Dynamic Multidimensional Histograms

Nitin Thaper
MIT
nitin@theory.lcs.mit.edu

Sudipto Guha
University of Pennsylvania
sudipto@cis.upenn.edu

Piotr Indyk
MIT
indyk@theory.lcs.mit.edu

Nick Koudas
AT&T Research
koudas@research.att.com

ABSTRACT

Histograms are a concise and flexible way to construct summary structures for large data sets. They have attracted a lot of attention in database research due to their utility in many areas, including query optimization, and approximate query answering. They are also a basic tool for data visualization and analysis.

In this paper, we present a formal study of dynamic multidimensional histogram structures over continuous data streams. At the heart of our proposal is the use of a dynamic summary data structure (vastly different from a histogram) maintaining a succinct approximation of the data distribution of the underlying continuous stream. On demand, an accurate histogram is derived from this dynamic data structure. We propose algorithms for extracting such an accurate histogram and we analyze their behavior and tradeoffs. The proposed algorithms are able to provide approximate guarantees about the quality of the estimation of the histograms they extract.

We complement our analytical results with a thorough experimental evaluation using real data sets.

1. INTRODUCTION

The explosive growth of networking in recent years has impacted the way we carry our every day tasks. We transmit enormous amounts of information through the internet, in forms of emails, streaming media (audio, video), images or documents on a daily basis. It is estimated that approximately 2.5×10^{16} bits flows through the internet on a single day.

This increase in network connectivity and usage has inevitably exacerbated the complexity of network management operations. Network operators are faced with challenging tasks including capacity planning, fault management, alarm and fault correlation and dynamic bandwidth allocation on a large number of network elements. Operators, as well as network management applications, rely increasingly on *data analysis* to facilitate these tasks. For example, commonly, operators require to understand or visualize the network traffic between two or more entities at various levels of detail. Such entities include internet domains, routers or even individual IP addresses. Traffic can be represented either as total number of bytes or packets from one entity to the other. Consider for example two network domains each encompassing a number of IP addresses. It is often desired to understand the traffic volume between individual IP addresses in the two domains. Analysis of such information in terms of visualization, can provide valuable insight about congestion, bandwidth allocation or planning. Moreover, query capabilities are also desirable, such as requesting the aggregate traffic from a range of addresses to another. Clearly, such a scenario can be generalized to more than two domains. A similar scenario could involve other network entities, such as individual routers and their interfaces etc.

A natural way to view information flow through a network, is that of a *continuous data stream*. The management solution consists of inspecting the data as it flows by and perform necessary computation for purposes of analysis without storing most of the data. Each entity in the stream or *stream tuple*, consists of a number of attributes. For example, in the network traffic domain, the stream tuple might have as attributes the source and destination of the packet information as well as a a measure attribute, such as bytes sent [1].

The database community has been on the forefront of providing the data management solutions. However much remains to be done in this context. Network elements generate enormous amount of data at very high rates. The amounts of data as well as their generation rates render materialization of data in secondary storage impossible. Even logging or accumulating the information for a small period of time can give rise to hundreds of gigabytes of data. As a result, we are seeking techniques that can effectively approximate the distribution of continuous streams of data in an *incremental* and highly efficient way. In particular, the techniques have to be able to maintain important traffic statistics and summary diagrams *without* storing all (or even a significant fraction of) the data.

One of the most natural and useful summary represen-

[1] This is exactly the way data is represented on IP packet headers.

ACM SIGMOD '2002 June 4-6, Madison, Wisconsin, USA

tations of the data for the purpose of data visualization and analysis are *histograms*. Histograms are a very popular and flexible way to track the distribution of the data in a database. They have been studied extensively and a plethora of algorithms exists for their efficient construction on a single [22, 28, 7, 31, 14, 13] or on multiple [30, 33, 25, 16, 5, 35] attributes. With a few exceptions however, the bulk of the work in this area, has addressed the *static* version of the problem; that is, given a multi attribute data set which is assumed static and materialized on secondary storage, and a fixed amount of space, construct the "best" histogram, i.e., the histogram minimizing estimation error, for suitably defined notions of error, depending on the particular application context. A common assumption in various works in this direction is that a new histogram is re-computed from the data, when changes in the data take place. As such, these proposals for approximating data distributions do not gracefully address the problem in a continuous data stream context. This is because (a) it is impossible to access the data in a stream on demand, while storing all the data on disk for future use is infeasible and (b) stream tuples arrive dynamically, so the distribution needs to be updated all the time.

In this paper, we address the problem of computing and maintaining *dynamic histogram* structures in a continuous data stream context. The techniques presented in this paper enable us to compute *multi-dimensional* histograms of the data[2]. At the heart of our proposal is the use of a *dynamic summary data structure*, which we refer to as a *sketch*. The sketch maintains succinctly the stream tuple distribution. Arrivals of new stream tuples are very efficiently reflected on the sketch. The sketch essentially acts as a dynamic snapshot of the stream tuple distribution. A histogram structure of the multi-attribute stream, can be efficiently and on demand derived from the sketch.

This approach opens numerous opportunities for effective management of continuous streams. For example, assuming that a sketch is associated with each network element, querying, analysis or visualization of the continuous streams from each network element can be efficiently performed by examination of the histogram extracted from the corresponding sketch. Moreover, it offers an efficient way of comparing continuous streams *temporally*. This can be done by comparing the histograms extracted from the sketches tracking the distribution of the same streams at different time periods. For example, consider comparing the histogram of the traffic distribution on a router today with the corresponding histogram of yesterday. Alternatively, consider observing correlations between traffic, in terms of number of bytes versus number of packets, by comparing the histograms of byte and packet distributions, and so on. In addition, sketches of different data streams can be *composed* (by simply adding them together), yielding a sketch of the union of the individual data streams. This becomes useful in scenarios where the data is gathered by many separate agents (e.g., routers) and needs to be combined together to obtain a summary of the overall data flow. Figure 1 presents an overview of our approach.

This paper is organized as follows: In Section 2 we review related work. Section 3 provides definitions necessary for our subsequent discussions. Section 4 introduces sketches for multidimensional distributions and presents their operation, properties and incremental behavior. In Section 5 we present algorithms with approximate guarantees, to extract a multidimensional histogram from the sketch, and analyze their complexity as well as introduce various improvements on the basic algorithmic approaches introduced. Section 6, builds on the algorithmic intuition gained, and proposes empirical approaches, improving the performance of the proposed algorithms further. Section 7 presents a thorough experimental evaluation of the algorithms presented herein using real data sets. Section 8 concludes the paper and points to problems of interest for further study.

[2]Thus we are able to maintain e.g., overall summary of the amount of traffic from each source to each destination, by maintaining a two-dimensional histogram of the traffic data.

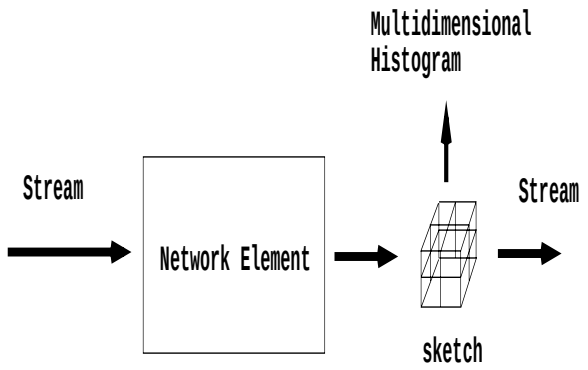


Figure 1: Overview of our approach: A *sketch* is incrementally updated from the stream, tracking the stream distribution succinctly. A multidimensional histogram is efficiently derived from the sketch on demand.

2. RELATED WORK

Histogram structures have been studied extensively in the database community, due to their utility in selectivity estimation for query optimization and approximate query answering. Early approaches to selectivity estimation and approximate query answering, focused on the problem of maintaining the distribution of a single attribute using histograms [24]. A large body of work addresses this problem with the use of sampling [19, 17, 18]. Various histogramming algorithms [31, 27, 34, 2] as well as the provably optimal, [22] and near-optimal [14, 13], approaches have been proposed in the case of a single attribute. Dynamic maintenance of histograms in one dimension has also been addressed [1, 7, 28, 10, 8]. The last two papers used sketches as a way of summarizing the data.

