Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Algorithms 55 (2005) 58–75
Journal of Algorithms
www.elsevier.com/locate/jalgor

# An improved data stream summary: the count-min sketch and its applications

Graham Cormode [a,*,1], S. Muthukrishnan [b,2]

[a] *Center for Discrete Mathematics and Computer Science (DIMACS), Rutgers University, Piscataway, NJ, USA*
[b] *Division of Computer and Information Systems, Rutgers University and AT&T Research, USA*

Received 6 June 2003
Available online 4 February 2004

**Abstract**

We introduce a new *sublinear space* data structure—the *count-min sketch*—for summarizing data streams. Our sketch allows fundamental queries in data stream summarization such as point, range, and inner product queries to be approximately answered very quickly; in addition, it can be applied to solve several important problems in data streams such as finding quantiles, frequent items, etc. The time and space bounds we show for using the CM sketch to solve these problems significantly improve those previously known—typically from $1/\varepsilon^2$ to $1/\varepsilon$ in factor.

## 1. Introduction

We consider a vector $\boldsymbol{a}$, which is presented in an implicit, incremental fashion. This vector has dimension $n$, and its current state at time $t$ is $\boldsymbol{a}(t) = [a_1(t), \ldots, a_i(t), \ldots, a_n(t)]$. Initially, $\boldsymbol{a}$ is the zero vector, $a_i(0) = 0$ for all $i$. Updates to individual entries of the vector are presented as a stream of pairs. The $t$th update is $(i_t, c_t)$, meaning that $a_{i_t}(t) = a_{i_t}(t-1) + c_t$, and $a_{i'}(t) = a_{i'}(t-1)$ for all $i' \neq i_t$. At any time $t$, a *query* calls for computing certain functions of interest on $\boldsymbol{a}(t)$.

This setup is the *data stream* scenario that has emerged recently. Algorithms for computing functions within the data stream context need to satisfy the following desiderata.

[*] Corresponding author.
*E-mail addresses:* graham@dimacs.rutgers.edu (G. Cormode), muthu@cs.rutgers.edu (S. Muthukrishnan).
[1] Supported by NSF ITR 0220280 and NSF EIA 02-05116.
[2] Supported by NSF CCR 0087022, NSF ITR 0220280 and NSF EIA 02-05116.

0196-6774/$ – see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jalgor.2003.12.001

First, the space used by the algorithm should be small, at most polylogarithmic in $n$, the space required to represent $\boldsymbol{a}$ explicitly. Since the space is sublinear in data and input size, the data structure used by the algorithms to represent the input data stream is merely a summary—aka a *sketch* or synopsis [17]—of it; because of this compression, almost no function that one needs to compute on $\boldsymbol{a}$ can be done precisely, so some approximation is provably needed. Second, processing an update should be fast and simple; likewise, answering queries of a given type should be fast and have usable accuracy guarantees. Typically, accuracy guarantees will be made in terms of a pair of user specified parameters, $\varepsilon$ and $\delta$, meaning that the error in answering a query is within a factor of $\varepsilon$ with probability $1 - \delta$. The space and update time will consequently depend on $\varepsilon$ and $\delta$; our goal will be limit this dependence as much as is possible.

Many applications that deal with massive data, such as Internet traffic analysis and monitoring contents of massive databases, motivate this one-pass data stream setup. There has been a frenzy of activity recently in the Algorithm, Database and Networking communities on such data stream problems, with multiple surveys, tutorials, workshops and research papers. See [3,12,28] for detailed description of the motivations driving this area.

In recent years, several different sketches have been proposed in the data stream context that allow a number of simple aggregation functions to be approximated. Quantities for which efficient sketches have been designed include the $L_1$ and $L_2$ norms of vectors [2,14,23], the number of distinct items in a sequence (i.e., number of non-zero entries in $\boldsymbol{a}(t)$) [6,15,18], join and self-join sizes of relations (representable as inner-products of vectors $\boldsymbol{a}(t), \boldsymbol{b}(t)$) [1,2], item and range sum queries [5,20]. These sketches are of interest not simply because they can be used to directly approximate quantities of interest, but also because they have been used considerably as "black box" devices in order to compute more sophisticated aggregates and complex quantities: quantiles [21], wavelets [20], histograms [19,29], database aggregates and multi-way join sizes [10], etc. Sketches thus far designed are typically linear functions of their input, and can be represented as projections of an underlying vector representing the data with certain randomly chosen projection matrices. This means that it is easy to compute certain functions on data that is distributed over sites, by casting them as computations on their sketches. So, they are suited for distributed applications too.

While sketches have proved powerful, they have the following drawbacks.

- Although sketches use small space, the space used typically has a $\Omega(1/\varepsilon^2)$ multiplicative factor. This is discouraging because $\varepsilon = 0.1$ or $0.01$ is quite reasonable and already, this factor proves expensive in space, and consequently, often, in per-update processing and function computation times as well.
- Many sketch constructions require time linear in the size of the sketch to process each update to the underlying data [2,21]. Sketches are typically a few kilobytes up to a megabyte or so, and processing this much data for every update severely limits the update speed.
- Sketches are typically constructed using hash functions with strong independence guarantees, such as $p$-wise independence [2], which can be complicated to evaluate, particularly for a hardware implementation. One of the fundamental questions is to what extent such sophisticated independence properties are needed.

- Many sketches described in the literature are good for one single, pre-specified aggregate computation. Given that in data stream applications one typically monitors multiple aggregates on the same stream, this calls for using many different types of sketches, which is a prohibitive overhead.
- Known analyses of sketches hide large multiplicative constants inside big-Oh notation.

Given that the area of data streams is being motivated by extremely high performance monitoring applications—e.g., see [12] for response time requirements for data stream algorithms that monitor IP packet streams—these drawbacks ultimately limit the use of many known data stream algorithms within suitable applications.

We will address all these issues by proposing a new sketch construction, which we call the *count-min*, or CM, sketch. This sketch has the advantages that:

(1) space used is proportional to $1/\varepsilon$;
(2) the update time is significantly sublinear in the size of the sketch;
(3) it requires only pairwise independent hash functions that are simple to construct;
(4) this sketch can be used for several different queries and multiple applications; and
(5) all the constants are made explicit and are small.

Thus, for the applications we discuss, our constructions strictly improve the space bounds of previous results from $1/\varepsilon^2$ to $1/\varepsilon$ and the time bounds from $1/\varepsilon^2$ to 1, which is significant.

Recently, a $\Omega(1/\varepsilon^2)$ space lower bound was shown for a number of data stream problems: approximating frequency moments $F_k(t) = \sum_k (a_i(t))^k$, estimating the number of distinct items, and computing the Hamming distance between two strings [30].[3] It is an interesting contrast that for a number of similar seeming problems (finding heavy hitters and quantiles in the most general data stream model) we are able to give an $O(1/\varepsilon)$ upper bound. Conceptually, CM sketch also represents progress since it shows that pairwise independent hash functions suffice for many of the fundamental data stream applications. From a technical point of view, CM sketch and its analyses are quite simple. We believe that this approach moves some of the fundamental data stream algorithms from the theoretical realm to the practical.

