Data Streams: Algorithms and Applications[1]

Data Streams: Algorithms and Applications[2]

now

the essence of knowledge

Contents

0.1 Abstract

In the data stream scenario, input arrives very rapidly and there is limited memory to store the input. Algorithms have to work with one or few passes over the data, space less than linear in the input size or time significantly less than the input size. In the past few years, a new theory has emerged for reasoning about algorithms that work within these constraints on space, time, and number of passes. Some of the methods rely on metric embeddings, pseudo-random computations, sparse approximation theory and communication complexity. The applications for this scenario include IP network traffic analysis, mining text message streams and processing massive data sets in general. Researchers in Theoretical Computer Science, Databases, IP Networking and Computer Systems are working on the data stream challenges. This article is an overview and survey of data stream algorithmics and is an updated version of [173].

0.2 Introduction

We study the emerging area of algorithms for processing data streams and associated applications, as an applied algorithms research agenda. We begin with three puzzles.

0.2.1 Puzzle 1: Finding Missing Numbers

Let π be a permutation of $\{1, \ldots, n\}$. Further, let π_{-1} be π with one element missing. Paul shows Carole $\pi_{-1}[i]$ in increasing order i. Carole's task is to determine the missing integer. It is trivial to do the task if Carole can memorize all the numbers she has seen thus far (formally, she has an n-bit vector), but if n is large, this is impractical. Let us assume she has only a few—say $O(\log n)$—bits of memory. Nevertheless, Carole must determine the missing integer.

This starter puzzle has a simple solution: Carole stores

$$s = \frac{n(n+1)}{2} - \sum_{j \le i} \pi_{-1}[j],$$

which is the missing integer in the end. Each input integer entails one subtraction. The total number of bits stored is no more than $2 \log n$. This is nearly optimal because Carole needs at least $\log n$ bits in the worst case since she needs to output the missing integer. (In fact, there exists the following optimal algorithm for Carole using $\log n$ bits. For each i, store the parity sum of the ith bits of all numbers seen thus far. The final parity sum bits are the bits of the missing number.) A similar solution will work even if n is unknown, for example by letting $n = \max_{j \le i} \pi_{-1}[j]$ each time.

Paul and Carole have a history. It started with the "twenty questions" problem solved in [198]. Paul, which stood for Paul Erdos, was the one who asked questions. Carole is an anagram for Oracle. Aptly, she was the one who answered questions. Joel Spencer and Peter Winkler used Paul and Carole to coincide with Pusher and Chooser respectively in studying certain chip games in which Carole chose which groups the chips falls into and Paul determined which group of chips to push. In the puzzle above, Paul permutes and Carole cumulates.

Generalizing the puzzle a little further, let π_{-2} be π with two elements missing. The natural solution would be for Carole to store $s = \frac{n(n+1)}{2} - \sum_{j \le i} \pi_{-2}[j]$ and $p = n! - \prod_{j \le i} \pi_{-2}[j]$, giving two equations with two unknown numbers, but this will result in storing large number of bits since $n!$ is large. Instead, Carole can use far fewer bits tracking

$$s = \frac{n(n+1)}{2} - \sum_{j \le i} \pi_{-2}[j] \quad \text{and} \quad ss = \frac{n(n+1)(2n+1)}{6} - \sum_{j \le i} (\pi_{-2}[j])^2$$

In general, what is the smallest number of bits needed to identify the k missing numbers in π_{-k}? Following the approach above, the solution is to maintain *power sums*

$$s_p(x_1, \ldots, x_k) = \sum_{i=1}^{k} (x_i)^p,$$

for $p = 1, \ldots, k$ and solving for x_i's. A different method uses *elementary symmetric polynomials* [167]. The ith such polynomial $\sigma_i(x_1, \ldots, x_k)$ is the sum of all possible i term products of the parameters, i.e.,

$$\sigma_i(x_1, \ldots, x_k) = \sum_{j_1 < \ldots < j_i} x_{j_1} \cdots x_{j_i}.$$

Carole continuously maintains σ_i's for the missing k items in field F_q for some prime $n \le q \le 2n$, as Paul presents the numbers one after the other (the details are in [167]). Since

$$\prod_{i=1,\ldots,k} (z - x_i) = \sum_{i=0}^{k} (-1)^i \sigma_i(x_1, \ldots, x_k) z^{k-i},$$

Carole needs to factor this polynomial in F_q to determine the missing numbers. No deterministic algorithms are known for the factoring problem, but there are randomized algorithms take roughly $O(k^2 \log n)$ bits and time [210]. The elementary symmetric polynomial approach above comes from [167] where the authors solve the *set reconciliation problem* in the communication complexity model. The *subset* reconciliation problem is related to our puzzle.

Generalizing the puzzle, Paul may present a multiset of elements in $\{1, \cdots, n\}$ with a single missing integer, i.e., he is allowed to *re*-present integers he showed before; Paul may present updates showing which integers to insert and which to delete, and Carole's task is to find the integers that are no longer present; etc. All of these problems are no longer (mere) puzzles; they are derived from motivating data stream applications.

0.2.2 Puzzle 2: Fishing

Say Paul goes fishing. There are many different fish species $U = \{1, \cdots, u\}$. Paul catches one fish at a time, $a_t \in U$ being the fish species he catches at time t. $c_t[j] = |\{a_i | a_i = j,\ i \leq t\}|$ is the number of times he catches the species j up to time t. Species j is *rare* at time t if it appears precisely once in his catch up to time t. The *rarity* $\rho[t]$ of his catch at time t is the ratio of the number of rare j's to u:

$$\rho[t] = \frac{|\{\ j \mid c_t[j] = 1\ \}|}{u}.$$

Paul can calculate $\rho[t]$ precisely with a $2u$-bit vector and a counter for the current number of rare species, updating the data structure in $O(1)$ operations per fish caught. However, Paul wants to store only as many bits as will fit his tiny suitcase, i.e., $o(u)$, preferably $O(1)$ bits.

Suppose Paul has a deterministic algorithm to compute $\rho[t]$ precisely. Feed Paul any set $S \subset U$ of fish species, and say Paul's algorithm stores only $o(u)$ bits in his suitcase. Now we can check if any $i \in S$ by simply feeding Paul i and checking $\rho[t+1]$: the number of rare items *de*creases by one if and only if $i \in S$. This way we can recover entire S from his suitcase by feeding different i's one at a time, which is impossible in general if Paul had only stored $o(|S|)$ bits. Therefore, if Paul wishes to work out of his one suitcase, he can not compute $\rho[t]$ exactly. This argument has elements of lower bound proofs found in the area of data streams.

However, proceeding to the task at hand, Paul can *approximate* $\rho[t]$. Paul picks k random fish species each independently, randomly with probability $1/u$ at the beginning and maintains the number of times each of these fish types appear in his bounty, as he catches fish one after another. Say $X_1[t], \ldots, X_k[t]$ are these counts after time t. Paul outputs $\hat{\rho}[t] = \frac{|\{\ i \mid X_i[t]=1\ \}|}{k}$ as an estimator for ρ. We have,

$$\mathsf{Pr}(X_i[t] = 1) = \frac{|\{j \mid c_t[j] = 1\}|}{u} = \rho[t],$$

for any fixed i and the probability is over the fish type X_i. If $\rho[t]$ is large, say at least $1/k$, $\hat{\rho}[t]$ is a good estimator for $\rho[t]$ with arbitrarily small ε and significant probability.

However, typically, ρ is unlikely to be large because presumably u is much larger than the species found at any spot Paul fishes. Choosing a random species from $\{1, \ldots, u\}$ and waiting for it to be caught is ineffective. We can make it more realistic by redefining rarity with respect to the species Paul in fact sees in his catch. Let

$$\gamma[t] = \frac{|\{\ j \mid c_t[j] = 1\ \}|}{|\{\ j \mid c_t[j] \neq 0\ \}|}.$$

As before, Paul would have to approximate $\gamma[t]$ because he can not compute it exactly using a small number of bits. Following [24], define a family of hash functions $\mathcal{H} \subset [n] \to [n]$ (where $[n] = \{1, \ldots, n\}$) to be

min-wise independent if for any $X \subset [n]$ and $x \in X$, we have

$$\mathsf{Pr}_{h \in \mathcal{H}} [h(x) = \min\ h(X)] = \frac{1}{|X|},$$

where, $h(X) = \{h(x) : x \in X\}$. Paul chooses k min-wise independent hash functions $h_1, h_2, \ldots, h_k$ for some parameter k to be determined later and maintains $h_i^*(t) = \min_{j \leq t}\ h_i(a_j)$ at each time t, that is, min hash value of the multi-set $\{\ldots, a_{t-2}, a_{t-1}, a_t\}$. He also maintain k counters $C_1(t), C_2(t), \ldots, C_k(t)$; $C_i(t)$ counts an item with (the current value of) hash value $h_i^*(t)$ in $\{\ldots, a_{t-2}, a_{t-1}, a_t\}$. It is trivial to maintain both $h_i^*(t)$ and $C_i(t)$ as t progresses and new items are seen. Let

$$\hat{\gamma}[t] = \frac{|\{\ i \mid 1 \leq i \leq k,\ C_i(t) = 1\ \}|}{k}.$$

Notice that $\mathsf{Pr}(C_i(t) = 1)$ is the probability that $h_i(t)$ is the hash value of one of the items that appeared precisely once in $a_1, ..., a_t$ which equals $\frac{|\{\ j \mid c[j]=1\ \}|}{|\{\ j \mid c[j]\neq 0\ \}|} = \gamma[t]$. Hence, $\hat{\gamma}[t]$ is a good estimator for $\gamma[t]$ provided $\gamma[t]$ is large, say at least $1/k$. That completes the sketch of the Paul's algorithm.

The remaining detail is that Paul needs to pick h_i's. If Paul resorts to his tendency to permute, i.e., if he picks a randomly chosen permutation π over $[u] = \{1, \ldots, u\}$, then h_i's will be min-wise hash functions. However, it requires $\Theta(u \log u)$ bits to represent a random permutation from the set of all permutations over $[u]$. Thus the number of bits needed to store the hash function will be close to u which is prohibitive!

To overcome this problem, Paul picks a family of *approximate* min-hash functions. A family of hash functions, $\mathcal{H} \subset [n] \to [n]$ is called *ϵ-min-wise independent* if for any $X \subset [n]$ and $x \in X$, we have

$$\mathsf{Pr}_{h \in \mathcal{H}} [h(x) = \min\ h(X)] = \frac{1}{|X|}(1 \pm \epsilon).$$

Indyk [132] presents a family of ϵ-min-wise independent hash functions such that any function from this family can be represented using $O(\log u \log(1/\epsilon))$ bits only and each hash function can be computed efficiently in $O(\log(1/\epsilon))$ time. This family is a set of polynomials over $GF(u)$ of degree $O(\log(1/\epsilon))$. Plugging this result into the solution above, Paul uses $O(k \log u \log(1/\epsilon))$ bits and estimates $\hat{\gamma}[t] \in (1 \pm \epsilon)\gamma[t]$, provided $\gamma[t]$ is large, that is, at least $1/k$. It will turn out that in applications of streaming interest, we need to only determine if $\gamma[t]$ is large, so this solution will do.