Unlike the one-dimensional case, constructing optimal histograms in multiple dimensions is NP-hard [29]. Thus, many proposals exist for this problem. Poosala and Ioannidis [30] proposed algorithms for multidimensional histogram construction. Several heuristics with *provable* worst-case guarantees have been also proposed in [23, 29]. In particular, the algorithm of [23] used greedy approach to solve a histogram construction problem in 2D. Others studied the application of various transforms [25, 33] to this problem. Kollios et. al., [16] proposed a kernel based algorithm to construct histograms in many dimensions and experimentally showed the algorithm superior in accuracy to previous approaches. Wu

et al., [35] applied the golden rule of sampling to query estimation.

All these works deal with the static version of the multidimensional histogram construction problem; that is, a common assumption is that a histogram is rebuilt periodically from the data to reflect changes in the underlying data distribution. Recently Bruno et al., [5] studied the problem of dynamic histogram construction in multiple dimensions by observation of query results. They proposed an algorithm named `STHoles` and experimentally demonstrated that it is comparable in accuracy to the best algorithm proposed for the static version of the multidimensional histogram construction problem.

Continuous data streams, have attracted lots of recent research attention in both the database [4, 26, 13, 10, 32, 12] as well as the theory community [6, 3, 20, 15, 14, 9, 8]. We mention that the paper [9] investigated stream problems occurring in networking. Also, the paper [8] presented a method for reconstructing one-dimensional histograms from sketches. Their algorithms were obtained either by reconstructing histogram from wavelet representation (obtained as in [10]), or via greedy reconstruction of histogram (as in [23] or in this paper). However, their paper deals exclusively with histograms in one dimension.

3. DEFINITIONS

Let r be a continuous data stream on an attribute set [3] $\{A_1, \ldots A_\ell\}$. Without loss of generality assume that each attribute $A_i, 1 \leq i \leq \ell$ has a numerical domain $\Delta = \{1 \ldots n\}$. A tuple $t \in r$ can be viewed as a multidimensional point in $\{1 \ldots n\}^\ell$. The *frequency distribution* of r is a function $D : \{1 \ldots n\}^\ell \rightarrow \{1 \ldots M\}$. For $t \in \{1 \ldots n\}^\ell$ the value $D(t)$ measures the number of times tuple t appears in r. This function $D(t)$ defines a distribution over the tuples. The streaming data we will encounter will be a sequence of tuples t_i. A simple generalization of the data would be to represent the data as a sequence $(t_i, \pm)$ where the positive symbol would signal the arrival of a new tuple t_i and the negative symbol would indicate that the tuple t_i has expired and is no longer relevant. If we are conceptualizing a snapshot of the distribution at a point of time, an arrival corresponds to an insert operation and an expiry, a delete operation. We can model an update by a combination of the two. Thus the streaming model already captures dynamic databases, if we inspect the stream of transactions on the database.

In this paper we address the problem of approximating the multidimensional frequency distribution of a stream of tuples. Our discussion equally applies if D is a general distribution over some discrete domain, as it would be appropriate for approximating a datacube [11, 33]. We restrict the bulk of our discussion on the use of piecewise-constant functions as basis functions for the approximation; we generalize to other functions of interest in section 5.2.

Our goal is to approximate the distribution D by a *histogram*. Formally a histogram is a function $H : \{1 \ldots n\}^\ell \rightarrow \{1 \ldots M\}$. Each histogram is defined by a sequence of hyperrectangles $S_1 \ldots S_k$ each $S_i \subset \{1 \ldots n\}^\ell$ and a sequence of values $v_1 \ldots v_k$, each corresponding to a hyperrectangle. For $t \in \{1 \ldots n\}^\ell$, $H(t)$ represents an estimate to $D(t)$. Depending on the type of histogram, as explained below, $H(t)$ is derived from one or more v_i values. In practice we represent histogram H as a sequence $\{(S_1, v_1) \ldots (S_k, v_k)\}$. We will consider the following classes of histograms:

- *Tiling histograms*: the hyperrectangles form a tiling of $\{1 \ldots n\}^\ell$ (i.e., they are disjoint and cover the whole domain). For any t we have $H(t) = v_i$, where $t \in S_i$
- *Non-overlapping histograms*: the hyperrectangles are disjoint. For any t we have $H(t) = v_i$, if there exists S_i containing t; $H(t) = 0$ otherwise.
- *Priority histograms*: the hyperrectangles can overlap. For any t we have $H(t) = v_i$ where i is the largest index such that $t \in S_i$; if none exists, $H(t) = 0$.
- *Additive histograms*: the hyperrectangles can overlap. For any t we have $H(t) = \sum_{i:t \in S_i} v_i$ (the value of $H(t)$ is 0 if there is no S_i containing t).

Commonly each hyperrectangle S_i is referred to as a bucket. We will refer to a histogram that consists of k buckets, as a *k-histogram*. Observe that in all the above models, if we increase k we can capture the distribution more accurately, and if we were to store n^ℓ buckets, we would capture the data exactly.

Observe that both the distributions and the histograms can be viewed as vectors in an N-dimensional space. This observation is immediate for one-dimensional distributions, since they are represented by a vector of $N = n$ numbers. However, a similar situation holds for the multidimensional case. For example a two-dimensional distribution D defined over an $n \times n$ square can be viewed as a point in an N-dimensional space for $N = n^2$ and in general for an ℓ-dimensional distribution as a point in an $N = n^\ell$ space. To view a distribution D this way however, we assume a fixed way to linearize the domain, such as row major. The same holds for ℓ-dimensional histograms; in this case the coordinates corresponding to regions of the n^ℓ space covered by a hyperrectangle S_i have the same value v_i. In this case we also assume a row major linearization order.

In the remainder of this paper we will treat D and H both as functions (represented as sets) as well as N-dimensional vectors derived from a row major linearization, whenever convenient. The multidimensional histogram construction problem is defined as follows:

DEFINITION 1 (OPTIMAL MULTIDIMENSIONAL HISTOGRAMS). *Given a distribution $D : \{1 \ldots n\}^\ell \rightarrow \{1 \ldots M\}$ and a fixed budget of buckets k, construct the k-histogram, H, minimizing $||D-H||_2$ (the L_2 distance between the two distributions)*

Each stream operation, in the form of a tuple arrival or expiry, can potentially change D. Such operations can be interleaved arbitrarily and take place on subsets of attribute values of a set of tuples in r. In fact this is an important aspect of stream analysis. Even if a k-histogram according to definition 1 is identified, this histogram no longer satisfies the criteria of the definition, when distribution D changes. A new histogram has to be identified using the new distribution resulting from the change. We will present algorithms to maintain a snapshot of D under any sequence of changes

[3] We can view the stream as a dynamic realization of the relation schema $R(A_1, \ldots A_\ell)$ but since we will not be storing the relation we will avoid this representation.

and subsequently to extract a k-histogram according to definition 1 with approximate guarantees. We will present our proposal in the following steps:

- First we introduce a summary data structure, which we refer to as a *sketch*, capable of maintaining succinctly a snapshot of D whenever changes occur, showing its properties and incremental behavior.
- We will introduce algorithms to extract a k-histogram according to definition 1 with approximate guarantees and analyze the running time of these algorithms.
- Building on the intuition gained from these algorithms, we will propose heuristic approaches to extract multidimensional histograms from a sketch and study their performance and tradeoffs.

4. SKETCHING MULTIDIMENSIONAL DISTRIBUTIONS

We will show how to maintain an accurate snapshot of any distribution D, and show how to incrementally maintain it, so it remains accurate under arbitrary modifications to D. For this purpose, we introduce the following Johnson-Lindenstrauss theorem.