Our results have some technical nuances:

- The accuracy estimates for individual queries depend on the $L_1$ norm of $\boldsymbol{a}(t)$ in contrast to the previous works that depend on the $L_2$ norm. This is a consequence of working with simple counts. The resulting estimates are often not as tight on individual queries since $L_2$ norm is never greater than the $L_1$ norm. But nevertheless, our estimates for individual queries suffice to give improved bounds for the applications here where it is desired to state results in terms of $L_1$.

[3] This bound has virtually been met for distinct items by results in [4], where clever use of hashing improves previous bounds of $O((\log n/\varepsilon^2)\log(1/\delta))$ to $\widetilde{O}((1/\varepsilon^2 + \log n)\log(1/\delta))$.

- Most prior sketch constructions relied on embedding into small dimensions to estimate norms. For example, [2] relies on embedding inspired by the Johnson–Lindenstrauss lemma [24] for estimating $L_2$ norms. But accurate estimation of the $L_2$ norm of a stream requires $\Omega(1/\varepsilon^2)$ space [30]. Currently, all data stream algorithms that rely on such methods that estimate $L_p$ norms use $\Omega(1/\varepsilon^2)$ space. One of the observations that underlie our work is while embedding into small space is needed for small space algorithms, it is not necessary that the methods accurately estimate $L_2$ or in fact any $L_p$ norm, for most queries and applications in data streams. Our CM sketch does not help estimate $L_2$ norm of the input, however, it accurately estimates the queries that are needed, which suffices for our data stream applications.
- Most data stream algorithm analyses thus far have followed the outline from [2] where one uses Chebyshev and Chernoff bounds in succession to boost probability of success as well as the accuracy. This process contributes to the complexity bounds. Our analysis is simpler, relying only on the Markov inequality. Perhaps surprisingly, in this way we get tighter, cleaner bounds.

The remainder of this paper is as follows: in Section 2 we discuss the queries of our interest. We describe our count-min sketch construction and how it answers queries of interest in Sections 3 and 4 respectively, and apply it to a number of problems to improve the best known complexity in Section 5. In each case, we state our bounds and directly compare it with the best known previous results.

All previously known sketches have many similarities. Our CM sketch lies in the same framework, and finds inspiration from these previous sketches. Section 6 compares our results to past work, and shows how all relevant sketches can be compared in terms of a small number of parameters. This should prove useful to readers in contrasting the vast number of results that have emerged recently in this area. Conclusions are in Section 7.

## 2. Preliminaries

We consider a vector $\boldsymbol{a}$, which is presented in an implicit, incremental fashion. This vector has dimension $n$, and its current state at time $t$ is $\boldsymbol{a}(t) = [a_1(t), \ldots, a_i(t), \ldots, a_n(t)]$. For convenience, we shall usually drop $t$ and refer only to the current state of the vector. Initially, $\boldsymbol{a}$ is the zero vector, $\mathbf{0}$, so $a_i(0)$ is 0 for all $i$. Updates to individual entries of the vector are presented as a stream of pairs. The $t$th update is $(i_t, c_t)$, meaning that

$$a_{i_t}(t) = a_{i_t}(t-1) + c_t,$$
$$a_{i'}(t) = a_{i'}(t-1), \quad i' \neq i_t.$$

In some cases, $c_t$s will be strictly positive, meaning that entries only increase; in other cases, $c_t$s are allowed to be negative also. The former is known as the *cash register* case and the latter the *turnstile* case [28]. There are two important variations of the turnstile case to consider: whether $a_i$s may become negative, or whether the application generating the updates guarantees that this will never be the case. We refer to the first of these as the *general* case, and the second as the *non-negative* case. Many applications that use sketches to compute queries of interest—such as monitoring database contents, analyzing IP traffic seen in a network link—guarantee that counts will never be negative. However, the general case occurs in important scenarios too, for example in distributed settings where one considers the subtraction of one vector from another, say.

At any time $t$, a *query* calls for computing certain functions of interest on $\boldsymbol{a}(t)$. We focus on approximating answers to three types of query based on vectors $\boldsymbol{a}$ and $\boldsymbol{b}$:

- A *point query*, denoted $\mathcal{Q}(i)$, is to return an approximation of $a_i$.
- A *range query* $\mathcal{Q}(l,r)$ is to return an approximation of $\sum_{i=l}^{r} a_i$.
- An *inner product query*, denoted $\mathcal{Q}(\boldsymbol{a},\boldsymbol{b})$ is to approximate $\boldsymbol{a} \odot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i$.

These queries are related: a range query is a sum of point queries; both point and range queries are specific inner product queries. However, in terms of approximations to these queries, results will vary. These are the queries that are fundamental to many applications in data stream algorithms, and have been extensively studied. In addition, they are of interest in a non-data stream context. For example, in databases, the point and range queries are of interest in summarizing the data distribution approximately; and inner-product queries allow approximation of join size of relations. Fuller discussion of these aspects can be found in [16,28].

We will also study use of these queries to compute more complex functions on data streams. As examples, we will focus on the two following problems. Recall that $\|\boldsymbol{a}\|_1 = \sum_{i=1}^{n} |a_i(t)|$; more generally, $||\boldsymbol{a}||_p = (\sum_{i=1}^{n} |a_i(t)|^p)^{1/p}$.

- ($\phi$*-quantiles*) The $\phi$-quantiles of the cardinality $\|\boldsymbol{a}\|_1$ multiset of (integer) values each in the range $1 \ldots n$ consist of those items with rank $k\phi\|\boldsymbol{a}\|_1$ for $k = 0 \ldots 1/\phi$ after sorting the values. Approximation comes by accepting any integer that is between the item with rank $(k\phi - \varepsilon)\|\boldsymbol{a}\|_1$ and the one with rank $(k\phi + \varepsilon)\|\boldsymbol{a}\|_1$ for some specified $\varepsilon < \phi$.
- (*heavy hitters*) The $\phi$-heavy hitters of a multiset of $\|\boldsymbol{a}\|_1$ (integer) values each in the range $1 \ldots n$, consist of those items whose multiplicity exceeds the fraction $\phi$ of the total cardinality, i.e., $a_i \geqslant \phi\|\boldsymbol{a}\|_1$. There can be between 0 and $1/\phi$ heavy hitters in any given sequence of items. Approximation comes by accepting any $i$ such that $a_i \geqslant (\phi - \varepsilon)\|\boldsymbol{a}\|_1$ for some specified $\varepsilon < \phi$.

