Space-efficient Algorithms for Data Streams and Histograms

Paul van Tilburg
Department of Mathematics and Computer Science
Eindhoven University of Technology
paul@luon.net

February 27, 2007

Abstract

This survey paper looks into algorithms that deal with massive data sets, streaming data in particular. In addition to reviewing data streaming models and some common problems and applications where space-efficiency is required, the paper explores a technique for constructing approximate representations of the data set by summarising the frequency of the values called histograms. Three types of histograms are discerned and discussed separately: V-optimal, equi-width and end-biased histograms.

1 Introduction

In the field of information processing, database systems in particular [10], more and more algorithms have to deal with massive data sets. This eventually leads to the problem that the data either is not storable anymore or to the situation that it takes to long to retrieve the entire data set or just a parts of it. This means that the data has to be processed as it flows by. *Streaming algorithms* are the algorithms that process the data in this flowing fashion using mostly a single pass, and in a few cases multiple passes.

In [8], a *data stream* is defined as "a sequence of data items that can be read once by algorithm". We use this definition as basis for the term "data stream" throughout the paper. The data streaming models that are often used in the area of streaming algorithms and also in this paper, will be discussed in the next section.

This paper first explores the area of data stream algorithms in general in Section 2 where we will introduce data stream models that are used throughout this paper and we will discuss two common *space-efficient* algorithms.

Another interesting area where space-efficiency has to be applied is considered in this paper. In Section 3 we will focuses on creating approximate space-efficient representations of the data set by constructing *histograms.*

These data set distribution representing histograms are often used in large database systems, where it is useful to monitor the database contents so that query execution and processing can be optimised. Moreover, an often occurring usage of histograms is in the monitoring of Internet traffic [10]. Internet packet headers contain source and destination addresses. Based on a histogram of packet destinations, the router could make real-time routing or load-balancing decisions. To be able to log traffic under given space constraints, routers can summarise the traffic in a space-efficient manner instead of keeping count of all traffic which is unfeasible. The following three types of histograms, each with a different purpose, are discussed separately: V-optimal, equi-width and end-biased histograms.

The last section, Section 4, will contain some concluding remarks and references to related work.

2 Common Algorithms

This section focuses on the part that most algorithms have in common, namely the data stream models. Besides the models two space-efficient algorithms that are commonplace and "famous" are discussed: the approximate L^1-DIFFERENCE algorithm and the F_k-APPROXIMATION frequency moments algorithm.

In [8], M. Henzinger et al. introduce a dichotomy for data stream algorithms that we use to characterise the algorithms presented in this paper:

1. *one-pass* or *multi-pass* algorithm:
   the algorithm processes the items of the data stream in one or multiple iterations respectively.
2. *deterministic* or *randomised* algorithm:
   the algorithm will either always yield the same fixed result or have an random element in it thus yielding different results.
3. *exact* or *approximate* algorithm:
   the algorithm will either yield the optimal solution or an ρ-approximation of the optimal solution.

S. Guha and N. Koudas also mention another important distinction between data stream algorithms in [7] that should be considered:

4. *agglomerative* or *fixed-window* algorithm:
   the algorithm either considers all items from the start of the stream or it only considers the last M elements, mostly due to a memory constraint.

This paper will mostly deal with one-pass algorithms, since most algorithms consider the data in a streaming fashion without any processing afterwards. This is even more so the case for the histogram-constructing algorithms. Moreover, most of the discussed algorithms are either deterministic or randomised, exact and agglomerative.

2.1 Data Stream Models

To be able to talk about data streams, we have to have a model to work with. S. Muthukrishnan presents three common models in [10] that deal with data streams. For the models we have a given *input data* stream of items $a_1, a_2, \ldots$ and an underlying *signal* $\mathbf{A}$, a function $\mathbf{A} : [1 \ldots N] \rightarrow R$ where N is the number of *signal elements* and R some arbitrary range, such as $\mathbb{N}$ or $\mathbb{R}$. Let $\mathbf{A}_i$ denote the state of $\mathbf{A}$ after a_i is received. The three models give an interpretation of how the items a_i relate to the (elements of) signal $\mathbf{A}$.