As an aside, the problem of estimating the rarity is related to a different problem. Consider fishing again and think of it as a random sampling process. There is an unknown probability distribution P on the countable set of fish types with p_t being the probability associated with fish type t. A *catch* is a sample S fishes drawn independently from fish types according to the distribution P. Let $c[t]$ be the number of times fish type t appears in S. The problem is to estimate the probability of fish type t being the next catch. Elementary reasoning would indicate that this probability is $c[t]/|S|$. However, it is unlikely that all (of the large number of) fish types in the ocean are seen in Paul's catch, or even impossible if the number of fish types is infinite. Hence, there are fish types t^* that do *not* appear in the sample (i.e., $c[t^*] = 0$) and the elementary reasoning above would indicate that they have probability 0 of being caught next. This is a conundrum in the elementary reasoning since t^* is indeed present in the ocean and has nonzero probability of being caught in a given probability distribution P. Let $m = \sum_{t^* \notin S} p_t^*$. The problem of estimating m is called the *missing mass problem*. In a classical work by Good (attributed to Turing too) [111], it is shown that m is estimated by $s[1]/|S|$, where $s[k]$ is the number of fish types that appear k times in S, provably with small bias; recall that our rarity γ is closely related to $s[1]/|S|$. Hence, our result here on estimating rarity in data streams is of independent interest in estimating the missing mass. See recent work on Turing-Good estimators in [183].

Once we generalize the fishing puzzle—letting the numerator be more generally $|\{\ j \mid q[j] \leq \alpha\ \}|$ for some α, letting Carole go fishing too, or letting Paul and Carole throw fish back into the Ocean as needed—there are some real data streaming applications [67]. In the reality of data streams, one is confronted with fishing in a far larger domain, that is, u is typically very large.

0.2.3 Puzzle 3: Pointer and Chaser

We study yet another game between Paul and Carole. There are $n+1$ positions numbered $1, ..., n+1$ and for each position i, Paul points to some position given by $P[i] \in \{1, 2, \cdots, n\}$. By the Pigeonhole principle, there must exist at least two positions i, j such that $P[i] = P[j] = D$ say, for a *duplicate*. Several different duplicates may exist, and the same duplicate may appear many times since P is an arbitrary many-to-one function. Carole's goal is to find *any* one duplicate using $O(\log n)$ bits of storage and using no more than $O(n)$ queries to Paul.

The trivial solution to this problem is to take each item $i \in \{1, \ldots, n\}$ in turn and count how many positions j have $P[j] = i$ by querying $P[1], P[2], \ldots, P[n+1]$ in turn. This solution takes $O(\log n)$ bits of extra storage but needs $\Theta(n)$ queries per i in the worst case for a total of $O(n^2)$ queries in all. One of the interesting aspects of this solution is that P is accessed in *passes*, that is, we query $P[1], P[2], \ldots, P[n+1]$ in order and repeat that many times.

One suspects that this problem should be solvable along the lines in Section 0.2.1 using a few passes. The only such solution we know is not optimally efficient and works as follows. In the first pass by querying $P[1], P[2], \ldots, P[n+1]$ in order, Carole counts the number of items below $n/2$ and those above $n/2$. Whichever counter is strictly larger than $n/2$ (one such counter exists) shows a range of size $n/2$ with a duplicate. Now we recurse on the range with a duplicate in the next pass and so on. This method uses only $O(\log n)$ bits of storage (2 counters together with the endpoints of the range of interest in the current pass), but needs $O(n)$ queries per pass, taking $O(n \log n)$ queries in all. Further, this solution uses $O(\log n)$ passes. This solution can be generalized to use $O(k \log n)$ bits of storage and use only $O(\log_k n)$ passes. As it is, this approach similar to Section 0.2.1 does not meet the desired bound of $O(n)$ queries.

Jun Tarui [202] has presented a lower bound on the number of passes needed to solve this problem. Consider odd n and a restricted class of inputs such that the numbers in the first half (and likewise the second half) are distinct. Hence the duplicate item must appear once in the first half and once in the second half. A solution with s bits of memory and r passes implies a two-party protocol with r rounds and s communication bits in each round for the game where Paul and Carole get $(n+1)/2$-sized subsets PA and CA respectively of $\{1, ..., n\}$ and the task is to find and agree on some w that is in the intersection of PA and CA. Such a protocol corresponds to a monotone (i.e. AND/OR) circuit computing the Majority function with depth at most r and fan-in at most 2^s. This leads to a lower bound of $\Omega(\log n / \log \log n)$ passes for $s = O(\log n)$.

In the final solution, Carole does not count, but chases pointers. Start chasing the pointers from $X = n+1$ going successively to the location pointed to by $P[X]$ for current X. Now the problem of finding a duplicate is the same as that of finding if a "linked list" has a loop not containing the start "node" $n+1$. This is easy to solve in $O(n)$ queries to P with 2 pointers. Notice that this solution makes use of the random access given by $P[X]$. (As an aside, the problem is interesting if $P[i] \in S$ where S is *some* set of size n, not necessarily $\{1, 2, \ldots, n\}$. Then, we do not know of an algorithm that takes less than $O(n^2)$ time within our space constraints.)

This puzzle is related to Pollard's rho method for determining the greatest common divisor between integers [91]. The primary focus here is on separating the complexity of the problem using passes vs using random accesses.

0.2.4 Lessons

The missing-number puzzle in Section 0.2.1 shows the case of a data stream problem that can be deterministically solved precisely with $O(\log n)$ bits (when $k = 1, 2$ etc.). Such algorithms—deterministic and exact—are uncommon in data stream processing. In contrast, the puzzle in Section 0.2.2 is solved only up to an approximation using a randomized algorithm in polylog bits. This—randomized and approximate solution—is more representative of currently known data stream algorithms. Further, the estimation of γ in Section 0.2.2 is accurate *only* when it is large; for small γ, the estimate $\hat{\gamma}$ is arbitrarily bad. This points to a feature that generally underlies data stream algorithmics. Such features—which applied algorithmicists need to keep in mind while formulating problems to address data stream issues—will be discussed in more detail later. In Section 0.2.3, the puzzle demonstrates the difference between passes (the best known solution using only passes takes $O(n \log n)$ time) and random access (solution takes $O(n)$ time using random access).

0.3 Map

Section 0.4 describes the data stream phenomenon, and avoids specific models here because the phenomenon is real. Models are the means and may change over time. Section 0.5 will present currently popular data stream models, motivating scenarios and other applications for algorithms in these models beyond dealing with data streams.

Section 0.6 abstracts mathematical ideas, algorithmic techniques as well as lower bound approaches for data stream models; together they comprise the foundation of the theory of data streams. Section 0.7 discusses applied work on data streams from different systems areas.

Section 0.8 contains new directions and open problems that arise in several research areas when the data streaming perspective is applied. Some traditional areas get enriched, new ones emerge. Conclusions are in Section 0.10.

0.4 The Data Stream Phenomenon

Data stream represents input data that comes at a very high rate. High rate means it stresses communication and computing infrastructure, so it may be hard to

- *transmit* (T) the entire input to the program,
- *compute* (C) sophisticated functions on large pieces of the input at the rate it is presented, and
- *store* (S), capture temporarily or archive all of it long term.

Most people typically do not think of this level of stress in TCS capacity. They view data as being stored in files. When transmitted, if links are slow or communication is erroneous, we may have delays but correct and complete data eventually gets to where it should go. If computing power is limited or the program has high complexity, it takes long time to get the desired response, but in principle, we would get it. Also, we save all the data we need. This simplified picture of TCS requirements is reasonable because so far, the need has balanced available resources: we have produced the amount of data that technology we could ship, process, store, or we have the patience to manipulate.

There are two recent developments that have confluenced to produce new challenges to the TCS infrastructure.

- *Ability to generate automatic, highly detailed data feeds comprising continuous updates.*

  This ability has been built up over the past few decades beginning with networks that spanned banking and credit transactions. Other dedicated network systems now provide massive data

streams: satellite-based high resolution measurement of earth geodetics, radar derived meteorological data, continuous large scale astronomical surveys in optical, infrared and radio wavelengths, and atmospheric radiation measurements.

The Internet is a general purpose network system that has distributed both the data sources as well as the data consumers over millions of users. It has scaled up the rate of transactions tremendously generating multiple streams: browser clicks, user queries, IP traffic logs, email data and traffic logs, web server and peer-to-peer downloads etc. The Internet also makes it to easier to deploy special purpose, continuous observation points that get aggregated into vast data streams: for example, financial data comprising individual stock, bond, securities and currency trades can now get accumulated from multiple sources over the internet into massive streams. Wireless access networks are now in the threshold of scaling this phenomenon even more. In particular, the emerging vision of sensor networks combines orders of more observation points (sensors) than are now available with wireless and networking technology and is posited to challenge TCS needs even more. Oceanographic, bio, seismic and security sensors are such emerging examples. Mark Hansen's recent keynote talk discusses Slogging, citizen-initiated sensing, in good detail [123].

- *Need for sophisticated analyses of update streams in near-real time*.

  With traditional data feeds, one modifies the underlying data to reflect the updates, and real time queries are fairly simple such as looking up a value. This is true for the banking and credit transactions. More complex analyses such as trend analysis and forecasting are typically performed offline in warehouses. However, the automatic data feeds that generate modern data streams arise out of monitoring applications, be they atmospheric, astronomical, networking, financial or sensor-related. They need to detect outliers, extreme events, fraud, intrusion, and unusual or anomalous activity, monitor complex correlations, track trends, support exploratory analyses and perform complex tasks such as classification and signal analysis. These are time critical tasks and need to be done in near-real time to accurately keep pace with the rate of stream updates and reflect rapidly changing trends in the data.

These two factors uniquely challenge the TCS needs. We in the computer science community have traditionally focused on scaling in size: how to efficiently manipulate large disk-bound data via suitable data structures [211], how to scale to databases of petabytes [112], synthesize massive data sets [113], etc. However, far less attention has been given to benchmarking, studying performance of systems under rapid updates with *near-real time analyses*. Even benchmarks of database transactions [217] are inadequate.

There are ways to build workable systems around these TCS challenges. TCS systems are sophisticated and have developed high-level principles that still apply. *Make things parallel*. A lot of data stream processing is highly parallelizable in computing (C) and storage (S) capacity; it is somewhat harder to parallelize transmission (T) capacity on demand. *Control data rate by sampling or shedding updates*. High energy particle physics experiments at Fermilab and CERN will soon produce 40TBytes/s which will be reduced by real time hardware into 800Gb/s data stream. Statisticians have the sampling theory;it is now getting applied to IP network streams. *Round data structures to certain block boundaries*. For example, the "Community of Interest" approach to finding fraud in telephone calls uses a graph data structure up to and including the previous day to perform the current day's analysis, thereby rounding the freshness of the analysis to period of a day [62]. This is an effective way to control the rate of real-time data processing and use pipelining. *Use hierarchically detailed analysis*. Use fast but simple filtering or aggregation at the lowest level and slower but more sophisticated computation at higher levels with smaller data. This hierarchical thinking stacks up against memory hierarchy nicely. Finally, often asking imaginative questions can lead to effective solutions

within any given resource constraint, as applied algorithms researchers well know.

Nevertheless, these natural approaches are ultimately limiting. They may meet ones' expectations in short term, but we need to understand the full potential of data streams for the future. Given a certain amount of resources, a data stream rate and a particular analysis task, what can we (not) do? Most natural approaches to dealing with data streams discussed above involves approximations: what are algorithmic principles for data stream approximation? One needs a systematic theory of data streams to get novel algorithms and to build data stream applications with ease and proven performance. What follows is an introduction to the emerging theory of data streams.

The previous few paragraphs presented a case for data stream research. Readers should use their imagination and intuit the implications of data streaming. Imagine that we can (and intend to) collect so much data that we may be forced to drop a large portion of it, or even if we could store it all, we may not have the time to scan it before making judgements. That is a new kind of uncertainty in computing *beyond* randomization and approximation.

0.5 Data Streaming: Formal Aspects

We define various models for dealing with data streams and present a motivating application.