We will assume the RAM model, where each machine word can store integers up to $\max\{\|\boldsymbol{a}\|_1, n\}$. Standard word operations take constant time and so we count space in terms of number of words and we count time in terms of the number of word operations. So, if one estimates our space and time bounds in terms of number of *bits* instead, a multiplicative factor $\log\max\{\|\boldsymbol{a}\|_1, n\}$ is needed. Our goal is to solve the queries and the problems above using a sketch data structure, that is using space and time significantly sublinear—polylogarithmic—in input size $n$ and $\|\boldsymbol{a}\|_1$. All our algorithms will be approximate and probabilistic; they need two parameters, $\varepsilon$ and $\delta$, meaning that the error in answering a query is within a factor of $\varepsilon$ with probability $\delta$. Both these parameters will affect the space and time needed by our solutions. Each of these queries and problems has a rich history of work in the data stream area. We refer the readers to surveys [3,28], tutorials [16], as well as the general literature.

## 3. Count-min sketches

We now introduce our data structure, the count-min, or CM, sketch. It is named after the two basic operations used to answer point queries, counting first and computing the minimum next. We use $e$ to denote the base of the natural logarithm function, ln.

### 3.1. Data structure

A *count-min* (*CM*) *sketch* with parameters $(\varepsilon, \delta)$ is represented by a two-dimensional array counts with width $w$ and depth $d$: $\mathit{count}[1,1] \ldots \mathit{count}[d,w]$. Given parameters $(\varepsilon, \delta)$, set $w = \lceil e/\varepsilon \rceil$ and $d = \lceil \ln(1/\delta) \rceil$. Each entry of the array is initially zero. Additionally, $d$ hash functions

$$h_1 \ldots h_d : \{1 \ldots n\} \to \{1 \ldots w\}$$

are chosen uniformly at random from a pairwise-independent family.

### 3.2. Update procedure

When an update $(i_t, c_t)$ arrives, meaning that item $a_{i_t}$ is updated by a quantity of $c_t$, then $c_t$ is added to one count in each row; the counter is determined by $h_j$. Formally, set $\forall 1 \leqslant j \leqslant d$,

$$\mathit{count}\big[j, h_j(i_t)\big] \leftarrow \mathit{count}\big[j, h_j(i_t)\big] + c_t.$$

This procedure is illustrated in Fig. 1.

The space used by count-min sketches is the array of $wd$ counts, which takes $wd$ words, and $d$ hash functions, each of which can be stored using 2 words when using the pairwise functions described in [27].

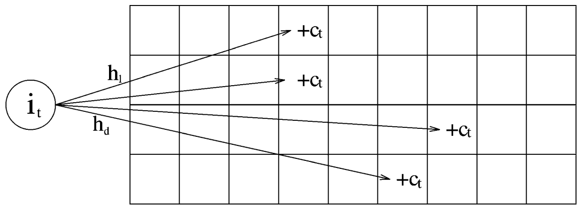

Fig. 1. Each item $i$ is mapped to one cell in each row of the array of counts: when an update of $c_t$ to item $i_t$ arrives, $c_t$ is added to each of these cells.

## 4. Approximate query answering using CM sketches

For each of the three queries introduced in Section 2: point, range, and inner product queries, we show how they can be answered using count-min sketches.

### 4.1. Point query

We first show the analysis for point queries for the non-negative case.

*Estimation procedure*
The answer to $\mathcal{Q}(i)$ is given by $\hat{a}_i = \min_j count[j, h_j(i)]$.

**Theorem 1.** *The estimate $\hat{a}_i$ has the following guarantees: $a_i \leqslant \hat{a}_i$; and, with probability at least $1-\delta$,*

$$\hat{a}_i \leqslant a_i + \varepsilon \|\boldsymbol{a}\|_1.$$

**Proof.** We introduce indicator variables $I_{i,j,k}$, which are 1 if $(i \neq k) \wedge (h_j(i) = h_j(k))$, and 0 otherwise. By pairwise independence of the hash functions, then

$$\mathsf{E}(I_{i,j,k}) = \mathsf{Pr}\big[h_j(i) = h_j(k)\big] \leqslant \frac{1}{\mathrm{range}(h_j)} = \frac{\varepsilon}{e}.$$

Define the variable $X_{i,j}$ (random over the choices of $h_i$) to be $X_{i,j} = \sum_{k=1}^{n} I_{i,j,k} a_k$. Since all $a_i$ are non-negative in this case, $X_{i,j}$ is a non-negative variable. By construction, $count[j, h_j(i)] = a_i + X_{i,j}$. So, clearly, $\min count[j, h_j(i)] \geqslant a_i$. For the other direction, observe that

$$\mathsf{E}(X_{i,j}) = \mathsf{E}\left(\sum_{k=1}^{n} I_{i,j,k} a_k\right) \leqslant \sum_{k=1}^{n} a_k \mathsf{E}(I_{i,j,k}) \leqslant \frac{\varepsilon}{e} \|\boldsymbol{a}\|_1$$

by pairwise independence of $h_j$, and linearity of expectation. By the Markov inequality,

$$\begin{aligned}
\mathsf{Pr}\big[\hat{a}_i > a_i + \varepsilon\|a\|_1\big] &= \mathsf{Pr}\big[\forall_j.\, count\big[j, h_j(i)\big] > a_i + \varepsilon\|\boldsymbol{a}\|_1\big] \\
&= \mathsf{Pr}\big[\forall_j.\, a_i + X_{i,j} > a_i + \varepsilon\|\boldsymbol{a}\|_1\big] \\
&= \mathsf{Pr}\big[\forall_j.\, X_{i,j} > e\mathsf{E}(X_{i,j})\big] \\
&< e^{-d} \leqslant \delta. \qquad \square
\end{aligned}$$

The time to produce the estimate is $O(\ln(1/\delta))$ since finding the minimum count can be done in linear time; the same time bound holds for updates. The constant $e$ is used here to minimize the space used: more generally, we can set $w = \varepsilon/b$ and $d = \log_b(1/\delta)$ for any $b > 1$ to get the same accuracy guarantee. Choosing $b = e$ minimizes the space used, since this solves $\mathrm{d}(wd)/\mathrm{d}b = 0$, giving a cost of $(2 + e/\varepsilon)\ln(1/\delta)$ words. For implementations, it may be preferable to use other (integer) values of $b$ for simpler computations or faster updates. Note that for values of $a_i$ that are large relative to $\|\boldsymbol{a}\|_1$, the bound in terms of $\varepsilon\|\boldsymbol{a}\|_1$ can be translated into a relative error in terms of $a_i$. This has implications for certain applications which rely on retrieving large values, such as large wavelet or Fourier coefficients.