THEOREM 1. *Consider a random linear mapping $A : \Re^N \to \Re^d$, such that each entry of the matrix A is chosen independently from a certain distribution[4]. If $d = O(\log(1/P)/\epsilon^2)$, then the mapping A has the property that for any fixed $x \in \Re^N$ we have $\|x\|_2 \leq \|Ax\|_2 \leq (1+\epsilon)\|x\|_2$ with probability at least $1 - P$.*

Consider any distribution D viewed as a vector in an N dimensional space. Similarly consider a set K of N dimensional vectors. A straightforward application of the union bound implies that a random mapping, A, for $d = O(\log(|K|/\epsilon^2))$ has (with high probability) the property that for any vector $v \in K$, $\|v - D\|_2 \leq \|Av - AD\|_2 \leq (1+\epsilon)\|v - D\|_2$. Thus, if we maintain only the "sketch" AD, by minimizing $\|Av - AD\|_2$ we can recover the element $v \in K$ which is closest to D in L_2 sense. We will show how to perform this minimization in section 5. Notice however, that this is clearly beneficial, because both D and v are N dimensional vectors, but AD and Av are d-dimensional. Also, the matrix A doesn't have to be stored explicitly. Its entries can be generated by using a pseudorandom number generator with jumpahead capability. Provided that $d \leq N$, significant savings in space and computation time can be achieved.

Now consider the distribution D under dynamic changes (arrivals,expiry). Recall that we modeled distribution D as an N dimensional vector. An arrival corresponds to an entry $(t_i, +)$ in the stream; this can be represented by an N dimensional vector (say U), which is only non-zero at the coordinate corresponding to t_i. The same approach works for $(t_i, -)$. The non zero value determines the type of change, being positive for an arrival operation and negative for an expiry. The change is reflected to D by a linear operation between the two vectors.

[4]Many distributions can be used here, e.g., Gaussian distribution or uniform distribution over $\{-1, 1\}$ (after scaling). We use a variant of the latter.

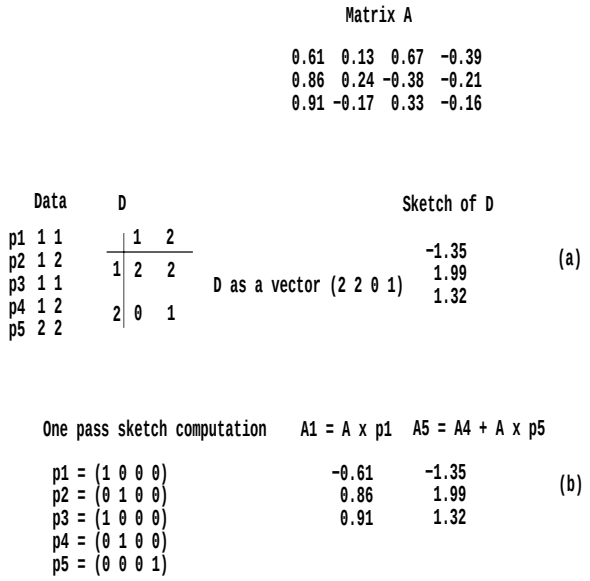


Figure 2: A two dimensional frequency distribution (a) Computing the sketch of a known frequency distribution and (b) One pass sketch computation via incremental change, for each data point $p_i, A_i = A_{i-1} + Ap_i$

EXAMPLE 1. *Consider a two-dimensional distribution D expressed as a four dimensional vector $D = (1, 3, 4, 1)$. Incrementing 3 to 4 as a result of an arrival of a value (which is the linear index numbering the tuple) can be performed via a vector $U = (0, 1, 0, 0)$ with the linear operation $D + U$. Similarly for expiry operations. Observe that the result is an insert or delete in the linear realization. However as mentioned before, this is only an analogy. Since the data is not stored in this model of computation, an insert is not concretely defined. Thus we will stick to our description of arrival/expiry.*

Clearly, such an operation can be performed in $O(1)$ time. Notice, that this way of reflecting change to D can handle bulk insertions or deletions on single or multiple attribute values. If we compute the sketch AD, we can maintain it efficiently since for any vector U expressing change we have, $A(D+U) = AD + AU$. If U is non-zero only at one position as before, we can compute AU in $O(d)$ time. Bulk arrivals or expiry are handled in a similar way. Given a specific relation r, of known frequency distribution, deriving AD involves a simple matrix multiplication. Since sketches are amenable to incremental updates, this suggests a strategy to compute the sketch of a multidimensional data set from scratch with a *single* pass on r, without knowing r's frequency distribution in advance. Provided that the domain of each attribute is known, we initialize matrix A according to Theorem 1 and a sketch S of size d setting each coordinate to zero. For each tuple t of r we perform incremental updates to S by adding vector At. This operation, requires a single scan of r and can be performed in main memory requiring $O(d \times |r|)$ operations. Figure 2 shows an example of this operation.

5. EXTRACTING A HISTOGRAM FROM THE SKETCH

The key idea behind our algorithms is to treat the distribution and the histograms as points in high-dimensional space. By maintaining a sketch of the distribution, the problem of finding the optimum histogram can be solved by computing a histogram whose sketch is "close to" the sketch of the data distribution. Consider any distribution D and the set $\mathcal{H}$ of all k-histograms. Recall that we view D and elements of $\mathcal{H}$ as points in an N-dimensional space. Observe that the number of all k-histograms is at most $n^{2\ell k}M^k$, since there are $n^{2\ell}$ possible hyperrectangles to be considered for the k-histogram and each possible k hyperrectangle collection, is a candidate. Moreover in each collection of k hyperrectangles there are M^k possible values to assign. Therefore, a random mapping A for $d = O(\log(n^{2\ell k}M^k)/\epsilon^2) = O(k\ell \log n/\epsilon^2)$ has (with high probability) the property that for any histogram H we have $\|H - D\|_2 \leq \|AH - AD\|_2 \leq (1+\epsilon)\|H-D\|_2$. Thus, by maintaining the "sketch" AD, we can recover the best k-histogram by minimizing $\|AH-AD\|_2$ over all histograms H.

To recover the "best" histogram, we need to solve an optimization problem over the space of histograms. Observe that if we knew the intervals in the domain of each attribute defining the histogram (without knowing the functions within each interval), we could solve the optimization problem via the Least Squares method. This immediately gives an $n^{2k\ell}d^{O(1)}$ algorithm by enumerating all possible subsets of hyperrectangles and finding the best value to set them to. Unfortunately, such algorithms have unreasonably large running time.

We will present a technique to extract a histogram with provable properties from the sketch AD of a multidimensional frequency distribution D, where A is chosen according to Theorem 1. We will show that if we change the sketch error making it $1+\epsilon/k$ instead of $1+\epsilon$ as dictated by Theorem 1, we can recover approximately the structure of the best k-histogram by extracting only one bucket at a time. This effectively allows us to apply greedy search methods and reduce the time of histogram construction.

In the remainder of this section we focus on the problem of retrieving the best histogram from the sketch AD. Consider the ℓ-dimensional distribution D defined over $N = n^\ell$. Algorithm GREEDY shown in Figure 3 retrieves a *priority* histogram from the sketch AD. The algorithm will output an optimum histogram with B buckets for $B > k$. The exact relationship of B to k will be established in Theorem 2.

At first, the algorithm initializes histogram H to empty. The main loop of the algorithm iterates B times, and at each iteration a bucket of the histogram is extracted. In each iteration, the algorithm enumerates all hyperrectangles in the domain space $\{1 \ldots n\}^\ell$. There are $n^{2\ell}$ such hyperrectangles. Given a currently optimum histogram H, it considers each hyperrectangle S for addition to the optimum histogram solution. Let H_S be the histogram obtained by adding S to the optimum solution H. The value that hyperrectangle S will assume, if added to the optimum solution, is yet to be determined and is initialized to the indeterminate variable x.

The algorithm proceeds computing the sketch of H_S. Conceptually, the sketch of H_S can be computed by viewing the histogram H_S as a vector $\bar{H_S}$ of size N. Each coordinate of this vector is a point p in the domain of D. If $p \notin S$, then the value of this coordinate is $H(p)$, exactly the same as the estimate for p in the current optimum histogram H. To compute value $H(p)$ from H, since H is a priority histogram, we have to search the buckets of H and find the bucket (hyperrectangle) with the largest index, containing p. Searching through the buckets of H can be performed in $O(B)$ time in the worst case. If however, the point p belongs to S, then the corresponding coordinate is set to the (yet to be determined) value x. In step (2) of the algorithm, the sketch $A\bar{H_S}$ is computed by multiplying the $d \times N$ matrix A with vector $\bar{H_S}$. This multiplication is performed in time $O(n^\ell d)$.