The Time Series model: Each a_i corresponds to $\mathbf{A}[i]$ in increasing order of i. Here, N is the number of items received until now and R the possible values of the items. An example could be a data stream of stock values or an amount IP traffic from a certain router that are received every 5 minutes.

The Cash Register model: Each a_i is an increment of a value of one of the elements of $\mathbf{A}$. An item can be seen as a tuple (j, Inc_i): the element of $\mathbf{A}$ indexed by j and the increment Inc_i. This would mean for every arriving item: $\mathbf{A}_i[j] = A_{i-1}[j] + Inc_i$. According to [10] this is the most popular data stream model. A possible application of this model is the monitoring of IP traffic from or to $[1 \ldots N]$ destination hosts.

The Turnstile model: This model is known under other names, but we will use this one since the name suits the intuition of the model. The Turnstile model is a general version of the Cash Register model. It not only considers increments to elements of $\mathbf{A}$ but all updates so it can fully suit dynamic situations in which one can have both inserts and deletes. The model is inspired by a subway train station where turnstiles keep track of the number of people in the station by counting the people moving in and out of the station.

Each a_i is an update of a value of one of the elements of $\mathbf{A}$. This means that a_i is some tuple (j, Upd_i) with j some element (index) of $\mathbf{A}$ and the update Upd_i. After every item that has arrived, we have that: $\mathbf{A}_i[j] = A_{i-1}[j] + Upd_i$. For the *strict* version of the Turnstile model we will assume that $A_i[j] \geq 0$ for all i and j.

The Turnstile model will suit the application of histogram construction when we are trying to approximate the number of entries in a certain database while they are inserted and deleted. Note that in the case that we just have to count incoming traffic, the Cash Register model will suffice. However, since this is a more specific form of the Turnstile model, we will assume usage of the strict Turnstile model from now on. For B-histogram construction we can instantiate the model with the number of signal elements $N = B$ and value range $R = \mathbf{N}$.

2.2 Difference Algorithm

The L^1-DIFFERENCE algorithm presented in in [4] solves the following problem: given two data streams a_i and b_i, compute the L^1-difference $D_1 = \sum_i |a_i - b_i|$ between the two functions in a space-efficient way. The presented algorithm can be categorised as an one-pass, randomised, approximate, agglomerative algorithm by our earlier presented dichotomy. A practical application of this algorithm could be to indicate the extent to which amount traffic differ between different routers or between different time intervals.

Algorithm The algorithm shown in algorithm listing 2.1 assumes the input stream to be an arbitrary interleaving of the two data streams and assumes that the maximal value M and number of items N are known. The data stream items c_i (either some a_i or b_i) that arrive can be considered as a tuple $(i, c_i, +1)$ or $(i, c_i, -1)$.

For each of the sample spaces, indexed by k, l (of which the ranges are specified later), the algorithm defines a family of $M \cdot N$ random variables $V_{i,j}$ for $0 \leq i < N$ and $0 \leq j < M$. Each variable $V_{i,j}$ has the value ± 1 and is constructed from a seeds s_i, which in turn is derived from some master seed S. The random variables are constructed in such a way that they are *n^2-bad 4-wise independent* [4, Section 3.2]. Per sample space, the algorithm also keeps a running sum Z, that is initialised with 0.

