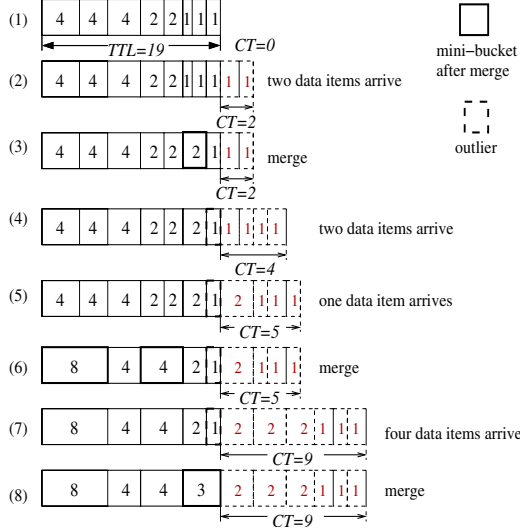


Figure 6. An example of H^{ce} with $\epsilon = 0.3$

arriving between the current time and the ending time of the histogram. When a new data item arrives with its value between the value range of an inactive exponential histogram, the CT increases by one. Meanwhile, the count CT is viewed as a set of mini-buckets virtually. A cascading merge technique is introduced to keep the mini-bucket pattern in an inactive exponential histogram, and hence generates a condensed exponential histogram. In Figure 6(2), after two data items arrive, the total number of real and virtual mini-buckets containing 1 element increases to 5 which is greater than the upper-bound 4 (refer to Figure 1). Hence the oldest two mini-buckets are merged into one with depth 2 shown in Figure 6(3). However, the virtual mini-buckets cannot merge with real mini-buckets. There could be a dangling mini-bucket which cannot consolidate with another. It is allowed in the condensed exponential histogram as a special case of mini-buckets and *at most one outlier* mini-bucket can exist. All the dangling mini-buckets will be combined as one outlier. In Figure 6(4), the last two mini-buckets cannot merge since one is real the other is virtual. In this case, the dangling mini-bucket becomes an outlier. In Figure 6(5), there are 3 real mini-buckets and 1 virtual mini-bucket having depth 2, and the total number exceeds the upper-bound 3. Hence the last two merged into one with depth 4 (Figure 6(6)) and for the same reason, the last two mini-buckets merge into one with depth 8. Figure 6(7)-(8) depict the consolidation of two dangling mini-buckets into one outlier.

6.2 Maintaining the Condensed Exponential Histograms

For each condensed exponential histogram in the hybrid histogram, other than the information stored in exponential histogram as described in Session 4, following sets of values

are stored:

- CT : the number of data elements arriving between TS_1 of H^{ce} and current time T_c , and having value $v \in [v_l, v_r)$.
- Pnt and f^o , where Pnt is the pointer to the timestamp of an outlier mini-bucket and f^o is the frequency of the outlier.

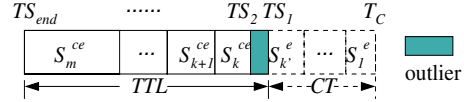


Figure 7. H^{ce} with mini-bucket sets

Figure 7 depicts a set of mini-buckets in condensed exponential histogram. Let S_i^{ce} be a set of consecutive mini-buckets with depth 2^{i-1} in H^{ce} and $|S_j^{ce}|$ be the total number of mini-buckets in set S_j^{ce} . Given ϵ , H^{ce} consists of mini-buckets $S_k^{ce}, S_{k+1}^{ce}, \dots, S_m^{ce}$. $S_1^e, \dots, S_{k'}^e$ are sets of the virtual mini-buckets derived from CT . For instance, in Figure 6(10), there are three sets of mini-buckets besides outlier. $k = 3$ because the second mini-bucket contains 4 data items, which is 2^{3-1} . Similarly, $m = 4$ and $k' = 2$. We have that $|S_3^{ce}| = 4$, $|S_4^{ce}| = 1$ and $|S_2^e| = 2$.

