Maintaining Statistical Summaries over Dynamic Data

by

Yuan QIU

Department of Computer Science and Engineering
Hong Kong University of Science and Technology

Supervised by Prof. Ke YI

March 2020, Hong Kong

Contents

List of Tables

Maintaining Statistical Summaries over Dynamic Data

Yuan QIU

Department of Computer Science and Engineering

Abstract

Since the introduction of Morris Counter in 1977, decades of research have been devoted to summary maintenance over dynamic data. For many problems, efficient summaries have been proposed that occupy small space while providing strong accuracy guarantees. The most important ones are statistical summaries for distinct count, frequency estimation, heavy hitter and quantile problems. They are discussed under various models, including cash register streams, turnstile streams, sliding windows and distributed streams. While the streaming context is almost well-understood by matching bounds for many problems, new directions arise in applications of summaries and their ideas. One of them is differential privacy, which guarantees the privacy of any user is not compromised by any post-processing of outputs. Several summaries have been applied or extended to work under privacy constraints. In this survey, we review the literature of maintaining statistical information over dynamic data, and propose possible directions for future research.

Chapter 1

Introduction

With the ever-growing volume of data, extracting information from the whole dataset has become an expensive or even infeasible job. As a solution, one can maintain a summary of the data using small space, usually irrelevant to the data size, to provide accurate estimates of frequently queried information. Of particular importance are summaries for *statistical* information, which include distinct count, frequencies, heavy hitters and quantiles. They are the cornerstones of describing the dataset with numerous applications. Extra attention has been paid to statistical summaries recently due to their communication-saving nature that works well in distributed environments.

A central challenge of summary design is maintenance through updates. This is straightforward: If the data were static, statistical information can be stored itself without the summary. Since the introduction of the Morris Counter [1], summary maintenance has been studied for decades. Various models emerge. There are two orthogonal directions. On the one hand, data can be centralized or distributed among sites. On the other, the updates through time can contain insertions and/or deletions, or we may be interested in only the recently arrived elements. They outline the broad picture of the streaming literature about summary maintenance. In this survey, we focus on statistical summaries, which are designed to estimate corresponding statistics like distinct count, frequencies, heavy hitters and quantiles.

A related broad topic is sample maintenance. A uniform (or weighted) sample drawn from the dataset can also utilize estimation of statistical information. But there are several constraints to sampling techniques: Firstly, the accuracy can be low for certain queries like distinct count. Secondly, the sample size can be proportional to the data size if a constant sampling rate is chosen, which can cause a large overhead. The concept is important though as many summaries use the idea of sampling.

The idea of statistical summaries can also be used in combination with other topics, for example, machine learning [2], security [3] and privacy [4].

The survery is organized as follows. Chapter 2 introduces background information and defines relevant terms. The distinct count problem is discussed in Chapter 3, frequency estimation and heavy hitters are discussed in Chapter 4, and quantile problems are discussed in Chapter 5. For each problem, we discuss about centralized solutions for cash register and turnstile streams, and for sliding windows. We also include solutions for distributed streams. For analysis, we focus on space-error trade-off for centralized algorithms and communication-error trade-off for distributed algorithms. Properties like update time and query time may not be compared. In Chapter 6, we conclude by giving open problems.

Chapter 2

Preliminaries

In the centralized, static setting, we assume there is a multiset A of elements from a universe U. For simplicity we assume $U = [u] = \{1, \ldots, u\}$ unless otherwise specified. It is sometimes easier to represent A by its frequency vector $\boldsymbol{x} = (x_1, \ldots, x_u)$ where x_i is the number of occurrences of i in A. The target is to estimate $y = f(\boldsymbol{x})$ for some function f, where f can be a numerical function like count-distinct, or a vector-valued function like top-k-values.

For numerical functions, we say a deterministic algorithm produces an ϵ-approximation $\hat{y}$ of y iff $|\hat{y} - y| < \epsilon y$. A randomized algorithm produces an (ϵ, δ)-approximation $\hat{y}$ of y iff $\Pr[|\hat{y} - y| > \epsilon y] < \delta$. It usually suffices to consider a constant δ as independent repetitions decrease the fail probability exponentially. Accordingly, we abbreviate it to ϵ-approximation as well.

In practice, the multiset A can be dynamic and/or distributed, which requires introducing data stream models and distributed models.

2.1 Data Stream Models

We may assume the multiset A is initially empty, and updated through a data stream $S = s_1, \ldots, s_n$ of length n. Each item s_i at timestamp i is a pair (a, c), which updates the frequency of a from x_a to $x_a + c$. There are two distinct models of data streams [5, 6]. In the *Cash Register* Model, c is always positive, *i.e.* all the updates are insertions. In the *Turnstile* Model, deletions are allowed so c can be negative. In this survey, we adopt the definition of *strict* turnstile model, which requires frequencies x_i to be non-negative at any time. Under this constraint, an item can only be deleted if it was inserted before. It is usually easier to consider $c \in \{\pm 1\}$, where insertions and deletions are performed one-by-one.

A special case of the turnstile model is the sliding window model. In sliding windows, we are always interested in the latest elements. In a count-based window of size w, the active elements are $A_t = \{s_{t-w+1}, \ldots, s_t\}$ at time t.[1] The difficulty of turnstile model and sliding window model are generally incomparable. For turnstile streams, arbitrary deletions are allowed, giving the adversary much power to generate bad inputs. For sliding windows, deletions are not explicitly given, so the algorithm needs to "remember" the lifespan of elements and automatically delete expired ones.

Standard applications only require an output to be reported at the end of processing the entire stream. In the *strong tracking* (or *continuous tracking*) model, an algorithm has to report an (ϵ, δ)-approximation after *every* update.

[1] Here we assume $c = 1$.

2.2 Distributed Stream Model

We consider the *distributed function monitoring* model [7, 8]. There are k remote sites and one coordinator. Two-way communication exists between the coordinator and each remote site, but sites are not allowed to communicate within themselves.[2] Each remote site i receives a stream S_i that updates its underlying multiset A_i, whose frequency vector is $\boldsymbol{x}_i$. We assume S_i is an insertion-only stream unless otherwise specified. We use n_i to denote the *current* length of S_i, which changes over time. n denotes the total number of items $\sum_i n_i$. We use N_i and N to denote upper bounds on n_i and n respectively. Bounds will be represented in N in distributed models. The target is to estimate a function f on the union of all streams $\biguplus_i A_i$,[3] while minimizing the messages communicated.

A closely related problem of *value tracking* is the *threshold monitoring* problem, which sets a threshold τ. The algorithm outputs 1 if $f(\biguplus_i A_i) \geq \tau$ and 0 if $f(\biguplus_i A_i) \leq (1-\epsilon)\tau$. A tracking algorithm naturally solves this problem, while a solution to the threshold monitoring problem indicates a solution to the tracking problem by setting $\tau = 1, (1+\epsilon), (1+\epsilon)^2, \ldots, T$ for some upper bound T on $f(\biguplus_i A_i)$. This requires creating $O(\log_{(1+\epsilon)} T) = O(\epsilon^{-1} \log T)$ independent repetitions.

Round-based algorithms are usually more attractive in this model, since it better suits industrial implementations like Flink [9]. In a round-based algorithm, the coordinator begins a round by sending messages to each site based on the history. During the round, each site reads the stream and sends messages from time to time to the coordinator. At some point, the coordinator decides to end the current round. After collecting the final message from all sites, each site resets itself and for the new round to begin.

2.3 Inequalities

We introduce useful inequalities here to help the analyses in following chapters.

2.3.1 Markov's Inequality

Let Y be a random variable whose expectation is $\mathbf{E}[Y]$ Markov's inequality says that

$$\Pr\left[Y > \frac{\mathbf{E}[Y]}{\delta}\right] < \delta$$

We use it to bound low expectation error terms.

2.3.2 Chebyshev's Inequality

We use Chebyshev's inequality to bound the error of unbiased estimators. Let $\hat{y}$ be an unbiased estimator of y, By Chebyshev's inequality we have

$$\Pr[|\hat{y} - y| > \epsilon y] < \frac{\mathbf{Var}[\hat{y}]}{\epsilon^2 y^2}.$$

If we can bound the variance by $\mathbf{Var}[\hat{y}] \leq \epsilon^2 y^2 \delta$, $\hat{y}$ will be an (ϵ, δ)-approximation of y. Alternatively, we can bound the variance by $\epsilon^2 y^2/2$, and take the median over $\log \delta^{-1}$ independent estimators. This is sometimes space-saving. Generally, we consider δ to be a constant and bound the variance by $O(\epsilon^2 y^2)$.

[2] They may still communicate through the coordinator, which requires 2 messages.

[3] $\biguplus$ adds up frequencies of multisets

2.3.3 Additive Chernoff-Hoeffding bound

Let $y_1, \ldots y_n$ be *independent* random variables bounded by $a_i \le y_i \le b_i$, and $Y = \sum_{i=1}^{n} y_i$, then

$$\Pr\left[|Y - \mathbf{E}[Y]| > k\right] \le 2\exp\left(\frac{-2k^2}{\sum_i (b_i - a_i)^2}\right).$$

Specifically, if the bound $-b \le y_i \le b$ holds for all i, we will have

$$\Pr\left[|Y - \mathbf{E}[Y]| > 2b\sqrt{n}\right] \le 2\exp\left(\frac{-8b^2 n}{4b^2 n}\right) = \frac{2}{e^2}.$$

So the error only grows by a $\sqrt{n}$ factor if we sum n zero-mean error terms.

2.4 Notations

Table 2.1 summarizes notations used in this survey.

Table 2.1: Summary of Notations.

Notation	Meaning
u	Size of the universe (domain) of inputs.
$[u], U$	The input domain, $[u] = \{1, \ldots, u\}$.
$\boldsymbol{x}$	The (total) frequency vector $(x_1, \ldots, x_u)$.
$\boldsymbol{x}_i$	The frequency vector of site i in distributed model.
n	(current) Length of the stream.
n_i	Current length of the stream at site i.
w	Window size in sliding window model.
$N, (N_i)$	Total length of the stream (at site i) in distributed streams.
k	Number of sites in distributed model.
$\|\boldsymbol{x}\|_1$	ℓ_1-norm, $\|\boldsymbol{x}\|_1 = \sum_i x_i$, abbreviated to n in cash register.
$\|\boldsymbol{x}\|_0$	ℓ_0-norm, $\|\boldsymbol{x}\|_0 = \|\{i : x_i > 0\}\|$, abbreviated to D.
ϵ	Error factor for additive error $\epsilon D, \epsilon w, \epsilon n$.
δ	Fail Probability.

Chapter 3

Distinct Count

In this chapter, we discuss the problem of distinct count estimation. In the literature, it is also referred to as the distinct elements problem, (distinct) cardinality estimation, computing the Hamming norm (*a.k.a.* ℓ^0 norm, 0-th frequency moment) *etc.* The target is to estimate D, the number of distinct elements in multiset A. Given frequency vector $\boldsymbol{x}$, $D = \|\boldsymbol{x}\|_0 = |\{i : x_i \neq 0\}|$. This problem is fundamental with applications to network monitoring [10], query optimization, OLAP and graph theory [11].

In general, there are two main types of approach [12] to the distinct counting problem: sampling-based [13, 14, 15, 16, 17, 18, 19, 20, 21, 22] and synopsis-based [23, 24, 25, 26, 27, 28, 29, 30, 31, 32]. We focus on synopsis-based solutions as most sampling based solutions are inaccurate [20].

3.1 Cash Register Model

First note there is a trivial upper bound of $O(\min\{u, n \log u\})$. One may either store every element in the stream using $O(n \log u)$ bits, or use a bit for every element in $[u]$ to record whether it appears ($O(u)$). Throughout this section we assume $\epsilon = \Omega(1/\sqrt{u})$, otherwise the trivial algorithm is already space-optimal.

Alon, Matias and Szegedy [33] showed that any *deterministic* algorithm that computes a constant approximation of D must use $\Omega(u)$ bits of memory. They also showed an lower bound of $\Omega(\log u)$ bits to compute a $(0.1, \frac{1}{4})$-approximation of D, which matches the upper bound of $O(\log u)$ in [23] for constant ϵ. Bar-Yossef [34] included ϵ as a parameter, and showed for $6 \leq \epsilon^{-1} \leq n$ a lower bound of $\Omega(\epsilon^{-1})$. Indyk and Woodruff [35] improved it to $\Omega(\epsilon^{-2})$ when $\epsilon^{-1} = o(u^{\frac{1}{9}})$. Woodruff [36] further relaxed the constraint to $\epsilon^{-1} = O(\sqrt{u})$. This is the best possible constraint for the $\Omega(\epsilon^{-2})$ lower bound: otherwise the $O(u)$ upper bound is optimal.

The lower bound was proved by a reduction to the Gap-Hamming problem. The Hamming distance between two n-bit strings x and y is defined as $H(\boldsymbol{x}, \boldsymbol{y}) = |\{i : x_i \neq y_i\}|$. The problem is to distinguish between $H(\boldsymbol{x}, \boldsymbol{y}) \leq n/2 - \sqrt{n}$ and $H(\boldsymbol{x}, \boldsymbol{y}) \geq n/2 + \sqrt{n}$. There is a lower bound [36, 37] that $\Omega(n)$ communication is needed when the two vectors are held by Alice and Bob respectively. The reduction works as follows: Alice encodes $\boldsymbol{x}$ into n tuples, where x_i is mapped to (i, x_i). She then runs the ϵ-approximation algorithm on the tuples, and transmit the states to Bob. Bob encodes $\boldsymbol{y}$ similarly, and feed them into the same algorithm initialized by states received from Alice. The full input to the algorithm contains $2n$ tuples, where $n - H(\boldsymbol{x}, \boldsymbol{y})$ are duplicated ones, meaning the real distinct count is $n + H(\boldsymbol{x}, \boldsymbol{y})$ for Bob's merged streams. In one case, $D \leq 3n/2 - \sqrt{n}$ while in the other $D \geq 3n/2 + \sqrt{n}$. Setting $\epsilon D = \Theta(\sqrt{n})$, *i.e.* $\epsilon = \Theta(1/\sqrt{n})$ suffices to solve the Gap-Hamming problem, which means the space consumption must be $\Omega(n) = \Omega(\epsilon^{-2})$.

Woodruff [38] showed that for a wide range of parameters, this lower bound still holds even in

the average case: when items are independently randomly chosen from an unknown distribution, instead of being provided by an adversary. Kane, Nelson and Woodruff [26] matched this lower bound by providing an algorithm using $O(\epsilon^{-2} + \log u)$ bits of memory with $O(1)$ update time. Taking δ into account, this implies an algorithm of $O((\epsilon^{-2} + \log u) \log \delta^{-1})$ space complexity. Jayram and Woodruff [39] showed the lower bound to be $\Omega(\epsilon^{-2} \log \delta^{-1} + \log u)$, namely the dependency of $\log \delta^{-1}$ does not multiply $\log u$. This was matched by Blasiok [40, 41].

In strong tracking, the algorithm is required to report after every update. A standard technique [42] is to run $O(\log n)$ independent trials to reduce the fail probability to $O(n^{-1})$, then use union bound to guarantee a constant success probability. Blasiok [40, 41] showed that $O(\epsilon^{-2}(\log \delta^{-1} + \log \log u) + \log u)$ is a tight upper bound to the strong tracking problem.