Algorithm 2.1 The L^1-DIFFERENCE algorithm

```
L^1-DIFFERENCE($\langle (i, c_i, \pm 1) \rangle$)
 1  for $k \leftarrow 1$ to $\log(\frac{1}{\delta})$ do
 2    for $l \leftarrow 1$ to $(8 \cdot C)/\epsilon^2$ do
 3      $Z_{k,l} = 0$
 4      pick a master seed $S_{k,l}$
 5      this implicitly defines $V_{i,j,k,l}$
 6          for $0 \leq i < N$ and $0 \leq j < M$
 7  for each tuple $(i, c_i, \theta_i)$ do
 8    for $k \leftarrow 1$ to $4\log(\frac{1}{\delta})$ do
 9      for $l \leftarrow 1$ to $(8 \cdot C)/\epsilon^2$ do
10        $Z_{k,l} \leftarrow Z_{k,l} + \theta_i \sum_{j=0}^{c_i - 1} V_{i,j,k,l}$
11  return median$_k$ avg$_l$ $Z_{k,l}^2$
```

When the algorithm encounters a tuple $(i, a_i, +1)$, it will add $\sum_{j=0}^{a_i - 1} V_{i,j}$ to the running sum Z for all sample spaces. Conversely, when the tuple $(i, b_i, -1)$ is encountered, the algorithm subtracts $\sum_{j=0}^{b_i - 1} V_{i,j}$ from Z. The overall effect is that Z contains the sum of the $|a_i - b_i|$ random variables by cancelling out the first $\min(a_i, b_i)$ variables. For the end result, when $i = N$, we know that for Z^2 there are $D_1 = \sum_{i=1}^{N} |a_i - b_i|$ terms which are squares of the random variables that contribute to square of the running sum. If the cross terms $V_{i,j} V_{k,l}$ do not contribute much to Z^2, then Z^2 is a good approximation of D_1.

The steps of algorithm described above are simultaneously performed for sample spaces indexed by k, l for $1 \leq k < \log(\frac{1}{d})$ and $1 \leq l < (8 \cdot C)/\epsilon^2$ with some C between 7.5 and 9 to ensure that the performance requirements are met. Hence, the output is the median of the average of the square of the running sums $Z_{k,l}$ of all sample spaces.

Performance J. Feigenbaum et al. presented the above algorithm (given in Figure 2.1) with the following performance: given the data stream items a_i and b_i with a maximal value of M, then for every $\epsilon > 0$ and $\delta > 0$ the algorithm computes with a probability of at least $1-\delta$ an approximation W to the L^1-difference D_1 of the two functions with $|W - D_1| \leq \epsilon D_1$, using space $O(\log(M)\log(N)\log(\frac{1}{\delta})/\epsilon^2)$ and time $O(\log(N)\log\log(N) + \log(M)\log(\frac{1}{\delta})/\epsilon^2)$ to process each item. A proof that the algorithm meets this performance is given in Section 3.5 of [4].

Correctness We will discuss a sketch of the correctness proof here to show that the algorithm outputs a random variable $W = \text{median}_k \ \text{avg}_l \ Z_{k,l}^2$ such that $|W - D_1| < \epsilon D_1$ with a probability of at least $1 - \delta$. For a full proof see Section 3.4 of [4]. We know that

$$Z_{k,l} = \sum_{i=0}^{N} \sum_{j=\min(a_i,b_i)}^{\max(a_i,b_i)} \pm V_{i,j,k,l}$$

since for each $j \leq \min(a_i, b_i)$ both $+V_{i,j,k,l}$ and $-V_{i,j,k,l}$ are added and for each $j > \max(a_i, b_i)$ nothing is added. Now, we can compute $E[Z_{k,l}^2]$ and $E[Z_{k,l}^4]$. Using a relabelling of the indices i, j to m and the pairwise independence of V_m and V'_m, we can compute that:

$$\begin{aligned} E[Z_{k,l}^2] &= E\left[\left(\sum_{m=1}^{D_1} \pm V_m\right)^2\right] \\ &= \sum_{m=1}^{D_1} E[(\pm V_m)^2] + \\ &\quad 2 \sum_{1 \leq m < m' < D_1} E[(\pm V_m)(\pm V'_m)] \\ &= \sum_{m=1}^{D_1} 1 = D_1. \end{aligned}$$