0.5.1 Data Stream Models

The input *stream* $a_1, a_2, \ldots$ arrives sequentially, item by item, and describes an underlying *signal* $\mathbf{A}$, a one-dimensional function $\mathbf{A} : [1 \ldots N] \rightarrow R$. Input may comprise multiple streams or multidimensional signals, but we do not consider those variations for now. Models differ on how the a_i's describe $\mathbf{A}$.

- *Time Series Model*. Each a_i equals $\mathbf{A}[i]$ and they appear in increasing order of i. This is a suitable model for time series data where, for example, you are observing the traffic volume at an IP link every 5 minutes, or NASDAQ volume of trades each minute, etc. At each such time "period", we observe the next new update.
- *Cash Register Model*. Here a_i's are increments to $\mathbf{A}[j]$'s. Think of $a_i = (j, I_i), I_i \geq 0$, to mean $\mathbf{A}_i[j] = \mathbf{A}_{i-1}[j] + I_i$ where $\mathbf{A}_i$ is the state of the signal after seeing the ith item in the stream. Much as in a cash register, multiple a_i's could increment a given $\mathbf{A}[j]$ over time.[3] This is perhaps the most popular data stream model. It fits applications such as monitoring IP addresses that access a web server, source IP addresses that send packets over a link etc. because the same IP addresses may access the web server multiple times or send multiple packets on the link over time. This model has appeared in literature before, but was formally christened in [102] with this name. A special, simpler case of this model is when a_i's are an arbitrary *permutation* of $\mathbf{A}[j]$'s, that is, items do not repeat in the stream, but they appear out of order.
- *Turnstile Model*. Here a_i's are updates to $\mathbf{A}[j]$'s. Think of $a_i = (j, I_i)$, to mean $\mathbf{A}_i[j] = \mathbf{A}_{i-1}[j] + I_i$ where $\mathbf{A}_i$ is the signal after seeing the ith item in the stream, and I_i may be positive or negative. This is the most general model. It is mildly inspired by a busy NY subway train station where the turnstile keeps track of people arriving and departing continuously. At any time, a large number of people are in the subway. This is the appropriate model to study fully dynamic situations where there are inserts as well deletes, but it is often hard to get powerful bounds in this model. This model too has appeared before under different guises, but it gets christened here

[3] For intuition, say $j \in \{\text{pennies}, \text{nickels}, \text{dimes}\}$. The signal $\mathbf{A}$ represents the number of coins in the cash register at any instant of time. Each new input adds some coins of a particular type to the cash register.

with its name.
There is a small detail:

- In some cases, $\mathbf{A}_i[j] \geq 0$ for all i, at all times. We refer to this as the *strict* Turnstile model.

  For example, in a database, you can only delete a record you inserted. Hence, the distribution of the number of records with a given attribute value can not be negative.

- On the other hand, there are instances when streams may be *non-strict*, that is, $\mathbf{A}_i[j] < 0$ for some i.

  This happens for instance as an artifact of studying say $\mathbf{A}_1 - \mathbf{A}_2$ with streams $\mathbf{A}_1$ and $\mathbf{A}_2$ distributed at two different sites respectively as cash register or strict turnstile streams.

We will avoid making a distinction between the two Turnstile models unless necessary. Typically we work in the strict Turnstile model unless the non-strict model is explicitly needed.

The models in decreasing order of generality are as follows: Turnstile, Cash Register, Time Series. (A more conventional description of models appears in [102].) From a theoretical point of view, of course one wishes to design algorithms in the Turnstile model, but from a practical point of view, one of the other models, though weaker, may be more suitable for an application. Furthermore, it may be (provably) hard to design algorithms in a general model, and one may have to settle for algorithms in a weaker model.

We wish to compute various functions on the signal $\mathbf{A}$ at different times during the stream. These functions can be thought of as queries to the data structure interspersed with the updates. There are different performance measures.

- Processing time per item a_i in the stream. (*Proc. Time*)
- Space used to store the data structure on $\mathbf{A}_t$ at time t. (*Storage*)
- Time needed to compute functions on $\mathbf{A}$. (*Compute* or *query time*.)

There is also the work space needed to compute the function or execute the query. We do not explicitly discuss this because typically this is of the same order as the storage.

Here is a rephrasing of our solutions to the two puzzles at the start in terms of data stream models and performance measures.

The puzzle in Section 0.2.1 is in the cash register model (although items do not repeat). The function to be computed is $\{j|\mathbf{A}[j] = 0\}$, given $|\{j|\mathbf{A}[j] = 0\}| \leq k$. For the simplest solution we presented for $k = 1$, the processing time per item and storage was $O(\log n)$ and the compute time was $O(1)$. The function is computed at the end of the stream, so compute time here applies only once. In the fishing exercise in Section 0.2.2, it is again the cash register model. The function to be computed is $\gamma[t]$ at any time t. Processing time per item was $O(k \log(1/\epsilon))$, storage was $O(k \log u \log(1/\epsilon))$ and compute time was $O(k)$ to estimate $\gamma[t]$ to $1+\epsilon$ approximation provided it was large (at least $1/k$) with desired, fixed probability of success. The query to estimate $\gamma[t]$ may arise at any time t during the stream, so compute time applies each time $\gamma[t]$ is estimated. The puzzle in Section 0.2.3 does not fit into the data stream models above since its solution uses multiple passes or random access over the signal. The discussion above is summarized in the table below.

Puzzle	Model	Function	Proc. Time	Storage	Compute Time
Section 0.2.1	cash register	$k = 1, \{j\|\mathbf{A}[j] = 0\}$	$O(\log N)$	$O(\log N)$	$O(1)$
Section 0.2.2	cash register	$\gamma[t]$	$O(k \log(1/\epsilon))$	$O(k \log u \log(1/\epsilon))$	$O(k)$

We can now state the ultimate *desiderata* that are generally accepted:

> *At any time t in the data stream, we would like the per-item processing time, storage as well as the computing time to be simultaneously $o(N,t)$, preferably, $polylog(N,t)$.*

Readers can get a sense for the technical challenge this desiderata sets forth by contrasting it with a traditional dynamic data structure like say a balanced search tree which processes each update in $O(\log N)$ time and supports query in $O(\log N)$ time, but uses linear space to store the input data. Data stream algorithms can be similarly thought of as maintaining a dynamic data structure, but restricted to use *sublinear* storage space and the implications that come with it. Sometimes, the desiderata is weakened so that:

> *At any time t in the data stream, per-item processing time and storage need to be simultaneously $o(N,t)$ (preferably, $polylog(N,t)$), but the computing time may be larger.*

This was proposed in [125], used in few papers, and applies in cases where computing is done less frequently than the update rate. Still, the domain N and input t being so large that they warrant using only $polylog(N,t)$ storage may in fact mean that computing time even linear in the domain or input may be prohibitive in applications for a particular query.

A comment or two about the desiderata follows.

First, why do we restrict ourselves to only a small (sublinear) amount of space? Typically, one says this is because the data stream is so massive that we may not be able to store all of what we see. That argument is facetious. Even if the data stream is massive, if it describes a compact signal (i.e., N is small), we can afford space linear in N, and solve problems within our conventional computing framework. For example, if we see a massive stream of peoples' IDs and their age in years, and all we wish to calculate were functions on peoples' age distribution, the signal is over N less than say 150, which is trivial to manage. What makes data streams unique is that there are applications where data streams describe signals over a very large universe. For example, N may be the

- number of source, destination IP address *pairs* (which is currently 2^{64} and may become larger if we adopt larger IP addresses in the future),
- number of time intervals where certain observations were made (which increases rapidly over time), or
- http addresses on the web (which is potentially infinite since web queries get sometimes written into http headers).

More generally, and this is significantly more convincing, data streams are observations over multiple attributes and any subset of attributes may comprise the domain of the signal in an application and that leads to potentially large domain spaces even if individual attribute domains are small. As we will see, there are serious applications where available memory is severely limited. It is the combination of large universe size of the signals and the limited available memory that leads to the data stream models.

Second, why do we use the polylog function? This is because the log of the input size is the lower bound on the number of bits needed to index and represent the signal, and poly gives us a familiar room to play.

Finally, there is a cognitive analogy that explains the desiderata qualitatively, and may be appealing. As human beings, we perceive each instant of our life through an array of sensory observations (visual, aural, nervous, etc). However, over the course of our life, we manage to abstract and store only part of the observations, and function adequately even if we can not recall every detail of each instant of our lives. We are biological data stream processing machines.

0.5.2 Motivating Scenarios

There are many motivating scenarios for data streams, some more fitting than the others. The one that has been worked out to the most detail is that of traffic analysis in the Internet as described below.

The Internet comprises routers connected to each other that forward IP packets. Managing such networks needs real time understanding of faults, usage patterns, and unusual activities in progress. This needs analyses of traffic and fault data in real time. Consider the traffic data. Traffic at the routers may be seen at many levels.

(1) At the finest level, we have the *packet log*. Each IP packet has a header that contains source and destination IP addresses, ports and other information. The packet log is a list of the relevant header attributes on the series of IP packets sent via a router.
(2) At a higher level of aggregation, we have the *flow log*. Each *flow* is a collection of packets with same values for certain key header attributes such as the source and destination IP addresses, seen within a short span of each other. The flow log comprises cumulative information about the number of bytes and packets sent, start time, end time, and protocol type for each flow passing through a router.
(3) At the highest level, we have the *SNMP log*, which is the aggregate data of the number of bytes sent over each link every few minutes.

Many other logs can be generated from IP networks (fault alarms, CPU usage at routers, etc), but the examples above suffice for our discussion.

One can collect and store SNMP data: one could store SNMP data up to a year or two for a Tier 1 Internet Service Provider without stressing one's laptop. The arguments we presented for data streaming apply to flow and packet logs which are far more voluminous than the SNMP log. The fundamental issue is that packet headers have to processed at the rate at which they flow through the IP routers. OC48 backbone routers work at about $2.5Gbps$ speed handling up to a few million packets a second comprising several hundreds of thousands of concurrent flows during normal times. Memory that has access times compatible with IP packet forwarding times are expensive and are used sparingly in the routers. This presents challenges for both approaches of (a) taking the dump of the traffic logs at packet level and populating a backend database for offline analysis, and (b) analyzing the traffic on line at or near the router at near real time. A more detailed description and defense of streaming data analysis in IP network traffic data is presented in [80, 103]. In general, making a case for a particular application for theoretical models is tricky, but over the past few years, there seems to be significant convergence that data stream methods are useful for IP traffic analysis.

Here are some examples of queries one may want to ask on IP traffic logs.

(1) How much HTTP traffic went on a link today from a given range of IP addresses? This is an example of a slice and dice query on the multidimensional time series of the flow traffic log.
(2) How many distinct IP addresses used a given link to send their traffic from the beginning of the day, or how many distinct IP addresses are currently using a given link in ongoing flows?
(3) What are the top k heaviest flows during the day, or currently in progress? Solution to this problem in flow logs indirectly provides a solution to the puzzle in Section 0.2.1.
(4) How many flows comprised one packet only (i.e., *rare* flows)? Closely related to this is the question: Find the number (or fraction) of TCP/IP SYN packets without matching ACK packets. This query is motivated by the need to detect denial-of-service attacks on networks as early as possible. This problem is one of the motivations for the fishing exercise in Section 0.2.2.

(5) How much of the traffic yesterday in two routers was common or similar? This is a distributed query that helps track the routing and usage in the network. A number of notions of "common" or "similar" apply.
(6) What are the top k correlated link pairs in a day, for a given correlation measure. In general a number of correlation measures may be suitable.
(7) For each source IP address and each five minute interval, count the number of bytes and number of packets related to HTTP transfers. This is an interesting query: how do we represent the output which is also a stream and what is a suitable approximation in this case?