We develop a maintenance algorithm for H^{ce} . The following lemma establishes the theoretical basis to compute the mini-bucket patterns from the stored meta-data of an H^{ce} including ϵ , TTL , CT and f^o .

Lemma 6.1 *The size of H^{ce} mini-bucket sets, $|S_k^{ce}|, |S_{k+1}^{ce}|, \dots, |S_m^{ce}|$, k and m are uniquely determined by ϵ , TTL , CT and f^o .*

Update_ $H^{ce}(d)$
 IF ($d.v \in [H^{ce}.v_l, H^{ce}.v_r)$)
 Calculate k, m and $|S_k^{ce}|, \dots, |S_m^{ce}|$ of H^{ce}
 from ϵ, TTL, CT and f^o ;
 $CT++$;
 Calculate k' and $|S_{k'}^e|$ from ϵ and CT ;
 IF ($k = k'$) AND ($|S_k^{ce}| + |S_{k'}^e| > UB(k', m)$)
 IF ($|S_k^{ce}| = 1$) AND (no outlier) $f^o = 2^{k-1}$;
 IF ($|S_k^{ce}| = 1$) AND (has outlier) $f^o += 2^{k-1}$; Delete TS_2 ;
 IF ($|S_k^{ce}| > 1$) Delete the last timestamp in S_k^{ce} ;
 Cascade merging from S_{k+1}^{ce} to S_m^{ce} ;
 IF ($TS_{last} \leq T_c - w + 1$) Delete the last mini-bucket;

Figure 8. The Algorithm to Update H^{ce}

The maintenance algorithm for the condensed exponential histogram is presented in Figure 8. First, the pattern of the mini-bucket sets are derived from ϵ , TTL , CT and f^o . If the data item has value in the value range of H^{ce} ,

CT is increased by one. We calculate the value k' and $|S_{k'}^e|$ from the updated CT . If the index k equals to k' , the first mini-bucket set in H^{ce} , i.e. S_k^{ce} , will be considered to bind with $S_{k'}^e$. We further verify whether the summation of the number of mini-buckets in S_k^{ce} and $S_{k'}^e$ exceeds the upper bound or not. If it is greater than $UB(k', m)$, we consider to merge the extra mini-buckets. There are several cases: 1) There is only one mini-bucket left in S_k^{ce} and no outlier has been created, the mini-bucket is set to be the new outlier. 2) If only one mini-bucket is in S_k^{ce} and there is already an outlier, we merge the existing outlier with this mini-bucket. 3) If in S_k^{ce} there are more than one mini-bucket, merge the last two mini-buckets and just delete the last timestamp in S_k^{ce} . After that, we cascade the merging process from the mini-bucket set with index $k + 1$ to m .

Finally, we prove that the H^{ce} preserves the ϵ relative error bound for the answers to the aligned window queries for unary value domain, as shown in Lemma 6.2.

Lemma 6.2 *The condensed exponential histogram is ϵ -approximate histogram. Let F_1, \dots, F_l be the frequencies of its mini-buckets. For $1 \leq j \leq l$ and $F_j > 1$, $F_j / (2(1 + \sum_{i=1}^{j-1} F_i + CT)) \leq \epsilon$.*

7 Experimental Study

In this section, we discuss some performance results the hybrid histogram in supporting the aligned sliding window query.

7.1 Experiment Setup

Data streams The incoming tuples of a data stream are generated from both *synthetic* and *real* datasets. The real dataset we consider consists of network packets in a day, referred to as *Network*. Packet size distribution is characteristic of network traffic that describes the size of packets traveling across the network. We use *tcpdump* to capture the packets flowing in the network. For each packet, we extract the packet size and its timestamp as a streaming data item. We truncate the real timestamp of a network packet into milli-seconds. Many data items can have the same timestamp in *Network*.