Using the n^2-bad 4-wise independence properties of the random variables $V_{i,j}$, we can compute that:

$$\begin{aligned} E[Z_{k,l}^4] &\leq 6D_1^2 + \sum_{i=1}^{N} 4(b_i - a_i)^2 \\ &\leq 10D_1^2. \end{aligned}$$

Hence, $\text{Var}(Z_{k,l}^2) = E[Z_{k,l}^4] - E^2[Z_{k,l}^2] \leq C \cdot D_1^2$ for $C = 9$. If $Y_k = \text{avg}_l \ Z_{k,l}$ is an average for all l, then $\text{Var}(Y_k) \leq \frac{\epsilon^2}{8} D_1^2$. We then know that $\Pr\{|Y_k - D_1| > \epsilon D_1\} \leq \frac{\text{Var}(Y_k)}{\epsilon^2 D_1^2} \leq \frac{1}{8}$ by Chebyshev's inequality. If $W = \text{median}_k \ Y_k$ is the median for all k, then $|W - D_1| > \epsilon D_1$ iff $|Y_k - D_1| > \epsilon D_1$ for half of the k's. The probability that this happens is at most δ by Chernoff's inequality.

2.3 Frequency Moments

Frequency moments provide a generic way of looking at several properties of a stream of data items. Given the Cash Register model, the frequency moments of a data stream are the numbers $F_k = \sum_{i=1}^{N} \mathbf{A}[i]$ where $\mathbf{A}[i]$ is the kth power of the amount of the occurrences (frequency) of the value i: $|\{j \in \{1, \ldots, N\} \mid a_j = i\}|^k$. Hence, for $k = 0$ we have that F_0 is the number of distinct values in the data stream. For $k = 1$, F_1 is the length of the stream. Also note that $\mathbf{A}$ contains an approximated histogram of the data stream. And finally, for $k \geq 2$, F_k captures the skew of the data.

Algorithm In [1], N. Alon et al. present a basic algorithm for determining an approximation of F_k for a given data stream. They also provide an optimised version for F_2 and optimisation suggestions for F_0 that are not discussed in this paper.

The basic approach is similar to the L_1-DIFFERENCE algorithm: attempt a space-efficient computation of a random variable of

Algorithm 2.2 The F_k-APPROXIMATION algorithm

```
$F_k$-APPROXIMATION($< c_i >$)
1  for $k \leftarrow 1$ to $2\log(\frac{1}{\delta})$ do
2    for $l \leftarrow 1$ to $\frac{8kn^{1-(1/k)}}{\epsilon^2}$ do
3      choose a random $p$ from $\{1, \ldots, N\}$
4      $r \leftarrow |\{q \in \{p, \ldots, N\} \mid a_q = a_p\}|$
5      $X_{k,l} \leftarrow N(r^k - (r-1)^k)$
6  return median$_k$ avg$_l$ $X_{k,l}$
```

which the expected value equals F_k. Therefore, the algorithm shown in algorithm listing 2.2 is also an one-pass, randomised, approximate, agglomerative algorithm.

Performance The correctness proof sketch that follows below will show that the algorithm given in Figure 2.2 has the following performance: given a data stream of N items that have a maximal value of M, then for every $k \geq 1$, $\epsilon > 0$ and $\delta > 0$ the algorithm computes an approximation W of the frequency moment F_k in such a way that $|W - F_k| < \epsilon F_k$ with a probability of at least $1-\delta$ using space $O\left(\frac{k\log(\frac{1}{\delta})}{\epsilon^2} n^{1-(1/k)}(\log(M) + \log(N))\right)$.