The best known previous result using sketches was in [5]: there sketches were used to approximate point queries. Results were stated in terms of the frequencies of individual items. For arbitrary distributions, the space used is $O((1/\varepsilon^2)\log(1/\delta))$, and the dependency on $\varepsilon$ is $1/\varepsilon^2$ in every case considered. A significant difference between CM sketches and previous work comes in the analysis:

- All prior analyses of sketch structures compute the variance of their estimators in order to apply the Chebyshev inequality, which brings the dependency on $\varepsilon^2$. Directly applying the Markov inequality yields a more direct analysis which depends only on $\varepsilon$. Practitioners may have discovered that less than $O(1/\varepsilon^2)$ space is needed in practice: here, we give proof of why this is so and the tighter bound.
- Because only positive quantities are added to the counters then it is possible to take the minimum instead of the median for the estimate. This allows a simple calculation of the failure probability, without recourse to Chernoff bounds. This significantly improves the constants involved: in [5], for example, the constant factors within the big-Oh notation is at least 256; here, the constant factor is less than 3.
- The error bound here is one-sided, as opposed to all previous constructions which gave two-sided errors. This brings benefits for many applications which use sketches.

In Section 6 we show how all existing sketch constructions can be viewed as variations of a common procedure. This emphasizes the importance of our attempt to find the simplest sketch construction which has the best guarantees and smallest constants. A similar result holds when entries of the implicit vector $\boldsymbol{a}$ may be negative, which is the general case.

*Estimation procedure for general case*
This time $Q(i)$ is answered with $\hat{a}_i = \text{median}_j\, \mathit{count}[j, h_j(i)]$.

**Theorem 2.** *With probability* $1 - \delta^{1/4}$,

$$a_i - 3\varepsilon\|\boldsymbol{a}\|_1 \leqslant \hat{a}_i \leqslant a_i + 3\varepsilon\|\boldsymbol{a}\|_1.$$

**Proof.** Observe that $\mathsf{E}(|\mathit{count}[j, h_j(i)] - a_i|) \leqslant \varepsilon\|\boldsymbol{a}\|_1/e$, and so the probability that any count is off by more than $3\varepsilon\|\boldsymbol{a}\|_1$ is less than $1/8$. Applying Chernoff bounds tells us that the probability of the median of $\ln(1/\delta)$ copies of this procedure being wrong is less than $\delta^{1/4}$. $\square$

The time to produce the estimate is $O(\ln(1/\delta))$ and the space used is $(2 + e/\varepsilon)\ln(1/\delta)$ words. The best prior result for this problem was the method of [5]. Again, the dependence on $\varepsilon$ here is improved from $1/\varepsilon^2$ to $1/\varepsilon$. By avoiding analyzing the variance of the estimator, again the analysis is simplified, and the constants are significantly smaller than in previous works.

### 4.2. Inner product query

*Estimation procedure*

Set $(\widehat{\boldsymbol{a} \odot \boldsymbol{b}})_j = \sum_{k=1}^{w} count_{\boldsymbol{a}}[j,k] * count_{\boldsymbol{b}}[j,k]$. Our estimation of $\mathcal{Q}(\boldsymbol{a},\boldsymbol{b})$ for non-negative vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is $\widehat{\boldsymbol{a} \odot \boldsymbol{b}} = \min_j (\widehat{\boldsymbol{a} \odot \boldsymbol{b}})_j$.

**Theorem 3.** *$\boldsymbol{a} \odot \boldsymbol{b} \leqslant \widehat{\boldsymbol{a} \odot \boldsymbol{b}}$ and, with probability $1-\delta$, $\widehat{\boldsymbol{a} \odot \boldsymbol{b}} \leqslant \boldsymbol{a} \odot \boldsymbol{b} + \varepsilon\|\boldsymbol{a}\|_1\|\boldsymbol{b}\|_1$.*

**Proof.**

$$(\widehat{\boldsymbol{a} \odot \boldsymbol{b}})_j = \sum_{i=1}^{n} a_i b_i + \sum_{\substack{p \neq q \\ h_j(p)=h_j(q)}} a_p b_q.$$

Clearly, $\boldsymbol{a} \odot \boldsymbol{b} \leqslant \widehat{\boldsymbol{a} \odot \boldsymbol{b}}_j$ for non-negative vectors. By pairwise independence of $h$,

$$\mathsf{E}(\widehat{\boldsymbol{a} \odot \boldsymbol{b}}_j - \boldsymbol{a} \odot \boldsymbol{b}) = \sum_{p \neq q} \mathsf{Pr}[h_j(p) = h_j(q)] a_p b_q \leqslant \sum_{p \neq q} \frac{\varepsilon a_p b_q}{e} \leqslant \frac{\varepsilon \|\boldsymbol{a}\|_1 \|\boldsymbol{b}\|_1}{e}.$$

So, by the Markov inequality, $\mathsf{Pr}[\widehat{\boldsymbol{a} \odot \boldsymbol{b}} - \boldsymbol{a} \odot \boldsymbol{b} > \varepsilon\|\boldsymbol{a}\|_1\|\boldsymbol{b}\|_1] \leqslant \delta$, as required. □

The space and time to produce the estimate is $O((1/\varepsilon)\log(1/\delta))$. Updates are performed in time $O(\log(1/\delta))$.

Note that in the special case where $b_i = 1$, and $\boldsymbol{b}$ is zero at all other locations, then this procedure is identical to the above procedure for point estimation, and gives the same error guarantee (since $\|\boldsymbol{b}\|_1 = 1$). A similar result holds in the general case, where vectors may have negative entries. Taking the median of $(\widehat{\boldsymbol{a} \odot \boldsymbol{b}})_j$ would give a guaranteed good quality approximation; however, we do not know of any application which makes use of inner products of such vectors, so we do not give the full details here. We next consider the application of inner-product computation to Join size estimation, where the vectors generated have non-negative entries.

Join size estimation is important in database query planners in order to determine the best order in which to evaluate queries. The *join size* of two database relations on a particular attribute is the number of items in the cartesian product of the two relations which agree the value of that attribute. We assume without loss of generality that attribute values in the relation are integers in the range $1 \ldots n$. We represent the relations being joined as vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ so that the values $a_i$ represents the number of tuples which have value $i$ in the first relation, and $b_i$ similarly for the second relation. Then clearly $\boldsymbol{a} \odot \boldsymbol{b}$ is the join size of the two relations. Using sketches allows estimates to be made in the presence of items being inserted to and deleted from relations. The following corollary follows from the above theorem.

**Corollary 1.** *The join size of two relations on a particular attribute can be approximated up to $\varepsilon\|\boldsymbol{a}\|_1\|\boldsymbol{b}\|_1$ with probability $1-\delta$, by keeping space $O((1/\varepsilon)\log(1/\delta))$.*

Previous results have used the “tug-of-war” sketches [1]. However, here some care is needed in the comparison of the two methods: the prior work gives guarantees in terms of the $L_2$ norm of the underlying vectors, with additive error of $\varepsilon\|\boldsymbol{a}\|_2\|\boldsymbol{b}\|_2$; here, the result is in terms of the $L_1$ norm. In some cases, the $L_2$ norm can be quadratically smaller than the $L_1$ norm. However, when the distribution of items is non-uniform, for example when certain items contribute a large amount to the join size, then the two norms are closer, and the guarantees of the CM sketch method is closer to the existing method. As before, the space cost of previous methods was $\Omega(1/\varepsilon^2)$, so there is a significant space saving to be had with CM sketches.