3.1.1 Probabilistic Counting (FM-sketch)

In 1983, Flajolet and Martin [23] introduced the probabilistic counting algorithm, also known as the FM-sketch. It is often regarded as the first summary for the distinct counting problem. The core is to remove duplicates through hashing. The sketch is a $BITMAP$ of length L associated with a (perfect) hash function $h : [u] \to \{0,1\}^L$ that *uniformly* hashes elements in the universe to L-bit binary strings. Denote by $h(a) = b_0 b_1 \dots b_{L-1}$ the hash value of $a \in [u]$. Uniform here means the occurrences of 0's and 1's are the same at each digit b_i, over the image of the function. The FM-sketch computes the position of leading 1's of all hash values, defined as[1]

$$\rho(h(a)) = \min\{k : b_k = 1\},$$

and stores them in the BITMAP. It can be observed that repetitions of a value does not affect the BITMAP, as their hash values are the same. Since h is uniform, $\rho(h(a))$ follows a geometric distribution[2] where $\Pr[\rho(h(a)) = k] = 2^{-k-1}$. Expected number of values hashed to position k is therefore $2^{-k-1}D$. Intuitively, we expect the beginning of the BITMAP to be 1's, and the ending to be 0's. In the middle we expect to see a mixture of 0's and 1's, which happens when $2^{-k-1}D$ is a constant, *i.e.* $k = \Theta(\log D)$. Let R denote the position of leading 0 in the BITMAP, defined as

$$R = \min\{k : BITMAP[k] = 0\}.$$

Analysis shows that $E[R] \approx \log_2 \varphi D$, where $\varphi = 0.77351\dots$, and its standard deviation $\sigma(R) \approx 1.12$. An immediate estimator (MLE) for D is therefore $\hat{D}_{PC} = 2^R/\varphi$. To achieve an ϵ-approximation, the algorithm uses a stochastic averaging technique by creating $O(\epsilon^{-2})$ BITMAPs. Instead of independent repetitions of the algorithm, the values are randomly partitioned and estimated seperately. This helps reduce the number of BITMAPs to be updated for an insertion from $O(\epsilon^{-2})$ to $O(1)$. As the algorithm uses $O(\epsilon^{-2})$ L-bit words, its space consumption is $O(\epsilon^{-2}L) = O(\epsilon^{-2} \log u)$, optimal for constant ϵ.

There are various extensions to the Probabilistic Counting algorithm as mentioned. [33] relaxed the assumptions on the hash function by showing that a linear hash function suffices. [43, 10] used linear space, but their algorithm manages achieve a higher accuracy in practice using the same actual space [44]. [45] combined it with sampling to further reduce space at the cost of accuracy. Among all, the HyperLogLog algorithm [25] has received the most attention.

3.1.2 (Hyper)LogLog

Originating from LogLog [24], HyperLogLog [25, 27] is now the preferred summary for distinct count implemented in many real database systems, including Redis [46], Redshift [47], Spark

[1] We ignore the fact that $\rho(h(a))$ is bounded by L.

[2] One may consider $\rho(h(a))$ as a new hash function $g : [u] \to \{0, \dots, L-1\}$ such that $\Pr[g(a) = k] = 2^{-k-1}$.

SQL [48] and Presto [49]. Compared to Probabilistic Counting, HyperLogLog improves the estimates to be unbiased. It also uses smaller memory units[3] while achieving the same asymptotic error bound.

The algorithm is similar, use a uniform hash function to hash values into binary strings. But instead of maintaining the *ALL* positions of leading ones by *BITMAP*, HyperLogLog only tracks the *MAX* position of leading ones $M = \max_a \rho(h(a))$, which explains the $\log\log$ space cost. As analyzed before, each $\rho(h(a))$ follows a geometric distribution. So M is the maximum of D geometrically distributed random variables, where

$$\Pr[M \le k] = \prod \Pr[\rho(h(a)) \le k] = (1 - 2^{-k-1})^D\,, \quad k \ge 0\,.$$

Equivalently,

$$\Pr[M = k] = (1 - 2^{-k-1})^D - (1 - 2^{-k})^D\,, \quad k \ge 1\,.$$

Analysis shows that $\hat{D}_{HLL} = 2^{M+1}$ is a good estimator of D. Intuitively, the pattern $0^M 1\ldots$ occurs with probability 2^{-M-1}. So we expect 2^{M+1} distinct hash values for this event to occur, resulting in a leading one at position M. *Stochastic averaging* is also involved by dividing the input into m buckets and taking harmonic mean of these estimators. Finally, it performs bias reduction and achieves a relative accuracy of $1.04/\sqrt{m}$ when using m counters, equivalently using $O(\epsilon^{-2}\log\log u + \log u)$ space to provide an ϵ-approximation.

Using this algorithm as building block, [26] introduced space counters and applied balls-and-bins analysis to reduce the space consumption to $O(\epsilon^{-2} + \log u)$. But the algorithm is not practical. [40, 41] further improved the dependency on δ, giving a tight upper bound of $O(\epsilon^{-2}\log\delta^{-1} + \log u)$, in terms of all u, ϵ and δ.

3.1.3 KMV and MinCount

HyperLogLog-like algorithms are based on *bit-pattern observables* [25], where the input is hashed to binary strings and their beginning bit patterns $0^\rho 1\ldots$ are observerd. Another broad category is based on *order statistics observables*, namely treating the binary strings as real values in $[0, 1]$, and observe the smallest value(s). The k Minimum Values synopsis (KMV) [28, 29, 30] and MinCount summary [31, 32] are typical examples.

Assuming a *uniform* hash function $h : [u] \to [0, 1)$. Hashing here is only a constraint that the same values are *hashed* to the same number.[4] the D distinct hash values can be viewed as D random numbers, taken independently uniformly at random from $(0, 1]$. Consider the smallest hash value v_1. Apparently, v_1 gets smaller when D is larger. More precisely, we have

$$\Pr[v_1 \le x] = 1 - (1 - x)^D\,,$$

so $E[v_1] = 1/(D+1)$. An immediate estimator (MLE) for D is $v_1^{-1} - 1$. However, its expectation is infinite as v_1 can be arbitrarily close to 0. In [29, 30], this is solved by using the k-th smallest hash value v_k instead of the minimum one. Their analysis shows that $\hat{D}_{KMV} = (k-1)/v_k$ is an unbiased estimator of D, and its relative error is $O(1/\sqrt{k})$. Since k words are needed to store the minimum hash values, the space consumption is $O(k\log u) = O(\epsilon^{-2}\log u)$.[5] Maintaining the smallest hash values by a min-heap, each update will cost $O(\log k) = O(\log\epsilon^{-1})$. [31, 32] solve the divergence of inverse by composing it with a sublinear function like square root, and achieve similar results.

[3] FM-sketch uses words of $O(\log u)$ bits, while HyperLogLog uses bytes of $O(\log\log u)$ bits.

[4] By definition the image of a hash function is discrete. In practice we use $g : [u] \to [1, u^3]$ to hash the values and take $h(a) = g(a)/u^3$ as the hash value to avoid collisions. Ignoring it here makes the description simpler.

[5] This is not tight. $O(\log\epsilon^{-1} + \log\log u)$ bits suffices for the hash values.

3.1.4 Coordinated (Distinct) Sampling

Gibbons [50, 21] proposed the coordinated (distinct) sampling algorithm to solve the poor accuracy in answering distinct count queries using uniform samples. Similar ideas appear in [51, 52]. In brief, the algorithm samples each *distinct* value with probability p for some appropriate p, and scale the sample size by p^{-1} to obtain an estimator of D. Limited to distinct counting itself, the algorithm works similar to KMV, except for that it reduces the number of sample values by half[6] wherever the budget is exceeded, so that there are always $O(\epsilon^{-2})$ samples kept. Whereas in KMV, inserting a new hash value only requires popping *one* largest hash value in the heap. It is worth mentioning that the author considered augmenting the distinct values with tuples to support filter conditions.

3.1.5 S-BITMAP

Chen et al. [53, 54] introduced the S-BITMAP as a follow-up work of [43, 55]. This is a linear-space algorithm whose actual space consumption is

$$\frac{\log(1+2u\epsilon^2)}{\log(1+2\epsilon^2/(1-\epsilon^2))}$$

bits. The algorithm adapts its sampling probability according to the current distinct count. Consider a $BITMAP$ of length L, and let S track the sum of bits in the BITMAP, initialized to 0. To insert an element a, first *uniformly* hash a to a bucket $h(a) \in \{0, \ldots, L-1\}$. If $BITMAP[h(a)] = 1$, skip to the next element. Otherwise, update the bit to 1 with probability p_S, depending on the current S. Increase S by 1 if the bit is updated. This sampling procedure also uses a as a key, so that repetitions are irrelevant. Intuitively, S is larger when the distinct count D is larger. Also p_S should be decreasing with respect to S to avoid filling the BITMAP too quickly. It can be verified that a new distinct value will update S by 1 with probability

$$q_S = \frac{L-S}{L} p_S \,.$$

Assume the distinct values are fed to the algorithm in the order $1, 2, \ldots, D$. Let T_S be the index of the distinct element that increased S to $S+1$, the number of distinct values between T_{S+1} and T_S follows a geometric distribution, that is

$$\Pr[T_{S+1} - T_S = t] = (1-q_S)^{t-1} q_S \,.$$

A natural estimator is then $\hat{D} = \mathbf{E}[T_S]$ at the last state of the bitmap. By analyzing this Markov chain, the authors derived that assigning

$$p_S = \frac{L}{L+1-S}(1+\epsilon^2) r^S \,, \quad \text{where } r = \frac{1-\epsilon^2}{1+\epsilon^2}$$

makes the estimator unbiased with root mean square error $\sqrt{\mathbf{E}(\hat{D}/D-1)^2} = \epsilon$. This is not related to D, improving algorithms like HyperLogLog. [56] extended the idea of using a Markov chain and provided a practical algorithm with space complexity $O(\epsilon^{-2}\log\epsilon^{-1} + \log u)$.

3.2 Turnstile Model and Sliding Windows

For turnstile model, Kane, Nelson and Woodruff showed in the appendix of [57] an lower bound of $\Omega(\epsilon^{-2}\log\epsilon^2 n)$. They nearly matched in [26] by an algorithm using $O(\epsilon^{-2}\log n(\log\epsilon^{-1} + \log\log n))$ bits.

[6] More precisely, reduce the sample probability p by half.

For sliding windows, Datar el al. [58] showed an lower bound of $\Omega(w+\epsilon^{-1}\log^2 w)$. Braverman et al. [59] improved it together with the $\Omega(\epsilon^{-2})$ lower bound from cash register model to $\Omega(\epsilon^{-1}\log^2 w + \epsilon^{-2}\log w)$ for $\epsilon = O(1/\sqrt{w})$. They also provided a nearly optimal upper bound of $O(\epsilon^{-1}\log^2 w + \epsilon^{-2}\log\epsilon^{-1}\log w\log\log w)$. Assaf et al. [60] introduced a Sliding Bloom Filter of size $O(w\log w)$ to work for distinct count and frequency counting simultaneously, when $\epsilon = w^{-o(1)}$.

There is also work [61] using an intermediate model: If number of distinct items at the end of the stream is at least α^{-1} of the number of distinct items ever appeared during the stream, the stream satisfies ℓ_0-α property. An cash-register stream has $\alpha = 1$ and a turnstile stream is not lower bounded. In this case, there is an upper bound of $O(\log n + \epsilon^{-2}\log(\alpha/\epsilon)(\log\epsilon^{-1} + \log\log n))$.

3.2.1 Augmenting FM/HLL/KMV with Counters

In a FM-Sketch-like summary, when we delete a value a whose hash value happens to have the most leading zeros, there are two scenarios. If there are still duplicates of a so that its $x_a \neq 0$ after deletion, the summary needs not to be modified. Otherwise, we should find the maximum position of leading one in remaining hash values. This is costly in the worst case as every deletion can cause a rescan of the whole stream. A straightforward solution is to store for each position the number of hash values having its leading one there. The idea was first mentioned by Flajolet and Martin in their journal version of the paper [62], and stated for similar sketches using exact or approximate counters [63, 64, 65, 66, 67, 68, 22]. Note that the improvement from $\log u$ to $\log\log u$ in (Hyper)LogLog vanishes, as a separate counter must be maintained for each bit. A naive counter uses $O(\log n)$ space to represent the frequencies, meaning a total space consumption of $O(\epsilon^{-2}\log u\log n)$. The $O(\log n)$ factor can be improved using approximate counters at the cost of accuracy guarantee.

For sliding windows, there is no need to maintain a count as we are always interested in the latest elements. Instead, a counter can be replaced by the timestamp of the latest element hashed to each bit. At query time, the summary is filtered to obtain a sketch corresponding to the latest window. The multiplicative factor to space is still $O(\log w)$ for windows of size w, as a timestamp needs $O(\log w)$ bits to represent. This was first mentioned in [58, 69], and restated in [70] as *Sliding MinCount* and in [71] as *Sliding HyperLogLog*.

Similar problems exist for KMV, MinCount and distinct sampling. In KMV, when we delete an a whose hash value $h(a)$ is among the k smallest, we need to check the frequency of a to decide whether $h(a)$ should be removed from the summary. If so, the summary only contains $k-1$ hash values after deletion, which may violate the accuracy guarantee. In the same paper [29], the authors included an augmented KMV synopsis (AKMV) as a solution. Similarly, each hash value is associated with its frequency. Deletion is lazy in the sense that a 0-frequency value is not removed from the summary until it can be replaced by a new one. In other words, the summary always contains k hash values, where only $k' \leq k$ of them have positive frequencies. The error depends now on k', so that a full refresh is required if k' is too small. Similarly, augmenting with frequencies poses an $O(\log n)$ factor to the space consumption. After augmenting frequencies, this problem is simply top-k view maintenance, a well-studied topic. Similar sample recovery techniques are also discussed for distinct sampling [67].

3.2.2 Approximation by Stable Distribution Sketch

[12, 72] introduced an algorithm targeting the turnstile model. Their algorithm allows negative frequencies, where it computes the Hamming Norm. It relies on *stable distributions*. A stable distribution enjoys the property that, if random variables $X_1, \ldots X_l$ have stable distribution

with stability parameter p, then $\sum_i a_i X_i$ is distributed as $(\sum_i |a_i|^p)^{1/p} X_0$, where X_0 is also a random variable with p stable distribution.

The sketch unit is simply $sk(\boldsymbol{x}) = \sum_i c_i x_i$ for frequency vector $\boldsymbol{x}$, where each c_i is drawn independently from a p stable distribution. By the property, $sk(\boldsymbol{x})$ is distributed as $(\sum_i |x_i|^p)^{1/p} X$, for a random variable X chosen from a p stable distribution. This allows us to estimate $\|\boldsymbol{x}\|_p^p = \sum_i |x_i|^p$. When p is small, it is a good approximation of the Hamming Norm. To see it, note $\|\boldsymbol{x}\|_\infty \leq n$, we have (define $0^0 = 0$)

$$\|\boldsymbol{x}\|_0 = \sum_i |x_i|^0 \leq \sum_i |x_i|^p \leq \sum_i n^p |x_i|^0 = n^p \|\boldsymbol{x}\|_0$$

If we set $n^p \leq 1 + \epsilon$, equivalently $p \leq \log(1+\epsilon)/\log n$, $\|f\|_p^p$ will be a $(1+\epsilon)$-approximation of $\|f\|_0$. The accuracy guarantee is achieved by taking the median over $O(\epsilon^{-2})$ independent sketch units, which means a space consumption of $O(\epsilon^{-2} \log^2 n)$ for constant δ [67]. Nevertheless, generating strictly stable distributions is costly. This puts a constraint to this solution.

3.2.3 Nearly Optimal Algorithm on Sliding Windows

Braverman et al. [59] gave a near optimal algorithm for sliding windows using [40] as a subroutine. Their counters are similar to [50, 21], which are of size $O(\epsilon^{-2} \log u)$. As was in [40], $O(\log \log w)$ repetitions are used to reduce the variance. They observed the counters can be maintained incrementally for partial windows $(t_1, t) \supset (t_2, t) \supset \ldots$ at current time t, where $t_1 < t_2 < \cdots < t$. It suffices to use units of size $O(\log \epsilon^{-1} + \log \log w)$ to represent the position of the last partial window containing bit 1, accompanied by the starting timestamp. With encoding applied, the final bound achieved was $O(\epsilon^{-2} \log w \log \epsilon^{-1} \log \log w + \epsilon^{-1} \log^2 w)$. This is optimal up to $\log \epsilon^{-1}$ and $\log \log w$ factors.