Then, in step (3) the algorithm assesses the L_2 error between the sketch of the new histogram and the sketch of the distribution. The resulting function $C_S(x)$ is a quadratic function which is minimized in step (4). Notice that this corresponds to computing $\min_x(ax-b)^2$ for some coefficients a, b and thus the minimum is achieved by setting $x = 2b/a$, which takes constant time. The factors in the running time are, the number of repetitions in the outer loop of the algorithm, B, the number of hyperrectangles, $n^{2\ell}$, the number of coordinates of the sketch, d and the time needed to compute the sketch of H_S, $n^\ell B$. The complexity of the algorithm is $O(n^{3\ell}dB^2)$. Figure 4 presents an example showing the operation of the algorithm.

The guarantee for the quality of the histogram returned by algorithm GREEDY is established by the following Theorem.

THEOREM 2. *Let D be the distribution and let H^* be the tiling k-histogram which minimizes $\|D-H^*\|_2^2$. If the sketching procedure preserves the distances exactly, then the priority histogram H reported by* GREEDY *satisfies $\|D-H\|_2^2 \leq \|D-H^*\|_2^2$.*

PROOF. The initial squared error of H is at most NM^2, since all coordinates of D are smaller than or equal to M. Consider H at any stage of the algorithm. If we added all rectangles from H^* to H with appropriate values, the error of H would be reduced from $\|D-H\|_2^2$ to $\|D-H^*\|_2^2$. Thus, one of the rectangles must reduce the error by at least $1/k \cdot (\|D-H\|_2^2 - \|D-H^*\|_2^2)$. Therefore, if we add the best rectangle S to H with the best value (forming H_S), we have that

$$\|D-H_S\|_2^2 - \|D-H^*\|_2^2 \leq (1-1/k)(\|D-H\|_2^2 - \|D-H^*\|_2^2)$$

After i stages we have,

$$\|D-H\|_2^2 - \|D-H^*\|_2^2 \leq (1-1/k)^i NM^2$$

If we set $i \approx k\ln(NM^2)$, then the difference becomes at most

$$(1-1/k)^{k\ln(NM^2)}NM^2 < e^{-\ln(NM^2)}NM^3 = 1$$

Since the difference must be an integer, it is equal to 0. □

In the case of our algorithm, the sketches preserve the distances between D and H only approximately. However, the following holds:

THEOREM 3. *Let D and H^* be as before. If the sketching procedure preserves the distances up to a factor of $(1+\epsilon/k)$, then the priority histogram H reported by* GREEDY *satisfies $\|D-H\|_2^2 \leq (1+\epsilon)\|D-H^*\|_2^2$.*

ALGORITHM GREEDY:
Distribution $D : \{1 \dots n\}^\ell \to \{1 \dots M\}$
Histogram H *with* B *buckets, represented as a sequence of hyperrectangles* (S_i, v_i)
Matrix A *chosen according to Theorem 1*
Sketch AD *of* D *computed with a single pass over the data set*
Set $N = n^\ell$
Initialize the histogram H *to empty*
For $i = 1$ *to* $B = k \ln(NM)$
  For all hyperrectangles $S \subset \{1 \dots n\}^\ell$
    (1)Create the histogram $H_S[x]$ *obtained by adding the rectangle* S *to* H *and setting its value to the indeterminate variable* x
    (2)Transform $H_S[x]$ *to its vector representation* $\bar{H}[x]$
    Compute the sketch $A\bar{H}_S[x]$*; note that the sketch is a linear function in* x *with values in* $\Re^d$
    (3)Define $C_S(x) = \|A\bar{H}_S[x] - AD\|_2^2$*; observe that* $C_S(x)$ *is a quadratic function of* x
    (4)Compute x *which minimizes* $C_S(x)$ *and denote it by* x_S
  Let S *be the rectangle with the smallest value of* $C_S(x_S)$
  Add S *to* H *with value* x_S

Figure 3: Algorithm GREEDY

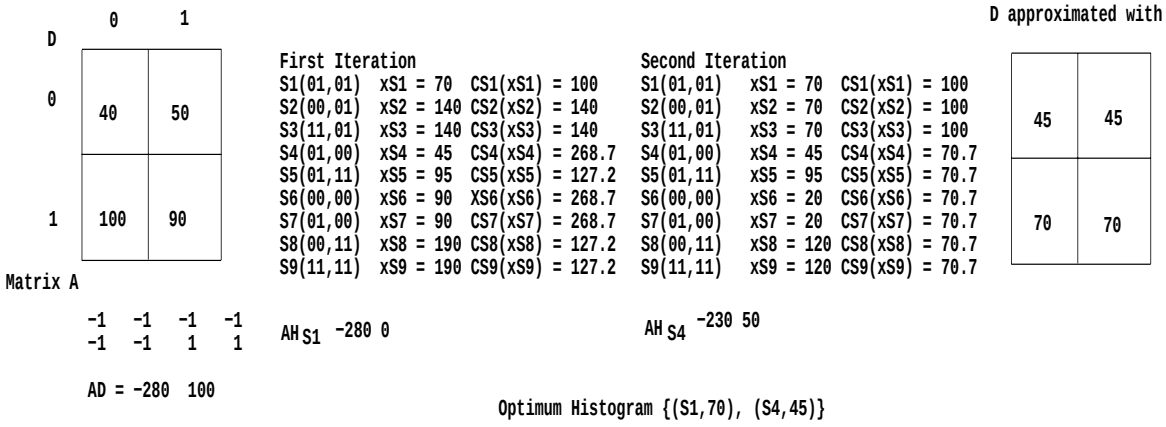


Figure 4: Example run of algorithm GREEDY, using $d = 2$, and two dimensional data space D ($n = 2$). Rectangles S_i represented with their extent in each dimension (horizontal,vertical). After the first iteration rectangle S_1 is added to the optimum histogram. At the end of the second iteration S_4 is added.

Thus, algorithm GREEDY provides a method for extracting a near-optimal histogram from a sketch which can be incrementally maintained under insertions, deletions and updates, in polynomial time. However, the histogram recovery time is very high. We present a sequence of modifications to the basic GREEDY algorithm which will allow us to decrease the running time by several orders of magnitude.

5.1 Improving the Running Time

We will be considering a series of improvements to the basic strategy of the algorithm in order to improve the running time. To ease presentation we will restrict our discussion to the case of two dimensions ($\ell = 2$). Generalization to more dimensions is straightforward.

Our first modification involves only the way we compute the sketch $A\bar{H}_S[x]$ and does not change the semantics of the GREEDY algorithm. The basic idea of the improvement is to observe that we can compute *all* sketches $A\bar{H}_S[x]$ much faster than in time n^6 times the cost of computing *one* sketch. Notice that in step (2) of algorithm GREEDY, a new sketch is computed for each rectangle S; this computation requires time $O(n^2 B)$ for *each* rectangle S, in the worst case. We will take advantage of the fact that computing each coordinate of $A\bar{H}_S[x]$ essentially involves summing up all entries of A corresponding to points in S. By enumerating the rectangles S in a proper order this can be done in constant (rather than $O(n^2 B)$) time per rectangle, using only n additional units of storage.