## 4.3. *Range query*

Define $\boldsymbol{\chi}(l,r)$ to be the vector of dimension $n$ such that $\boldsymbol{\chi}(l,r)_i = 1 \Leftrightarrow l \leqslant i \leqslant r$, and 0 otherwise. Then $Q(l,r)$ can straightforwardly be re-posed as $Q(\boldsymbol{a}, \boldsymbol{\chi}(l,r))$. However, this method has two drawbacks: first, the error guarantee is in terms of $\|\boldsymbol{a}\|_1\|\boldsymbol{\chi}(l,r)\|_1$ and therefore large range sums have an error guarantee which increases linearly with the length of the range; and second, the time cost to directly compute the sketch for $\boldsymbol{\chi}(l,r)$ depends linearly in the length of the range, $r-l+1$. In fact, it is clear that computing range sums in this way using our sketches is not much different to simply computing point queries for each item in the range, and summing the estimates. One way to avoid the time complexity is to use range-sum random variables from [20] to quickly determine a sketch of $\boldsymbol{\chi}(l,r)$, but that is expensive and still does not overcome the first drawback. Instead, we adopt the use of *dyadic ranges* from [21]: a dyadic range is a range of the form $[x2^y+1\ldots(x+1)2^y]$ for parameters $x$ and $y$.

*Estimation procedure*

Keep $\log_2 n$ CM sketches, in order to answer range queries $Q(l,r)$ approximately. Any range query can be reduced to at most $2\log_2 n$ *dyadic range* queries, which in turn can each be reduced to a single point query. Each point in the range $[1\ldots n]$ is a member of $\log_2 n$ dyadic ranges, one for each $y$ in the range $0\ldots\log_2 n-1$. A sketch is kept for each set of dyadic ranges of length $2^y$, and update each of these for every update that arrives. Then, given a range query $Q(l,r)$, compute the at most $2\log_2 n$ dyadic ranges which canonically cover the range, and pose that many point queries to the sketches, returning the sum of the queries as the estimate.

**Example 1.** For $n=256$, the range $[48,107]$ is canonically covered by the non-overlapping dyadic ranges $[48\ldots48], [49\ldots64], [65\ldots96], [97\ldots104], [105\ldots106], [107\ldots107]$.

Let $a[l,r]=\sum_{i=l}^{r} a_i$ be the answer to the query $Q(l,r)$ and let $\hat{a}[l,r]$ be the estimate using the procedure above.

**Theorem 4.** *$a[l,r] \leqslant \hat{a}[l,r]$ and with probability at least $1-\delta$,*

$$\hat{a}[l,r] \leqslant a[l,r] + 2\varepsilon\log n\|\boldsymbol{a}\|_1.$$

**Proof.** Applying the inequality of Theorem 1, then $a[l,r] \leqslant \hat{a}[l,r]$. Consider each estimator used to form $\hat{a}[l,r]$; the expectation of the additive error for any of these is $2\log n(\varepsilon/e)\|\boldsymbol{a}\|_1$, by linearity of expectation of the errors of each point estimate. Applying the same Markov inequality argument as before, the probability that this additive error is more than $2\varepsilon\log n\|\boldsymbol{a}\|_1$ for any estimator is less than $1/e$; hence, for all of them the probability is at most $\delta$. $\square$

The time to compute the estimate or to make an update is $O(\log n \log(1/\delta))$. The space used is $O((\log n/\varepsilon)\log(1/\delta))$.

The above theorem states the bound for the standard CM sketch size. The guarantee will be more useful when stated without terms of $\log n$ in the approximation bound. This can be changed by increasing the size of the sketch, which is equivalent to rescaling $\varepsilon$. In particular, if we want to estimate a range sum correct up to $\varepsilon'\|\boldsymbol{a}\|_1$ with probability $1-\delta$ then set $\varepsilon = \varepsilon'/(2\log n)$. The space used is $O((\log^2 n/\varepsilon')\log(1/\delta))$. An obvious improvement of this technique in practice is to keep exact counts for the first few levels of the hierarchy, where there are only a small number of dyadic ranges. This improves the space, time and accuracy of the algorithm in practice, although the asymptotic bounds are unaffected.

For smaller ranges, ranges that are powers of 2, or more generally, any range whose endpoints can be expressed in binary using a small number of 1s, then improved bounds are possible; we have given the worst case bounds above.

One way to compute approximate range sums is via approximate quantiles: use an algorithm such as [22,25] to find the $\varepsilon$ quantiles of the stream, and then count how many quantiles fall within the range of interest to give an $O(\varepsilon)$ approximation of the range query. Such an approach has several disadvantages:

(1) Existing approximate quantile methods work in the cash register model, rather than the more general turnstile model that our solutions work in.
(2) The time cost to update the data structure can be high, sometimes linear in the size of the structure.
(3) Existing algorithms assume single items arriving one by one, so they do not handle fractional values or large values being added, which can be easily handled by sketch-based approaches.
(4) The worst case space bound depends on $O((1/\varepsilon)\log(\|\boldsymbol{a}\|_1/\varepsilon))$, which can grow indefinitely. The sketch based solution works in fixed space that is independent of $\|\boldsymbol{a}\|_1$.

The best previous bounds for this problem in the turnstile model are given in [21], where range queries are answered by keeping $O(\log n)$ sketches, each of size $O((1/\varepsilon'^2)\log n \times \log(\log n/\delta))$ to give approximations with additive error $\varepsilon\|\boldsymbol{a}\|_1$ with probability $1-\delta'$. Thus the space used there is $O((\log^2 n/\varepsilon'^2)\log(\log n/\delta))$ and the time for updates is linear in the space used. The CM sketch improves the space and time bounds; it improves the constant factors as well as the asymptotic behavior. The time to process an update is significantly improved, since only a few entries in the sketch are modified, rather than a linear number.

## 5. Applications of count-min sketches

By using CM sketches, we show how to improve best known time and space bounds for the two problems from Section 2.

### 5.1. Quantiles in the turnstile model

In [21] the authors showed that finding the approximate $\phi$-quantiles of the data subject to insertions and deletions can be reduced to the problem of computing range sums. Put simply, the algorithm is to do binary searches for ranges $1 \ldots r$ whose range sum $a[1, r]$ is $k\phi\|\boldsymbol{a}\|_1$ for $0 \leqslant k \leqslant 1/\phi$. The method of [21] uses *random subset sums* to compute range sums. By replacing this structure with count-min sketches, the improved results follow immediately. By keeping $\log n$ sketches, one for each dyadic range and setting the accuracy parameter for each to be $\varepsilon/\log n$ and the probability guarantee to $\delta\phi/\log n$, the overall probability guarantee for all $1/\phi$ quantiles is achieved.