3.3 Distributed Streams

The hardness of distinct counting in the distributed setting is that it is cannot be obtained by simple arithmetic operations on local estimates. A lower bound for the threshold monitoring problem also holds for the tracking problem. Cormode, Muthukrishnan and Yi [73, 8] showed an $\Omega(k)$ lower bound when $\epsilon \leq 1/4$ and $u \geq k^2$. The lower bound also applies to one-shot algorithms. In the same paper they presented an algorithm using $O(k(\log u + \epsilon^{-2} \log \epsilon^{-1}))$ bits of communication for monitoring. Arackaparambil, Brody and Chakrabarti [74] showed another lower bound of $\Omega(\epsilon^{-1})$ when $\epsilon \leq 1/2$. They also show that if the algorithm is round-based, the lower bound can be improved to $\Omega(\epsilon^{-2})$. Chakrabarti and Regev [75] removed the constraint and showed the lower bound to $\Omega(\epsilon^{-2})$ applies to all one-pass algorithms. Woodruff and Zhang [76] improved these bounds (*i.e.* $\Omega(k + \epsilon^{-2})$) to $\Omega(k\epsilon^{-2}/\log(k\epsilon^2))$ when $k\epsilon^2 > 1$, and later [77] to $\Omega(k(\epsilon^{-2} + \log N))$.[7] This is a tight lower bound for one-shot algorithms: Since there is a centralized algorithm [26] using $O(\epsilon^{-2} + \log u)$ space, this implies a distributed algorithm using $O(k(\epsilon^{-2} + \log u))$ bits of communication. The sites consecutively run the algorithm on their input and pass the state to the next player.

If the stream allows arbitrary deletions, [74] showed that no good upper bound can be provided: the lower bound is proportional to the length of the stream.

3.3.1 Merging FM/HLL/KMV

FM-sketch, HyperLogLog and KMV are all mergeable summaries [78], provided that the same hash functions are used among all sites. This comes from the associative nature of the max / min

[7] We consider $\epsilon = \Omega(1/\sqrt{u})$.

operator. A trick [79] in merging KMV summaries is to use $\min_j \max_i v_{i,j} = \min_j v_{m,j}$ as a threshold where hash values $\{v_{i,j}\}_{i=1}^m$ are reported by site j. This may keep more values in the merged summary compared to finding the m minimum values among the all hash values reported.

They are desirable in one-shot setting, but merging the summaries for every update is clearly unnecessary and inefficient.

3.3.2 Lazy Update Algorithms

To save communication in the distributed setting, a natural idea is to communicate with the coordinator only when local changes are too large. [80] formalized this by giving a subroutine for monitoring distinct counts over the union of distributed streams. Let S_i be the set of distinct of elements on site i. Each site also maintains an approximate set $\hat{S}_i$, which is synchronized with the coordinator. To report the distinct count, the coordinator calculates $\cup\hat{S}_i$. As long as $|S_i - \hat{S}_i| < \epsilon$, $|\cup_i S_i| - |\cup_i \hat{S}_i| \le |\cup_i S_i - \cup_i \hat{S}_i| \le \sum_i |S_i - \hat{S}_i| \le k\epsilon$ if there are k sites. Similarly, $|\cup_i \hat{S}_i| - |\cup_i S_i| \le k\epsilon$ if $|S_i - \hat{S}_i| < \epsilon$. This implies an intuitive algorithm as follows: Charge $\phi_i^+(e) = 1$ for (inserting) an element $e \in S_i - \hat{S}_i$. Charge $\phi_i^-(e) = 1$ for (deleting the last copy of) an element $e \in \hat{S}_i - S_i$. The other elements are not charged. Whenever $\sum_e \phi_i^+(e) > \epsilon$ or $\sum_e \phi_i^-(e) > \epsilon$, synchronize with the coordinator so that $\hat{S}_i = S_i$. The communication cost is high as a synchronize is needed upon receiving every ϵ items in the worst case. An improvement was made due to the fact that frequent items may appear in multiple streams. For any item $e \in \cup_i \hat{S}_i$, the coordinator computes the number of streams containing it, $C(e) = |\{i : e \in \hat{S}_i\}|$. If $C(e) > \tau$, the item is frequent, and the coordinator also maintains a *lower bound* $\theta(e)$ where $\theta(e) < C(e) < 4\theta(e)$. Now if $\theta(e) > 0$, we can set $\phi_i^+(e) = 0$ since e is already in $\cup_i \hat{S}_i$. Also we can set $\phi_i^-(e) = 1/\theta(e)$ because there are at least $\theta(e)$ repetitions of e. They must all be deleted for e to be removed from the output. This is inefficient in the worst case, if there is no duplicated elements. But the output is a set instead of a count, which can support further operations such as intersections.

Cormode et al. [7] designed a distinct count tracking algorithm using similar heuristics. Each update applies on the local sketch. If the local distinct count estimated is above a threshold, the sketch is synchronized with the coordinator. The algorithm can achieve $O(\epsilon^{-3}k^2 \log u)$ communication.

3.3.3 Distributed Distinct Sampling

Cormode et al. [73, 8] improved the bound to $O(k(\epsilon^{-2} \log \epsilon^{-1} + \log u))$ for the monitoring problem, using ideas from distinct sampling [50, 21]. $O(k \log u)$ communication is needed to transmit the hash functions to each site. Given a parameter τ, the algorithm uses a sampling probability of $p = \Theta(\epsilon^{-2}\tau^{-1})$ to perform distinct sampling. The sampled values will be sent to the coordinator once they appear locally. To save communication, instead of sending the real value using $\log u$ bits, it suffices to send hash values in a domain of size $O(\epsilon^{-4})$, whose values can be represented using $O(\log \epsilon^{-1})$ bits. Collisions only affects the fail probability by a constant. τ distinct values are received before the algorithm terminates, which explains for the communication of $O(\epsilon^{-2} \log \epsilon^{-1})$ on each site It is also clear that the coordinator essentially receives a distinct sample from remote sites (represented by hash values) at sampling rate p, so that an estimator can be obtained by scaling the sample size by p^{-1}. It is obvious that it provides an unbiased estimator of D, where the variance is bounded by D/p. Thus setting $p = \Theta(\epsilon^{-2}\tau^{-1})$ suffices to bound the error by $\epsilon\tau$.

This implies a tracking algorithm. We divide the tracking into rounds. Within each round, τ is always an upper bound of D so the communication is bounded. Whenever the algorithm

monitors $D > \tau$, a new round is started by doubling τ. This guarantees that $\tau < 2D$ always holds, which is necessary to bound the variance. There are only $O(\log D) = O(\log u)$ rounds. Within each round, $O(k/\epsilon^2)$ messages are communicated.[8]

3.4 Remarks

Table 3.1 summarizes the lower bounds and upper bounds on space/communication in different models. Experimental evaluations can be found in [44]. Detailed discussions can be found in [81, 82, 83].

Table 3.1: Lower Bounds and Upper Bounds for Distinct Counting.

	Model	Lower Bound	Upper Bound	Remarks
One-Shot	Centralized	$\Omega(\log u)$ [33]	$O(\log u)$ [23]	Constant ϵ, δ
		$\Omega(\epsilon^{-2} + \log u)$ [36]	$O(\epsilon^{-2} + \log u)$ [26]	Constant δ
		$\Omega(\epsilon^{-2} \log \delta^{-1} + \log u)$ [39]	$O(\epsilon^{-2} \log \delta^{-1} + \log u)$ [41]	Optimal
	Distributed	$\Omega(k(\epsilon^{-2} + \log u))$ [77]	$O(k(\epsilon^{-2} + \log u))$ [26]	Optimal
Tracking	Cash Register	$\Omega(\log u)$ [33]	$O(\log u)$ [23]	Constant ϵ, δ
		$\Omega(\frac{\log \delta^{-1} + \log\log u}{\epsilon^2} + \log u)$ [41]	$O(\frac{\log \delta^{-1} + \log\log u}{\epsilon^2} + \log u)$ [41]	Optimal
	Sliding Window	$\Omega(\frac{\log^2 w}{\epsilon} + \frac{\log w}{\epsilon^2})$ [59]	$O(\frac{\log^2 w}{\epsilon} + \frac{\log w}{\epsilon^2}(\log \epsilon^{-1} \log\log w))$ [59]	Nearly Optimal
			$O(w \log w)$ [60]	$\epsilon = w^{-o(1)}$
	Turnstile	$\Omega(\epsilon^{-2} \log \epsilon^2 u)$ [57]	$O(\epsilon^{-2} \log u(\log \epsilon^{-1} + \log\log n))$ [26]	Nearly Optimal
	Distributed	$\Omega(k(\epsilon^{-2} + \log u))$	$O(k\epsilon^{-2} \log \epsilon^{-1} \log u)$ [8]	

It was shown [74] that solving the distinct count tracking problem under distributed update streams is hard in the worst case. Useful heuristics may be applicable in this setting. For example, [84] separately counted the number of remaining items and the number of deleted items. There is also a gap in *strong tracking* of the distinct count in insertion-only streams.

[8] Hash functions only needs to be sent once.

Chapter 4

Frequency Estimation and Heavy Hitters

In this chapter, we study two closely related problems: Frequency Estimation and Finding Heavy Hitters. We first study the frequency estimation problem in its basic form: point queries. Given a frequency vector $\boldsymbol{x}$, $f(\boldsymbol{x}, i) = x_i$ returns the frequency of item i in the stream. Similar to the distinct count problem, an exact solution is to use $O(u \log n)$ bits to store the histogram.

For this problem, we use a different definition of ϵ-approximation. Requiring $\Pr[|\hat{x}_i - x_i| > \epsilon x_i] < \delta$ is obviously too restrictive: the algorithm must identify **all** zero-frequency items and singletons when $\epsilon < 1$,which is not realistic in small space and usually unnecessary. Instead, we let the error magnitude depend on $\|\boldsymbol{x}\| = \sum_i |x_i|$. We say an algorithm has error $\epsilon\|\boldsymbol{x}\|_1$ if for all i the estimate $\hat{x}_i$ satisfies $\Pr[|\hat{x}_i - x_i| > \epsilon\|\boldsymbol{x}\|] < \delta$, *i.e.* $\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_\infty \leq \epsilon\|\boldsymbol{x}\|$ with probability $1 - \delta$. Under this setting , we are free to return 0 for any value whose frequency $x_i \leq \epsilon\|\boldsymbol{x}\|$. Only $O(\epsilon^{-1})$ non-zero frequencies needs to be reported, which is achievable using small space. This topic is therefore referred to as finding frequent items sometimes.

4.1 Basic Counting

We first introduce the basic counting problem. The target is to count the occurrence of 1's in a bit stream. It can be regarded as frequency estimation on a domain $U = \{0, 1\}$. A trivial algorithm uses $O(\log n)$ bits. The Morris Counter [1, 85, 86] for this problem was the first streaming algorithm. It uses a counter c with a parameter $1 < b \leq 2$. Each bit 1 will increase the counter with probability b^{-c}. An estimate of x_1 is $X = (b^c - 1)/(b - 1)$. To analyze the accuracy, let random variable C_i denote the value of the counter when i bits are inserted. $C_0 = 0$ deterministically. The estimator is then $X_i = (b^{C_i} - 1)/(b - 1)$. We have

$$\begin{aligned}
\mathbf{E}[X_i] &= \sum_c \Pr[C_i = c] \cdot \frac{b^c - 1}{b - 1} \\
&= (b-1)^{-1} \sum_c (\Pr[C_{i-1} = c] \cdot (1 - b^{-c}) + \Pr[C_{i-1} = c - 1] \cdot b^{-c+1})(b^c - 1) \\
&= (b-1)^{-1} \sum_c \Pr[C_{i-1} = c] \cdot ((1 - b^{-c})(b^c - 1) + b^{-c}(b^{c+1} - 1)) \\
&= (b-1)^{-1} \sum_c \Pr[C_{i-1} = c] \cdot (b^c + b - 2) \\
&= \sum_c \Pr[C_{i-1} = c] \cdot \frac{b^c - 1}{b - 1} + \sum_c \Pr[C_{i-1} = c] = \mathbf{E}[X_{i-1}] + 1\,.
\end{aligned}$$

By $\mathbf{E}[X_0] = \mathbf{E}[(b^{C_0} - 1)/(b-1)] = 0$, we conclude $\mathbf{E}[X_i] = i$, thus X_i is an unbiased estimator. Similarly we can derive $\mathbf{E}[X_i^2] = (b+1)i(i-1)/2$, so its variance is bounded by $\mathbf{Var}[X_i] = (b+1)i(i-1)/2 - i^2 \le (b-1)i^2/2$. Setting $b = 1 + 2\epsilon^2\delta$ suffices to bounds the variance to provide an (ϵ, δ)-approximation. Correspondingly, the counter c can be as large as $1 + \log_b(b-1)n = O(\epsilon^{-2}\delta^{-1} \log \epsilon^2\delta n)$, thus only requires $O(\log \epsilon^{-1} + \log \delta^{-1} + \log\log n)$ bits to represent, improving over the $O(\log n)$ upper bound. It is recently shown [87] that a slight modification of the algorithm will make it optimal matching the lower bound of $\Omega(\min\{\log n, \log \epsilon^{-1} + \log\log \delta^{-1} + \log\log n\})$.

The Morris Counter also implies a frequency estimation protocol: create $O(u)$ counters for each item in the universe, and update the corresponding counter for values in the stream. This improves the space consumption to $O(u \log\log n)$ compared to the exact algorithm for constant ϵ and δ.

4.2 Cash Register Model

Bose et al. [88] showed that for any randomized algorithm using m counters, there is an error lower bound of $\Omega(\|x\|_1/m)$. In other words, to achieve an $\epsilon\|\boldsymbol{x}\|_1$ guarantee, we must use $\Omega(\epsilon^{-1})$ counters, or $\tilde{\Omega}(\epsilon^{-1})$ bits.[1] Several Algorithms match this lower bound using $O(\epsilon^{-1})$ counters, including MG Summary [89], SpaceSaving [90, 91], Count Sketch [92] and Count-Min Sketch [93]. As there are matching deterministic algorithms, this is also a tight deterministic lower bound up to log factors.

4.2.1 Misra-Gries Summary and SpaceSaving

The Misra-Gries (MG) summary [89] originates from the *majority* algorithm by Moore and Boyer [94, 95]. The algorithm is deterministic. View the MG summary as a histogram. On insertion, it updates the corresponding frequency. Wherever the histogram contains more than $m = \epsilon^{-1}$ non-zero entries, simultaneously reduce all non-zero frequencies until (at least) one of them becomes 0 and is removed. When the algorithm terminates, it reports the histogram.

The correctness can be verified as follows. Each frequency reduction process simultaneously reduces m counters, thus reduces the sum of frequencies in the summary by m. This can happen at most $\|\boldsymbol{x}\|_1/m$ times. So for each value i, its final reported value $\hat{x}_i$ satisfies $x_i - \|\boldsymbol{x}\|/m \le \hat{x}_i \le x_i$, which means $|\hat{x}_i - x_i| \le \epsilon\|\boldsymbol{x}\|_1$.

Since there are only $O(\epsilon^{-1})$ non-zero frequencies at any time of the algorithm, they can be maintained by a dictionary of size $O(\epsilon^{-1} \log un)$. For update time, the original paper [89] used a binary search tree for the dictionary to achieve $O(1)$ amortized update time. [96] showed the update time can be improved to worst case $O(1)$ and [97] implemented the dictionary using a hash table.