The questions above is a sparse sample of interesting questions; by being imaginative or being informed by network practitioners, one can discover many other relevant ones.

Let us formalize one of the examples above in more detail in the data stream terminology, say the question (2) above.

Example 1. Consider the question: how many distinct IP addresses used a given link to send their traffic since the beginning of the day? Say we monitor the packet log. Then the input stream $a_1, a_2, \ldots$ is a sequence of IP packets on the given link, with packet a_i having the source IP address s_i. Let $\mathbf{A}[0 \ldots N-1]$ be the number of packets sent by source IP address i, for $0 \leq i \leq N-1$, initialized to all zero at the beginning of the day. Each packet a_i adds one to $\mathbf{A}[s_i]$; hence, the model is Cash Register. Counting the number of distinct IP addresses that used the link during the day thus far can be solved by determining the number of nonzero $\mathbf{A}[i]$'s at any time. (Alternately, we could consider each new packet a_i to be setting $\mathbf{A}[s_i]$ to 1. We have used a more general signal to be useful for the example below.) ∎

Example 2. Consider the question: how many distinct IP addresses are *currently* using a given link? More formally, at any time t, we are focused on IP addresses s_i such that some flow f_j began at time before t and will end after t, and it originates at s_i. In the packet log, there is information to identify the first as well as the last packets of a flow. (This is an idealism; in reality, it is sometimes hard to tell when a flow has ended.) Now, let $\mathbf{A}[0 \ldots N-1]$ be the number of flows that source IP address i is currently involved in, for $0 \leq i \leq N-1$, initialized to all zero at the beginning of the day. If packet a_i is the beginning of a flow, add one to $\mathbf{A}[s_j]$ where s_j is the source of packet a_i; if it is the end of the flow, subtract one from $\mathbf{A}[s_j]$ where s_j is the source of packet a_i; else, do nothing. Thus the model is Turnstile. Counting the number of distinct IP addresses that are currently using a link can be solved by determining the number of nonzero $\mathbf{A}[i]$'s at any time. ∎

Similarly other examples above can be formalized in terms of the data stream models and suitable functions to be computed. This is a creative process: the same example may be formalized in terms of different signals with associated functions to be computed, each formulation with potentially unique complexity. A recent paper represents a nice exercise showing a set of case studies, in each case, starting from a network traffic analysis problem and mapping it neatly to a data stream instance [157]. Another recent paper maps network security problems to data stream queries [141].

There has been some frenzy lately about collecting and analyzing IP traffic data in the data stream context. Analyzing traffic data is not new. In telephone and cellular networks, call detail records (CDRs) are routinely collected for billing purposes, and they are analyzed for sizing, forecasting, troubleshooting, and network operations. The angst with IP traffic is that the data is far more voluminous, and billing is not usage based. The reason to invest in measurement and analysis infrastructure is mainly for network maintenance, value-added services and more recently, network security. So, the case for making this investment has to be strong, and it is now being made across the communities and service providers. Both Sprint (the IPMON project) and AT&T seem to be engaged on this topic. That presents the possibility of getting suitable data

stream sources in the future, at least within these companies. Other smaller companies are also developing methods for these large service providers.

0.5.3 Other Data Streaming Applications

The IP traffic analysis case is perhaps the most developed application for data streaming. Consider a different data streaming scenario from Section 0.4: it is a mental exercise to identify suitable analyses and develop data stream problems from them. This is an opportunity to be imaginative. For example,

Exercise 1. Consider multiple satellites continuously gathering multiple terrestrial, atmospheric and ocean-surface observations of the entire earth. What data analysis questions arise with these *spatial* data streams?

This is a good homework exercise. The queries that arise are likely to be substantially different from the ones listed above for the IP traffic logs case. In particular, problems naturally arise in the area of Computational Geometry: an example is detecting the changes in the shape of clouds over time. A wealth of (useful, fundamental) research remains to be done. Algorithms for geometric streams have begun emerging the past year.

Those who want more mental exercises can consider the following three exercises.

Exercise 2. We have a stream of financial transactions including securities, options etc. collected from multiple sources. What are interesting data analysis queries on such data streams?

Many financial companies already face and deal with the challenges; database community has begun addressing some of these problems in research and demos [16] recently. The problems here seem less to do with bounded space and more to do with real time processing.

Exercise 3. We have continuous physical observations—temperature, pressure, EMG/ECG/EEG signals from humans, humidity—from an ad hoc network of sensors. What are interesting data analysis queries on such data streams?

The area of sensor-based data collection is moving from concept to prototypes and slowly beyond. See Zebranet [142], Duck Island [160] and *in loco* monitoring [36] for some examples and more pointers. More and more data stream algorithms are emerging in the context of sensor streams. In particular, the space and power constraints on sensors are real. Also, there are communication constraints which enrich the class of distributed data stream problems of interest.

Exercise 4. We have distributed servers each of which processes a stream of text files (instant messages, emails, faxes, say) sent between users. What are interesting data analysis queries on such *text* data streams?

For example, one may now look for currently popular topics of conversation in a subpopulation. More generally, this scenario involves text processing on data streams which entails information retrieval problems.

We need to develop the examples above in great detail, much as we have done with the IP traffic analysis scenario earlier. We are far from converging on the basic characteristics of data streams or a building block of queries that span different application scenarios, but each of these applications point to new algorithmic problems and challenges.

0.5.4 Other Applications for Data Stream Models

The data stream models are suitable for other applications besides managing rapid, automatic data feeds. In particular, they find applications in the following two scenarios (one each for cash register and Turnstile models).

One pass, Sequential I/O. Besides the explicit data stream feeds we have discussed thus far, there are implicit streams that arise as an artifact of dealing with massive data. It is expensive to organize and access sophisticated data structures on massive data. Programs prefer to process them in one (or few) scans. This naturally maps to the (*Time Series or Cash Register*) data stream model of seeing data incrementally as a sequence of updates. Disk, bus and tape transfer rates are fast, so one sees rapid updates (inserts) when making a pass over the data. Thus the data stream model applies to some extent.

Focus on one (or few) pass computing is not new. Automata theory studied the power of one versus two way heads. Sorting tape-bound data relied on making few passes. Now we are seeking to do more sophisticated computations, with far faster updates.

This application differs from the data streaming phenomenon we saw earlier in a number of ways. First, in data streaming, the analysis is driven by monitoring applications and that determines what functions you want to compute on the stream. In contrast, here, you may have in mind to compute *any* common function (say transitive closure or eigenvalues) and want to do it on massive data in one or more passes. Thus, there is greater flexibility in the functions of interest. Second, programming systems that support scans often have other supporting infrastructure such as a significant amount of fast memory etc. which may not exist in IP routers or satellites that generate data streams. Hence the one pass algorithms may have more flexibility. Finally, the rate at which data is consumed can be controlled and smoothed, data can be processed in chunks etc. In contrast, some of the data streams are more phenomena-driven, and can potentially have higher and more variable update rates. So, the data stream model applies to one pass algorithms, but some of the specific concerns are different.

Monitoring database contents. Consider a large database undergoing transactions: inserts/deletes and queries. In many cases, we would like to monitor the database contents. As an example, consider *selectivity estimation*. Databases need fast estimates of result sizes for simple queries in order to determine an efficient query plan for complex queries. The estimates for simple queries have to be generated fast, without running the queries on the entire database which will be expensive. This is the selectivity estimation problem. It maps to the data stream scenario as follows. The inserts or deletes in the database are the updates in the (*Turnstile model of a data*) stream, and the signal is the database. The selectivity estimation query is the function to be computed on the signal. Data stream algorithms therefore have the desirable property that they represent the signal (i.e., the database) in small space and the results are obtained without looking at the database, in time and space significantly smaller than the database size and scanning time. Thus, data stream algorithms in the Turnstile model naturally find use as algorithms for selectivity estimation. Readers should not dismiss the application of monitoring database content as thinly disguised data streaming. This application is motivated even if updates proceed at a slow rate; it relies only on small space and fast compute time aspect of data stream algorithms to avoid rescanning the database for quick monitoring. In addition, having small memory during the processing is useful: for example, even if the data resides in the disk or tape, the small memory "summary" can be maintained in the main memory or even the cache and that yields very efficient query implementations in general.

Other reasons to monitor database contents are *approximate query answering* and *data quality monitoring*, two rich areas in their own right with extensive literature and work. Again, data stream algorithms find

direct applications in these areas.

0.6 Foundations

The main mathematical and algorithmic techniques used in data stream models are collected here, so the discussion below is technique-driven rather than problem-driven.

0.6.1 Basic Mathematical Ideas

0.6.1.1 Sampling

Many different sampling methods have been proposed: domain sampling, universe sampling, reservoir sampling, priority sampling, distinct sampling etc. Sampling in the data stream context means every input/update is seen but only a (polylogarithmic sized) subset of items are retained, possibly with associated satellite data such as the count of times it has been seen so far, etc. Which subset of items to keep can be chosen deterministically or in a randomized way. Sampling algorithms are known for:

- Finding the quantiles on a Cash Register data stream. See [114] for most recent results.
- Finding frequent items in a Cash Register data stream. See [162].
- Estimating the inverse distribution in a Cash Register data stream. See [58].
- Finding rangesums of items in a Cash Register data stream. See [8].

These problems/solutions have many nice applications. Further, it is quite practical to implement sampling even on high speed streams. In fact, some of the systems that monitor data streams—especially IP packet sniffers—end up sampling the stream just to slow the rate down to a reasonable level, but this should be done in a principled manner, otherwise valuable signals may be lost. Also, keeping a sample helps one estimate many different statistics or apply predicates on the samples *a posteriori*, and additionally, actually helps one return certain *sample answers* to non-aggregate queries that return sets. As a result, sampling methods have been popular and many of these stream sampling methods have even become embedded into an operator in an operational data stream management systems within an SQL-like language [140].

In what follows, we will provide a technical overview of these sampling methods.

Quantile Sampling. The problem is typically thought of as follows. Consider some set S of items: ϕ-quantiles for $0 < \phi < 1$ are items with *rank* $k\phi|S|$ for $k = 0 \ldots 1/\phi$, when S is sorted. This is easily generalized to when S is a multiset, that is, items have frequency greater than 1. For our purpose, we need to define an approximate version of this problem. We will define the problem in the context of the data stream model from Section 0.5.1 equivalently as follows. Recall that $||\mathbf{A}||_1 = \sum_i \mathbf{A}[i]$.

Definition 1. ((ϕ, ε)-*Quantiles*) The (ϕ, ε)-quantiles problem is to output k-quantile j_k for $k = 0 \ldots 1/\phi$, such that $(k\phi - \varepsilon)||\mathbf{A}||_1 \leq \sum_{i \leq j_k} \mathbf{A}[i] \leq (k\phi + \varepsilon)||\mathbf{A}||_1$ for some specified $\varepsilon < \phi$.

The (ϕ, ε)-Quantiles problem is trivial to solve *exactly* on a time series model, that is, with $\varepsilon = 0$ provided we know $||\mathbf{A}||_1$. The problem is more interesting in the cash register model. Computing quantiles exactly requires space linear in N [186, 172]. The folklore method of sampling values randomly and returning the sample median works in $O(1/\varepsilon^2 \log(1/\delta)$ space to return the median, i.e., the $(1/2, \varepsilon)$-quantile, to ε approximation with probability at least $1 - \delta$; improved randomized sampling algorithms were presented in [163] using space $O((1/\varepsilon)(\log^2(1/\varepsilon) + \log^2 \log(1/\delta)))$.

