DIMACS Technical Report 2003-20
June 2003

Improved Data Stream Summary: The Count-Min Sketch and its Applications

by

Graham Cormode[1,2]   S. Muthukrishnan[3,4]

[1]DIMACS Postdoctoral Fellow
[2]graham@dimacs.rutgers.edu Center for Discrete Mathematics and Computer Science (DIMACS) Rutgers University, Piscataway NJ. Supported by NSF ITR 0220280 and NSF EIA 02-05116.
[3]DIMACS Permanent Member
[4]muthu@cs.rutgers.edu Division of Computer and Information Systems, Rutgers University and AT&T Research. Supported by NSF CCR 0087022, NSF ITR 0220280 and NSF EIA 02-05116.

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 quickly; in addition, it can be applied to solve several important problems in data streams such as finding quantiles, frequent items, etc. In both cases, the time and space bounds we show for using the CM sketch significantly improve those known for previous sketch constructions — 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 $\boldsymbol{a}(t) =< a_1(t), \dots a_i(t), \dots a_n(t) >$. Initially, $\boldsymbol{a}$ is the zero vector, 0 for all i. Updates to individual entries of the vector are presented as a stream of pairs. The tth 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. First, the space used by the algorithm should be small, at most poly-logarithmic in n, the space required to represent $\boldsymbol{a}$ explicitly. Since the space is sublinear in data and input size, the data structures used by the algorithms to represent the input data stream is merely a summary—aka a *sketch* or synopsis [12])—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, ε and δ, meaning that the error in answering a query is within a factor of ε with probability δ. The space and update time will consequently depend on ε and δ; our goal will be limit this dependence as much as is possible. Many applications that deal with massive data, such as Internet traffic log analysis and monitoring contents of massive databases, motivate this one-pass data stream setup. See [3, 22] for detailed description of the motivations.

In recent years, the data stream setup has become popular. Several different sketches have been proposed which allow a number of simple aggregate functions to be approximated. Quantities for which efficient sketches have been designed include the L_1 and L_2 norms of vectors [2, 9, 17], the number of distinct items in a sequence (ie number of non-zero entries in $\boldsymbol{a}(t)$) [10, 13, 5], join and self-join sizes of relations (representable as inner-products of vectors $\boldsymbol{a}(t), \boldsymbol{b}(t)$) [2, 1], item and range sum queries [15, 4]. 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 and structures: quantiles [16], wavelets [15], histograms [23, 14], database aggregates and multi-way join sizes [7], 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 following drawbacks.

- Although sketches use small space, 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.

- Many sketch constructions require time linear in the size of the sketch to process each update to the underlying data. Sketches are typically a few kilobytes upto a megabyte or so, and processing this much data for every update limits the update speed.

- Sketches are typically constructed using hash functions with strong independence guarantees, such as p-wise independence, which can be complicated to evaluate, especially for a hardware implementation.

- Many sketches described in the literature are good for one single, pre-specified aggregate computation.

- Known analyses of sketches hide large multiplicative constants inside big-Oh notation.

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 simple pairwise independent hash functions; (4) this sketch can be used for several different queries and multiple applications; and (5) all the constants are made explicit and are small. Our constructions strictly improve the space or time bounds of previous results for the applications we discuss.

One of the technical nuances in our result is that 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. Another technical nuance is that 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 [18] for estimating L_2 norms. But this requires $\Omega(\frac{1}{\varepsilon^2 \log \frac{1}{\varepsilon}})$ dimensions if one wants accurate estimation of the L_2 norm [20]. Consequently, all data stream algorithms that rely on such methods 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, and suffices for data stream applications. Finally, from a technical point of view, CM Sketch and its analyses are quite simple, so we expect it to be highly practical, perhaps even as a hardware implementation. We also expect CM Sketch to find other applications.

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.

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. 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 tth update is (i_t, c_t), meaning that

$$\begin{aligned} a_{i_t}(t) &= a_{i_t}(t-1) + c_t \\ a_{i'}(t) &= a_{i'}(t-1) \qquad\quad i' \neq i_t \end{aligned}$$

In some cases, c_ts will be strictly positive, meaning that entries only increase; in other cases, c_ts are allowed to be negative also. The former is known as the *cash register* case and the latter the *turnstile* case [22]. There are two important variations of the turnstile case to consider: whether a_is 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 *non-negative* case, and the second as the *general* 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 cases 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 series 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 extensively studied. In addition, they are of interest in 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.

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

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

We will assume the RAM model, where each word can store integers in the the range $1 \ldots n$ (the range of item indices) and $||\boldsymbol{a}||_1$; standard word operations take constant time and we count space in terms of number of words and 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, ε and δ, meaning that the error in answering a query is within a factor of ε with probability δ. 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 the surveys [22, 3], tutorials [11], as well as the general literature. Our result improves the previously best known bounds for several problems; we discuss the relationship to the previous work in more detail later.