Many algorithms use similar ideas to MG summary, including Sticky Sampling, Loosy Counting [98] and SpaceSaving [90, 91]. In SpaceSaving, instead of deducting frequencies when the space budget is reached, the algorithm replaces the value with minimum frequency by the newly inserted value, and add its frequency to the new one. This is identical to a MG summary with $m - 1$ counters, if we subtract every frequency by the minimum frequency in the dictionary [78].

The estimator can be made unbiased [99].

[1] We suppress log factors in $\tilde{O}$ and $\tilde{\Omega}$.

4.3 Turnstile Model and Sliding Windows

An $\tilde{\Omega}(\epsilon^{-1})$ lower bound is inherited from the cash register model for any randomized algorithm, which is also matched by Count-Min sketch [93] and Count Sketch [92].

The MG summary does not handle deletions, so there is a gap between the $\tilde{O}(\epsilon^{-2})$ deterministic upper bound [100] and this lower bound. Ganguly [101] matched this up to log factors by showing an $\Omega(\epsilon^{-2}\log^{-1}\epsilon^{-1}\log u\log n)$ lower bound.

For sliding windows, Arasu and Manku [102] gave an algorithm using $O(\epsilon^{-1}\log^2\epsilon^{-1})$ space. Lee and Ting [103] improved it to $O(\epsilon^{-1})$, at the cost of a higher query time. Zhang and Guan [104] solved it using $O(\epsilon^{-1})$ space with $O(1)$ update time. Similar results are achieved by Hung et al. [105]. Ben-Basat et al. [106] further improved the query time to $O(1)$.

4.3.1 Count-Min Sketch

Count-Min sketch [93] is an efficient data sketch for frequency estimation. It is a linear transform of the frequency vector, which can be represented by a matrix. Each level of a Count-Min sketch is an array CM of L counters associated with a pairwise independent hash function $h : [u] \to [L]$. Updating an element $a \in [u]$ simply changes the frequency of counter $CM[h(a)]$ by the same amount, allowing the algorithm to support deletions. Intuitively, each bucket i stores $CM[i] = \sum_{j:h(j)=i} x_j$. The estimator for x_a is $\hat{x}_a = CM[h(a)]$. It is easy to see that $\hat{x}_a \geq x_a$. The surplus is

$$\hat{x}_a - x_a = \sum_{j\neq a:h(j)=h(a)} x_j = \sum_{j\neq a} x_j \cdot \mathbf{I}[h(j) = h(a)]\,.$$

Since h is pairwise independent, $\Pr[h(j) = h(a)] = 1/L$. So we have in expectation

$$\mathbf{E}\big[\hat{x}_a - x_a\big] = \sum_{j\neq a} x_j/L \leq \|\boldsymbol{x}\|_1/L\,.$$

By Markov's inequality,

$$\Pr\left[\hat{x}_a - x_a \geq 2\|\boldsymbol{x}\|_1/L\right] \leq \frac{1}{2}\,.$$

Taking $L = 2\epsilon^{-1}$ provides an $(\epsilon, 1/2)$-approximation. Using $\log\delta^{-1}$ independent levels of count sketch will boost the success probability to $1-\delta$. The space consumption is therefore $O(\epsilon^{-1}\log\delta^{-1})$ words. Since $CM[h(a)]$ is always an upper bound of x_a, the minimum is taken over all the independent estimates.

The analysis can be tightened by using residual norm $\|\boldsymbol{x}\|_1^{\text{res}}$, which is $\|\boldsymbol{x}\|_1$ subtracting $O(\epsilon^{-1})$ largest frequencies. We only need to fail the algorithm when $h(a) = h(j)$ for some j within the $O(\epsilon^{-1})$ most-frequent values. For each a, this fails the algorithm with probability $1-(1-2\epsilon)^{O(\epsilon^{-1})}$, which is a constant that can be restricted to δ through repetitions. This helps explain why the Count-Min sketch usually performs better than its worst-case bound.

Count-Min sketch always produces an upper bound, so its estimator is biased. Observing that $\mathbf{E}\big[CM[h(a)]\big] = x_a + \sum_{j\neq a} x_j/L = (L-1)x_a/L + \|x\|_1/L$, the bias can be removed using the estimator

$$\hat{x}_a^{\text{unbiased}} = \frac{L\cdot CM[h(a)] - \|x\|_1}{L-1} = CM[h(a)] - \frac{\sum_{i\neq h(a)} CM[h(i)]}{L-1}\,.$$

In other words, $(\sum_{i\neq h(a)} CM[h(i)])/(L-1)$ is an estimator for the (upward) bias. Since it is specific to hash function h, it does not concentrate over repetitions. However, this helps us to

provide a bound using $\|\boldsymbol{x}\|_2 = \sqrt{\sum_i x_i^2}$. We see that

$$\begin{aligned}
\mathbf{Var}[\hat{x}_a^{\text{unbiased}}] &= \mathbf{Var}\left[\frac{L \cdot CM[h(a)] - \|x\|_1}{L-1}\right] \\
&= \left(\frac{L}{L-1}\right)^2 \mathbf{Var}[CM(h(a))] \\
&= \left(\frac{L}{L-1}\right)^2 \sum_{j \neq a} \mathbf{Var}[x_j \cdot \mathbf{I}[h(j) = h(a)]] \\
&= \frac{L^2}{(L-1)^2} \sum_{j \neq a} x_j^2 \cdot \frac{L-1}{L^2} \\
&\leq \frac{\|\boldsymbol{x}\|_2^2}{L-1}\,.
\end{aligned}$$

Apply Chebyshev's inequality, we have

$$\Pr\left[|\hat{x}_a^{\text{unbiased}} - x_a| \geq \sqrt{\epsilon}\|x\|_2\right] \leq \frac{\|\boldsymbol{x}\|_2^2}{L-1} \cdot \frac{1}{\epsilon\|\boldsymbol{x}\|_2^2} = \frac{1}{\epsilon(L-1)}\,.$$

Taking $L = 1 + 2\epsilon^{-1}$ will make the fail probability $1/2$, which can be boosted through $\log \delta^{-1}$ independent trials. In this case, the error bound is $\sqrt{\epsilon}\|\boldsymbol{x}\|_2$ using only $\log \delta^{-1}$ more counters compared to achieving an $\epsilon\|\boldsymbol{x}\|_1$ error. When the frequencies are flat, for example when $x_i = v$ for all i, $\|\boldsymbol{x}\|_2 = \sqrt{v\|\boldsymbol{x}\|_1} \ll \|\boldsymbol{x}\|_1$. the $\sqrt{\epsilon}\|\boldsymbol{x}\|_2$ bound is better.

It has also been extended [107] to work for sliding windows.

4.3.2 Count Sketch

Count Sketch [92] can be regarded as an unbiased version of Count-Min sketch. Each level still has L counters $CS[1], \ldots, CS[L]$. In addition to a pairwise independent hash function $h : [u] \to [L]$, it also uses a pairwise independent hash function $g : [u] \to \{\pm 1\}$. Inserting an element $a \in [u]$ updates counter $CS[h(a)]$ to $CS[h(a)] + g(a)$. It is straightforward to handle updates with frequencies larger than 1, or deletions. Similarly, we derive $CS[i] = \sum_{j:h(j)=i} x_j \cdot g(j)$. An estimator for x_a is $\hat{x}_a = CS[h(a)] \cdot g(a)$. We have

$$\begin{aligned}
\hat{x}_a &= \sum_{j:h(j)=h(a)} x_j g(j) g(a)] \\
&= x_a g^2(a) + \sum_{j \neq a} x_j g(j) g(a) \mathbf{I}[h(j) = h(a)]
\end{aligned}$$

Since $g^2 \equiv 1$, and g is pairwise independent, $\mathbf{E}[\hat{x}_a] = x_a + \sum_{j \neq a} x_j \mathbf{E}[g(j)]\mathbf{E}[g(a)]\mathbf{E}\big[\mathbf{I}[h(j) = h(a)]\big] = x_a$, proving the estimator is unbiased.

Similar to Count-Min sketch, there are both ℓ^1 and ℓ^2 bounds. Note that

$$\mathbf{E}\left[|\hat{x}_a - x_a|\right] \leq \sum_{j \neq a} x_j \mathbf{E}\left[|g(j)g(a)| \cdot \mathbf{I}[h(j) = h(a)]\right] = \sum_{j \neq a} x_j \Pr[h(j) = h(a)] = \sum_{j \neq a} x_j / L\,.$$

By Markov's inequality, we have

$$\Pr\left[|\hat{x}_a - x_a| \geq 2\|\boldsymbol{x}\|_1 / L\right] \leq \frac{1}{2}\,.$$

To achieve an (ϵ, δ)-approximation, we set $L = 2\epsilon^{-1}$ and create $O(\log \delta^{-1})$ repetitions to apply the median estimator. Although both Count-Min sketch and Count Sketch use $O(\epsilon^{-1} \log \delta^{-1})$ counters to achieve $O(\epsilon \|\boldsymbol{x}\|_1)$ error, the hidden constants are worse in Count Sketch. This is due to the difference between taking min and a median. For Count-Min, the algorithm fails if all the repetitions fail, while for Count Sketch, it fails when half the repetitions fail.

For an ℓ^2 bound, observe $\hat{x}_a$ is already unbiased. Its variance is

$$\begin{aligned}
\mathbf{Var}[\hat{x}_a] &= \sum_{j \neq a} x_j^2 \mathbf{Var}\big[g(j)g(a)\mathbf{I}[h(j) = h(a)]\big] \\
&= \sum_{j \neq a} x_j^2 \left(\mathbf{E}\big[g^2(j)g^2(a)\mathbf{I}^2[h(j) = h(a)]\big] - \mathbf{E}^2\big[g(j)g(a)\mathbf{I}[h(j) = h(a)]\big] \right) \\
&= \sum_{j \neq a} x_j^2 (\Pr[h(j) = h(a)] - 0^2) = \|\boldsymbol{x}\|_2^2 / L \,.
\end{aligned}$$

Apply Chebyshev's inequality, we see

$$\Pr\left[|\hat{x}_a - x_a| \geq \sqrt{\epsilon} \|\boldsymbol{x}\|_2 \right] \leq \frac{\|\boldsymbol{x}\|_2^2}{L} \cdot \frac{1}{\epsilon \|\boldsymbol{x}\|_2^2} \leq \frac{1}{\epsilon L} \,.$$

Taking $L = 2\epsilon^{-1}$ will make the probability $1/2$, which can be boosted by $O(\log \delta^{-1})$ repetitions. Same to the Count-Min sketch, the bound is $O(\sqrt{\epsilon}\|\boldsymbol{x}\|_2)$ when using $O(\epsilon^{-1} \log \delta^{-1})$ counters.

4.3.3 Sliding-Windows Sketches

Arasu and Manku [102] introduced a framework that turns cash register algorithms into sliding window ones. This is similar to the dyadic ranges to be introduced later. To ease the proof, we assume a fixed window length w. The stream is duplicated L times, denoted by level $0, \ldots, L$. At level 0, the stream is partitioned into blocks of size $c_0 = \Theta(\epsilon w)$. Each block at level i is the union of two adjacent blocks at level $i-1$, thus contains 2^i epochs, or $c_i = \Theta(2^i \epsilon w)$ elements. Correspondingly we only need $L = O(\log \epsilon^{-1})$ levels. Level i contains $w/c_i = \Theta(1/(2^i\epsilon))$ blocks, so $O(\epsilon^{-1})$ blocks in total. A block is

- Active, if all its elements are inside the current window.
- Under construction, if some of its elements have not arrived yet.
- Expired, if some of its elements are out of the current window.

It can be verified that the window can always be decomposed into $O(L)$ active blocks, with at most two partial blocks at level 0 that are under construction or has expired. The error due to incomplete blocks can be bounded by $O(\epsilon w)$ as the block only contains $O(\epsilon w)$ elements. A natural idea is to let each block hold an ϵ-approximation protocol. Summing over the decomposition will generate an $O(\epsilon w)$ estimation. This uses $O(\epsilon^{-2})$ space as there are $O(\epsilon^{-1})$ sketches of size $O(\epsilon^{-1})$. An improvement can be made by observing that there are more blocks at bottom levels, so we may assign a larger ϵ for them to save space. The problem now becomes to minimize the space consumption of $\sum_i (w/(c_i\epsilon_i)$ subject to the error $\sum_i \epsilon_i c_i = O(\epsilon w)$. It is optimized when $\epsilon_i \propto 1/c_i$, *i.e.* when $\epsilon_i c_i = \Theta(\epsilon w \log^{-1} \epsilon^{-1})$. The space consumption is $O(\epsilon^{-1} \log^2 \epsilon^{-1})$.

4.3.4 Window Counter

[103] proposed the idea of window counter. A counter is separately maintained for each value v, using $O(x_v/(\epsilon n))$ space, so the total space consumption is $O(\epsilon^{-1})$. Each counter "samples" the position of every λ items, so that the error due to sampling is at most 2λ. The authors further combined it with the MG summary to show that most counters are unnecessary to be stored. Similar ideas are used in [105].

4.4 Distributed Multisets

In distributed setting, we denote by $\boldsymbol{x}_i$ the frequency vector at site i, and $\boldsymbol{x}$ to denote the frequency vector of all elements, namely the frequency of $j \in [u]$ is $x_j = \sum_{i=1}^{k} x_{i,j}$. The error threshold is set to $\epsilon\|\boldsymbol{x}\|_1$. Note that computing $\|\boldsymbol{x}\|_1$ requires $O(k)$ messages of $O(\log n)$ bits, which we do not include in the following bounds. We first discuss frequency estimation in static setting.

4.4.1 Deterministic Solution

There is a simple deterministic solution to the problem: each site reports all values whose frequency is above $\epsilon\|\boldsymbol{x}\|_1/k$. Unreported values are assumed by the coordinator to have 0 frequency. The cumulative error is bounded by $k \cdot \epsilon\|\boldsymbol{x}\|_1/k = \epsilon\|\boldsymbol{x}\|_1$. For site i, there are at most $\|\boldsymbol{x}_i\|_1/(\epsilon\|\boldsymbol{x}\|_1/k) = (k/\epsilon)(\|\boldsymbol{x}_i\|_1/\|\boldsymbol{x}\|)$ values above the threshold, the total communication is bounded by

$$\sum_{i=1}^{k} \frac{k}{\epsilon} \cdot \frac{\|\boldsymbol{x}_i\|_1}{\|\boldsymbol{x}\|_1} = k/\epsilon$$

item-frequency pairs, which consumes $O(\log un)$ bits, provided $\|\boldsymbol{x}\|$ was pre-computed and sent to all sites. This is conjectured to be the best deterministic algorithm. Equivalently, we may apply any approximate algorithm above to achieve an $\epsilon\|\boldsymbol{x}_i\|_1$ guarantee locally using $O(\epsilon^{-1})$ counters, and send the states using $O(k/\epsilon)$ messages to obtain error $\sum_i \epsilon\|\boldsymbol{x}_i\|_1 = \epsilon\|\boldsymbol{x}\|_1$. Both the MG summary and Count(-Min) Sketch are mergable [78]

4.4.2 Coin-Flip Sampling

To save communication, a natural idea is to apply sampling. One of the simplest techniques is *coin-flip sampling*, namely independently sample each *item* with the same probability. Let p be the probability of sampling any item j, and $\|\boldsymbol{y}\|$ be the frequencies in the sample, it is known that $\hat{x}_j = y_j/p$ is an unbiased estimator for x_j. Since $y_j \sim B(x_j, p)$, $\mathbf{Var}[y_j] = p(1-p)x_j$,

$$\mathbf{Var}[\hat{x}_j] = \frac{\mathbf{Var}[y_j]}{p^2} \le x_j/p \le \|\boldsymbol{x}\|_1/p\,.$$

Therefore by taking $p = 2\epsilon^{-2}\|\boldsymbol{x}\|_1^{-1}$, the variance is bounded by $\epsilon^2\|\boldsymbol{x}\|_1^2/2$, which guarantees an $\epsilon\|\boldsymbol{x}\|_1$ error with probability $1/2$ by Chebyshev's inequality. The communication is $O(p\|\boldsymbol{x}\|_1) = O(\epsilon^{-2})$ messages of $O(\log u)$ bits, provided that $\|\boldsymbol{x}\|_1$ was already computed and sent to all sites.