Correctness The correctness proof of the F_k-APPROXIMATION algorithm [1, p. 22] follows exactly the same structure as the proof of the L_1-DIFFERENCE algorithm. First it shows for the random variables $X_{k,l}$ of all sample spaces that the expected value $E[X_{k,l}] = F_k$ and then it computes $E[X_{k,l}^2] \leq kn^{1-(1/k)}F_k^2$. Likewise it defines Y_k to be the average avg$_l$ $X_{k,l}$ for all $0 \leq l < \frac{8kn^{1-(1/k)}}{\epsilon^2}$, then we know that the expected value

$$E[Y_k] = E[X_{k,l}] = F_k$$

and the variance

$$\mathrm{Var}(Y_k) \leq \frac{E[X_{k,l}^2]\epsilon^2}{8kn^{1-(1/k)}} = \frac{F_k^2\epsilon^2}{8}.$$

Hence, by Chebyshev's inequality the probability that the approximation is off by more than a factor ϵ is

$$\Pr\{|Y_k - F_k| > \epsilon F_k\} \leq \frac{\mathrm{Var}(Y_k)}{\epsilon^2 F_k^2} \leq \frac{1}{8}.$$

Finally, it assumes that W is the median median$_k$ Y_k for all $1 \leq k < 2\log(\frac{1}{\delta})$. Then, by Chernoff's inequality the probability that half of the Y_k variables deviate more than ϵF_k from F_k is at most δ.

Considering the space-wise performance, each variable costs $O(\log(M) + \log(N))$ space and there are $2\log(\frac{1}{\delta}) \cdot \frac{8kn^{1-(1/k)}}{\epsilon^2}$ variables. Hence, the space usage of the algorithm is in $O\left(\frac{k\log(\frac{1}{\delta})}{\epsilon^2} n^{1-(1/k)}(\log(M) + \log(N))\right)$.

3 Histogram algorithms

Histograms are used to approximate a representation of a data set by summarising (or counting) the number of occurrence of each value in the set called the *value frequency.* Therefore, the histogram captures the distribution of the data stream item values.

Useful applications of histograms (of data streams) are visualisations of data distributions, often used in statistical analysis, or database engines that use a histogram to optimise query executions or approximately processing of queries [5]. As mentioned earlier in the introduction, histograms are also useful as traffic logs that summarise IP traffic.

In [2], B. Babcock et al. discern three types of histograms that we will discuss separately in the remaining part of this section:

- *V-optimal* histograms:
  approximations of the distribution of the values minimising an error function that pairwise considers the estimated number occurrences of the values and the real number of occurrences.
- *Equi-width* histograms:
  histograms where the domain is split up into buckets such that the number of occurrences in each bucket is uniform across all buckets.

- *End-biased* histograms:
  histograms where the frequencies of the values above some threshold are kept in an exact manner, while the frequencies below the threshold are approximated. These histograms are used for what is called *iceberg querying*.

Out of the three types, the V-optimal histogram is the most common and will be discussed first.

3.1 V-optimal Histograms

Numerous space-efficient algorithms have been devised to deal with approximating the data distribution. Though, most of them use heuristics that have no provable quality guarantees. H.V. Jagadish et al. give several algorithms in [9] that do have these guarantees.

We define a data stream as it is defined in either of our earlier presented models. Then V is the set of all values that a_i has for all $i \in \{1, \ldots, N\}$. For each $v \in V$ we denote the frequency that v occurs in the data stream as f_v. We define F as the ordered (by value) frequency vector $F = (f_1\ f_2\ \ldots\ f_M)$ for M distinct values.

A histogram of the data distribution $\{1, \ldots, M\}$ can be found by splitting the frequency vector F up into B intervals, called buckets. Given the fact that we are discussing V-optimal histograms, we must make sure that the values and frequencies of these values in each bucket approximate the real data distribution. H.V. Jagadish et al. propose two kinds of histograms: the space-bounded histogram where there is a limit B on the length of the histogram H and the algorithm has to find such a H that minimises some error function, and an error-bounded histogram where there is a limit on the error ϵ and the algorithm has to find the smallest histogram H such that the error is at most ϵ. We will discuss the algorithms for the first kind, for the second kind please refer to Section 3 of [9].