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, $\ln$.

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

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

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

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 $\forall 1 \leq j \leq d$

$$count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t$$

This procedure is illustrated in Figure 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 [21].

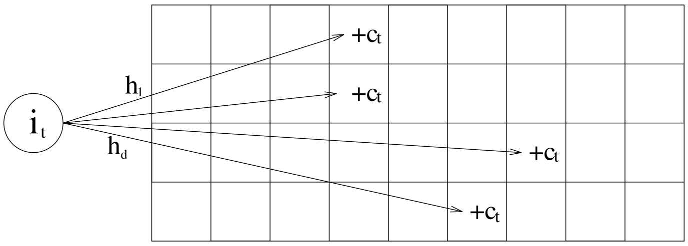

Figure 1: Each item i is mapped to one cell in each row of the array of counts: when an update of c_t to item i_t arrives, c_t is added to each of these cells

4 Approximate Query Answering Using Count-Min 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:* $a_i \leq \hat{a}_i$*; and, with probability* $1 - \delta$*,*

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

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

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

Define the random 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, $\min count[j, h_j(i)] \geq a_i$. For the other direction, observe that

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

by pairwise independence of h_j, and linearity of expectation. Then

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

by the Markov inequality. □

The time to produce the estimate is $O(\ln \frac{1}{\delta})$ since finding the minimum count can be done in linear time; the same time bound holds for updates. The space used is $(2 + \frac{e}{\varepsilon}) \ln \frac{1}{\delta}$ words.

The best known previous result using sketches was in [4]: 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 $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$, and the dependency on ε is always $\frac{1}{\varepsilon^2}$. 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 ε^2. Directly applying the Markov inequality yields a more direct analysis which depends only on ε.
- Because only add positive quantities are added to the counters then it is possible to take the minimum instead of the median for the esimate. This allows a simple calculation of the failure probability, without recourse to Chernoff bounds. This significantly improves the constants involved: in [4] 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 a 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. This time $\mathcal{Q}(i)$ is answered with $\hat{a}_i = \mathrm{median}\, count[j, h_j(i)]$.

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

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

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

Again, the time to produce the estimate is $O(\ln \frac{1}{\delta})$ and the space used is $(2 + \frac{e}{\varepsilon}) \ln \frac{1}{\delta}$ words.

The best prior result for this problem was the method of [4]. Again, the dependence on ε 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 $(\widehat{\boldsymbol{a} \odot \boldsymbol{b}})_j = \sum_{k=1}^{w} count_a[j,k] * count_b[j,k]$. 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. *$\boldsymbol{a} \odot \boldsymbol{b} \leq \widehat{\boldsymbol{a} \odot \boldsymbol{b}}$ and, with probability $1 - \delta$,*

$$\widehat{\boldsymbol{a} \odot \boldsymbol{b}} \leq \boldsymbol{a} \odot \boldsymbol{b} + \varepsilon ||\boldsymbol{a}||_1 ||\boldsymbol{b}||_1.$$

Proof.

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

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

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

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

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

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 $O(\frac{1}{\varepsilon} \log \frac{1}{\delta})$.*

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; 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 $O(\frac{1}{\varepsilon^2})$, 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 $\boldsymbol{\chi}(l, r)_i = 1 \iff l \le i \le r$, and 0 otherwise. Then $\mathcal{Q}(l, r)$ can straightforwardly be re-posed as $\mathcal{Q}(\boldsymbol{a}, \boldsymbol{\chi}(l, r))$. 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, $r - l$. 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 [15] 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 [16]: a dyadic range is a range of the form $[x2^y + 1 \dots (x + 1)2^y]$ for parameters x and y.

Estimation Procedure. Keep $\log_2 n$ CM sketches, in order to answer range queries $\mathcal{Q}(l, r)$ 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 $[1 \dots n]$ is a member of $\log_2 n$ dyadic ranges, one for each y in the range $0 \dots \log_2(n) - 1$. 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 $\mathcal{Q}(l, r)$, 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. Let $a[l, r] = \sum_{i=l}^{r} a_i$ be the answer to the query $\mathcal{Q}(l, r)$ and let $\hat{a}[l, r]$ be the estimate using the procedure above.

Theorem 4. $a[, r] \le \hat{a}[l, r]$ *and with probability at least* $1 - 2\delta \log n$*,*

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

Proof. Applying the inequality of Theorem 1, then $a[, r] \le \hat{a}[l, r]$. By the error guarantees, each answer is correct up to additive $\varepsilon||\boldsymbol{a}||_1$ with probability $1 - \delta$, so by the union bound, all of them simultaneously succeed with probability at least $1 - 2\delta \log_2(n)$. □

The time to compute the estimate or to make an update is $O(\log(n) \log \frac{1}{\delta})$. Space used is $O(\frac{\log(n)}{e} \log \frac{1}{\delta})$.