A deterministic sampling method for the (ϕ, ε)-quantiles problem follows from [172] taking space $O((\log^2(\varepsilon||\mathbf{A}||_1))/\varepsilon)$. We will describe a deterministic sampling procedure from [114], which has the currently best known space bounds. For exposition here, let us assume each update $a_t = (j, I_i)$ is in fact $(j, 1)$. The data structure at time t, $S(t)$, consists of a sequence of s tuples $\langle t_i = (v_i, g_i, \Delta_i)\rangle$, where each v_i is a sampled item from the data stream and two additional values are kept:

- g_i is the difference between the lowest possible rank of item v_i and the lowest possible rank of item v_{i-1}; and
- Δ_i is the difference between the greatest possible rank of item v_i and the lowest possible rank of item v_i.

The total space used is therefore $O(s)$. For each entry v_i, let $r_i = \sum_{j=1}^{i-1} g_j$. Hence, the true rank of v_i is bounded below by $r_i + g_i$ and above by $r_i + g_i + \Delta_i$. Thus, r_i can be thought of as an overly conservative bound on the rank of the item v_i. The algorithm will maintain the invariant:

$$\forall i: \quad g_i + \Delta_i \leq 2\epsilon||\mathbf{A}||_1$$

When the update at time $t+1$ is to insert a new item, $v \notin S(t)$, find i such that $v_i < v \leq v_{i+1}$, compute r_i and insert the tuple $(v, g = 1, \Delta = \lfloor 2\varepsilon||\mathbf{A}||_1\rfloor)$. Periodically, the algorithm scans the data structure and merges adjacent nodes when this does not violate the invariant. That is, remove nodes (v_i, g_i, Δ_i) and $(v_{i+1}, g_{i+1}, \Delta_{i+1})$ and replace with $(v_{i+1}, (g_i + g_{i+1}), \Delta_{i+1})$ provided that $(g_i + g_{i+1} + \Delta_{i+1}) \leq 2\varepsilon||\mathbf{A}||_1$. Given a value $0 \leq \phi \leq 1$, let i be the smallest index so that $r_i + g_i + \Delta_i > (\phi + \varepsilon)||\mathbf{A}||_1$. Output v_{i-1} as the approximated quantile when desired.

Theorem 5. [114] The deterministic sampling procedure above outputs (ϕ, ε)-quantiles on a cash register stream using space $O(\frac{\log(\varepsilon||\mathbf{A}||_1)}{\varepsilon})$.

The algorithm above can be implemented carefully by batching updates and scheduling the pruning to minimize the amortized processing time. It will be of interest to develop a simpler proof for Theorem 5 than the one in [114].

A recent paper [197] gives a simple, alternative way to compute approximate quantiles over streaming data. Impose the obvious full *binary* tree of height $\log N$ over the domain $[1, N]$. With each node v, associate a count $c(v)$ corresponding to a count of some subset of items appearing in the range covered by the node. A *q-digest* is a set of carefully chosen $(v, c(v))$'s. Let $\alpha < 1$. A node v and its count $c(v)$ is in the q-digest if and only if: $c(v) \leq \alpha||\mathbf{A}||_1$ (unless v is a leaf) and

$$c(v) + c(v_{parent}) + c(v_{sibling}) > \alpha||\mathbf{A}||_1.$$

It is quite simple to see that on a static tree, one can maintain this invariant by bottom-up traversal. Further, the size of q-digest is at most $O(1/\alpha)$ and error in estimating the frequency of any node is at most $(\alpha \log N)$. This algorithm can be made incremental by considering each new update as a q-digest by itself and merging the two q-digests to still maintain the invariant above. It will follow that:

Theorem 6. [197] The deterministic sampling procedure above outputs (ϕ, ε)-quantiles on a cash register stream using space $O(\frac{\log N}{\varepsilon})$.

Which of the two above algorithms is more efficient depends on the domain and nature of the stream.

Problem 1. Are there deterministic algorithms for the (ϕ, ε)-quantiles problem on cash register streams using space $o((1/\varepsilon)\min\{\log N,\ \log \varepsilon||\mathbf{A}||_1\})$?

While $1/\varepsilon$ term is necessary, it is not known if the multiplicative $\log(\varepsilon||\mathbf{A}||_1)$ or $\log N$ factor is necessary. Is $O(1/\varepsilon + \log(\varepsilon||\mathbf{A}||_1))$ doable?

None of the algorithms mentioned thus far work in the Turnstile model for which algorithms will be discussed later.

Frequent Elements/Heavy Hitters Sampling. Again consider the cash register model. Estimating $\max_i \mathbf{A}[i]$ is impossible with $o(N)$ space [10]. Estimating the k most frequent items (k largest $\mathbf{A}[i]$'s) is at least as hard. Hence, research in this area studies related, relaxed versions of the problems, in particular, the heavy hitters problem. The ϕ-heavy hitters of a multiset of values consist of those items whose multiplicity exceeds the fraction ϕ of the total cardinality. There can be between 0 and $\frac{1}{\phi}$ heavy hitters in any given sequence of items. If one is allowed $O(N)$ space, then a simple heap data structure will process each insert update in $O(\log N)$ time and find the heavy hitter items in $O((\log N)/\phi)$ time. However, this reduced goal of finding the heavy hitters is also difficult when working with only small space. The lower bound of [10] does not directly apply to finding ϕ-heavy hitters, but a simple information theory argument shows the following lower bound.

Theorem 7. Any algorithm which guarantees to find all and only items i such that $\mathbf{A}[i] > (1/k+1)||\mathbf{A}||_1$ must store $\Omega(N)$ bits.

Proof. Consider a set $S \subseteq \{1 \ldots N\}$. Transform S into a stream of length $n = |S|$. Process this stream with the proposed algorithm. We can use the algorithm to extract whether $x \in S$ or not: for some x, insert n/k copies of x. Suppose $x \notin S$: the frequency of x is $(n/k)/(n + n/k) \leq 1/(k+1)$, and so x will not be output. On the other hand, if $x \in S$ then $(n/k+1)/(n+n/k) > 1/(k+1)$ and so x will be output. Hence we can extract the set S and the space stored must be $\Omega(N)$ since, by an information theoretic argument, the space to store an arbitrary subset S is N bits. ∎

The lower bound above also applies to randomized algorithms. Any algorithm which guarantees to output all heavy hitters (with $\varepsilon = 0$) with probability at least $1 - \delta$, for some constant δ, must also use $\Omega(N)$ space. This follows by observing that the above reduction corresponds to the Index problem in communication complexity [155], which has one-round communication complexity $\Omega(N)$.

Hence, we need to allow some approximation. We formally work in the data stream model and define an approximate version.

Definition 2. *((ϕ, ε)-heavy hitters)* Return all i such that $\mathbf{A}[i] \geq \phi||\mathbf{A}||_1$ and no i such that $\mathbf{A}[i] \leq (\phi - \varepsilon)||\mathbf{A}||_1$ for some specified $\varepsilon < \phi$.

The problem of determining the (ϕ, ε)-heavy hitters has many (rediscovered) algorithms. Consider the simpler problem: given n items in a data stream model, find the *strict majority* if any, that is, any item that appears at least $||\mathbf{A}||_1/2 + 1$ times. A simple, elegant algorithm is in [90, 169] which relies on ignoring successive pairs of elements that differ from each other. If there is a majority, it will be the element left over. This can be generalized to reporting all items that appear at least $||\mathbf{A}||_1/\phi + 1$ times as follows. Maintain a set K of elements and counts for each. If the new element is in K, its count is incremented. Else, it is added to K with count 1. When $|K| > 1/\phi$, decrement each of the counts by 1 and eliminate the ones with count 0. There are no false negatives, that is, any heavy hitter item will be in K because each occurrence of i not in K is eliminated with at most $1/\phi - 1$ others. Space used is $O(1/\phi)$ and time per new element is amortized $O(1)$ since deletion can be charged to the time of the arrival of the element. This generalization appears in [144, 110]. While this algorithm has no false negatives, it potentially has *bad* false positives, that

is, there may be items i with $\mathbf{A}[i]$ arbitrarily smaller than $||\mathbf{A}||_1/\phi$. An additional observation shows that this algorithm in fact can eliminate the false positives with very low frequencies and solve the (ϕ, ε)-heavy hitters problem in space $O(1/\epsilon)$ [166] which is tight.

Curiously, the (ϕ, ε)-heavy hitters problem seems to have captured the imagination of the researchers. There are rediscoveries aplenty of the basic scheme above, each tuning one aspect or the other in terms of the worst case update time, batched updates or frequency estimates. A popular example is the sampling algorithm in [162]. The input stream is conceptually divided into buckets of width $w = \lceil \frac{1}{\varepsilon} \rceil$. There are two alternating phases of the algorithms: insertion and compression. The algorithm maintains triples (e, f, Δ) where e is an element on the stream, f is its estimated frequency and Δ is the maximum possible error in f. It is easy to see that items i with $\mathbf{A}[i] < (\phi - \epsilon)||\mathbf{A}||_1$ are not maintained; it requires an analysis to show that this algorithm takes $O((\log(\varepsilon||\mathbf{A}||_1))/\varepsilon)$ space to solve the (ϕ, ε) heavy hitters problem in the cash register model. Some empirical and special-case analysis is shown in [162] that the space used is closer to $O(1/\varepsilon)$.

Note. There is a strong relationship between the (ϕ, ε)-quantiles and (ϕ, ε)-heavy hitters problems.

- Using $(\phi, \varepsilon/2)$-quantiles will retrieve the (ϕ, ε)-heavy hitters.
- Consider a partition of the universe $1 \ldots N$ into $\log N$ problems, ith of which comprises $N/2^i$ disjoint intervals of size 2^i—called *dyadic intervals*—covering $[1, N]$. Use $(\phi/\log N, \varepsilon/\log N)$-heavy hitters in each of these subproblems. Observe that each range $[1, i] \in [1, N]$ can be thought of as the disjoint union of at most $\log N$ dyadic intervals. The quantiles can be obtained by binary searching for i such that $\sum_{j=1}^{j=i} \mathbf{A}[j]$ is the desired approximation to the kth ϕ-quantile for each k independently. Thus, the (ϕ, ε)-quantiles problem needs roughly $\log N$ times the resources needed for the $(\phi/\log N, \varepsilon/\log N)$-heavy hitters problem.

The discussion above shows that up to polylog N factors, the two problems are equivalent. This will help understand the relationship between [162], [114] and [197]. ∎

The problem of heavy hitters has been extended together with generalization of the sampling approach above to hierarchical domains [47]—there is a tree atop $1 \cdots N$ where internal nodes correspond to ranges and the tree specifies the containment of a range within others—as well as multidimensional signals—product of trees on each dimension [79, 48]. Here, one *discounts* heavy-hitter $\mathbf{A}[i]$'s while determining if j is a heavy-hitter provided i is contained in j's range for d-dimensional nodes i and j in the hierarchy. Still, there are inherent bottlenecks and one can not get as succinct space bounds for these extensions as we did for the one dimensional heavy-hitters problem [126].

Distinct Sampling. Consider the problem of estimating aspects of the inverse signal. Formally, let us assume $\mathbf{A}[i]$'s are non-negative integers. We define the *inverse signal*

$$\mathbf{A}^{-1}[i] = \frac{|\{j | j \in [1, N], \mathbf{A}[j] = i, i \neq 0\}|}{|\{j \ | \mathbf{A}[j] \neq 0\}|}.$$

Therefore, $\sum_i \mathbf{A}^{-1}[i] = 1$. Queries on the inverse signal give a variety of information about the signal itself.