It can be observed that the inequality $x_j \le \|\boldsymbol{x}\|_1$ in the analysis is quite loose. Indeed, setting $p_j = 2x_j/(\epsilon\|\boldsymbol{x}\|_1)^2$ is sufficient for the same the analysis. The problem is that x_j is the value to be estimated, thus we cannot use this exact probability in coin-flip sampling. Yet this call for a different sampling probability for each value, depending on its frequency.

4.4.3 Importance Sampling

Zhao et al. [108] introduced a frequency-aware sampling framework. Sampling is performed on *local frequencies* instead of on *items*. Specifically, site i reports the frequency of item j $x_{i,j} = x$ to the coordinator with probability $p(x)$. The coordinator estimates $\hat{x}_{i,j} = x/p(x)$, if value x is received, and 0 otherwise. Similar to the analysis before, $\hat{x}_{i,j}$ is unbiased and its variance is $\mathbf{Var}[\hat{x}_{i,j}] = x^2/p(x) - x^2$. Summing over all sites, the frequency estimator $\hat{x}_j = \sum_i \hat{x}_{i,j}$ is also unbiased with variance

$$\mathbf{Var}[\hat{x}_j] = \sum_{i=1}^{k} \frac{x_{i,j}^2}{p(x_{i,j})} - \sum_{i=1}^{k} x_{i,j}^2 \leq \sum_{i=1}^{k} \frac{x_{i,j}^2}{p(x_{i,j})} - x_j^2/k\,.$$

The problem now becomes finding the optimal function $p(x)$ to achieve a trade-off in minimizing both the variance and the communication $\sum_{i=1}^{k} p(x_{i,j})$. Observing that by Cauchy-Schwarz inequality,

$$\sum_{i=1}^{k} \frac{x_{i,j}^2}{p(x_{i,j})} \cdot \sum_{i=1}^{k} p(x_{i,j}) \geq \left(\sum_{i=1}^{k} \sqrt{\frac{x_{i,j}^2}{p(x_{i,j})} p(x_{i,j})} \right)^2 = x_j^2\,,$$

the best setting must choose $\frac{x_{i,j}^2}{p(x_{i,j})} \propto p(x_{i,j})$, that is $p(x_{i,j}) = cx_{i,j}$. Correspondingly the total communication for j is cx_j and the variance is bounded by $\mathbf{Var}[\hat{x}_j] = -x_j^2/k + x_j/c \leq k/(4c^2)$. To bound the variance by $\epsilon^2\|\boldsymbol{x}\|_1^2/2$, it suffices to set $c = \sqrt{k}/(\sqrt{2}\epsilon\|\boldsymbol{x}\|_1)$. This is the *importance sampling* function in [109]. The name comes from setting $p(x)$ proportional to x, where $p(x) = \sqrt{k}x/(\epsilon\|\boldsymbol{x}\|_1)$.[2] The total communication is then $O(\sqrt{k}/\epsilon)$ pairs. They also show that by using a Bloom Filter, communication can be reduced to $O(\sqrt{k}/\epsilon)$ bits for the same guarantee. This was proved optimal [110, 76].[3]

4.5 Distributed Streams

Since the total frequency is simply the sum of all local frequencies, as long as the coordinator tracks the local frequency of j at site i within ϵn_i error, it also tracks the total frequency within ϵn error. Yi and Zhang [111] solves it by an optimal deterministic algorithm with $O(k/\epsilon \cdot \log N)$ communication. Huang, Yi and Zhang [112] improved the upper bound to $O(\sqrt{k}/\epsilon \cdot \log N)$ using importance sampling as a building block and showed it is optimal: $\Omega(\sqrt{k}/\epsilon \cdot \log N)$ messages must be communicated.

4.5.1 Lazy Update Algorithm

[111] can be seen as a streaming version of the exact algorithm. The coordinator maintains $\hat{n}$, an ϵ-approximation of n. To guarantee this, each site sends n_i to the coordinator wherever it increases by a factor of $(1+\epsilon)$, denote it by $\bar{n}_i$. The coordinator uses $\bar{n}_i$ to approximate n_i until it receives another message. By definition, it is guaranteed that $\bar{n}_i \leq n_i \leq (1+\epsilon)\bar{n}_i$. This can be achieved using $O(\log_{1+\epsilon} N_i)$ communication at each site, thus consumes $O(k/\epsilon \cdot \log N/k)$ communication in total.

The algorithm then divides the tracking into $O(\log_{1+\epsilon} N)$ rounds: Within each round, the coordinator uses $O(k)$ communication to ensure each site hold the same $\bar{n}$. When $\hat{n} \geq (1+\epsilon)\bar{n}$, the round ends. Each site ensures that $x_{i,j} - \hat{x}_{i,j} \leq \epsilon\bar{n}/k \leq \epsilon n/k$, by sending the latest $x_{i,j}$ to the coordinator as $\hat{x}_{i,j}$ when the above condition fails. The error guarantee is trivial.

[2] We assume $p(x)$ is always upper bounded by 1.

[3] For $k = \omega(\epsilon^{-2})$, the coin-flip sampling is optimal.

For communication, note that each round only contains $\epsilon\bar{n}$ items, thus synchronization can only happen k times. Therefore $O(k)$ messages are sent within each round, resulting in an $O(k/\epsilon \cdot \log N)$ bound. This is also shown to be the optimal deterministic algorithm.

4.5.2 Distributed Importance Sampling

[112] extends the idea of importance sampling to the distributed setting. We first show how the algorithm tracks n using $O(\sqrt{k}/\epsilon \cdot \log N)$ communication.

Whenever an item arrives at site i, the site sends the latest n_i to the coordinator with probability p. Denote this value by $\bar{n}_i$. It is easy to see that the number of out-of-sync items $Y = n_i - \bar{n}_i$ follows a geometric distribution: It is 0 if the last item n_i is sampled; 1 if the last but one value was sampled, but the last value is not, *etc.* We have $\Pr[Y = y] = (1-p)^y p$.[4] Since $\mathbf{E}[Y] = p^{-1} - 1$ and $\mathbf{Var}[Y] \le p^{-2}$, it follows that $\hat{n}_i = \bar{n}_i + p^{-1} - 1$ is an unbiased estimator of n_i with its variance bounded by p^{-2}. This is achieved using $O(pN_i)$ communication. It is straightforward that $\hat{n} = \sum_i \hat{n}_i$ is an unbiased estimator of n, whose variance is bounded by k/p^2. Thus setting $p = \Theta(\sqrt{k}/\epsilon n)$ suffices to provide an ϵ-approximation by Chebyshev's inequality.

Again the problem is that n is the value we want to estimate. To solve it, the authors used the deterministic algorithm in the previous section to track n' where $n' \le n \le 2n'$. This only consumes $O(k \log N)$ communication since $\epsilon = 1$. Whenever n' doubles, the coordinator initiates a new round by broadcasting $p = \sqrt{k}/\epsilon n'$ to all sites to be the sampling probability in this round. Since n can be as large as $4n'$ within a round, the communication in the round is bounded by $O(\sqrt{k}/\epsilon)$ with constant probability. Counted over all rounds, it sums to $O(\sqrt{k}/\epsilon \cdot \log N)$. It is also easy to see that the same sample probability and estimator suffice to estimate individual frequencies: each $x_{i,j}$ is estimated within ϵn error, using $O(\sqrt{k}/\epsilon \cdot \log N)$ communication. The authors also show that by approximately counting local frequencies, the space consumption can be reduced to $O((\epsilon\sqrt{k})^{-1})$.

4.6 Finding Heavy Hitters

A closely related problem to frequency estimation is finding heavy hitters. Denote by $n = \|\boldsymbol{x}\|_1$. In a ϕ-heavy hitter query, the algorithm must report all values whose frequency is at least ϕn, and must report no value whose frequency is below $(\phi - \epsilon)n$.

4.6.1 Histogram-Based Solutions

Many frequency estimation protocols release histograms that naturally answer heavy hitter queries. For example, the MG summary releases a histogram where $x_i - \epsilon n \le \hat{x}_i \le x_i$. We only need to report values with $\hat{x}_i \ge (\phi - \epsilon)n$.

- Reported values satisfy $x_i \ge \hat{x}_i \ge (\phi - \epsilon)n$;
- If $x_i \ge \phi n$,we have $\hat{x}_i \ge x_i - \epsilon n \ge (\phi - \epsilon)n$ and the value is reported.

It solves the heavy hitter problem for any $\phi > \epsilon$. Indeed, if we have any any $\epsilon/2$-approximation frequency estimation protocol where $|\hat{x}_i - x_i| \le \epsilon n/2$, using $\hat{x}'_i = \hat{x}_i - \epsilon n/2$ will guarantee $x_i - \epsilon n \le \hat{x}'_i \le x_i$. We can use it to answer the heavy hitter query similarly.

This also works in the distributed case: as long as the coordinator tracks n and all frequencies within $\epsilon n/3$ error, it can also solve the heavy hitter problem.

[4] A more careful analysis takes into account that Y is upper bounded by n_i. We ignore this.

4.6.2 Sketch-Based Solutions

For sketch-based solutions like count sketch and count-min sketch, no histogram is explicitly stored. Obtaining all frequencies consumes $O(u)$ time, which is inefficient. Below we show using a supporting data structure can reduce this cost. This applies to any solution that can provide an ϵ-approximation for point queries.

In the cash register model, it is easy as $\|\boldsymbol{x}\|_1$ always increases. We only need to perform a query after every insertion to obtain the latest estimated frequency $\hat{x}_i$ of the inserted item, and track the values with $\hat{x}_i \geq \phi n$ through a min-heap.

In the turnstile model, a solution is to use dyadic intervals [113], as was described in [114]. In a dyadic data structure, the leaf nodes (level 0) are individual elements. Level j contains dummy elements, each representing 2 elements from level $j-1$. The i-th item at level j can be represented by an interval $[(i-1)2^j+1, i2^j]$, for $j = 0, \ldots, \log u$ and $i = 1, \ldots, u/2^j$.[5] The intervals can be organized into a full binary tree: the frequency of a parent node is the sum of frequencies of both its children. A sketch is maintained for each level, thus increasing the space consumption by $O(\log u)$. At query time, we perform a breadth-first search, only expanding nodes whose frequency is at least $(\phi - \epsilon/2)n$. Subject to **all** frequency estimations are $(\epsilon/2)$-approximations, any interval containing a heavy hitter will be expanded. Therefore all heavy hitters are found. There are only $(\phi - \epsilon/2)^{-1} = O(\epsilon^{-1})$ nodes to be expanded at each level, leading to a query time of $O(\epsilon^{-1} \log u)$ multiplied by the query time of the sketch.

4.7 Remarks

Table 4.1 is a summary of algorithms. Experimental evaluations can be found in [115]. For simplicity we refer to space and communication in units of words (instead of bits).

Table 4.1: Algorithms for Frequency Estimation.

	Model	Algorithm	Space/Communication	Error Guarantee	Properties
Deterministic	Cash Register	MG Summary [89]	$O(\epsilon^{-1})$	$\epsilon\|\boldsymbol{x}\|_1$	always lower bound
	Turnstile	CR-Precis [100]	$O(\epsilon^{-2})$	$\epsilon\|\boldsymbol{x}\|_1$	
	Sliding Window	Lee & Ting [103]	$O(\epsilon^{-1})$	ϵw	
	Distributed Tracking	Yi & Zhang [111]	$O(k/\epsilon \cdot \log N)$	$\epsilon\|\boldsymbol{x}\|_1$	When $k = O(\epsilon^{-2})$
Randomized	Turnstile	Count-Min Sketch [93]	$O(\epsilon^{-1})$	$\min\{\epsilon\|\boldsymbol{x}\|_1, \sqrt{\epsilon}\|\boldsymbol{x}\|_2\}$	always upper bound
	Turnstile	Count Sketch [92]	$O(\epsilon^{-1})$	$\min\{\epsilon\|\boldsymbol{x}\|_1, \sqrt{\epsilon}\|\boldsymbol{x}\|_2\}$	unbiased
	Distributed One-Shot	Importance Sampling [109]	$O(\sqrt{k}/\epsilon)$	$\epsilon\|\boldsymbol{x}\|_1$	When $k = O(\epsilon^{-2})$
	Distributed Tracking	Huang et al. [112]	$O(\sqrt{k}/\epsilon \cdot \log N)$	$\epsilon\|\boldsymbol{x}\|_1$	When $k = O(\epsilon^{-2})$

[5] Assume u is a power of 2.

Chapter 5

Quantiles

For ordered data, quantiles are useful to provide an overlook of the data distribution. For example, the most important quantile, *median*, is a representative of the data that is robust against outliers. Formally, let $[u] = \{1, \ldots, u\}$ be an ordered domain, and $\boldsymbol{x}$ be the frequency vector where $\|\boldsymbol{x}\|_1 = n$. The rank of value $i \in [u]$ is defined as $\text{rank}(i) = \sum_{j<i} x_j$: the total frequency of values less than x. A quantile query specifies a given rank r, usually represented by a fraction $\phi = r/n$, and queries the largest value i such that $\text{rank}(i) \leq r$ and $x_i > 0$. It is usually meaningful to apply the approximation constraint on ranks, that is to return a value $\hat{i}$ where $|\text{rank}(\hat{i}) - r| \leq \epsilon n$.[1]

A closely related problem is rank query, which returns the approximate rank $\hat{\text{rank}}(i)$ of any given value i such that $|\text{rank}(i) - \hat{\text{rank}}(i)| \leq \epsilon n$. The all-quantile problem and all-ranks problem are asymptotically equivalent. Suppose a protocol supports solving the rank problem with error $\epsilon n/2$, to solve the quantile problem, let $r = \text{rank}(i)$, then $|r - \hat{\text{rank}}(i)| \leq \epsilon n/2$. We only need to search for $\hat{i}$ whose approximate rank is most close to r, so that

$$|\text{rank}(\hat{i}) - r| \leq |\text{rank}(\hat{i}) - \hat{\text{rank}}(\hat{i})| + |\hat{\text{rank}}(\hat{i}) - r| \leq \epsilon n\,.$$

The reverse is similar. To solve the all quantile problem, bound the fail probability of every quantile problem by $O(\epsilon)$, so that a union bound will make it constant for all $1/\epsilon$ quantiles.

It is usually easier to consider a set of distinct elements in this problem.

5.1 Cash Register Model

If the frequency vector $\boldsymbol{x}$ is available, all ranks can be computed through a prefix-sum. The problem is to maintain a small summary in the streaming setting. Munro and Paterson [116] showed a lower bound of $\Omega(n^{1/p})$ for any p-pass algorithm that computes the median, which means an $\Omega(n)$ lower bound for exact solutions. Manku, Rajagopalan and Lindsa [117] developped the ideas in [116] to provide a deterministic algorithm storing $O(\epsilon^{-1}\log^2(\epsilon n))$ items, assuming n was known. The best deterministic algorithm was by Greenwald and Khanna [118], which stores $O(\epsilon^{-1}\log \epsilon n)$ items and makes no assumption on knowledge of the stream. It also has a practical version, which was the state-of-art for many years. For deterministic *comparison-based* algorithm, there are lower bounds of $\Omega(\epsilon^{-1}\log \epsilon^{-1})$ [119] and $\Omega(\epsilon^{-1}\log \epsilon n)$ [120]. If the universe $[u]$ is known in advance, the Q-digest by Shivastava et al. [121] uses $O(\epsilon^{-1}\log u)$ items, which is better when u is small.