Since almost all histograms algorithms require an error function, we need a common way to define the error between the real value frequency and an approximated value frequency. A common way is the *squared error*: $|\hat{f}_k - f_k|^2$. In [9], a sum squared error is defined for an interval $[a, b]$:

$$\mathbf{Sse}([a,b]) = \sum_{k=a}^{b} (F[k] - \mathbf{Avg}([a,b]))^2$$

where

$$\mathbf{Avg}([a,b]) = \frac{\sum_{k=a}^{b} F[k]}{b-a+1}.$$

For the following algorithm and optimisations we will use **Sse** as our error metric. Note however, that another error metric could be used just as easily.

Algorithm The algorithm given in algorithm listing 3.1 is the basic optimal algorithm discussed in [9]. It is an one-pass, deterministic, exact, agglomerative algorithm.

For the algorithm we define $\mathbf{Sse}^*(i,k)$ to represent the smallest **Sse** for the prefix vector $F[1,i]$ using at most k buckets. The algorithm uses the dynamic programming technique to determine the smallest sum square error $\mathbf{Sse}^* = \mathbf{Sse}^*(N,B)$ which is the optimal **Sse** for the histogram. During the computation, the set of bucket boundaries S_b is be updated to reflect the situation with the smallest **Sse**.

Algorithm 3.1 The Space-Bound-Histogram algorithm

```
Space-Bound-Histogram($F, B$)
1  for $i \leftarrow 1$ to $N$ do
2    for $k \leftarrow 1$ to $B$ do
3      $\mathbf{Sse}^*(i,k) \leftarrow$
4        $\min_{1 \le j < i} (\mathbf{Sse}^*(j, k-1) + \mathbf{Sse}([j+1, i]))$
5      update $S_b$ accordingly
6  return $S_b$
```

Note that for each computation of $\mathbf{Sse}^*(i,k)$ table lookups are used of values that are already known. The algorithm given above does not explicitly mention how the boundary cases are calculated because they are trivial to compute.

Performance Since the algorithm calculates the value of $\mathbf{Sse}^*$ for each of the N frequencies and each of the B buckets, it performs $O(NB)$ calculations of $\mathbf{Sse}^*$. Each computation of Sss^* considers $O(N)$ possible values to ultimately select the minimum, where each value takes a constant time to be computed: a lookup of $\mathbf{Sse}^*(j, k-1)$ and a call to $\mathbf{Sse}([j+1, i])$. Hence, the Space-Bound-Histogram algorithm uses $O(N^2B)$ time. Evidently, it uses $O(N\log(N)B)$ space since it tabulates sum square errors for each frequency value and each bucket. A value frequency can be at most N since there are only N values, and consequently the sum square error can be at most N^2 which is storable in $O(\log(N))$ bits.

Optimisations An optimisation of the above algorithm that is proposed in Section 4.2 of [9] is to compute $\mathbf{Sse}^*$ by iterating from $i-1$ down to 1. The exact details will be omitted in this paper, however the end result is that the inner loop being terminated much before **Sse** has to be calculated for all possible values of j. Hence, it is unlikely that the algorithm takes $O(N^2B)$ time.

A second optimisation proposed in Section 4.3 of [9] is to partition the frequency vector F in l disjoint chunks (for some arbitrary l) and call the Space-Bound-Histogram algorithm for each of the chunks. Dynamic programming is used to determine how the B buckets are divided over the l chunks. H.V. Jagadish et al. ascertain that the running time of this approximation approach uses $O(\frac{N^2B}{l})$ time.

More algorithms As noted in the introduction of this subsection, there are numerous algorithms available for this problem. Another "famous" algorithm is the fast, small-space algorithm presented by A.C. Gilbert et al. This algorithm, actually a set of related algorithms, is much more contrived than the basic algorithm presented above and would require too much space to explain. In essence it first uses *array sketches* to approximate the result of a data stream of updates (see the Turnstile model), then it constructs a histogram with the desired accuracy ϵ and number of buckets B. The time to process the updates and maintain the sketch, the time to construct the B-histogram and the space used by the sketch are all bounded by $poly(B, \log(N), log(||A||), \frac{1}{\epsilon})$.