Any function or query on the inverse signal can be rewritten explicitly as a query or function on the original signal, so the inverse signal does not provide any additional information, but it is an intuitive convenience. Given full space, we can convert the signal to its inverse easily. However, in the data stream model, this conversion is not possible. Hence, it is a challenge to support queries on the inverse signal directly in any of the data stream models.

In general, queries over the inverse signal are strictly harder than their counterparts over the original signal. For example, answering $\mathbf{A}[i]$ query for a *prespecified* i over the cash register or turnstile models is trivial to do exactly, but answering $\mathbf{A}^{-1}[i]$ even for a prespecified $i = 1$ requires $\Omega(N)$ space as we know from Section 0.2.2. The number of distinct values, that is $|\{j \ |\mathbf{A}[j] \neq 0\}|$, can be approximated up to a fixed error with constant probability in $O(1)$ space by known methods [18]. However, the number of distinct values in the inverse signal, that is $|\{j \ |\mathbf{A}^{-1}[j] \neq 0\}|$ requires linear space to approximate to a constant factor which can be shown by reducing the communication complexity problem of disjointness to this problem [58].

Consider estimating $\mathbf{A}^{-1}[i]$ for any i in the cash register model. This corresponds to finding the fraction of items that occurred exactly i times on the stream. For example, finding $\mathbf{A}^{-1}[1]$ over a stream of IP flows corresponds to finding IP flows consisting of a single packet — possible indication of a probing attack if $\mathbf{A}^{-1}[1]$ is large. This quantity is the rarity discussed in Section 0.2.2. Other problems like heavy hitters and quantiles can be defined on inverse signal as well.

The technique in Section 0.2.2 from [67] will solve these problems as shown below; in addition, the procedure below uses only pair-wise independence (and not approximate min-wise independence as in [67]).

There are two steps.

- Produce multiset S consisting of $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ samples (x, i), where x is an item in $[1, N]$ and i, its count in the input stream; further, the probability that any $x \in [1, N]$ is in S is $1/|\{j \ |\mathbf{A}[j] \neq 0\}|$. That is, each *distinct* item in the input stream is equally likely to be in S.
- Subsequently, approximate $\mathbf{A}^{-1}[i]$ by $\frac{|\{(x,i)\in S\}|}{|S|}$. Let $Y_j = 1$ if the jth sample in S is a pair (x, i), and $Y_j = 0$ if the jth sample is a pair (x', i') for $i' \neq i$. Since each sample is drawn uniformly,

$$\mathsf{Pr}[Y_j = 1] = \frac{|\{x|\mathbf{A}[x] = i\}|}{|\{j \ |\mathbf{A}[j] \neq 0\}|} = \mathbf{A}^{-1}[i].$$

  So the estimate is correct in expectation. By applying the Hoeffding inequality to$\sum_j Y_j/|S|$,

$$\mathsf{Pr}[\left|\frac{\sum_j Y_j}{|S|} - \mathbf{A}^{-1}[i]\right| \leq \epsilon] \geq 1 - \delta.$$

The technical crux is therefore the first step. We consider pairwise-independent hash functions $h_\ell : [1, N] \to [0, 2^\ell - 1]$ for $\ell = 1, \cdots, \log N$. For each i, we maintain the *unique* item i, if any, that maps to 0, as well as $\mathbf{A}[i]$ for such an i. If more than one item mapped to 0 for that ℓ, then we can ignore that ℓ thereafter. Each item on the cash register stream is hashed using each of these hash function and updates are handled in the straightforward way. Generate a sample by picking the largest ℓ such that something mapped to 0 and if a unique item i did, return the item and its count $\mathbf{A}[i]$. One outcome is that no item is output when for example more than one item mapped to 0 for some ℓ. However, it can be proved that

- With constant probability, some $(i, \mathbf{A}[i])$ is returned, and
- If a sample is returned, probability that i is returned is uniformly $O(1/|\{j \ |\mathbf{A}[j] \neq 0\}|)$.

In fact, the above can be proved by considering a *single* hash function h and defining $h_\ell(x)$ to be the trailing ℓ bits of $h(x)$ for any x. Combining the two steps:

Theorem 8. [58] There exists a randomized $O(1/\varepsilon^2)\log N\log(1/\delta))$ space algorithm that with probability at least $1 - \delta$, returns an estimate for any $\mathbf{A}^{-1}[i]$ correct to $\pm\varepsilon$.

The sampling procedure above can be extended to the turnstile case too: the crux is in maintaining the unique element that maps to 0 under the hash function at any given time even if at times earlier, many items were mapped to 0 but subsequently removed. Independently, solutions have been proposed for this in [92] and [58]. Inverse signals may be sometimes more effective in detecting anomalies [109]. Problems on inverse signals have not been studied as much as those on the basic signals, so more work will be useful.

Subset-Sum Sampling. Recall that in the cash register model, the ith update is (j, I_i), $I_i \geq 0$. So far we have studied problems that collect the tuples based on the first coordinate—e.g., all updates to j—and support various estimation queries on the summation of associated I_i's. There are many known methods for estimating $\mathbf{A}[i]$ for query i, $\sum_{k=i}^{k=j} \mathbf{A}[k]$ for given range $[i, j]$, heavy hitters and quantiles on i's described earlier.

Let us consider a more general collection method in which we specify an *arbitrary subset* T of these updates by some predicate (all those where j's are odd or all odd numbered updates etc.) *at query time* and seek to estimate the sum of I_i's in T. One special case of this problem is the one in which we specify the subset T to be all updates to a particular j which will coincide with querying $\mathbf{A}[i]$ for a given i, but this framework is more general.

Subset-sum sampling works as follows. For each update i, generate an independent uniformly random $\alpha_i \in (0, 1)$ and assign *priority* $q_i = I_i \alpha_i^{-1}$. For any subset T, let q be the $k+1$st largest priority amongst items in T. The *subset-sum sample* $S \subseteq T$ of size k is the top k priority updates in T, that is, $i \in S$ iff $q_i > q$. Using S, for subset T, $W = \sum_{i \in T} I_i$ can be estimated as $W^* = \sum_{i \in S} \max\{I_i, q\}$ in an unbiased way, that is, $E(W^*) = W$ [77, 8]. In a very interesting result, Szegedy recently proved that $var(W^*) \leq W^2/(k-1)$ [201] (this is known to be tight).

This scheme can be applied in a way that all priorities are stored in a data structure and when some subset-sum needs to be estimated, S can be retrieved from the data structure [8]. However, in applications to IP networks in earlier versions [77], it is used to determine a single S for $\{1, \ldots, N\}$ and used for any T specified at the query time by projecting S on the given T. By asking for a sample which will provide such subset-sum estimates for *any* T, one has to necessarily settle for extremely weak accuracy guarantees in the worst case since the set of possible T's is exponential in $||\mathbf{A}||_1$. It is also not clear one needs to aim for such a general query since one needs to identify suitable applications where we need *arbitrary* subset sums—points or ranges seemed to be adequate for many applications thus far. Finally, statistical sampling methods such as this may not be accurate (even some heavy hitters may not be accurately estimated) or efficient (to get relative approximation of $1 \pm \varepsilon$, these methods need $\Omega(\log(1/\delta)/\varepsilon^2)$ samples; in contrast, the heavy hitters and quantiles can be solved with $O(1/\varepsilon)$ samples) or powerful to model (subset-sum sampling does not solve the inverse signal estimation problem directly) for many specific problems of interest. Thus this scheme has some challenges in applicability for data streams.

Still, this sampling scheme can be executed in the cash register model easily. Further, it is an interesting sampling scheme and its analysis has proved to be nontrivial. The problem itself seems to be a generalization of the $I_i = 1$ case studied in order statistics. Better understanding of this method will be useful. Also, it will be a scholarly exercise to precisely show the relationship between subset-sum sampling and other data stream sampling methods. ∎

In general, the sampling approach to solving problems on the data stream models has some positives. The algorithms are often simple and once one has a sample, any predicate can be applied on the sample itself and produce candidate items which is sometimes useful. On the other hand, there are two main difficulties with sampling for data stream problems. First, sampling is not a powerful primitive for many problems. One needs far too many samples for performing sophisticated analyses. See [34] for some lower bounds.

Second, sampling methods typically do not work in the Turnstile data stream model: as the stream unfolds, if the samples maintained by the algorithm get deleted, one may be forced to resample from the past, which is in general, expensive or impossible in practice and in any case, not allowed in data stream models. An exception is the inverse sampling above where if the randomness used by the algorithm is unknown to the adversary, the algorithm still provides a guarantee on the sample size it outputs [92, 58]. This is the only such dynamic sampling scheme we know.

Problem 2. Say we have data streams over two observed variables (x_t, y_t). An example *correlated aggregate* at time t is $\{g(y_i) \mid x_i = f(x_1 \cdots x_t)\}$, that is, computing some aggregate function g—SUM, MAX, MIN—on those y_t's when the corresponding x_t's satisfy certain relationship f. For what f's and g's (by sampling or otherwise) can such queries be approximated on data streams? See [148] for the motivation.

0.6.1.2 Random Projections

This approach relies on dimensionality reduction, using projection along pseudo-random vectors. The pseudo-random vectors are generated by space-efficient computation of limitedly independent random variables. These projections are called the *sketches*. This approach typically works in the Turnstile model and is therefore quite general.

Moments estimation. The problem is to estimate the kth moment, $F_k = \sum_i \mathbf{A}[i]^k, k = 0, 1, 2, \ldots$.

First focus on F_2. In a seminal paper [10], Alon, Matias and Szegedy presented an algorithm in the turnstile model. For each i and j, let $\mathbf{X}_{i,j}$ be a random vector of size N where $\mathbf{X}_{i,j}[\ell]$'s are ± 1-valued 4-wise independent random variables. Let $X_{i,j} = \langle \mathbf{A}, \mathbf{X}_{i,j} \rangle$. We have

$$E(X_{i,j}^2) = F_2 \text{ and } \mathrm{var}(\mathrm{X}_{\mathrm{i,j}}^2) \leq 2\mathrm{F}_2^2$$

using the $4-$wise independence property of $\mathbf{X}_{i,j}[\ell]$'s. Define Y_i to be the average of $X_{i,1}, \ldots, X_{i,O(\log(1/\varepsilon^2)}$. By Chebyshev's inequality, $\mathsf{Pr}(|Y_i - F_2| > \varepsilon F_2) \leq O(1)$. Now define Z to be the median of $Y_1, \ldots, Y_{O(\log(1/\delta)}$. By Chernoff's bound, $\mathsf{Pr}(|Z - F_2| > \varepsilon F_2) < \delta$. Note that it takes $O(\log N)$ bits to generate each $\mathbf{X}_{i,j}$, and $X_{i,j}$ can be maintained in the turnstile model with $O(1)$ time per update using constant time word operations on $O(\log N + \log ||\mathbf{A}||_1)$ bits. Thus:

Theorem 9. [10] There exists an $O((1/\varepsilon^2)\log(1/\delta))$ space randomized algorithm in the turnstile model that outputs an estimate $\tilde{F}_2$ such that with probability at least $1 - \delta, \tilde{F}_2 \in (1 \pm \varepsilon)F_2$. The time to process each update is $O((1/\varepsilon^2)\log(1/\delta))$.

The study of $F_k, k > 2$, has a rich history of successively improved upper bounds from [10], via eg [41], to finally [137] where $O(N^{1-2/k})$ space (to polylog terms) algorithm is presented for $k > 2$. This is tight following successively improved lower bounds from [10], via e.g. [193] and [17], to finally [27].