For randomized algorithms, it is well-known that a random sample of size $O(\epsilon^{-2})$ suffices

[1] There may not be a value satisfying this constraint if some value j satisfies $\text{rank}(j) < r - \epsilon n$ and $\text{rank}(j) + x_j > r + \epsilon n$. In this case j is the output.

to answer any quantile query.[2] The space requirement of randomized algorithms was improved by Manku et al. [122] to $O(\epsilon^{-1}\log^2\epsilon^{-1})$; by Agarwal [78] to $O(\epsilon^{-1}\log^{1.5}\epsilon^{-1})$; by Felber and Ostrovsky [123] to $O(\epsilon^{-1}\log\epsilon^{-1})$; and finally by Karnin et al. [124] to $O(\epsilon^{-1})$, matching the lower bound.

There are also works focusing on relative error [125] and historical tracking [126]. We omit them for simplicity.

5.1.1 Q-Digest and T-Digest

The Q-digest [121] consists of a binary tree whose leaf nodes are elements in $[u]$. This is similar to dyadic ranges. Having to know the universe in advance is a limitation to this algorithm. Only $O(\epsilon^{-1}\log u)$ nodes will be stored out of the $O(u)$ nodes. Each non-leaf node is weighted and has a maximum capacity of $C = \epsilon n/\log u$. To insert an item with frequency x, the algorithm execute a **find** operation in the binary tree. Along the path from root to leaf, it fills the total weight of x to the node until the capacity is reached, and carries remaining weights to the next node, until the leaf node is reached where there is no capacity constraint.

To perform a rank query for element i, divide the nodes into 3 categories depending on their corresponding ranges: strictly less than i, containing i and strictly larger than i. The rank of i is estimated by the cumulative frequencies of the first category, and it is clear that error only rises from non-leaf nodes of the second category, whose weight is at most $C\log u = \epsilon n$.

Note that the capacity C grows with n, so a full node will become underfull after inserting elements, requiring many nodes to be stored. To solve it, the algorithm uses a compress operation. When the capacity increases, the node borrows weights from its descendants to be as full as possible. The borrowed amount is carried to the lowest level possible, where the descendant node lent all its weights and need not be maintained. After a compress operation, each node is either full or has no children. There are at most $n/C = \epsilon^{-1}\log u$ full nodes, so $O(\epsilon^{-1}\log u)$ nodes in total, binding underfull nodes to their parents. The compress operation can be done whenever needed.

T-digest [127, 128] is based on similar clustering ideas and is widely used in industry.

5.1.2 Greenwald-Khanna Summary

The Greenwald-Khanna (GK) summary [118] is a comparision-based algorithm, where it does not require knowledge about the domain. We describe the practical version.

The summary is an ordered list of tuples (v_i, g_i, Δ_i), where v_i is a value from the universe, g_i is the number of values before v_i that is represented by v_i, and Δ_i is the uncertainty of the position of v_i. They help bound the rank of v_i by

$$\sum_{j\le i} g_j \le \operatorname{rank}(v_i) + 1 \le \Delta_i + \sum_{j\le i} g_j\,.$$

It follows that $\operatorname{rank}(v_i) - \operatorname{rank}(v_{i-1}) \le \Delta_i + g_i$. As long as we bound $\Delta_i + g_i < 2\epsilon n$, we can solve the quantile query problem by finding a v_i such that

$$r - \epsilon n \le \operatorname{rank}(v_i) \le r + \epsilon n\,.$$

Such an v_i must exist: If $r > n - \epsilon n - 1$, we can simply take the maximum value whose rank is at most $n - 1 < r + \epsilon n$. Otherwise we pick the maximum v_i whose rank is $\operatorname{rank}(v_i) \le r + \epsilon n$. It follows that $\operatorname{rank}(v_{i+1}) > r + \epsilon n$ and therefore $\operatorname{rank}(v_i) \ge \operatorname{rank}(v_{i+1}) - (\Delta_i + g_i) > r - \epsilon n$. So v_i satisfies the property.

[2]The ϕ-quantile in the sample is centered around the ϕ-quantile in the population with bounded variance $O(n^2/s)$ for sample size s.

The remaining problem is to maintain the summary so that the properties are maintained. To insert a value v, we find in the summary the smallest $v_i > v$.[3] Clearly the rank of v_j is not affected for $j \leq i-1$ and is increases by 1 for $j \geq i$, so we may simply increase g_i by 1 if $\Delta_i + g_i < 2\epsilon n$ still holds after increasing. Otherwise we have to create a new tuple for v between v_{i-1} and v_i. Since $\text{rank}(v) - \text{rank}(v_{i-1}) \leq \text{rank}(v_i) - \text{rank}(v_{i-1}) \leq \Delta_i + g_i$, the tuple $(v, 1, \Delta_i + g_i - 1)$ satisfies all constraints. As this creates new tuples, we need to compress the summary. Consecutive tuples $(v_{i-1}, g_{i-1}, \Delta_{i-1})$ and (v_i, g_i, Δ_i) can be merged into $(v_i, g_{i-1} + g_i, \Delta_i)$, provided that $g_{i-1} + g_i + \Delta_i < 2\epsilon n$.

The formal space guarantee is stated in [118]. If the universe is known, we may feed a sample of size $O(\epsilon^{-2} \log \epsilon^{-1})$ into the GK summary to improve the space consumption to $O(\epsilon^{-1} \log \epsilon^{-1]})$.

5.1.3 Karnin-Lang-Liberty

The Karnin-Lang-Liberty (KLL) summary [124] consists of h buffers $B[1], \ldots, B[h]$ that store values from the data. The capacity of each buffer $B[i]$ is $c[i] = \alpha^{h-i} c[h]$, where $0.5 < \alpha < 1$ is a constant. The smallest capacity $c[1]$ is guaranteed to be at least 2.

A new element is always inserted into buffer $B[1]$. Whenever a buffer $B[i]$ reaches its capacity, a compress operation is run. The buffer sorts all its elements and sends half of them, either all at *odd* positions or all at *even* positions to $B[i+1]$ and clears itself. This may trigger consecutive compression and create additional levels.

It is clear that the buffers holds elements from disjoint partitions of the stream. Only 2^{i-1} elements goes to buffer $B[i]$. So an fair estimate of the rank $\text{rank}(v)$ is $\sum_i 2^{i-1} \text{rank}_i(v)$ where $\text{rank}_i(v)$ is number of elements in buffer $B[i]$ that is less than v.

To analyze the error, consider the first compression when half the elements are sent from $B[1]$ to $B[2]$. Essentially we use $2\text{rank}_2(v)$ to estimate $\text{rank}_1(v)$ where $\text{rank}_2(v)$ is a random variable that with probability 0.5 equals the number of odd/even elements less than v. If $\text{rank}_1(v)$ is even, $2\text{rank}_2(v) \equiv \text{rank}_1(v)$. Otherwise, $2\text{rank}_2(v) \equiv \text{rank}_1(v) \pm 1$, whose expectation is still $\text{rank}_1(v)$. It can be verified that a compression by $B[i]$ keeps the estimator unbiased, while creating an error of $2^{i-1}X$, for a random variable $X = \pm 1$ with equal probability. To bound the total number of compression, note the the last level $B[h]$ is never compressed. Level $h-1$ can be compressed at most $2/\alpha$ times, which will send $(\alpha c[h]) \cdot (2/\alpha) = c[h]$ elements to level h. By induction, number of compression at level i is $(2/\alpha)^{h-i}$. We therefore write the total error as

$$Err = \sum_{i=1}^{h} \sum_{j=1}^{(2/\alpha)^{h-i}} 2^{i-1} X_{i,j}$$

By Chernoff-Hoeffding bound,

$$\begin{aligned}
\Pr[|Err - 0| > \epsilon n] &\leq 2 \exp\left(\frac{-2\epsilon^2 n^2}{\sum_{i=1}^{h} \sum_{j=1}^{(2/\alpha)^{h-i}} 2^{2i}} \right) \\
&= 2 \exp\left(\frac{-2\epsilon^2 n^2}{(2/\alpha)^h \sum_{i=1}^{h} (2\alpha)^i} \right) \\
&\leq 2 \exp\left(\frac{-2\epsilon^2 n^2}{2^{2h}} \right)
\end{aligned}$$

To bound it by a constant, we require $2^h = O(\epsilon n)$: There cannot be too many levels. Since $B[h]$ is never compressed, we have its capacity $c[h] > n/2^h = \Omega(\epsilon^{-1})$. Thus setting $c[h] = \Theta(\epsilon^{-1})$ suffice.

[3] We may have $v_{i-1} = v$.

5.2 Turnstile Model and Sliding Windows

It can be argued that any comparison based algorithm in the turnstile model must store $\Omega(n)$ elements: if we insert n elements and delete $n-1$ of them, we may leave any element. A comparison-based algorithm must keep all the n elements as any of them can be the output after deletion. So most algorithms are under the fixed-universe model. Ganguly and Majumder [100] gave a deterministic frequency estimation protocol that supports quantile queries using $O(\epsilon^{-2}\log^5 u \log(\epsilon^{-1}\log u))$ space. Cormode and Muthukrishnan [93] shows by using a dyadic data structure, a frequency estimation sketch can answer quantile queries using $O(\epsilon^{-1}\log^2 u \log(\epsilon^{-1}\log u))$ space. Luo et al. [129] improved the consumption to $O(\epsilon^{-1}\log^{1.5} u \log^{1.5}(\epsilon^{-1}\log u))$.

For sliding windows, Lin et al. [130] designed an algorithm using $O(\epsilon^{-2}+\epsilon^{-1}\log(\epsilon^2 w))$ memory. Arasu and Manku [102] improved it to $O(\epsilon^{-1}\log\epsilon^{-1}\log w)$.

5.2.1 Dyadic Count Sketch

We introduce the Dyadic Count Sketch in [129]. It also uses the dyadic data structure [113]. We build dyadic intervals $[(i-1)2^j+1, i2^j]$ for $j=0,\ldots,\log u$ and $i=1,\ldots,u/2^j$. At each level j we build a count-sketch (or equivalent frequency estimation protocols), treating an interval as an element. To perform a quantile query r, the algorithm runs a binary search from root to leaf. At each node, it queries the frequency of the left child to decide whether to expand the left or the right child. The intuition is that if all frequencies are accurately estimated, the cumulative error is also small for the quantile. Let the size of each count sketch be $L\times d$. We need $d=\log\log u$ to bound the total fail probability by $O(\log^{-1} u)$, which is a constant summing over all $O(\log u)$ levels. Conditioned on this happening, the error at each level is bounded by w/L, so that by Chernoff-Hoeffding bound,

$$\begin{aligned}\Pr[|Err-0|>\epsilon w] &\le 2\exp\left(\frac{-2\epsilon^2 w^2}{(2w/L)^2\log u}\right)\\ &= 2\exp\left(\frac{-\epsilon^2 L^2}{2\log u}\right)\end{aligned}$$

Taking $L=\Theta(\epsilon^{-1}\sqrt{\log u})$ makes the probability constant. So to answer a quantile query, it suffices to use $O(\epsilon^{-1}\log^{1.5} u\log\log u)$ space. To answer the **all** quantile query, we need to bound fail probabilities at both places by $O(\epsilon)$, so that the space consumption is $O(\epsilon^{-1}\log^{1.5} u\sqrt{\log\epsilon^{-1}}\log\frac{\log u}{\epsilon})$

5.2.2 Sliding Windows

The framework proposed by Arasa and Manku [102] also solves the quantile problem. The only change is to replace the MG summary for frequency estimation by a GK sumamry for quantiles. By properly choosing the parameters, the space can be bounded by $O(\epsilon^{-1}\log\epsilon^{-1}\log w)$.

5.3 Distributed Streams

The heavy hitter problem and quantile problem are closely related. Agarwal et al. [78] discussed about merging quantile summaries to support one-shot estimation. Yi and Zhang [111] designed an optimal deterministic algorithm that can track any quantile using $\Theta(k/\epsilon\cdot\log N)$ communication, and all quantiles using $O(k/\epsilon\cdot\log N\cdot\log^2\epsilon^{-1})$ communication. Huang, Yi and Zhang improved it using a randomized algorithm with $O(\sqrt{k}/\epsilon\cdot\log N\cdot\log^{1.5}(\epsilon\sqrt{k})^{-1})$, while showing an lower bound of $\Omega(\sqrt{k}/\epsilon\cdot\log N)$.

5.3.1 Mergable Summaries

The Q-digest [121], KLL summary [124] and Dyadic Count Sketch [129] are easily mergable without affecting the summary size. For GK summary, merging is not out-of-the-box [131], and either the error or the summary size is increased [78].

5.3.2 Lazy Update Algorithm

The algorithm in [111] is similar to the heavy hitter tracking algorithm. Use the median for illustration. First divide the tracking into $O(\log N)$ rounds, so that within each round every site holds the same $\bar{n}$ in this round, satisfying $\bar{n} \leq n \leq 2\bar{n}$. When the round begins, each site computes $O(\epsilon^{-1})$ intervals, each containing $\Theta(\epsilon n_i)$ items. In $O(k/\epsilon)$ communication, they are all sent to the coordinator, so that the coordinator can compute the rank of any value within $O(\epsilon n)$ error. The coordinator then computes $O(\epsilon^{-1})$ intervals, each containing $\Theta(\epsilon n)$ items, and broadcast them to all sites in $O(k/\epsilon)$ communication. By the frequency tracking algorithm, $O(k/\epsilon)$ communication sufficies to track the number of items within each interval within $O(\epsilon n)$ error. We thus can ensure that the intervals always contain $\Theta(\epsilon n)$ items: divide it into two when it reaches the capacity. This costs $O(k)$ communication. With these intervals, the coordinator can count the number of items to the left (right) of the current approximate median within $\Theta(\epsilon n)$ error. Whenever they differ too much ($\Theta(\epsilon n)$), the coordinator computes a new median. The algorithm uses $O(k/\epsilon \cdot \log N)$ communication, which is optimal for deterministic algorithms. This also implies an $O(k/\epsilon \cdot \log N \log^2 \epsilon^{-1})$ for tracking all quantiles.

5.3.3 Importance Sampling Based Algorithm

Merging GK summaries solves the one-shot distributed quantile estimation problem using $O(\epsilon^{-1} \log^{1.5} \epsilon^{-1})$ space [78]. The summary is of size $O(\epsilon^{-1})$. [112] usesd it as a building block to track ranks. Similarly use $O(\log N)$ rounds so that $\bar{n} < n < 2\bar{n}$ within each round. A site groups its input every $\bar{n}/k$ elements. They are divided into blocks of capacity $c = \epsilon\bar{n}/\sqrt{k}$. They are organized into a binary tree of height $h = O(\log(\epsilon\sqrt{k})^{-1})$. The capacity of a block at level i is $c[i] = 2^i\epsilon\bar{n}/\sqrt{k}$, so there are $O((2^i\epsilon\sqrt{k})^{-1})$ blocks in level i. Whenever a block becomes active, it is sent to the coordinator. The summary size is set to $O(2^i\sqrt{h})$, so that the total communication is $O(\sqrt{k}/\epsilon \cdot h^{1.5}) = O(\sqrt{k}/\epsilon \cdot \log^{1.5}(\epsilon\sqrt{k})^{-1})$. The variance of any block is bounded by $O((c[i]/(2^i\sqrt{h}))^2) = O(c^2/h)$. Accumulated over h blocks through the dyadic decomposition, it becomes $O(c^2)$. There are at most c items that does not belong to any complete block. The authors solve it using a uniform sample with probability c^{-1}, which uses an additional $O(\sqrt{k}/\epsilon)$ communication. The variance from sampling is $O(c/c^{-1}) = O(c^2)$. So the total variance is $O(c^2)$ every $\bar{n}/k$ elements. Summing over all the groups, the variance is $O(kc^2) = O(\epsilon^2\bar{n}^2)$.