**Theorem 5.** *$\varepsilon$-approximate $\phi$-quantiles can be found with probability at least $1 - \delta$ by keeping a data structure with space $O((1/\varepsilon)\log^2 n \log(\log n/(\phi\delta)))$. The time for each insert or delete operation is $O(\log n \log(\log n/(\phi\delta)))$, and the time to find each quantile on demand is $O(\log n \log(\log n/(\phi\delta)))$.*

Choosing CM sketches over random subset sums improves both the query time and the update time from $O((1/\varepsilon^2)\log^2 n \log(\log n/(\varepsilon\delta)))$, by a factor of more than $(34/\varepsilon^2)\log n$. The space requirements are also improved by a factor of at least $34/\varepsilon$.

It is illustrative to contrast our bounds with those for the problem in the weaker cash register model where items are only inserted (recall that in our stronger turnstile model, items are deleted as well). The previously best known space bounds for finding approximate quantiles is $O((1/\varepsilon)(\log^2(1/\varepsilon) + \log^2\log(1/\delta)))$ space for a randomized sampling solution [25] and $O((1/\varepsilon)\log(\varepsilon\|\boldsymbol{a}\|_1))$ space for a deterministic solution [22]. These bounds are not completely comparable, but our result is the first on the more powerful turnstile model to be comparable to the cash register model bounds in the leading $1/\varepsilon$ term.

### 5.2. Heavy hitters

We consider this problem in both the cash register model (meaning that all updates are positive) and the more challenging turnstile model (where updates are both positive and negative, with the restriction that count of any item is never less than zero, i.e., $a_i(t) \geqslant 0$).

*Cash register case*

It is possible to maintain the current value of $\|\boldsymbol{a}\|_1$ throughout, since $\|\boldsymbol{a}(t)\|_1 = \sum_{i=1}^{t} c_i$. On receiving item $(i_t, c_t)$, update the sketch as before and pose point query $\mathcal{Q}(i_t)$: if estimate $\hat{a}_{i_t}$ is above the threshold of $\phi\|\boldsymbol{a}(t)\|_1$, $i_t$ is added to a heap. The heap is kept small by checking that the current estimated count for the item with lowest count is above threshold; if not, it is deleted from the heap as in [5]. At the end of the input, the heap is scanned, and all items in the heap whose estimated count is still above $\phi\|\boldsymbol{a}\|_1$ are output.

**Theorem 6.** *The heavy hitters can be found from an inserts only sequence of length $\|\boldsymbol{a}\|_1$, by using CM sketches with space $O((1/\varepsilon)\log(\|\boldsymbol{a}\|_1/\delta))$, and time $O(\log(\|\boldsymbol{a}\|_1/\delta))$ per item. Every item which occurs with count more than $\phi\|\boldsymbol{a}\|_1$ time is output, and with probability $1-\delta$, no item whose count is less than $(\phi-\varepsilon)\|\boldsymbol{a}\|_1$ is output.*

**Proof.** This procedure relies on the fact that the threshold value increases monotonically: therefore, if an item did not pass the threshold in the past, it cannot do so in the future without its count increasing. By checking the estimated value every time an items value increases, no heavy hitters will be omitted. By the one-sided error guarantee of sketches, every heavy hitter is included in the output, but there is some possibility of including non-heavy hitters in the output. To do this, the parameter $\delta$ is scaled to ensure that over all $\|\boldsymbol{a}\|_1$ queries posed to the sketch, the probability of mistakenly outputting an infrequent item is bounded by $1-\delta$, using the union bound. □

We can compare our results to the best known previous work. The algorithm in [5] solves this problem using Count sketches in worst case space $O((1/\varepsilon^2)\log(\|\boldsymbol{a}\|_1/\delta))$, which we strictly improve here. A randomized algorithm given in [26] has *expected* space cost $O((1/\varepsilon)\log(1/(\phi\delta)))$, slightly better than our worst case space usage. Meanwhile, a deterministic algorithm in the same paper solves the problem in worst case space $O((1/\varepsilon)\log(\|\boldsymbol{a}\|_1/\varepsilon))$. However, for both algorithms in [26] the time cost of processing each insertion can be high ($\Omega(1/\varepsilon)$): periodically, there are operations with cost linear in the space used. For high speed applications, our worst case time bound may be preferable.

*Turnstile case*

We adopt the solution given in [8], which describes a divide and conquer procedure to find the heavy hitters. This keeps sketches for computing range sums: $\log n$ different sketches, one for each different dyadic range. When an update $(i_t, c_t)$ arrives, then each of these is updated as before. In order to find all the heavy hitters, a parallel binary search is performed, descending one level of the hierarchy at each step. Nodes in the hierarchy (corresponding to dyadic ranges) whose estimated weight exceeds the threshold of $(\phi+\varepsilon)\|\boldsymbol{a}\|_1$ are split into two ranges, and investigated recursively. All single items found in this way whose approximated count exceeds the threshold are output.

We instead must limit the number of items output whose true frequency is less than the fraction $\phi$. This is achieved by setting the probability of failure for each sketch to be $\delta\phi/(2\log n)$. This is because, at each level there are at most $1/\phi$ items with frequency more than $\phi$. At most twice this number of queries are made at each level, for all of the $\log n$ levels. By scaling $\delta$ like this and applying the union bound ensures that, over all the queries, the total probability that any one (or more) of them overestimated by more than a fraction $\varepsilon$ is bounded by $\delta$, and so the probability that every query succeeds is $1-\delta$. It follows that

**Theorem 7.** *The algorithm uses space $O((1/\varepsilon)\log n \log(2\log n/(\delta\phi)))$ and time $O(\log n \times \log(2\log n/(\delta\phi)))$ per update. Every item with frequency at least $(\phi+\varepsilon)\|\boldsymbol{a}\|_1$ is output, and with probability $1-\delta$ no item whose frequency is less than $\phi\|\boldsymbol{a}\|_1$ is output.*

The previous best known bound appears in [8], where a non-adaptive group testing approach was described. Here, the space bounds agree asymptotically but have been improved in constant factors; a further improvement is in the nature of the guarantee: previous methods gave probabilistic guarantees about outputting the heavy hitters. Here, there is absolute certainty that this procedure will find and output every heavy hitter, because the CM sketches never underestimate counts, and strong guarantees are given that no non-heavy hitters will be output. This is often desirable.