The way to perform this computation is as follows. Consider a rectangle S=$\{1 \dots u\} \times \{1 \dots v\}$. We will show how to compute sketches for $H_{S'}[x]$ for all $O(n^2)$ S' that are obtained by "translating" S. Given rectangle S, the set of all $O(n^2)$ rectangles obtained from S by translation, has as lower left coordinates $i, j, 1 \le n - u, 1 \le j \le n - v$. We show how to compute the first coordinate of the sketch $A\bar{H}_{S'}[x]$; the remaining $d - 1$ coordinates are computed in the same way. Let a be the first row of A. Our goal is to compute the dot product $a \cdot \bar{H}_{S'}[x]$ for every S'. We will actually compute $a \cdot (\bar{H}_{S'}[x] - \bar{H})$, (where $\bar{H}$ corresponds to the vector representation of H) and then use the formula $a \cdot \bar{H}_{S'}[x] = a \cdot \bar{H} + a \cdot (\bar{H}_{S'}[x] - \bar{H})$. This formula demonstrates the computation we will perform for exposi-

tion purposes only; as it will become evident, we *do not* need to compute the vector representations of H and $H_{S'}$ ($\bar{H}$ and $\bar{H_{S'}}$ respectively). Notice that if a is the first row of A, then $a \cdot \bar{H}$ is the first coordinate of the sketch of H. It remains to show how to compute, $a \cdot (\bar{H_{S'}}[x] - \bar{H})$.

Let $T : \{1 \dots n\}^2 \to \Re$ be a function such that $T(p) = a_k \cdot (\bar{H_{S'}}[x] - \bar{H})(p)$, where $k \in \{1 \dots n^2\}$ is the index in a corresponding to the point p. Observe that $a \cdot (\bar{H_{S'}}[x] - \bar{H}) = \hat{T}(q)$, where q is the upper-left corner of S' (with lowest values of coordinates) and $\hat{T}(q) = \sum_{p \in S'} T(p)$. Thus, it suffices to compute $\hat{T}(q)$ for all points q, using small space (i.e., without explicitly maintaining the matrix T) and in $O(n^2)$ time. This is done as follows. First, for each $i = 1 \dots n$, compute and store the "column sum" $T_i = \sum_{j=1}^{u} T(j, i)$. Note that each T_i can be computed in $O(nB)$ time, directly from histogram H. Then $\hat{T}(1,1) = \sum_{i=1}^{v} T_i$, $\hat{T}(1,2) = \hat{T}(1,1) + T_{v+1} - T_1$, etc. Thus we can compute all values $\hat{T}(1, \cdot)$ in $O(nB)$ time. In order to compute values $\hat{T}(2, \cdot)$, we first update T_i's via assigning $T_i := T_i - T(1,i) + T(u+1,i)$. Then we proceed as before. Altogether, we can compute all values of $\hat{T}(q)$ in $O(n^2 B)$ time, using n units of storage.

We remark that although our algorithm is not in-place (i.e., it uses non-constant units of storage), the storage is used only temporarily for processing information. This means that (unlike the memory used to store sketches), the *same* memory region can be used to process histograms of *many* relations.

We can further reduce the running time in practice, without sacrificing the guarantees of Theorem 3. The idea is to choose (in the step (4) of the algorithm) a rectangle S which is "good-enough", as opposed to "the best one". Let H_S be the histogram resulting after adding histogram S to H. Specifically, we choose the *first* rectangle S such that

$$\|D - H\|_2 - \|D - H_S\|_2 > \alpha/k \cdot \|D - H\|_2$$

for a parameter $\alpha > 0$. Clearly, this method generates at most $k/\alpha \cdot \ln(NM^2)$ rectangles in the output histogram. At the same time, if no rectangle S satisfies the above inequality (i.e., no choice of S yields significant improvements to the quality of approximation), we can conclude that

$$\|D-H\|_2 - \|D-H^*\| \le k(\|D-H\|_2 - \min_S \|D-H_S\|_2) \le \alpha \|D-H\|_2$$

which implies that

$$\|D - H\|_2 \le \frac{1}{1-\alpha} \|D - H^*\|$$

and thus H is already an almost optimal solution. Therefore, we output a histogram with at most $O(k \ln(NM^2)/\alpha)$ buckets, with cost at most $(1+\alpha)$ larger than the cost of H^* (for small α).

The benefit of using this version of the algorithm is that during one enumeration of all rectangles S we can choose *several* rectangles to add to H. This version of the algorithm is expected to have running time reduced by a factor up to B. We will assume the running time of $O(n^4 dB)$ in further analysis. We will refer to algorithm GREEDY incorporating these improvements as IMPROVED GREEDY

5.2 Extending IMPROVED GREEDY to Other Basis Functions

It is possible to extend the histogram construction to other basis functions namely linear or quadratic functions, where each hyperrectangle is equipped with a function that computes the contribution of this hyperrectangle towards the distribution of the tuple t_i. For this purpose, the basic algorithm GREEDY needs to be modified, in order to optimize the choice of several parameters per bucket (e.g., a linear function in two dimensions is represented by 3 parameters). This can be done in a way similar to the 1-dimensional optimization employed for the piecewise constant case [22]. All the possible ways of combining hyperrectangles, namely, tiling, priority, non-overlap, additive etc. apply to this case as well. Linear or quadratic functions usually result in a better fit for a single hyperrectangular area, since we have more than one value to represent the function.

6. FASTER EMPIRICAL APPROACHES

Inspired by the operation of the algorithms and the improvements presented, in this section we reduce running time further and present empirical approaches which we subsequently evaluate. Again, we restrict our discussion to the two dimensional case ($\ell = 2$) to ease presentation. Our discussion generalizes in a straightforward way to more than two dimensions.

We consider replacing priority histograms in algorithm IMPROVED GREEDY with *additive* histograms. In this case, the running time can be reduced by a factor of B. In priority histograms, step (2) of IMPROVED GREEDY the sketch of the candidate histogram H_s is computed from the sketch of the currently optimum histogram H. Recall, that during this computation, one requires for each $p \in S$ the value $H(p)$, to assess the difference $H(p) - x$. Computing $H(p)$ takes $O(B)$ time in the worst case, since we might need to scan all rectangles in H to find one which contains p. However in the case of additive histograms, for each $p \in S$ the difference between the estimate of H_S and that of H for point p is x if $p \in S$ or 0 otherwise, and thus updating $AH_S[x]$ is much faster. This leads to an algorithm with empirical running time roughly $O(n^2 \log^2 nd)$ and worst-case running time $O(n^2 \log^2 ndB)$.

The second modification involves restricting the search for the optimal rectangle S only among rectangles whose side lengths are powers of 2; we call such rectangles *regular*. This decreases the number of rectangles to consider from $\Theta(n^4)$ to $\Theta(n^2 \log^2 n)$. Furthermore, the rectangles found by our algorithms, usually have bounded aspect ratios and therefore can be represented as a union of a few squares. Thus, we restrict the search space in algorithm IMPROVED GREEDY even further, by considering all squares of various sizes instead of rectangles. This shaves off a factor of $\log n$, giving us a running time of $O(n^2 \log ndB)$.

Finally, we adapt the idea of considering "good-enough" rectangles in the approach of algorithm IMPROVED GREEDY, in this case as well. A drawback of the "good-enough" algorithm is a fixed choice of the parameter α. If α is too large, the resulting histogram can have large error. On the other hand, small value of α creates many buckets. To circumvent this issue, we use the following approach: after enumerating all rectangles S as candidates for extending H, we divide α by 2 and proceed further. In this way we require new rect-

angles to produce large gains at the beginning, and much smaller gains at the end when we are close to optimum. The empirical GREEDY algorithm (EGREEDY) we propose, incorporating these properties is shown in Figure 5.

7. EXPERIMENTAL EVALUATION

In order to assess the performance and accuracy of the proposed algorithms, we conducted a detailed performance evaluation. We start by presenting the data sets used in our study and continue with the description and presentation of our evaluation.

Data sets

We used both synthetic and real data sets in our experiments. The real data sets that we used, reflect real traffic information collected from operational router devices. The first real data set, which we refer to as *Traffic1*, represents the amount of traffic information, for a specific measure of traffic, at the granularity of a second, flowing through a number of network elements for the duration of an entire day [5]. The second data set, which we refer to as *Traffic2* is similar, but the measure used to quantify traffic demands is different. These data sets can be treated as two dimensional, by computing for every network element source and destination pair, the total amount of traffic for the corresponding traffic measure in each data set. There are 100 distinct sources and destinations in these data set, thus the domain size is 100x100 in these data sets.

We also use synthetic data sets in our experiments. The synthetic data sets are generated by a mixture of three Gaussians, centered at random points, with variances 3, 3 and 5, respectively; we refer to this data set as *Gauss*. All of our experiments were performed on a dual-processor Intel machine (Pentium II, 300 Mhz) with 256 Mb main memory and 512 Kb cache on each processor, running Redhat Linux 6.2.