The above theorem states that the bound for the standard CM sketch size. Typically, the guarantee will be more useful when stated without terms of $\log n$. This can be changed by increasing the size of the sketch, which is equivalent to rescaling ε and δ. In particular, if we want to estimate a range sum correct up to $\varepsilon'||\boldsymbol{a}||_1$ with probability $1-\delta$ then set $\varepsilon = \frac{\varepsilon'}{2\log n}$ and $\delta = \frac{\delta'}{2\log n}$. The the space used is $O(\frac{\log(n)}{\varepsilon'} \log \frac{\log n}{\delta'})$. An obvious improvement of this technique is to keep exact counts for the first few levels of the hierarchy, where there are only a small number of dyadic ranges. This clearly improves the space, time and accuracy of the algorithm. For smaller ranges, ranges that are powers of 2, or more generally, any range whose size can be expressed in binary using a small number of 1s, then improved bounds are possible; we have given the worst case bounds above.

The best previous bounds for this problem are given in [16], where range queries are answered by keeping $O(\log n)$ sketches, each of size $O(\frac{1}{\varepsilon^2} \log \frac{n}{\delta})$ to give the same guarantees as our construction. 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 4.

5.1 Quantiles

In [16] the authors showed that finding the approximate ϕ-quantiles of the data subject to insertions and deletions can be reduced to the problem of computing range sums. Quite 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 $1 \leq k \leq \frac{1}{\phi} - 1$. The method of [16] 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. *ε-approximate ϕ-quantiles can be found with probability at least $1-\delta$ by keeping a data structure with space $O(\frac{1}{\varepsilon}\log^2(n)\log(\frac{\log n}{\phi\delta}))$. The time for each insert or delete operation is $O(\log(n)\log(\frac{\log n}{\phi\delta}))$, and the time to find each quantile on demand is $O(\log(n)\log(\frac{\log n}{\phi\delta}))$.*

Choosing CM sketches over Random Subset Sums improves both the query time and the update time from $O(\frac{1}{\varepsilon^2}\log^2 n \log \frac{\log n}{\varepsilon\delta})$, by a factor of more than $\frac{34}{\varepsilon^2}\log n$. The space requirements are also improved significantly.

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., $a_i(t) \geq 0$.).

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 [4]. 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 $O(\frac{1}{\varepsilon}\log\frac{||\boldsymbol{a}||_1}{\delta})$, and time $O(\log\frac{||\boldsymbol{a}||_1}{\delta})$ per item. Every item which occurs more than ϕ of the 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 δ 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. □

Note that this problem can be solved deterministically based on previous work, using a comparable amount of space, $O(\frac{1}{\varepsilon}\log\frac{||\boldsymbol{a}||_1}{\varepsilon})$ [19]. However, the time cost of processing each insertion is $O(\log\frac{||\boldsymbol{a}||_1}{\varepsilon})$ *amortized* time: periodically, there are operations with cost linear in the space used. For high speed applications, a worst case time bound is preferable. The time bound of our approach can be further improved in a hardware implementation, by computing the hash functions in parallel. Used in this fashion the method is similar to the parallel multistage filters of Estan and Varghese [8], so we do not discuss other optimizations further, but instead refer the interested reader to that paper. There, the method sets a fixed numeric threshold in advance, compared to this approach which expresses the threshold as a fraction.

Turnstile Case. We adopt the solution given in [6], 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 threshhold 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 ϕ. This is achieved by setting the probability of failure for each sketch to be $\frac{\delta\phi}{2\log n}$. This is because, at each level there are at most $1/\phi$ items with frequency more than ϕ. At most twice this number of queries are made at each level, for all of the $\log n$ levels. By scaling δ 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 ε is bounded by δ, and so the probability that every query succeeds is $1-\delta$. It follows that

Theorem 7. *The algorithm uses space $O(\frac{1}{\varepsilon}\log(n)\log\left(\frac{2\log(n)}{\delta\phi}\right))$, and time $O(\log n \log\left(\frac{2\log n}{\delta\phi}\right))$ 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.*

In [6], this algorithm was considered with the range sum query being approximated using a variety of sketch mechanisms. The best known previous bound using sketches is $O(\frac{1}{\phi\varepsilon^2}\log(n)\log(\frac{\log n}{\delta\phi}))$. Here, the space and time bounds have been improved; a further improvement is in the nature of the guarantee. Previous sketch 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.

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 jth entry of the sketch, which is a vector of length $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$, 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, 15] 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 [15], 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 $\mathcal{Q}(l,r)$: 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 worse: $\varepsilon(r-l+1)||\boldsymbol{a}||_1$.