Norms estimation. The problem is to estimate L_p-sums defined as $\sum_i |\mathbf{A}[i]|^p$ (L_p sums are pth powers of L_p norms). Moment estimation methods that work in the turnstile model can be used to estimate the L_p-sums, often with nearly tight space bounds. The interesting cases are for $p = 1$ (where F_1 estimation is trivial unlike L_1-sum estimation), or smaller.

Indyk [131] pioneered the use of *stable* distributions to estimate L_p-sums. In p-stable distributions, $\sum_{i=1}^n a_i X_i$ is distributed as $(\sum_{i=1}^n |a_i|^p)^{1/p} X_0$. Here, a_i are scalars, and $X_0 \ldots X_n$ are independent and identically distributed random variables drawn from p-stable distributions. Such distributions exist for all $p \in$

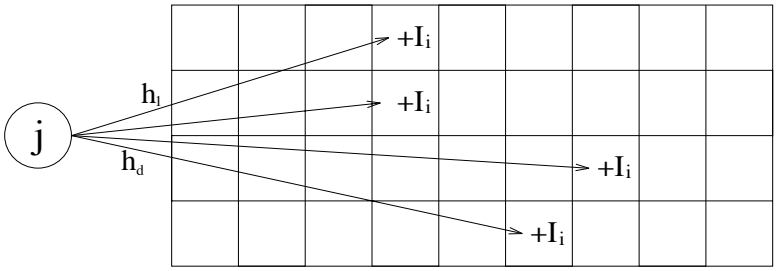

Fig. 1 Each item j is hashed to one cell in each row: on ith update (j, I_i), I_i is added to each such cell.

$(0, 2]$; Gaussian is stable with $p = 2$ and Cauchy is stable with $p = 1$. For estimating L_p-sum, the algorithm is similar to the one for F_2 estimation, but uses random variables from p-stable distributions for $\mathbf{X}_{i,j}[\ell]$. Stable distributions can be simulated for any value of p using transforms from uniform variables [28]. As before, the median of $O(1/\varepsilon^2 \log 1/\delta)$ estimators is close to the median of the distribution with probability at least $1 - \delta$, relying on the derivative of the distribution being bounded at the median. Hence, one obtains $1 \pm \varepsilon$ approximation to the L_p-sum with probability at least $1 - \delta$ in space $O(1/\varepsilon^2 \log 1/\delta)$. Many technical issues need to be explored about the use of stable distributions for L_p-sum estimation: see [42]. ∎

There are a few applications for computing moments and norms on streams, but not many. Mainly, they (and the techniques) are used as subroutine in other problems in the Turnstile data stream model. F_2 and L_2 are used to measure deviations in anomaly detection [153] or interpreted as self-join sizes [9]; with variants of L_1 sketches we can dynamically track most frequent items [51], quantiles [104], wavelets and histograms [105], etc. in the Turnstile model; using L_p sketches for $p \to 0$, we can estimate the number of distinct elements at any time in the Turnstile model [43]. But in many of these applications, the basic mathematical results above can be replaced by variations of random projections of simpler ilk (Random subset sums [102], counting sketches [32], Count-Min sketches [55]), and simple algorithmic strategies atop which give better bounds for specific problems. Here are three such specific sets of results.

Count-Min sketch and its applications. We discuss the strict turnstile case, that is, $\mathbf{A}[i] \geq 0$ at all times. A *Count-Min (CM) sketch* with parameters (ε, δ) is a two-dimensional array counts with width w and depth d: $\mathrm{count}[1, 1] \ldots \mathrm{count}[d, w]$. Given parameters (ε, δ), set $w = \lceil \frac{e}{\varepsilon} \rceil$ and $d = \lceil \ln \frac{1}{\delta} \rceil$. Each entry of the array is initially zero. Additionally, d hash functions $h_1 \ldots h_d : \{1 \ldots N\} \to \{1 \ldots w\}$ are chosen uniformly at random from a pairwise-independent family. When jth update (j, I_i) is seen, set $\forall 1 \leq k \leq d$, $\mathrm{count}[k, h_k(j)] \leftarrow \mathrm{count}[k, h_k(j)] + I_i$. See Figure 1.

The answer to a *point query*: "$\mathbf{A}[i] =$?" is given by

$$\hat{\mathbf{A}}[i] = \min_j \mathrm{count}[j, h_j(i)].$$

Clearly, $\mathbf{A}[i] \leq \hat{\mathbf{A}}[i]$. The claim is, with probability at least $1 - \delta, \hat{\mathbf{A}}[i] \leq \mathbf{A}[i] + \varepsilon ||\mathbf{A}||_1$. This is because for each cell i is mapped to, the expected contribution of other items is $||\mathbf{A}||_1/w$. Application of the Markov's Inequality to each row upper bounds the failure probability by δ. Summarizing:

Theorem 10. [55] The CM sketch data structure works in the Turnstile model, uses space $O((1/\varepsilon) \log(1/\delta))$ with update time $O(\log(1/\delta))$, and answers point query "$\mathbf{A}[i] =$?" with estimate $\hat{\mathbf{A}}[i]$ such that $\mathbf{A}[i] \leq \hat{\mathbf{A}}[i]$ and with probability at least $1 - \delta$,

$$\hat{\mathbf{A}}[i] \leq \mathbf{A}[i] + \varepsilon ||\mathbf{A}||_1.$$

For all previous sketches for point queries, the dependency of space on ε was $\frac{1}{\varepsilon^2}$; the result above is a significant improvement since $\varepsilon = 0.01$ is not uncommon in applications. All prior analyses of sketch structures compute the variance of their estimators in order to apply the Chebyshev inequality, which brings the dependency on ε^2. Directly applying the Markov inequality yields a more direct analysis which depends only on ε. Also, the constants are small and explicit. Further, CM Sketch is a building block for rangesum queries, quantiles, heavy hitters [55] and moments [56]; hence, it is a single sketch that can be used for a variety of analyses —something desirable in systems, as well as conceptually. CM Sketch is currently in the Gigascope data stream system [46], working at $2-3$ million updates per second without significantly taxing the resources.

Theorem 10 has some apparent strengths. For example, unlike the standard sampling methods, it works in the Turnstile model to summarize any data set. Consider inserting a million ($\approx 2^{20}$) IP addresses; the CM sketch still maintains $O((1/\varepsilon)\log(1/\delta))$ counts of size $\log ||\mathbf{A}||_1 \approx 20$ bits still, which is much less than the input size. Now delete all but 4. Asking point queries on the remaining signal to accuracy $\varepsilon = 0.1$ will exactly reveal the 4 remaining items by using Theorem 10 (since $||\mathbf{A}||_1 = 4$ now, and for i's with $A[i] = 1$, $\hat{\mathbf{A}}[i] \in [1, 1.4]$). The interesting part is that in the intermediate stages the input data was significantly larger than the CM data structure or the final signal size when the point query was posed.

Theorem 10 also has apparent weaknesses. Estimating $\mathbf{A}[i]$ to additive $\varepsilon||\mathbf{A}||_1$ accuracy is not effective for small $\mathbf{A}[i]$'s. Fortunately, in most data stream applications, one is interested in cases where $A[i]$'s are large with respect to $||\mathbf{A}||_1$. Further, the theorem above is tight:

Theorem 11. The space required to answer point queries correctly with any constant probability and error at most $\varepsilon||\mathbf{A}||_1$ is $\Omega(\varepsilon^{-1})$ over general distributions.

Proof. We reduce from the Index problem in communication complexity. Paul holds a bitstring of length N and is allowed to send a message to Carole who wants to compute the ith bit of the bitstring. Since Carole is not allowed to communicate with Paul, any protocol to solve this problem, even probabilistically, requires $\Omega(N)$ bits of communication [155]. Take a bitstring $B[1 \ldots \frac{1}{2\varepsilon}]$ of length $N = \frac{1}{2\varepsilon}$ bits and set $\mathbf{A}[i] = 2$ if $B[i] = 1$; otherwise, set $\mathbf{A}[i] = 0$ and add 2 to $\mathbf{A}[0]$. Whatever the value of B, $||\mathbf{A}||_1 = 1/\varepsilon$. If we can answer point queries with accuracy $\varepsilon||\mathbf{A}||_1 = 1$, we can test any $\mathbf{A}[i]$ and determine the value of $B[i]$ by reporting 1 if the estimated value of $\mathbf{A}[i]$ is above $\varepsilon||\mathbf{A}||_1$ and 0 otherwise. Therefore, the space used must be at least $\Omega(\frac{1}{\varepsilon})$ bits. ∎

We can sometimes overcome this lower bound in practical settings, since the distribution of frequencies of different items is often skewed in practice. The Zipf distribution accurately captures this skew in certain natural distributions. It was introduced in the context of linguistics where it was observed that the frequency of the ith most commonly used word in a language was approximately proportional to $1/i$ [216]. Zipf distributions are related to Pareto distributions and power-laws [4]. Formally, a Zipf distribution with parameter z has the property that f_i, the (relative) frequency of the ith most frequent item is given by $f_i = \frac{c_z}{i^z}$, where c_z is an appropriate scaling constant. For the signal $\mathbf{A}$ whose entries are distributed according to a Zipf distribution, the count of the ith most frequent item is simply $||\mathbf{A}||_1 f_i$. In real life, web page accesses have $z \in [0.65, 0.8]$; "depth" to which surfers investigate websites has $z \approx 1.5$; file transmission times over the internet has $z \approx 1.2$; size of files moved has $z \approx [1.06, 1.16]$; etc. [56].

Theorem 12. [56] Using the CM Sketch, the space required to answer point queries with error $\varepsilon||\mathbf{A}||_1$ with probability at least $1 - \delta$ is given by

$$O(\varepsilon^{-\min\{1,1/z\}} \ln 1/\delta)$$

for a Zipf distribution with parameter z.

The proof of above theorem uses the idea of separating the influence of certain heavy items and arguing that, with significant probability, only the influence of the remainder of the item counts, and bounding the latter using Zipfian properties. This result improves the "golden standard" of $O(1/\varepsilon)$ space used in data stream algorithms for skewed data, $z > 1$. The space used in the theorem above is tight can be shown to be tight for Zipfian data. The result above is useful in practice because one uses the same CM data structure one uses regularly, but gets the improvement because of just better analysis. In practice, z is unknown; hence, this result would be used with fixed space, but providing better accuracy guarantees for the given space bound. That is, theorem above alternatively implies: for $z \geq 1$,

$$\hat{\mathbf{A}}[i] \leq \mathbf{A}[i] + O(\varepsilon)^z ||\mathbf{A}||_1.$$

Estimating number of distinct elements. The problem is to estimate $D = |\{i \mid \mathbf{A}[i] \neq 0\}$. If $\mathbf{A}[i]$ is the number of occurrences of i in the stream, D is the number of distinct items. More generally, D may be thought of as the *Hamming* norm if $\mathbf{A}[i]$ is arbitrary. Estimating D is a fundamental problem.