7.1 Description of Experiments

There are two main parameters of interest in our approach, namely the time to construct the optimum histogram for the various algorithms and the accuracy of the resulting histograms. In this section we experimentally evaluate both parameters for the algorithms proposed.

To assess the quality of our algorithms for histogram extraction from a sketch, we compare them with histograms computed by an algorithm that operates directly on the data; that is, the algorithm does not use sketches, but instead assumes the distribution of the data is available and operates directly on the data distribution, computing a histogram from the actual data. For this purpose, we chose the recently proposed *STHoles* algorithm [5]. The nice feature of this algorithm is that it is dynamic in the sense that it learns a good multidimensional histogram from the data by posing queries. Moreover, it was experimentally demonstrated in [5] that the quality of the histograms constructed by the STHoles algorithm is comparable with the quality of histograms generated by other algorithms that have been previously shown to compute good multidimensional histograms. Thus, STHoles is a natural candidate to serve as a benchmark in our setting. As proposed by Bruno et. al., [5], we trained the STHoles algorithm using 1000 queries with 1% query volume. In contrast, our algorithms assume no a priori knowledge of the data distribution. With a single pass on the data (as the stream tuples arrive) we incrementally update a sketch of a specific size and, on demand, we run our algorithms to extract a histogram from the sketch.

For a query Q, let A_Q be the exact query answer computed by executing the query on the actual data, U_Q the query estimate assuming a uniform data distribution and H_Q the query result returned by the histogram. Following previous work [5, 21] we define the absolute relative ARE error as

$$ARE = \frac{|A_Q - H_Q|}{|A_Q - U_Q|}$$

The average absolute relative error (AARE) is computed by averaging ARE of a large number of queries uniformly distributed, chosen such that the volume of the range is equal to 1% of the total grid volume. It is given as a percentage in the graphs below.

7.2 Evaluating IMPROVED GREEDY

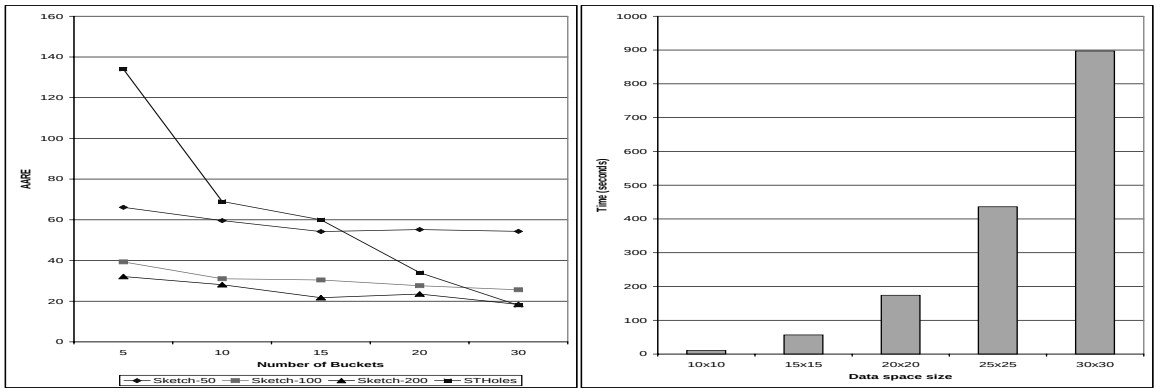
The first set of experiments evaluates the quality and performance of the IMPROVED GREEDY algorithm. Figure 6(a) presents the accuracy of the IMPROVED GREEDY algorithm as the number of buckets increases for different sizes of the sketch. The data set *Gauss* in used in this experiment with a domain of 20 in each dimension. The sketch size varies from 50 bytes to 200 bytes. One can observe from the figure that accuracy increases, with increasing number of buckets as expected. Moreover the histograms extracted by the algorithm become more accurate as the sketch size increases, since the sketch tracks the underlying distribution more accurately. For a small sketch size (50) the quality of the histogram remains low if we increase the number of buckets. In this case, the error induced by sketching is large enough to obscure any differences between accurate or inaccurate histograms. However, as we increase the sketch length to 100, we can see an improvement of the quality for larger number of buckets. This trend becomes even more visible for sketch length 200, where the error is reduced from 35% (for 5 buckets) to 18% (for 30 buckets). Figure 6(a) presents also the accuracy of algorithm STHoles as the number of buckets increases. For a small number of buckets and for various sketch sizes algorithm IMPROVED GREEDY outperforms STHoles by a large factor. Notice that STHoles has the exact data distribution at its disposal, but algorithm IMPROVED GREEDY operates only on an approximation of the distribution extracted from the sketch. As the number of buckets increases, STHoles improves; in this case algorithm IMPROVED GREEDY is comparable in accuracy.

Figure 6 presents the time algorithm IMPROVED GREEDY requires, to extract the optimum histogram for 10 buckets and a sketch of size 50 as the domain of the underlying data space increases. The running time is consistent with the analytical expectations, and increases fast as a function of the domain size of the underlying stream. Thus, although algorithm IMPROVED GREEDY is very accurate, being able to be comparable (as well as outperform) in accuracy algorithms having exact knowledge of the data distribution, the time required to extract the guaranteed optimum histogram is high. Similar results were obtained for the real data sets as well as additional synthetic data sets we experimented with during the course of this study. Algorithm EGREEDY compensates the high running time of IMPROVED GREEDY.

[5]The proprietary nature of these data sets prohibits us from providing additional details.

ALGORITHM EGREEDY:
Distribution $D : \{1 \dots n\}^\ell \to \{1 \dots M\}$, *represented as an* $N = n^\ell$ *vector*
Histogram H *with* B *buckets, represented as a sequence of rectangles* (S_i, v_i)
SH *the sketch of* H, *a* d*-dimensional vector*
Matrix A *chosen according to Theorem 1*
Sketch AD *of* D *computed with a single pass over the data set*
Parameter α
Initiate the histogram H *to empty*
For $i = 1$ *to* $B = k \ln(NM^2)$
  For all squares $S \subset \{1 \dots n\}^\ell$
    (1)Create the histogram $H_S[x]$ *obtained by adding the rectangle* S *to* H *and setting its value to the indeterminate variable* x
    (2)Compute the sketch $A\bar{H}_S[x]$ *from* SH *according to section 5.1*
    (3)Define $C_S(x) = \|A\bar{H}[x] - AD\|_2^2$; *observe that* $C_S(x)$ *is a quadratic function of* x. *Define* $C = \|SH - AD\|_2^2$
    (4)Compute x *with* $\|C - C_S(x)\| > \alpha/k \cdot C$ *and denote it by* x_S
  Let S *be the rectangle satisfying (4) and* $A\bar{H}'$ *the corresponding sketch*
  Add S *to* H *with value* x_S, *set* $SH = A\bar{H}'$ *and* $\alpha = \frac{\alpha}{2}$

Figure 5: Algorithm EGREEDY



(a) IMPROVED GREEDY accuracy (b) extraction time

Figure 6: Data set *Gauss*: (a) Accuracy of IMPROVED GREEDY **algorithm with increasing number of buckets for various sketch sizes (b) Histogram extraction time for the algorithm, for a sketch size of 50 and 10 buckets as the domain of the underlying data space increases**

We present an evaluation of this algorithm in the sequel.