For point queries only, then pairwise independence suffices for tug-of-war sketches, as observed by [4]. 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 $O(\log\frac{1}{\delta})$ for updating the sketch[1]. "Random subset sums" were introduced in [16] in order to compute point queries and range sums. Here, 2-universal hash functions[2] h_j map items to $\{0,1\}$, and the jth 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 Figure 2.

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

[2] In [16] the authors use a 3-wise independent hash function onto $\{0,1\}$, chosen because it is easy to compute, but pairwise suffices.

Method	Query	Space	Update Time	Query Time	Randomness
Tug-of-war [1]	Inner-product	$1/\varepsilon^2$	$1/\varepsilon^2$	$1/\varepsilon^2$	4-wise
Tug-of-war [15]	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
Count sketches [4]	Point	$1/\varepsilon^2$	1	1	Pairwise
Random subset-sums [16]	Range	$\log(n)/\varepsilon^2$	$\log(n)/\varepsilon^2$	$\log(n)/\varepsilon^2$	Pairwise
Count-Min sketches	Point	$1/\varepsilon$	1	1	Pairwise
	Inner-product	$1/\varepsilon$	1	$1/\varepsilon$	Pairwise
	Range	$\log(n)/\varepsilon$	1	$\log(n)/\varepsilon$	Pairwise

Figure 2: Comparison of different space and time requirements of sketching methods: the dependency on ε and n is shown, a factor of $O(\log \frac{1}{\delta})$ applies to each entry and is omitted.

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 $h_1 \dots h_d$ be pairwise independent hash functions mapping from $\{1 \dots n\}$ to $\{1 \dots w\}$, and let $g_1 \dots g_d$ 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

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

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 $w = 1, d = O(\frac{1}{\varepsilon^2} \log(\frac{1}{\delta}))$, $g(i)$ is $\{+1, -1\}$ with 4-wise independence.
- Count sketches have $w = O(\frac{1}{\varepsilon^2}), d = O(\log(\frac{1}{\delta}))$, $g(i)$ is $\{+1, -1\}$ with 2-wise independence.
- Random Subset sums have $w = 2, d = \frac{24}{\varepsilon^2} \log(\frac{1}{\delta})$, $g(i)$ is 1.
- Count-Min sketches have $w = \frac{e}{\varepsilon}, d = \ln(\frac{1}{\delta})$, $g(i) = 1$.

We briefly mention some other well-known sketch constructions, and explain why they are not appropriate for the queries we study here. [9] 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 [17] 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 the parallel multistage filters of Estan and Varghese [8]. Our construction differs for the following reasons: (1) their structure is not a sketch: it does not approximate any value, but instead returns only a binary answer about whether an item has exceeded a certain numeric threshold; (2) the algorithms used requires 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.

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 the sketch to the problem of change detection on data streams, which will be reported in future work, 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 [14].

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, and 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)*, pages 10–20, 1999.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996. Journal version in *JCSS: Journal of Computer and System Sciences*, 58:137–147, 1999.

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

[4] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Procedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 693–703, 2002.

[5] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using Hamming norms. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 335–345, 2002. Journal version in *IEEE Transactions on Knowledge and Data Engineering*, pages 529–540, 2003.

[6] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *Proceedings of ACM Principles of Database Systems*, 2003.

[7] A. Dobra, M. Garofalakis, J. E. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *Proceedings of the 2002 ACM Sigmod International Conference on Management of Data*, pages 61–72, 2002.

[8] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM-02)*, volume 32, 4 of *Computer Communication Review*, pages 323–338, 2002.

[9] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L_1-difference algorithm for massive data streams. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 501–511, 1999.

[10] P. Flajolet and G. N. Martin. Probabilistic counting. In *24th Annual Symposium on Foundations of Computer Science*, pages 76–82, 1983. Journal version appeared in *Journal of Computer and System Sciences*, 31:182–209, 1985.

[11] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. In *Proceedings of the*

ACM SIGMOD International Conference on Management of Data, `http://www.cs.ust.hk/vldb2002/program-info/tutorial-slides/T5garofalalis.pdf`, 2002.

[12] P. Gibbons and Y. Matias. Synopsis structures for massive data sets. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, A, 1999.

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

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

[15] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 79–88, 2001.

[16] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 454–465, 2002.

[17] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, pages 189–197, 2000.

[18] W.B. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[19] G.S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 346–357, 2002.

[20] J. Matoušek. *Lectures on Discrete Geometry*, chapter 15. Springer, 2002. Embedding finite metric spaces into Euclidean spaces.

[21] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[22] S. Muthukrishnan. Data streams: Algorithms and applications. In *ACM-SIAM Symposium on Discrete Algorithms*, `http://athos.rutgers.edu/~muthu/stream-1-1.ps`, 2003.

[23] N. Thaper, P. Indyk, S. Guha, and N. Koudas. Dynamic multidimensional histograms. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 359–366, 2002.