5.4 Remarks

Table 5.1 is a summary of algorithms. Space refers to the number of items stored by the summary. [132] gives a survey of results. Experimental evaluation of quantile algorithms are presented in [129, 133, 134].

Table 5.1: Algorithms for Quantiles.

Model		Algorithm	Space/Communication	Properties
Deterministic	Cash Register	Q-Digest [121]	$O(\epsilon^{-1} \log u)$	Fixed-Universe
		GK Summary [118]	$O(\epsilon^{-1} \log \epsilon n)$	Comparison-Based
	Turnstile	Ganguly & Majumder [100]	$O(\epsilon^{-2} \log^5 u \log(\epsilon^{-1} \log u))$	
	Sliding Window	Arasu & Manku [102]	$O(\epsilon^{-1} \log \epsilon^{-1} \log w)$	
	Distributed Tracking	Yi & Zhang [111]	$O(k/\epsilon \cdot \log N \log^2 \epsilon^{-1})$	
Randomized	Cash Register	KLL Summary [124]	$O(\epsilon^{-1})$	Optimal
	Turnstile	Dyadic Count Sketch [129]	$O(\epsilon^{-1} \log^{1.5} u \log^{1.5}(\epsilon^{-1} \log u))$	
	Distributed One-Shot	Hybrid Quantile [78]	$O(\epsilon^{-1} \log^{1.5} \epsilon^{-1})$	Summary size $O(\epsilon^{-1})$
	Distributed Tracking	Huang et al. [112]	$O(\sqrt{k}/\epsilon \cdot \log N \log^{1.5}(\epsilon\sqrt{k})^{-1})$	

Chapter 6

Conclusion and Open Problems

In this survey, we reviewed summaries for statistical information like distinct count, frequencies, heavy hitters and quantiles. We also discussed about how they can be maintained under different models, including cash register stream, turnstile stream, sliding windows and distributed cases. The following algorithms are implemented in Apache DataSketches [135].

- HyperLogLog and KMV for the distinct count problem.
- A variant of the MG summary for the frequency estimation problem.
- KLL summary for the quantile problem.

The Count(-Min) sketch and T-digest are also widely used in practice. There are some possible directions here.

6.1 Distributed Turnstile Streams

Existing distributed algorithms mostly work on off-line sets or cash register streams. There are some works [69, 136, 137, 138, 139] under the distributed sliding window case but few [140] work for distributed turnstile streams, where each remote site can receive an arbitrary sequence of updates. The problem arises in many real applications, but it is hard under the worst case where the adversary has too much power to manipulate inputs. It may be solvable by making realistic assumptions on the sequence of updates [141] or on the number of deletions [61], so that the algorithm works in practice.

6.2 Multi-Functional Summary

Currently, different summaries accomplish different jobs. This means additional storage and maintenance costs. While a uniform sample can answer most queries, its accuracy is usually poor. There has been an interest [140, 142, 143, 60] in answering different queries by the same summary. It is worth investigating how to achieve a trade-off between the functionality of the summary and its cost, and how to generalize such summaries to different models.

6.3 Summaries under Differential Privacy

There is a recent focus on releasing summaries that satisfy privacy constraints. Differential Privacy [144] has become the standard for privacy analysis, which requires the output distribution

to be similar with or without any individual element. That is, algorithm M satisfies (ϵ, δ)-DP if for any two neighboring datasets $\boldsymbol{x}$ and $\boldsymbol{x}'$,

$$\Pr[M(\boldsymbol{x}) \in O] \leq e^{\epsilon} \cdot \Pr[M(\boldsymbol{x}') \in O] + \delta\,,$$

for any subset of output O. Depending on the definition of neighboring datasets, and the level where this condition holds, there are subcategories like user(node)-DP, event(edge)-DP, centralized-DP, local-DP etc.

In centralized-DP, the algorithm has access to the full dataset $\boldsymbol{x}$. The most efficient solution is to add a noise to the output calibrated to the *global sensitivity* of the query [144], which is the difference of output on neighboring datasets in the worst case. For statistical queries, the sensitivities of distinct count and frequency estimation are both 1, so they can be released accurately even under centralized-DP constraints. For quantile queries like median, its global sensitivity is large. Smooth Sensitivity [145, 146] is proposed to solve such problems.

In local-DP, the algorithm must protect the privacy of individual elements. The frequency estimation problem can be solved [147, 148] using randomized response based ideas, while the distinct count problem is known to be hard [149].

Differential Privacy is also discussed under streaming settings, where the community uses the term *continual observations* [150, 151, 152, 153]; and under distributed (streaming) settings, also named *multi-party* [154, 155, 156]. Most results focus on the (easiest) frequency estimation problem, leaving the problem unsolved for other statistical information, including distinct count and quantiles.

Bibliography

[1] R. H. M. Sr., "Counting large numbers of events in small registers," *Commun. ACM*, vol. 21, no. 10, pp. 840–842, 1978.

[2] C. C. Aggarwal and P. S. Yu, "On classification of high-cardinality data streams," in *Proceedings of the SIAM International Conference on Data Mining, SDM*, pp. 802–813, SIAM, 2010.

[3] C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, and S. Yekhanin, "Pan-private streaming algorithms," in *Innovations in Computer Science - ICS 2010*, pp. 66–80, Tsinghua University Press, 2010.

[4] S. G. Choi, D. Dachman-Soled, M. Kulkarni, and A. Yerukhimovich, "Differentially-private multi-party sketching for large-scale statistics," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 3, pp. 153–174, 2020.

[5] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Surfing wavelets on streams: One-pass summaries for approximate aggregate queries," in *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases*, pp. 79–88, Morgan Kaufmann, 2001.

[6] S. Muthukrishnan, "Data streams: Algorithms and applications," *Found. Trends Theor. Comput. Sci.*, vol. 1, no. 2, 2005.

[7] G. Cormode, S. Muthukrishnan, and W. Zhuang, "What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams," in *Proceedings of the 22nd International Conference on Data Engineering, ICDE*, p. 57, IEEE Computer Society, 2006.

[8] G. Cormode, S. Muthukrishnan, and K. Yi, "Algorithms for distributed functional monitoring," *ACM Trans. Algorithms*, vol. 7, no. 2, pp. 21:1–21:20, 2011.

[9] Apache Flink, "Apache Flink — Stateful Computations over Data Streams."

[10] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference, IMC*, pp. 153–166, ACM, 2003.

[11] E. Cohen, "Size-estimation framework with applications to transitive closure and reachability," *J. Comput. Syst. Sci.*, vol. 55, no. 3, pp. 441–453, 1997.

[12] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan, "Comparing data streams using hamming norms (how to zero in)," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 3, pp. 529–540, 2003.

[13] L. A. Goodman *et al.*, "On the estimation of the number of classes in a population," *The Annals of Mathematical Statistics*, vol. 20, no. 4, pp. 572–579, 1949.

[14] W. Hou, G. Özsoyoglu, and B. K. Taneja, "Statistical estimators for relational algebra expressions," in *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 276–287, 1988.

[15] J. F. Naughton and S. Seshadri, "On estimating the size of projections," in *ICDT'90, Third International Conference on Database Theory*, pp. 499–513, 1990.

[16] A. Chao, "Nonparametric estimation of the number of classes in a population," *Scandinavian Journal of statistics*, pp. 265–270, 1984.

[17] G. Özsoyoglu, K. Du, A. Tjahjana, W. Hou, and D. Y. Rowland, "On estimating count, sum, and AVERAGE," in *Proceedings of the International Conference on Database and Expert Systems Applications*, pp. 406–412, 1991.

[18] A. Chao and S.-M. Lee, "Estimating the number of classes via sample coverage," *Journal of the American statistical Association*, vol. 87, no. 417, pp. 210–217, 1992.

[19] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes, "Sampling-based estimation of the number of distinct values of an attribute," in *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases*, pp. 311–322, 1995.

[20] M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya, "Towards estimation error guarantees for distinct values," in *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 268–279, 2000.

[21] P. B. Gibbons, "Distinct sampling for highly-accurate answers to distinct values queries and event reports," in *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases*, pp. 541–550, 2001.

[22] M. J. Freitag and T. Neumann, "Every row counts: Combining sketches and sampling for accurate group-by result estimates," in *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research*, www.cidrdb.org, 2019.

[23] P. Flajolet and G. N. Martin, "Probabilistic counting," in *24th Annual Symposium on Foundations of Computer Science*, pp. 76–82, IEEE Computer Society, 1983.

[24] M. Durand and P. Flajolet, "Loglog counting of large cardinalities (extended abstract)," in *Algorithms - ESA 2003, 11th Annual European Symposium Proceedings*, vol. 2832 of *Lecture Notes in Computer Science*, pp. 605–617, Springer, 2003.

[25] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *AOFA '07: Proceedings of the 2007 International Conference on Analysis of Algorithms*, 2007.

[26] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 41–52, ACM, 2010.

[27] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm," in *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings*, pp. 683–692, ACM, 2013.

[28] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *Randomization and Approximation Techniques, 6th International Workshop, RANDOM*, vol. 2483 of *Lecture Notes in Computer Science*, pp. 1–10, Springer, 2002.

[29] K. S. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla, "On synopses for distinct-value estimation under multiset operations," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 199–210, ACM, 2007.

[30] K. S. Beyer, R. Gemulla, P. J. Haas, B. Reinwald, and Y. Sismanis, "Distinct-value synopses for multiset operations," *Commun. ACM*, vol. 52, no. 10, pp. 87–95, 2009.

[31] P. Chassaing and L. Gerin, "Efficient estimation of the cardinality of large data sets," *Discrete Mathematics & Theoretical Computer Science*, pp. 419–422, 2006.

[32] F. Giroire, "Order statistics and estimating cardinalities of massive data sets," *Discret. Appl. Math.*, vol. 157, no. 2, pp. 406–427, 2009.

[33] 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*, pp. 20–29, ACM, 1996.

[34] Z. Bar-Yossef, *The Complexity of Massive Data Set Computations*. PhD thesis, UNIVERSITY of CALIFORNIA at BERKELEY, 2002. Page 142.

[35] P. Indyk and D. P. Woodruff, "Tight lower bounds for the distinct elements problem," in *44th Symposium on Foundations of Computer Science*, pp. 283–288, IEEE Computer Society, 2003.

[36] D. P. Woodruff, "Optimal space lower bounds for all frequency moments," in *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pp. 167–175, SIAM, 2004.

[37] T. Jayram, R. Kumar, and D. Sivakumar, "The one-way communication complexity of hamming distance.," *Theory of Computing*, vol. 4, pp. 129–135, 01 2008.

[38] D. P. Woodruff, "The average-case complexity of counting distinct elements," in *Database Theory - ICDT 2009, 12th International Conference*, vol. 361 of *ACM International Conference Proceeding Series*, pp. 284–295, ACM, 2009.

[39] T. S. Jayram and D. P. Woodruff, "Optimal bounds for johnson-lindenstrauss transforms and streaming problems with subconstant error," *ACM Trans. Algorithms*, vol. 9, no. 3, pp. 26:1–26:17, 2013.

[40] J. Blasiok, "Optimal streaming and tracking distinct elements with high probability," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pp. 2432–2448, SIAM, 2018.

[41] J. Blasiok, "Optimal streaming and tracking distinct elements with high probability," *ACM Trans. Algorithms*, vol. 16, no. 1, pp. 3:1–3:28, 2020.

[42] Z. Huang, W. M. Tai, and K. Yi, "Tracking the frequency moments at all times," *CoRR*, vol. abs/1412.1763, 2014.

[43] K. Whang, B. T. V. Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, 1990.

[44] A. Metwally, D. Agrawal, and A. E. Abbadi, "Why go logarithmic if we can go linear?: Towards effective distinct counting of search traffic," in *EDBT 2008, 11th International Conference on Extending Database Technology*, vol. 261 of *ACM International Conference Proceeding Series*, pp. 618–629, ACM, 2008.

[45] P. Flajolet, "On adaptive sampling," *Computing*, vol. 43, no. 4, pp. 391–400, 1990.

[46] Redis, "PFCOUNT."

[47] Amazon Redshift, "Using HyperLogLog sketches in amazon redshift."

[48] Apache Spark, "Spark SQL, Built-in Functions."

[49] Presto, "8.21. HyperLogLog functions."

[50] P. B. Gibbons and S. Tirthapura, "Estimating simple functions on the union of data streams," in *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA*, pp. 281–291, ACM, 2001.

[51] G. Cormode, S. Muthukrishnan, and I. Rozenbaum, "Summarizing and mining inverse distributions on data streams via dynamic inverse sampling," in *Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 25–36, ACM, 2005.

[52] G. Frahling, P. Indyk, and C. Sohler, "Sampling in dynamic data streams and applications," in *Proceedings of the 21st ACM Symposium on Computational Geometry*, pp. 142–149, ACM, 2005.

[53] A. Chen and J. Cao, "Distinct counting with a self-learning bitmap," in *Proceedings of the 25th International Conference on Data Engineering, ICDE*, pp. 1171–1174, IEEE Computer Society, 2009.

[54] A. Chen, J. Cao, L. Shepp, and T. Nguyen, "Distinct counting with a self-learning bitmap," *CoRR*, vol. abs/1107.1697, 2011.

[55] C. Estan, G. Varghese, and M. E. Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, 2006.

[56] D. Ting, "Streamed approximate counting of distinct elements: beating optimal batch methods," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pp. 442–451, ACM, 2014.

[57] D. M. Kane, J. Nelson, and D. P. Woodruff, "On the exact space complexity of sketching and streaming small norms," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pp. 1161–1178, SIAM, 2010.

[58] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM J. Comput.*, vol. 31, no. 6, pp. 1794–1813, 2002.

[59] V. Braverman, E. Grigorescu, H. Lang, D. P. Woodruff, and S. Zhou, "Nearly optimal distinct elements and heavy hitters on sliding windows," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, vol. 116 of *LIPIcs*, pp. 7:1–7:22, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[60] E. Assaf, R. Ben-Basat, G. Einziger, and R. Friedman, "Pay for a sliding bloom filter and get counting, distinct elements, and entropy for free," in *2018 IEEE Conference on Computer Communications*, pp. 2204–2212, IEEE, 2018.

[61] R. Jayaram and D. P. Woodruff, "Data streams with bounded deletions," in *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 341–354, ACM, 2018.

[62] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 182–209, 1985.

[63] A. Shukla, P. Deshpande, J. F. Naughton, and K. Ramasamy, "Storage estimation for multidimensional aggregates in the presence of hierarchies," in *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases*, pp. 522–531, Morgan Kaufmann, 1996.

[64] S. Ganguly, M. N. Garofalakis, and R. Rastogi, "Processing set expressions over continuous update streams," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 265–276, ACM, 2003.

[65] S. Ganguly, M. N. Garofalakis, and R. Rastogi, "Tracking set-expression cardinalities over continuous update streams," *VLDB J.*, vol. 13, no. 4, pp. 354–369, 2004.

[66] S. Ganguly, "Counting distinct items over update streams," in *Algorithms and Computation, 16th International Symposium, ISAAC*, vol. 3827 of *Lecture Notes in Computer Science*, pp. 505–514, Springer, 2005.

[67] S. Ganguly, "Counting distinct items over update streams," *Theor. Comput. Sci.*, vol. 378, no. 3, pp. 211–222, 2007.