We also generate synthetic datasets for our experiments from *Zipf* [23] and *Gaussian* distributions. In the *Zipf* distribution, the value and frequency are randomly correlated and z varies from 0.1 to 1 for different data skewness. The gaussian distribution is generated from the *Mersenne Twister generator* and denoted as $Gauss(\mu, \sigma^2)$. The value and frequency are also randomly correlated. The timestamp function consists of two uniform random number generators. Each uniform generator has its own range and produces a random number for the time interval between two

consecutive timestamps. A coin is flipped by factor p to determine which uniform generator is chosen at the current time. The default values for the synthetic datasets parameter are described as below.

Dataset	Parameter	Value
All	R: # of distinct values	1,000
	N: Length of data streams	1,000,00
	w: the window size	5000
Gauss	μ : mean	300
	σ : standard deviation	50
Zipf	z: Skewness	0.4

Histograms We implement four histograms to summarize the sliding window as shown below.

1. Equi-width buckets on the value domain with the condensed exponential histogram embedded, referred as $h1$.
2. Equi-width buckets with fixed time-interval mini-buckets, referred to as $h2$.
3. The time interval is partitioned into buckets with fixed time-interval mini-buckets and those buckets are maintained by dynamic adjust algorithm. This technique is called $h3$.
4. Dynamically adjusted buckets using dynamic adjust algorithm with the condensed exponential histogram embedded, which is our proposed hybrid histogram. It is plotted as hh in the following figures.

Metrics To measure the accuracy of answers to aggregation queries, we used the *Average Relative Error* (ARE) as the error metric. The relative error is computed from the ratio of the difference between absolute error and accurate answer to the accurate answer $\frac{|est-acc|}{acc}$; A set of queries issued at one time contains 100 queries. We divide the sum of relative errors by the number of queries and get ARE .

All the experiments were run on an Intel machine with PIII pentium 1.13GHz cpu, 128MB main memory and 512KB cache. The operating system is Mandrake Linux 8.0.

7.2 Impact of ϵ and query selectivity

As we have discussed, ϵ is the error bound for all exponential histograms. It also affects the overall accuracy of H^h represented by ARE . The smaller ϵ value should result in lower overall error. On the other hand, query selectivity, referred to as sel , has great impact on the accuracy of estimation as well. To see the relationship between ϵ , query selectivity and ARE , we conduct the following experiment. We set ϵ to be 0.1, 0.2 and 0.3 respectively. For each ϵ value, ARE is measured according to the changing selectivity from 10% to 90%. The streaming data is generated from the *Zipf* distribution.

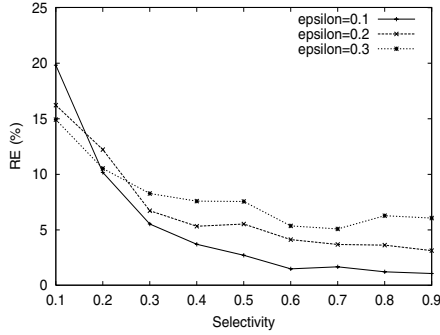


Figure 9. Impact of ϵ and sel on ARE

Figure 9 depicts the relative error under different ϵ and sel . The tendency is that when sel increases, ARE decreases, because higher query selectivity produces bigger accurate answer and hence reduce the relative error. It is interesting to see that when sel is low (10%), the estimation error for $\epsilon = 0.3$ is the smallest while H^h with $\epsilon = 0.1$ produces the biggest relative error. This is because H^h with smaller ϵ has finer partitions on the time domain. Given the memory constraint, in return, the partitioning on the value domain has larger granularity. However, when sel increases, the benefit ϵ produces quickly offsets the penalty it pays on the value domain. For instance, when $sel > 20\%$, ARE in the case $\epsilon = 0.1$ is the smallest while that of $\epsilon = 0.3$ becomes the largest. The H^h with smaller ϵ has steeper decreasing line of ARE . Consequently, the difference of ARE s for $\epsilon = 0.1, 0.2$ and 0.3 amplifies when sel grows higher.