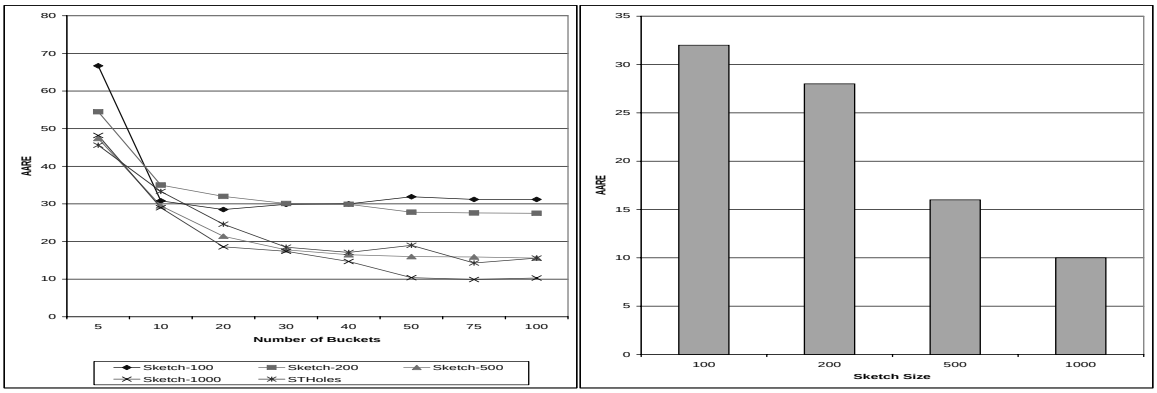
7.3 Evaluating EGREEDY

In this section, we evaluate the performance of EGREEDY using real data sets. Figure 7(a) presents the accuracy of the histograms extracted by EGREEDY as a function of the total number of buckets, for different sizes of the sketch. Figure 7(a) presents also for comparison, the accuracy of the corresponding histograms computed by algorithm STHoles for the same range of buckets.

As is evident in Figure 7 for all sketch sizes, the optimal number of buckets is around 50; increasing the number of buckets beyond this quantity essentially does not reduce the error any further. This can be explained by the fact that beyond certain ranges of bucket numbers, the differences between histograms become undetectable. A similar observation is evident for the STHoles algorithm as well. In particular, the improvement gained by increasing the number of buckets beyond 50 is fairly small and uneven. Algorithm EGREEDY is comparable in accuracy to STHoles for small sketch sizes and capable to outperform STHoles as the sketch size increases, for the same ranges of buckets as is evident in figure 7(a).

Although the optimal number of buckets seems invariant with respect to the sketch size, the resulting *error* of the approximation decreases significantly as the sketch size increases. For a total bucket budget of 50, we depict the error as a function of the sketch size in Figure 7(b). It should be noted that the error is roughly proportional to the square root of the sketch size, which is the dependence predicted by the Johnson-Lindenstrauss lemma (lemma 1). This behavior is very useful, since it allows us to predict the sketch length necessary for achieving certain error.

Figure 8(a) presents the running time of algorithm EGREEDY as a function of the number of buckets for different sketch sizes. For exposition purposes only, we also depict the time to construct the STHoles histogram. The construction time for STHoles is not really comparable with that of EGREEDY since STHoles operates assuming that the entire data is available and issues a large number of queries to "learn" the distribution. Figure 8(b) presents the running time of EGREEDY for two different bucket budgets increasing the to-



(a) Error, increasing number of buckets for various (b) Error versus sketch size for 50 buckets

Figure 7: Error trends for EGREEDY and STHoles for data set *Traffic1*: (a) Error, increasing number of buckets for various sketch sizes (b) Error as a function of sketch size for 50 buckets

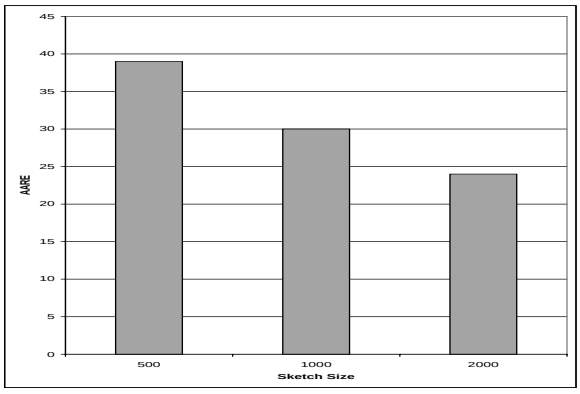


Figure 9: Accuracy of extracted histograms for 100 buckets as the sketch size increases, for data set *Traffic2*

tal sketch size. It is evident that the time EGREEDY requires to extract a good histogram from the sketch is clearly improved compared to that of GREEDY, without great loss in accuracy. This makes algorithm EGREEDY efficiently applicable to problems of larger scale (distribution domain sizes).

For the case of data set *Traffic2*, the overall observations and trends where very similar to that of *Traffic1*; thus, we omit these graphs for brevity. We present however, in Figure 9 the accuracy of the histograms extracted by EGREEDY for 100 buckets as the sketch size increases.

Finally, we visually demonstrate the quality of the histograms algorithm EGREEDY is able to extract from the sketch of a data set. Figure 10(a) presents the distribution of data set *Traffic1* and Figure 10(b) its histogram approximation, using algorithm EGREEDY, with 50 buckets and a sketch size of 1000. The quality of approximation is visually evident; we remark that this histogram is obtained by a single pass over data set *Traffic1* and subsequent extraction from the sketch.

8. CONCLUSIONS

In this paper we have introduced a very efficient method to track the distribution of a multiattribute continuous data stream. We have presented a sketch based approach amenable to incremental updates to maintain a snapshot of the underlying multidimensional distribution. We proposed algorithms with approximate guarantees to extract an optimum multidimensional histogram from the sketch and analytically demonstrated the accuracy and guarantees of our algorithms. These are the first algorithms proposed with these properties.

Since the running time of the optimum histogram extraction algorithm is high, we proposed efficient empirical approaches and we have experimentally demonstrated using real and synthetic data sets that the proposed methods are able to approximate the best histogram solution with high accuracy.

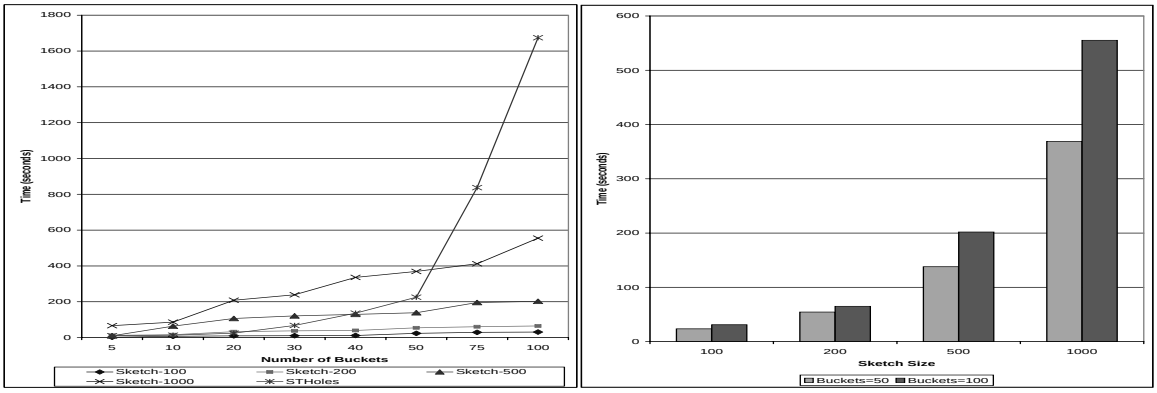
This work raises a variety of interesting questions for further exploration and study. In particular, the sketch-based technique for tracking the distribution of data streams seems quite versatile. Initial examination indicates that many *static* algorithms known in the literature (e.g., the hierarchical partitioning methods of [29]) can be re-implemented to work when only the sketch of the data is available, without any access to the actual data. This raises the possibility of improving the quality of computed histograms even further, by using more elaborate algorithms than the greedy approach used in this paper. We plan to investigate these approaches in our future work in this area.

Acknowledgments. The authors would like to thank Muthu Muthukrishnan, for several important suggestions on the preliminary version of this paper. In particular, we are grateful for pointing to us the references [23, 29] as well as indicating that the algorithms from [29] could be amenable to the sketching approach.