3.2 Equi-width Histograms

Equi-width histograms have quite a different kind of algorithms that deal with the problem. An often used paradigm to solve the problem is the *quantile* approach. A q-quantile divides a data set in q equal-sized partitions which is clearly analogous to an equi-width B-histogram with a data set represented by value frequencies.

In [6], M. Greenwald and S. Khanna present an one-pass, approximate, agglomerative algorithm that computes an ϵ-approximate quantile summary of a sequence of N elements with a precision of ϵN which has a worst-case space requirement of $O(\frac{1}{\epsilon}\log(\epsilon N))$.

The algorithm keeps a summery data structure S, which is a tree of tuples. When a data item a_i arrives, the algorithm will create a tuple t_i of a_i with supporting data such as its rank, etc. and insert in the tree accordingly. On every $\frac{1}{2\epsilon}$-th arrival of a data item, it is not inserted, but the tree S is compressed by deleting and merging tuples. At all times, the algorithm can answer a q-quantile query with an approximation of ϵn, where n is the number of the last element that arrived until the moment of the query. For more specifics of the algorithm, please refer to [6].

3.3 End-biased Histograms

End-biased histograms are mostly used for *iceberg queries* in database systems. Iceberg queries range over a huge amount of data items while the answer, i.e. the few frequently occurring data items that are above a certain threshold, is relatively small. The main problem with these queries is that the target data items over which the queries ranges is most of the time too big to fit in memory.

M. Fang et al. present several strategies and algorithms in [3] to deal efficiently with these iceberg queries. The presented algorithms are space-efficient algorithms that are sometimes multi-pass, and mostly deterministic and approximate. Note that in the case of iceberg queries there is a fixed-window notion since there is a bound on the memory use. The data items that do not meet the threshold are virtually omitted and approximated.

4 Conclusion

In this paper, we have discusses various space-efficient algorithms. First the L^1-DIFFERENCE and F_k-APPROXIMATION algorithms that can abstract over a vast number of data items and compute a property of the data stream in a space-efficient way. Then we looked at histograms that represent the data distribution in a space-efficient way. We've discussed briefly a few of the numerous algorithms available for constructing histograms. Following [2] we have distinguished three kinds of histograms: V-optimal, equi-width and end-based histograms, of which the V-optimal histograms are the most common.

A subtopic that has been surpassed in this paper is multidimensional histograms. A thorough formal study of this kind of histograms is presented in [11] by N. Thaper et al. For other algorithms and applications related to data streams that have not been discussed here, please refer to [10].

References

[1] N. Alon, Y. Matias, and M. Szegedy, *The Space Complexity of Approximating the Frequency Moments*, STOC '96: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing (1996), 20–29.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, *Models and Issues in Data Stream Systems*, Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (2002), 1–16.

[3] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J.D. Ullman, *Computing Iceberg Queries Efficiently*, Proceedings of the 24rd International Conference on Very Large Data Bases (1998), 299–310.

[4] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, *An approximate L^1-difference algorithm for massive data streams*, IEEE Symposium on Foundations of Computer Science, 1999, pp. 501–511.

[5] A.C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss, *Fast, Small-space Algorithms for Approximate Histogram Maintenance*, Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing (2002), 389–398.

[6] M. Greenwald and S. Khanna, *Space-efficient Online Computation of Quantile Summaries*, Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data (2001), 58–66.

[7] S. Guha and N. Koudas, *Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation*, Proc. of the 2002 Intl. Conf. on Data Engineering **576** (2002).

[8] M.R. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on data streams*, (1999), 107–118.

[9] HV Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, and T. Suel, *Optimal Histograms with Quality Guarantees*, Proceedings of the 24rd International Conference on Very Large Data Bases (1998), 275–286.

[10] S. Muthukrishnan, *Data Streams: Algorithms and Applications*, Now Publishers, 2006.

[11] N. Thaper, S. Guha, P. Indyk, and N. Koudas, *Dynamic Multidimensional Histograms*, Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (2002), 428–439.