[68] A. Nazi, B. Ding, V. R. Narasayya, and S. Chaudhuri, "Efficient estimation of inclusion coefficient using hyperloglog sketches," *Proc. VLDB Endow.*, vol. 11, no. 10, pp. 1097–1109, 2018.

[69] P. B. Gibbons and S. Tirthapura, "Distributed streams algorithms for sliding windows," in *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA*, pp. 63–72, ACM, 2002.

[70] É. Fusy and F. Giroire, "Estimating the number of active flows in a data stream over a sliding window," in *Proceedings of the Fourth Workshop on Analytic Algorithmics and Combinatorics, ANALCO*, pp. 223–231, SIAM, 2007.

[71] Y. Chabchoub and G. Hébrail, "Sliding hyperloglog: Estimating cardinality in a data stream over a sliding window," in *ICDMW 2010, The 10th IEEE International Conference on Data Mining Workshops*, pp. 1297–1303, IEEE Computer Society, 2010.

[72] P. Indyk, "Stable distributions, pseudorandom generators, embeddings, and data stream computation," *J. ACM*, vol. 53, no. 3, pp. 307–323, 2006.

[73] G. Cormode, S. Muthukrishnan, and K. Yi, "Algorithms for distributed functional monitoring," in *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pp. 1076–1085, SIAM, 2008.

[74] C. Arackaparambil, J. Brody, and A. Chakrabarti, "Functional monitoring without monotonicity," in *Automata, Languages and Programming, 36th International Colloquium, ICALP*, vol. 5555 of *Lecture Notes in Computer Science*, pp. 95–106, Springer, 2009.

[75] A. Chakrabarti and O. Regev, "An optimal lower bound on the communication complexity of gap-hamming-distance," in *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC*, pp. 51–60, ACM, 2011.

[76] D. P. Woodruff and Q. Zhang, "Tight bounds for distributed functional monitoring," *CoRR*, vol. abs/1112.5153, 2011.

[77] D. P. Woodruff and Q. Zhang, "An optimal lower bound for distinct elements in the message passing model," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pp. 718–733, SIAM, 2014.

[78] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi, "Mergeable summaries," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 23–34, ACM, 2012.

[79] D. Ting, "Towards optimal cardinality estimation of unions and intersections with sketches," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1195–1204, ACM, 2016.

[80] A. Das, S. Ganguly, M. N. Garofalakis, and R. Rastogi, "Distributed set expression cardinality estimation," in *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB*, pp. 312–323, Morgan Kaufmann, 2004.

[81] G. Cormode, "Sketch techniques for approximate query processing," *Foundations and Trends in Databases. NOW publishers*, 2011.

[82] P. B. Gibbons, "Distinct-values estimation over data streams," in *Data Stream Management - Processing High-Speed Data Streams*, Data-Centric Systems and Applications, pp. 121–147, Springer, 2016.

[83] G. Cormode and K. Yi, *Small Summaries for Big Data*. Cambridge University Press, 2020.

[84] W. Chen and Y. Guan, "Distinct element counting in distributed dynamic data streams," in *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*, pp. 2371–2379, IEEE, 2015.

[85] P. Flajolet, "Approximate counting: A detailed analysis," *BIT Comput. Sci. Sect.*, vol. 25, no. 1, pp. 113–134, 1985.

[86] A. Gronemeier and M. Sauerhoff, "Applying approximate counting for computing the frequency moments of long data streams," *Theory Comput. Syst.*, vol. 44, no. 3, pp. 332–348, 2009.

[87] J. Nelson and H. Yu, "Optimal bounds for approximate counting," *CoRR*, vol. abs/2010.02116, 2020.

[88] P. Bose, E. Kranakis, P. Morin, and Y. Tang, "Bounds for frequency estimation of packet streams," in *SIROCCO 10: Proceedings of the 10th Internaltional Colloquium on Structural Information Complexity*, vol. 17 of *Proceedings in Informatics*, pp. 33–42, Carleton Scientific, 2003.

[89] J. Misra and D. Gries, "Finding repeated elements," *Sci. Comput. Program.*, vol. 2, no. 2, pp. 143–152, 1982.

[90] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Database Theory - ICDT 2005, 10th International Conference*, vol. 3363 of *Lecture Notes in Computer Science*, pp. 398–412, Springer, 2005.

[91] R. Berinde, G. Cormode, P. Indyk, and M. J. Strauss, "Space-optimal heavy hitters with strong error bounds," in *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*, pp. 157–166, ACM, 2009.

[92] M. Charikar, K. C. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Automata, Languages and Programming, 29th International Colloquium, ICALP*, vol. 2380 of *Lecture Notes in Computer Science*, pp. 693–703, Springer, 2002.

[93] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," in *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium*, vol. 2976 of *Lecture Notes in Computer Science*, pp. 29–38, Springer, 2004.

[94] J. S. Moore, "A fast majority vote algorithm," tech. rep., Automated Reasoning: Essays in Honor of Woody Bledsoe, 1981.

[95] R. S. Boyer and J. S. Moore, "MJRTY: A fast majority vote algorithm," in *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pp. 105–118, Kluwer Academic Publishers, 1991.

[96] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," in *Algorithms - ESA 2002, 10th Annual European Symposium*, vol. 2461 of *Lecture Notes in Computer Science*, pp. 348–360, Springer, 2002.

[97] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Trans. Database Syst.*, vol. 28, pp. 51–55, 2003.

[98] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proceedings of 28th International Conference on Very Large Data Bases, VLDB*, pp. 346–357, Morgan Kaufmann, 2002.

[99] D. Ting, "Data sketches for disaggregated subset sum and frequent item estimation," in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pp. 1129–1140, ACM, 2018.

[100] S. Ganguly and A. Majumder, "Cr-precis: A deterministic summary structure for update data streams," in *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, First International Symposium, ESCAPE 2007, Hangzhou, China, April 7-9, 2007, Revised Selected Papers*, vol. 4614 of *Lecture Notes in Computer Science*, pp. 48–59, Springer, 2007.

[101] S. Ganguly, "Lower bounds on frequency estimation of data streams (extended abstract)," in *Computer Science - Theory and Applications, Third International Computer Science Symposium in Russia, CSR 2008, Moscow, Russia, June 7-12, 2008, Proceedings*, vol. 5010 of *Lecture Notes in Computer Science*, pp. 204–215, Springer, 2008.

[102] A. Arasu and G. S. Manku, "Approximate counts and quantiles over sliding windows," in *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 286–296, ACM, 2004.

[103] L. Lee and H. F. Ting, "A simpler and more efficient deterministic scheme for finding frequent items over sliding windows," in *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 290–297, ACM, 2006.

[104] L. Zhang and Y. Guan, "Frequency estimation over sliding windows," in *Proceedings of the 24th International Conference on Data Engineering*, pp. 1385–1387, IEEE Computer Society, 2008.

[105] R. Y. S. Hung, L. Lee, and H. Ting, "Finding frequent items over sliding windows with constant update time," *Inf. Process. Lett.*, vol. 110, no. 7, pp. 257–260, 2010.

[106] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in *35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, IEEE, 2016.

[107] N. Rivetti, Y. Busnel, and A. Mostéfaoui, "Efficiently summarizing data streams over sliding windows," in *14th IEEE International Symposium on Network Computing and Applications, NCA*, pp. 151–158, IEEE Computer Society, 2015.

[108] Q. Zhao, M. Ogihara, H. Wang, and J. J. Xu, "Finding global icebergs over distributed data sets," in *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 298–307, ACM, 2006.

[109] Z. Huang, K. Yi, Y. Liu, and G. Chen, "Optimal sampling algorithms for frequency estimation in distributed data," in *INFOCOM 2011. 30th IEEE International Conference on Computer Communications*, pp. 1997–2005, IEEE, 2011.

[110] D. P. Woodruff and Q. Zhang, "Tight bounds for distributed functional monitoring," in *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pp. 941–960, ACM, 2012.

[111] K. Yi and Q. Zhang, "Optimal tracking of distributed heavy hitters and quantiles," in *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pp. 167–174, ACM, 2009.

[112] Z. Huang, K. Yi, and Q. Zhang, "Randomized algorithms for tracking distributed count, frequencies, and ranks," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pp. 295–306, ACM, 2012.

[113] 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, VLDB 2002, Hong Kong, August 20-23, 2002*, pp. 454–465, Morgan Kaufmann, 2002.

[114] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[115] G. Cormode and M. Hadjieleftheriou, "Finding the frequent items in streams of data," *Commun. ACM*, vol. 52, no. 10, pp. 97–105, 2009.

[116] J. I. Munro and M. Paterson, "Selection and sorting with limited storage," in *19th Annual Symposium on Foundations of Computer Science*, pp. 253–258, IEEE Computer Society, 1978.

[117] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate medians and other quantiles in one pass and with limited memory," in *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pp. 426–435, ACM Press, 1998.

[118] M. Greenwald and S. Khanna, "Space-efficient online computation of quantile summaries," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pp. 58–66, ACM, 2001.

[119] R. Y. S. Hung and H. Ting, "An w(\frac1e log\frac1e)\omega(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}) space lower bound for finding *epsilon*-approximate quantiles in a data stream," in *Frontiers in Algorithmics, 4th International Workshop, FAW 2010*, vol. 6213 of *Lecture Notes in Computer Science*, pp. 89–100, Springer, 2010.

[120] G. Cormode and P. Veselý, "A tight lower bound for comparison-based quantile summaries," in *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database System*, pp. 81–93, ACM, 2020.

[121] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: new aggregation techniques for sensor networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pp. 239–249, ACM, 2004.

[122] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Random sampling techniques for space efficient online computation of order statistics of large datasets," in *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, pp. 251–262, ACM Press, 1999.

[123] D. Felber and R. Ostrovsky, "A randomized online quantile summary in o(1/epsilon * log(1/epsilon)) words," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, vol. 40 of *LIPIcs*, pp. 775–785, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[124] Z. S. Karnin, K. J. Lang, and E. Liberty, "Optimal quantile approximation in streams," in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS*, pp. 71–78, IEEE Computer Society, 2016.

[125] G. Cormode, Z. S. Karnin, E. Liberty, J. Thaler, and P. Veselý, "Relative error streaming quantiles," *CoRR*, vol. abs/2004.01668, 2020.

[126] Y. Tao, K. Yi, C. Sheng, J. Pei, and F. Li, "Logging every footstep: quantile summaries for the entire history," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD*, pp. 639–650, ACM, 2010.

[127] T. Dunning and O. Ertl, "Computing extremely accurate quantiles using t-digests," *CoRR*, vol. abs/1902.04023, 2019.

[128] T. Dunning, "The t-digest: Efficient estimates of distributions," *Softw. Impacts*, vol. 7, p. 100049, 2021.

[129] G. Luo, L. Wang, K. Yi, and G. Cormode, "Quantiles over data streams: experimental comparisons, new analyses, and further improvements," *VLDB J.*, vol. 25, no. 4, pp. 449–472, 2016.

[130] X. Lin, H. Lu, J. Xu, and J. X. Yu, "Continuously maintaining quantile summaries of the most recent N elements over a data stream," in *Proceedings of the 20th International Conference on Data Engineering, ICDE*, pp. 362–373, IEEE Computer Society, 2004.

[131] M. Greenwald and S. Khanna, "Power-conserving computation of order-statistics over sensor networks," in *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 275–285, ACM, 2004.

[132] C. Buragohain and S. Suri, "Quantiles on streams," in *Encyclopedia of Database Systems, Second Edition*, Springer, 2018.

[133] N. Ivkin, E. Liberty, K. J. Lang, Z. S. Karnin, and V. Braverman, "Streaming quantiles algorithms with small space and update time," *CoRR*, vol. abs/1907.00236, 2019.

[134] G. Cormode, T. Kulkarni, and D. Srivastava, "Constrained private mechanisms for count data," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 2, pp. 415–430, 2021.

[135] Apache, "DataSketches."

[136] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Optimal sampling from distributed streams," in *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pp. 77–86, ACM, 2010.

[137] H. Chan, T. W. Lam, L. Lee, and H. Ting, "Continuous monitoring of distributed data streams over a time-based sliding window," in *27th International Symposium on Theoretical Aspects of Computer Science, STACS*, vol. 5 of *LIPIcs*, pp. 179–190, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

[138] G. Cormode and K. Yi, "Tracking distributed aggregates over time-based sliding windows," in *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing*, pp. 213–214, ACM, 2011.

[139] S. Gayen and N. V. Vinodchandran, "New algorithms for distributed sliding windows," in *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT*, vol. 101 of *LIPIcs*, pp. 22:1–22:15, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[140] G. Cormode and M. N. Garofalakis, "Approximate continuous querying over distributed streams," *ACM Trans. Database Syst.*, vol. 33, no. 2, pp. 9:1–9:39, 2008.

[141] K. Yi, H. Yu, J. Yang, G. Xia, and Y. Chen, "Efficient maintenance of materialized top-k views," in *Proceedings of the 19th International Conference on Data Engineering*, pp. 189–200, IEEE Computer Society, 2003.

[142] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI*, pp. 29–42, USENIX Association, 2013.

[143] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proceedings of the ACM SIGCOMM*, pp. 101–114, ACM, 2016.

[144] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography, Third Theory of Cryptography Conference, TCC*, vol. 3876 of *Lecture Notes in Computer Science*, pp. 265–284, Springer, 2006.

[145] K. Nissim, S. Raskhodnikova, and A. D. Smith, "Smooth sensitivity and sampling in private data analysis," in *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pp. 75–84, ACM, 2007.

[146] N. M. Johnson, J. P. Near, and D. Song, "Towards practical differential privacy for SQL queries," *Proc. VLDB Endow.*, vol. 11, no. 5, pp. 526–539, 2018.

[147] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *26th USENIX Security Symposium, USENIX*, pp. 729–745, USENIX Association, 2017.

[148] T. Wang, M. Lopuhaä-Zwakenberg, Z. Li, B. Skoric, and N. Li, "Locally differentially private frequency estimation with consistency," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*, The Internet Society, 2020.

[149] L. Chen, B. Ghazi, R. Kumar, and P. Manurangsi, "On distributed differential privacy and counting distinct elements," in *12th Innovations in Theoretical Computer Science Conference, ITCS*, vol. 185 of *LIPIcs*, pp. 56:1–56:18, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[150] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum, "Differential privacy under continual observation," in *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pp. 715–724, ACM, 2010.

[151] T. H. Chan, E. Shi, and D. Song, "Private and continual release of statistics," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 3, pp. 26:1–26:24, 2011.

[152] T. H. Chan, E. Shi, and D. Song, "Privacy-preserving stream aggregation with fault tolerance," in *Financial Cryptography and Data Security - 16th International Conference*, vol. 7397 of *Lecture Notes in Computer Science*, pp. 200–214, Springer, 2012.

[153] C. Dwork, M. Naor, O. Reingold, and G. N. Rothblum, "Pure differential privacy for rectangle queries via private partitions," in *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security*, vol. 9453 of *Lecture Notes in Computer Science*, pp. 735–751, Springer, 2015.

[154] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, vol. 4004 of *Lecture Notes in Computer Science*, pp. 486–503, Springer, 2006.

[155] T. H. Chan, M. Li, E. Shi, and W. Xu, "Differentially private continual monitoring of heavy hitters from distributed streams," in *Privacy Enhancing Technologies - 12th International Symposium, PETS*, vol. 7384 of *Lecture Notes in Computer Science*, pp. 140–159, Springer, 2012.

[156] E. Shi, T. H. Chan, E. G. Rieffel, and D. Song, "Distributed private data analysis: Lower bounds and practical constructions," *ACM Trans. Algorithms*, vol. 13, no. 4, pp. 50:1–50:38, 2017.