9. REFERENCES

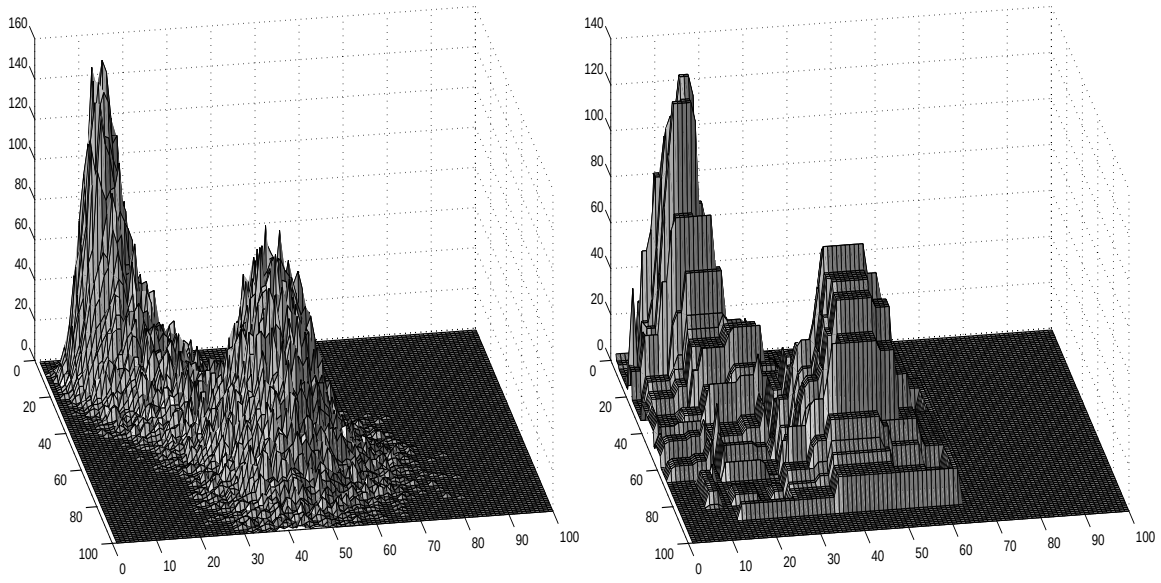
[1] A. Aboulnaga and S. Chaudhuri. Self Tuning Histograms: Building Histograms Without Looking at Data. *Proceedings of ACM SIGMOD*, pages 181–192, June 1999.

[2] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua Approximate Query Answering System. *Proceedings of ACM SIGMOD,*



(a) Extraction time for EGREEDY (b) Histogram extraction time for EGREEDY

Figure 8: Trends in time for EGREEDY and data set *Traffic1*: (a) Extract time for EGREEDY, increasing number of bucket for various sketch sizes (b) Histogram extracting time for EGREEDY as a function of sketch size for 50 and 100 buckets



(a) Distribution of data set *Traffic1* (b) Approximating TRAFFIC1 using EGREEDY

Figure 10: Distribution of *Traffic1* and its approximation with EGREEDY

Philladephia PA, pages 574–578, June 1999.

[3] B. Babcock, M. Datar, and R. Motwani. Sampling From a Moving Window Over Streaming Data. *Proceedings of the Symposium on Discrete Algorithms*, 2002.

[4] S. Babu and J. Widom. Contineous Queries Over Data Streams. *SIGMOD Record*, Sept. 2001.

[5] N. Bruno, L. Gravano, and S. Chaudhuri. STHoles: A Workload Aware Multidimensional Histogram. *Proceedings of ACM SIGMOD*, May 2001.

[6] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. *Proceedings of the Symposium on Discrete Algorithms*, 2002.

[7] P. Gibbons, Y. Mattias, and V. Poosala. Fast Incremental Maintenance of Approximate Histograms. *Proceedings of VLDB, Athens Greece*, pages 466–475, Aug. 1997.

[8] A. Gilbert, S. Guha, P. Indyk, Y. Kotadis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintanance. *Proc. STOC*, 2002.

[9] A. Gilbert, Y. Kotadis, S. Muthukrishnan, and M. Strauss. Quicksand: quick summary and analysis of network data. *DIMACS tech report.*

[10] A. Gilbert, Y. Kotadis, S. Muthukrishnan, and M. Strauss. Surfing Wavelets on Streams: One Pass Summaries for Approximate Aggregate Queries. *Proceedings of VLDB*, pages 79–88, 2001.

[11] J. Gray, A. Bosworth, A. Leyman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross Tab and Sub Total. *Proceedings of ICDE*, pages 152–159, May 1996.

[12] M. Greenwald and S. Khanna. Space-Efficient Online Computation of Quantile Summaries. *Proceedings of ACM SIGMOD, Santa Barbara*, May 2001.

[13] S. Guha and N. Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. *ICDE*, Feb. 2002.

[14] S. Guha, N. Koudas, and K. Shim. Data Streams and Histograms. *Symposium on the Theory of Computing (STOC)*, July 2001.

[15] S. Guha, N. Mishra, R. Motwani, and L. O'callahan. Clustering Data Streams. *Foundations of Computer Science (FOCS)*, Sept. 2000.

[16] D. Gunopulos, G. Kollios, V. Tsotras, and C. Domeniconi. Approximating Multi-Dimensional Aggregate Range Queries Over Real Attributes. *Proceedings of ACM SIGMOD*, June 2000.

[17] P. Haas, J. Naughton, S. Seshadri, and L. Stokes. Sampling Based Estimation Of the Number Of Distinct Values Of An Attribute. *Proceedings of VLDB*, pages 311–322, June 1995.

[18] P. Haas, J. Naughton, S. Seshadri, and A. Swami. Fixed Precision Estimation Of Join Selectivity. *Proceedings of ACM PODS*, pages 190–201, June 1993.

[19] P. Haas and A. Swami. Sequantial Sampling Procedures for Query Size Estimation. *Proceedings of ACM SIGMOD, San Diego, CA*, pages 341–350, June 1992.

[20] P. Indyk. Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation". *Foundations of Computer Science (FOCS)*, Sept. 2000.

[21] Y. Ioannidis and V. Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. *Proceedings of ACM SIGMOD, San Hose, CA*, pages 233–244, June 1995.

[22] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. *Proceedings of VLDB*, pages 275–286, Aug. 1998.

[23] S. Khanna, S. Muthukrishnan, and S. Skiena. Efficient array partitioning. *Proc. ICALP*, 1997.

[24] R. P. Kooi. The Optimization of Queries in Relational Databases. *PhD Thesis, Case Western Reserve University*, Sept. 1980.

[25] J. Lee, D. Kim, and C. Chung. Multidimensional Selectivity Estimation Using Compressed Histogram Inf ormation. *Proceedings of ACM SIGMOD*, pages 205–214, June 1999.

[26] S. Madden and M. Franklin. Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data. *Proceedings of ICDE*, Feb. 2002.

[27] Y. Mattias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. *Proc. of the 1998 ACM SIGMOD Intern. Conf. on Management of Data, Jun e 1998.*

[28] Y. Mattias, J. S. Vitter, and M. Wang. Dynamic Maintenance of Wavelet-Based Histograms. *Proceedings of the International Conference on Very Large Databases, (VLDB), Cairo, Egypt*, pages 101–111, Sept. 2000.

[29] S. Muthukrishnan, V. Poosala, and T. Suel. Partitioning two dimensional arrays: algorithms, complexity and applications. *Proc. Intl Conf. Database Theory*, 1998.

[30] V. Poosala and Y. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. *Proceedings of VLDB, Athens Greece*, pages 486–495, Aug. 1997.

[31] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *Proceedings of ACM SIGMOD, Montreal Canada*, pages 294–305, June 1996.

[32] G. Singh, S. Rajagopalan, and B. Lindsay. Random Sampling Techniques For Space Efficient Computation Of Large Dataset s. *Proceedings of SIGMOD, Philladelphia PA*, pages 251–262, June 1999.

[33] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates on sparse data using wavelets. *Proceedings of SIGMOD*, pages 193–204, June 1999.

[34] J. Vitter, M. Wang, and B. R. Iyer. Data Cube Approximation and Histograms via Wavelets. *Proc.of the 1998 ACM CIKM Intern. Conf. on Information and Knowledge M anagement, November 1998.*

[35] Y. Wu, D. Agrawal, and A. E. Abbadi. Applying the Golden Rule of Sampling for Selectivity Estimation. *Proceedings of ACM SIGMOD*, May 2001.