7.3 Adapting to changing data distribution

The data distribution may vary over time in the span of data stream. The granularity of H^h adapts to the changes in data distribution for better estimation. In this section, we verify the effectiveness for H^h to capture the moving distribution of the streaming data within a window. We first generate data from $Zipf(0.1)$, then abruptly shift to $Zipf(0.4)$. Note that the frequencies and data values are randomly correlated. We issue a set of queries with 50% selectivity after each data item arrives. The total number of incoming data items is 1800. The query set changes after the 600th data item arrives. We plot the result with $\epsilon = 0.1$ in Figure 10.

Before the change of the data distribution (the number of data items less than 600), the relative errors are mostly within the range of [4%, 8%]. Because of the shifting of the data distribution, the H^h incorporating the data items from $Zipf(0.1)$ does not represent the partition for those from $Zipf(0.4)$. Hence both ARE s and the variation of ARE s increase. However, H^h adapts quickly to the new data distribution. ARE becomes stable (within [4%, 6%]) roughly after the 820th data item arrives. For clarity, a curve is

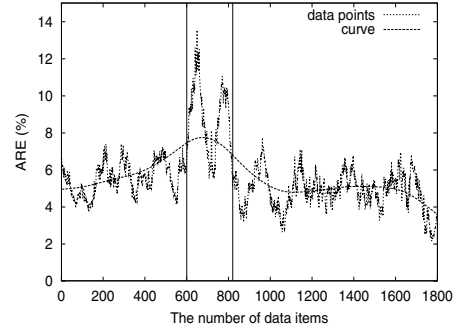


Figure 10. Adapting to changing data distribution

plotted to show the smooth spline generated from the data points. The top of the curve, in the region within the vertical lines, depicts the adjustment of partitions in H^h . It is clear that H^h can efficiently adapt to the changing data distributions.

7.4 Accuracy of the histograms

In this section, we conduct experiments to evaluate the accuracy of the histograms. In particular, we compare the four histograms discussed in Section 7.1 under various query selectivities. For $h1$ and hh , ϵ equals to 0.1.

As we have mentioned, when the query selectivity increases, the estimation error decreases because the query with large selectivity tends to cover more accurate data. Figure 11(a)-(c) depicts the accuracy of the four histograms over different query selectivities based on three different data distributions, namely, Zipf, Gauss and a real data set *Network* as described in Section 7.1. From the figures, first, we can see that the relative error of all four histograms decreases as the query selectivity increases, which is expected. Second, our proposed histogram H^h consistently outperforms the other three histograms under all three data distributions. ARE of H^h (hh in the figures) is always smaller than the others'. Moreover ARE of H^h decreases faster than others'. This contributes to the sophisticated mechanism of H^h in building histograms that better capture the data distribution.

8 Conclusion

Data streams are an increasingly important domain for data processing software. One of the main challenges is how to accurately summarize information contained in a data stream over a sliding window. Up to date, most work on sliding windows has only considered unary value domains. Histograms for static data sets, on the other hand, have been designed for multi-valued domains. In this pa-