In some situations in practice, it is vital that updates are as fast as possible, and here update time can be played off against search time: ranges based on powers of two can be replaced with an arbitrary branching factor $k$, which reduces the number of levels to $\log_k n$, at the expense of costlier queries and weaker guarantees on outputting non-heavy hitters.

*Hierarchical heavy hitters.* A generalization of this problem is finding hierarchical heavy hitters [7], which assumes that the items are leaves in a hierarchy of depth $h$. Here the goal is to find all nodes in the hierarchy which are heavy hitters, *after discounting the contribution of any descendant heavy hitter nodes*. Using our CM sketch, the cost of the solution given in [7] for the turnstile model can be improved from $O((h/\varepsilon^2)\log(1/\delta))$ space and time per update to $O((h/\varepsilon)\log(1/\delta))$ space and $O(h\log(1/\delta))$ time per update.

## 6. Comparison of sketch techniques

We give a common framework to summarize known sketch constructions, and compare the time and space requirements for each of the fundamental queries—point, range and inner products—using them.

Here is a brief summary of known sketch constructions. The first sketch construction was that of Alon, Matias and Szegedy [2], whose *tug-of-war* sketches are computed using 4-wise random hash functions $g_j$ mapping items to $\{+1,-1\}$. The $j$th entry of the sketch, which is a vector of length $O((1/\varepsilon^2)\log(1/\delta))$, is defined to be $\sum a_i * g_j(i)$, which is easy to maintain under updates. This structure was applied to finding inner products in [1,20] where, in our notation, it was shown that it is possible to compute inner products with additive error $\pm\varepsilon\|\boldsymbol{a}\|_2\|\boldsymbol{b}\|_2$. In [20], the authors use these sketches to compute large wavelet coefficients. In particular, they show how the structure allows point queries to be computed up to additive error of $\pm\varepsilon\|\boldsymbol{a}\|_2$. They also compute range sums $Q(l,r)$: here range-summable random variables are used to compute the sums efficiently, but this incurs a factor of $O(\log n)$ additional time and space to compute these. Also note that here the error guarantees are much worse for large ranges: $\varepsilon(r-l+1)\|\boldsymbol{a}\|_1$.

For point queries only, then pairwise independence suffices for tug-of-war sketches, as observed by [5]. These authors additionally used a second hash set of hash functions, $h_j$, to spread out the effect of high frequency items in their *count sketch*. For point queries, this gives the same space bounds, but an improved time bound to $O(\log(1/\delta))$ for updating the

Table 1
Comparison of different space and time requirements of sketching methods

| Method | Ref. | Query | Space | Update time | Query time | Randomness needed |
|---|---|---|---|---|---|---|
| Tug-of-war | [1] | Inner-product | $1/\varepsilon^2$ | $1/\varepsilon^2$ | $1/\varepsilon^2$ | 4-wise |
| Tug-of-war | [20] | Point | $\log n/\varepsilon^2$ | $\log n/\varepsilon^2$ | $\log n/\varepsilon^2$ | 4-wise |
| | | Range | $\log n/\varepsilon^2$ | $\log n/\varepsilon^2$ | $\log n/\varepsilon^2$ | 4-wise |
| Random subset-sums | [21] | Range | $\log^2 n/\varepsilon^2$ | $\log^2 n/\varepsilon^2$ | $\log^2 n/\varepsilon^2$ | Pairwise |
| Count sketches | [5] | Point | $1/\varepsilon^2$ | 1 | 1 | Pairwise |
| Count-min sketches | (this paper) | Point | $1/\varepsilon$ | 1 | 1 | Pairwise |
| | | Inner-product | $1/\varepsilon$ | 1 | $1/\varepsilon$ | Pairwise |
| | | Range | $\log n/\varepsilon$ | $\log n$ | $\log n/\varepsilon$ | Pairwise |

The dependency on $\varepsilon$ and $n$ is shown, a factor of $O(\log(1/\delta))$ applies to each entry and is omitted.

sketch.[4] *Random subset sums* were introduced in [21] in order to compute point queries and range sums. Here, 2-universal hash functions[5] $h_j$ map items to $\{0,1\}$, and the $j$th entry of the sketch is maintained as $\sum_{i=1}^n a_i * h_j(i)$. The asymptotic space and time bounds for different techniques in terms of their dependence on epsilon are summarized in Table 1.

All of the above sketching techniques, and the one which we propose in this paper, can all be described in a common way. Firstly, they can all be viewed as linear projections of the vector $\boldsymbol{a}$ with appropriately chosen random vectors, but more than this, the computation of these linear projections is very similar between all methods. Define a sketch to be a two dimensional array of dimension $w$ by $d$. Let $h_1 \ldots h_d$ be pairwise independent hash functions mapping from $\{1 \ldots n\}$ to $\{1 \ldots w\}$, and let $g_1 \ldots g_d$ be another hash function whose range and randomness varies from construction to construction. The $(j,k)$th entry of the sketch is defined to be

$$\sum_{i\colon h_k(i)=j} a_i * g_k(i).$$

The contents of the sketch for each of the above techniques is specified by the parameters $w$, $d$, and $g$; methods of answering queries using the sketch vary from technique to technique. The update time for each sketch is $O(d)$ computations of $g$ and $h$, and the space requirement is dominated by the $O(wd)$ counters, provided that the hash functions can be stored efficiently.

- Tug of war sketches have $w = 1$, $d = O((1/\varepsilon^2)\log(1/\delta))$, $g(i)$ is $\{+1,-1\}$ with 4-wise independence.
- Count sketches have $w = O(1/\varepsilon^2)$, $d = O(\log(1/\delta))$, $g(i)$ is $\{+1,-1\}$ with 2-wise independence.

[4] We observe that the same idea, of replacing the averaging of $O(1/\varepsilon^2)$ copies of an estimator with a 2-universal hash function distributing to $O(1/\varepsilon^2)$ buckets can also reduce the update time for "tug-of-war" sketches and similar constructions to $O(\log(1/\delta))$.

[5] In [21] the authors use a 3-wise independent hash function onto $\{0,1\}$, chosen because it is easy to compute, but pairwise suffices.

- Random subset sums have $w = 2$, $d = (24/\varepsilon^2)\log(1/\delta)$, $g(i)$ is 1.
- Count-min sketches have $w = e/\varepsilon$, $d = \ln(1/\delta)$, $g(i) = 1$.

We briefly mention some other well-known sketch constructions, and explain why they are not appropriate for the queries we study here. [14] gave a sketch construction for computing the $L_1$ difference between vectors. It extends the tug-of-war construction, the technical contribution being a demonstration of how to compute range sums of 4-wise random variables efficiently. Indyk [23] pioneered the use of stable distributions in sketch computations, again in order to compute $L_p$ norms of vectors presented as a sequence of updates. However, all these norm computations do not directly answer the three query types we consider here. Lastly, we note that our count-min sketches appear similar in outline to both the uniscan algorithm due to Fang et al. [13] and the parallel multistage filters of Estan and Varghese [11]. Our construction differs for the following reasons:

(1) the structures from prior work are not sketches: they do not approximate any value, but instead return only a binary answer about whether an item has exceeded a certain numeric threshold;
(2) the algorithms used require updates to be only positive; and
(3) the analysis does not consider any limited independence needed for the hash functions, in contrast to CM sketches, which require only pairwise independence.

The methods in [11,13] as such seem to use fully independent hash functions which is prohibitive in principle.

## 7. Conclusions

We have introduced the count-min sketch, and shown how to estimate fundamental queries such as point, range or inner product queries as well as solve more sophisticated problems such as quantiles and heavy hitters. The space and/or time bounds of our solutions improve previously best known bounds for these problems. Typically the improvement is from $1/\varepsilon^2$ factor to $1/\varepsilon$ which is significant in real applications. Our CM sketch is quite simple, and is likely to find many applications, including in hardware solutions for these problems.

We have recently applied these ideas to the problem of change detection on data streams [9], and we also believe that it can be applied to improve the time and space bounds for constructing approximate wavelet and histogram representations of data streams [19]. Also, the CM sketch can also be naturally extended to solve problems on streams that describe multidimensional arrays rather than the unidimensional array problems we have discussed so far.

Our CM sketch is not effective when one wants to compute the norms of data stream inputs. These have applications to computing correlations between data streams and tracking the number of distinct elements in streams, both of which are of great interest. It is an open problem to design extremely simple, practical sketches such as our CM sketch for estimating such correlations and more complex data stream applications.

## References

[1] N. Alon, P. Gibbons, Y. Matias, M. Szegedy, Tracking join and self-join sizes in limited storage, in: Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems (PODS ’99), 1999, pp. 10–20.

[2] N. Alon, Y. Matias, M. Szegedy, The space complexity of approximating the frequency moments, in: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, 1996, pp. 20–29; Journal version in J. Comput. System Sci. 58 (1999) 137–147.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of Symposium on Principles of Database Systems (PODS), 2002, pp. 1–16.

[4] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, L. Trevisan, Counting distinct elements in a data stream, in: Proceedings of RANDOM 2002, 2002, pp. 1–10.

[5] M. Charikar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, in: Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP), 2002, pp. 693–703.

[6] G. Cormode, M. Datar, P. Indyk, S. Muthukrishnan, Comparing data streams using Hamming norms, in: Proceedings of 28th International Conference on Very Large Data Bases, 2002, pp. 335–345; Journal version in IEEE Trans. Knowledge Data Engrg. 15 (3) (2003) 529–541.

[7] G. Cormode, F. Korn, S. Muthukrishnan, D. Srivastava, Finding hierarchical heavy hitters in data streams, in: International Conference on Very Large Databases, 2003, pp. 464–475.

[8] G. Cormode, S. Muthukrishnan, What’s hot and what’s not: tracking most frequent items dynamically, in: Proceedings of ACM Principles of Database Systems, 2003, pp. 296–306.

[9] G. Cormode, S. Muthukrishnan, What’s new: finding significant differences in network data streams, in: Proceedings of IEEE INFOCOM, 2004.

[10] A. Dobra, M. Garofalakis, J.E. Gehrke, R. Rastogi, Processing complex aggregate queries over data streams, in: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, 2002, pp. 61–72.

[11] C. Estan, G. Varghese, New directions in traffic measurement and accounting, in: Proceedings of ACM SIGCOMM, Computer Communication Review 32 (4) (2002) 323–338.

[12] C. Estan, G. Varghese, Data streaming in computer networks, in: Proceedings of Workshop on Management and Processing of Data Streams, 2003, http://www.research.att.com/conf/mpds2003/schedule/estanV.ps.

[13] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, J.D. Ullman, Computing iceberg queries efficiently, in: Proceedings of the Twenty-Fourth International Conference on Very Large Databases, 1998, pp. 299–310.

[14] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan, An approximate $L_1$-difference algorithm for massive data streams, in: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999, pp. 501–511.

[15] P. Flajolet, G.N. Martin, Probabilistic counting, in: 24th Annual Symposium on Foundations of Computer Science, 1983, pp. 76–82; Journal version in J. Comput. System Sci. 31 (1985) 182–209.

[16] M. Garofalakis, J. Gehrke, R. Rastogi, Querying and mining data streams: you only get one look, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2002.

[17] P. Gibbons, Y. Matias, Synopsis structures for massive data sets, in: DIMACS Ser. Discrete Math. Theoret. Comput. Sci. A, 1999.

[18] P. Gibbons, S. Tirthapura, Estimating simple functions on the union of data streams, in: Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures, 2001, pp. 281–290.

[19] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, M. Strauss, Fast, small-space algorithms for approximate histogram maintenance, in: Proceedings of the 34th ACM Symposium on Theory of Computing, 2002, pp. 389–398.

[20] A. Gilbert, Y. Kotidis, S. Muthukrishnan, M. Strauss, Surfing wavelets on streams: one-pass summaries for approximate aggregate queries, in: Proceedings of 27th International Conference on Very Large Data Bases, 2001, pp. 79–88; Journal version in IEEE Trans. Knowledge Data Engrg. 15 (3) (2003) 541–554.

[21] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, M. Strauss, How to summarize the universe: dynamic maintenance of quantiles, in: Proceedings of 28th International Conference on Very Large Data Bases, 2002, pp. 454–465.

[22] M. Greenwald, S. Khanna, Space-efficient online computation of quantile summaries, SIGMOD Record (ACM Special Interest Group on Management of Data) 30 (2) (2001) 58–66.
[23] P. Indyk, Stable distributions, pseudorandom generators, embeddings and data stream computation, in: Proceedings of the 40th Symposium on Foundations of Computer Science, 2000, pp. 189–197.
[24] W.B. Johnson, J. Lindenstrauss, Extensions of Lipshitz mapping into Hilbert space, Contemp. Math. 26 (1984) 189–206.
[25] G.S. Manku, S. Rajagopalan, B.G. Lindsay, Approximate medians and other quantiles in one pass and with limited memory, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1998, pp. 426–435.
[26] G.S. Manku, R. Motwani, Approximate frequency counts over data streams, in: Proceedings of 28th International Conference on Very Large Data Bases, 2002, pp. 346–357.
[27] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
[28] S. Muthukrishnan, Data streams: algorithms and applications, in: ACM–SIAM Symposium on Discrete Algorithms, 2003, http://athos.rutgers.edu/~muthu/stream-1-1.ps.
[29] N. Thaper, P. Indyk, S. Guha, N. Koudas, Dynamic multidimensional histograms, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2002, pp. 359–366.
[30] D. Woodruff, Optimal space lower bounds for all frequency moments, in: ACM–SIAM Symposium on Discrete Algorithms, 2004.