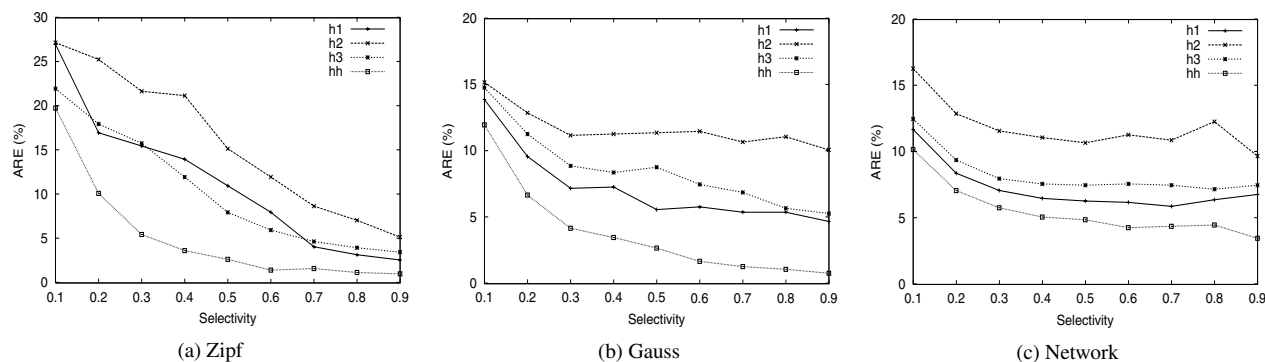


Figure 11. Accuracy of the Histograms

per, we proposed the first multi-valued histogram for data streams. Our approach exploits exponential histograms as building blocks. The resulting hybrid histogram dynamically adapts to variations in the data stream. New exponential histograms are generated to capture the new data distribution while old ones are gradually removed from the sliding window. We further compress the exponential histograms into condensed exponential histograms. More succinct information is stored in condensed exponential histograms and it preserves ϵ error bound. The performance analysis on different synthetic and real data sets show that the hybrid histogram is a valuable and accurate solution.

References

- [1] NEXARD. <http://www.roc.noaa.gov/>.
- [2] S. Babu and J. Widom. Continuous Queries over Data Streams. Technical report, Stanford University, 2001.
- [3] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: A Multidimensional Workload-Aware Histogram. In *Proceedings of the ACM SIGMOD Conference*, pages 211–222, 2001.
- [4] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002.
- [5] D. Gunopulos, G. Kollios, V.J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *Proceedings of ACM SIGMOD Conference*, 2000.
- [6] J. Gehrke, F. Korn, and D. Srivastava. On Computing Correlated Aggregates Over Continual Data Stream. In *Proceedings of ACM SIGMOD Conference*, pages 13–24, May 2001.
- [7] P. Gibbons, Y. Matias, and V. Poosala. Fast Incremental Maintenance of Approximate Histograms. In *Proceedings of 23rd VLDB Conference*, pages 466–475, 1997.
- [8] P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 63–72, 2002.
- [9] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In *Proceedings of the 27th VLDB Conference*, Italy, September 2001.
- [10] M. Greenwald and S. Khanna. Space-Efficient Online computation of Quantile Summaries. In *Proceedings of ACM SIGMOD Conference*, pages 58–66, May 2001.
- [11] S. Guha and N. Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. In *Proceedings of the 16th ICDE Conference*, 2002.
- [12] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical report, Compaq Systems Research Center, Palo Alto, California, May 1998. Technical Report TR 1998-011.
- [13] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantee. In *Proceeding of 24th VLDB Conference*, New York, USA, 1998.
- [14] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets. In *Proceedings of ACM SIGMOD Conference*, 1999.
- [15] Y. Matias, J. Vitter, and M. Wang. Dynamic Maintenance of Wavelet-Based histograms. In *Proceeding of 26th VLDB Conference*, 2000.
- [16] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitions in two dimensions: Algorithms, complexity, and applications. In *Database Theory - ICDT'99*, 1999.
- [17] G. Piatetsky-Shapiro and C. Connell. Accurate Estimation of the number of Tuples Satisfying a Condition. In *Proceedings of the ACM SIGMOD Conference*, pages 256–276, 1984.
- [18] V. Poosala and Y. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proceedings of the 23rd VLDB Conference*, pages 486–495, 1997.
- [19] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *Proceedings of the ACM SIGMOD Conference*, May 1996.
- [20] L. Qiao, D. Agrawal, and A. El Abbadi. Supporting sliding window queries for continuous data streams. In *Technical Report, University of California, Santa Barbara*, 2003.
- [21] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *Proceedings of the ACM SIGMOD Conference*, 2002.
- [22] Y. Wu, D. Agrawal, and A. El Abbadi. Applying the Golden Rule of Sampling for Query Estimation. In *Proceedings of ACM SIGMOD Conference*, pages 449–460, 2001.
- [23] G. K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, Reading, MA, 1949.