An early result for estimating D in a data stream model appears in [89]. We keep a bit vector $c[1] \ldots c[\log_2 N]$ initialized to zero and use a hash function $f : [1, N] \rightarrow \{1, 2, \ldots \log_2 N\}$ such that $\mathsf{Pr}[f(i) = j] = 2^{-j}$. Any update j to item i sets $c[f(i)]$ to 1. At any point, an unbiased estimate for the number of distinct items is given by $2^{k(c)}$, where $k(c)$ is the lowest index of a counter $c[j]$ that is zero ($k = \min\{j|c[j] = 0\}$). Intuitively this procedure works because if the probability that any item is mapped onto counter j is 2^{-j}, then if there are D distinct items, we expect $D/2$ to be mapped to $c[1]$, $D/4$ to be mapped to $c[2]$, and so on until there are none in $c[\log_2 D]$. This algorithm is not affected by repeated elements: if one item, i, is seen many times, it is mapped onto the same bit counter each time, and will not alter the estimate of the number of distinct elements. The procedure above relies on existence of fully random hash functions. In a series of papers including [10] and [101] more realistic hash functions were used and the space used was $O(1/\varepsilon^2 \log N \log(1/\delta))$, until the current best space bound of $O((\frac{1}{\varepsilon^2} \log\log m + \log m \log(1/\epsilon)) \log(1/\delta)$ was shown in [18]. These results [10, 101, 18] work only in the cash register model. The result in [89] works for the strict Turnstile model and fails for the non-strict Turnstile model. For example, if an item that has not been seen before is removed (that is $\mathbf{A}[i] < 0$), this method is not well defined. It could lead to decreasing a counter (even when generalized to be a large counter rather than being a bit counter) below zero, or to producing an estimate of the Hamming norm that is inaccurate.

In the most general data stream model—non-strict turnstile model—the algorithm for estimating D uses L_p-sum estimation. Observe that $|\mathbf{A}[i]|^0 = 1$ if $\mathbf{A}[i] \neq 0$; we can define $|\mathbf{A}[i]|^0 = 0$ for $\mathbf{A}[i] = 0$. Thus, $D = |\{i|\mathbf{A}[i] \neq 0\}| = \sum_i |\mathbf{A}[i]|^0$, L_0-sum defined earlier. We approximate L_0-sum by L_p for suitably small p. Say $|\mathbf{A}[i]| \leq M$ for some upper bound M, then

$$D = \sum_i |\mathbf{A}[i]|^0 \leq \sum_i |\mathbf{A}[i]|^p \leq \sum_i M^p |\mathbf{A}[i]|^0 \leq M^p \sum_i \mathbf{A}[i]^0 \leq (1+\epsilon) \sum_i \mathbf{A}[i]^0 = (1+\epsilon)D$$

if we set $p \leq \log(1+\varepsilon)/\log M \approx \varepsilon/\log M$. Hence, estimating $\sum_i |\mathbf{A}[i]|^p$ for $0 < p \leq \epsilon/\log M$ gives $1+\varepsilon$ approximation to D. Estimating such L_p-sum for small p can be done using stable distributions like described earlier. This approach of using small L_p sums first appeared in [43], is quite general and gives efficient results for distinct element estimation, Hamming norms and its generalization to dominance norms [53].

Theorem 13. [43] We can compute a sketch of a non-strict Turnstile stream that requires space $O(1/\epsilon^2 \cdot \log 1/\delta)$ words of size $\log(N + ||\mathbf{A}||_1)$ and allows approximation of D to $1+\epsilon$ factor with probability $1-\delta$.

There is a history of lower bounds as well for estimating D. Early results showed the lower bound of $\Omega(\log N + 1/\varepsilon)$ [10, 18]. Recently, a lower bound of $\Omega(1/\varepsilon^2)$ has been shown [137, 213], nearly optimal with the best known upper bound in [18] for the cash register model.

Estimating inner products. Given $\mathbf{A}$ and $\mathbf{B}$ as data streams, we want to estimate their *inner product* $\mathbf{A} \odot \mathbf{B} = \sum_{i=1}^{n} \mathbf{A}[i]\mathbf{B}[i]$. This is motivated by the important problem of join size estimation in databases. The *join size* of two database relations on a particular dimension is the number of tuples in the cartesian product of the two relations which agree on the value of that dimension. Representing the two relations as vectors $\mathbf{A}$ and $\mathbf{B}$ where $\mathbf{A}[i]$ is the number of tuples in $\mathbf{A}$ with value i in that dimension (likewise for $\mathbf{B}[i]$), join size is their inner product. Inner product estimation also arises in other contexts such as measuring the projection of the signal along different directions, document similarity and correlations between signals.

Use CM Sketch to summarize the signals $\mathbf{A}$ and $\mathbf{B}$ via $\text{count}_{\mathbf{A}}$ and $\text{count}_{\mathbf{B}}$ respectively. Set

$$(\widehat{\mathbf{A} \odot \mathbf{B}})_j = \sum_{k=1}^{w} \text{count}_{\mathbf{A}}[j,k] * \text{count}_{\mathbf{B}}[j,k].$$

Then our estimate $\widehat{\mathbf{A} \odot \mathbf{B}} = \min_j (\widehat{\mathbf{A} \odot \mathbf{B}})_j$. We have $\mathbf{A} \odot \mathbf{B} \leq \widehat{\mathbf{A} \odot \mathbf{B}}$ and with probability at least $1 - \delta$,

$$\widehat{\mathbf{A} \odot \mathbf{B}} \leq \mathbf{A} \odot \mathbf{B} + \varepsilon ||\mathbf{A}||_1 ||\mathbf{B}||_1.$$

The space and time to produce the estimate is $O(\frac{1}{\varepsilon} \log \frac{1}{\delta})$. Updates take time $O(\log \frac{1}{\delta})$. In the special case where $\mathbf{B}[i] = 1$, and $\mathbf{B}$ is zero elsewhere, this result coincides with the point query case above.

Using the moment estimation results from [9], an alternative method uses $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ space and produces an estimate for the inner product with additive error $\varepsilon ||\mathbf{A}|_2 ||\mathbf{B}||_2$ (by definition $||\mathbf{A}||_2 = \sqrt{\sum_i \mathbf{A}[i]^2}$). This alternative uses more space than the CM sketch approach, but provides a tighter error guarantee since the L_2 norm can be quadratically smaller than the L_1 norm.

No matter what alternative is used, the approximation is weak since inner-product estimation is a difficult problem in the communication complexity setting captured by the small space constraint of the data stream models. ∎

Problem 3. Provide improved estimates for L_p-sums including distinct element estimation if input stream has statistical properties such as being Zipfian.

Results are known for estimating the L_2-sum [56] and top-k [32] when the input is Zipfian, heavy-hitters [166] and heavy-hitters under "random, small tail" [162], but in general, only a handful of results are known that improve upon the worst case bounds.

Problem 4. Develop data stream algorithms for estimating the *entropy sum* of the stream $\sum_{i \mid \mathbf{A}[i] \neq 0} \mathbf{A}[i] \log \mathbf{A}[i]$ and the *harmonic sum* $(\sum_{i \mid \mathbf{A}[i] \neq 0} (\mathbf{A}[i])^{-1})$ or more generally, $(\sum_{i \mid \mathbf{A}[i] \neq 0} (\mathbf{A}[i])^{-p})$, under the Turnstile models.

Some of these problems have easy lower bounds: still, the goal would be to find workable approximations under suitable assumptions on input data distributions.

Problem 5. Design random projections using complex random variables or other generalizations, and find suitable streaming applications.

There are instances where considering random projections with complex numbers or their generalization have been useful. For example, let A be a $0,1$ matrix and B be obtained from A by replacing each 1 uniformly randomly with ± 1. Then $E[(det(B))^2] = per(A)$ where $det(A)$ is the determinant of matrix A and $per(A)$ is the permanent of matrix A. While $det(A)$ can be calculated in polynomial time, $per(A)$ is difficult to compute or estimate. The observation above presents a method to estimate $per(A)$ in polynomial time using $det(A)$, but this procedure has high variance. However, if C is obtained from A by replacing each 1 uniformly randomly by $\pm 1, \pm i$, then $E[(det(C))^2] = per(A)$ still, and additionally, the variance falls significantly. By extending this approach using quaternion and Clifford algebras, a lot of progress has been made on decreasing the variance further, and deriving an effective estimation procedure for the permanent [37].

Rangesum random variables. Data stream algorithms need to generate random variables x_i drawn from a stable distribution, for each value i of the domain $[u]$. The *range-summable property* was introduced in [85] where the authors additionally required that given a range $[l, r] \in [u]$, the random variables in the range be summed fast, that is, $\sum_{i=l}^{r} x_i$ be computed fast on-demand. They gave a 3-universal construction based on the Hamming error-correcting code. Subsequently, [102] gave a 7-universal construction based on the dual of the second-order Reed-Muller code. A very general construction was given for stable distributions with $p = 1, 2$ in [105] and for $p \to 0$ in [53]. These work in the Turnstile model and find many applications including histogram computation and graph algorithms on data streams. An improved 5-wise range-summable random variable construction is in [26]. It is $\#P$-complete to construct range-sum Reed-Muller codes of high order [156]. A detailed study of range-sum random variables will be of great interest.

0.6.2 Basic Algorithmic Techniques

There are a number of basic algorithmic techniques: binary search, greedy technique, dynamic programming, divide and conquer etc. that directly apply in the data stream context, mostly in conjunction with samples or random projections. Here are a few other algorithmic techniques that have proved useful.

0.6.2.1 Group Testing

This goes back to an earlier Paul and Carole game. Paul has an integer I between 1 and n in mind. Carole has to determine the number by asking "Is $I \leq x$?". Carole determines various x's, and Paul answers the questions truthfully. How many questions does Carole need, in the worst case? There is an entire area called Combinatorial Group Testing that produces solutions for such problems [76]. In the data stream case, each question is a group of items and the algorithm plays the role of Carole. This setup applies to a number of problems in data streams. Examples are found in:

- Determining the B heavy-hitter Haar wavelet coefficients in Turnstile data streams [105]. This was the first use of combinatorial group testing in data stream algorithms.
- Finding (ϕ, ε)-heavy hitters, quantiles and what are known as deltoids, in Turnstile data streams [51, 55, 54].
- Estimating highest B fourier coefficients by sampling [106].

We describe two specific results to show the framework of group testing for data stream algorithms.

Finding large differences. Say we have two signals $\mathbf{A}$ and $\mathbf{B}$. For any item i, let $D[i] = |\mathbf{A}[i] - \mathbf{B}[i]|$ denote the absolute difference of that item between the two signals. A *ϕ-deltoid* is an item i so that $D[i] >$

$\phi \sum_x D[x]$. It is a heavy-hitter in absolute differences. As before, we need an approximation version. Given $\varepsilon \leq \phi$, the ε-approximate ϕ-deltoid problem is to find all items i whose difference $D[i]$ satisfies $D[i] > (\phi + \varepsilon) \sum_x D[x]$, and to report no items where $D[i] < (\phi - \varepsilon) \sum_x D[x]$. Items between these thresholds may or may not be output.

At high level, the algorithm works as follows. Say $\phi > 1/2$: there is only one (if any) deltoid of interest. Do a standard non-adaptive grouping of items $[1, N]$ into $2 \log N$ "groups" based on the ith bit in their binary representation being a 1 or 0 for each i. For each such group, maintain the sum of all items that fall in it. Then comparing the absolute difference in the sum for 1-group in $\mathbf{A}$ and $\mathbf{B}$ versus that for 0-group for the ith bit will reveal the ith bit of the deltoid, if any. In the general case of up to $1/\phi$ deltoids, use a pairwise independent hash function to map items to $O(1/\varepsilon)$ groups. Then with constant probability no two deltoids will collide in the same group. Thereafter, within each group, repeat the $\log N$ grouping above to identify the unique deltoid if any. The entire process is repeated $O(\log \frac{1}{\delta})$ times in parallel to boost success probability. To sum,

Theorem 14. [54] Each ε-approximate ϕ-absolute deltoid is found by the above algorithm with probability at least $1 - \delta$. The space required is $O(\frac{1}{\varepsilon} \log N \log \frac{1}{\delta})$, time per update is $O(\log N \log \frac{1}{\delta})$, and the expected time to find deltoids is $O(\frac{1}{\varepsilon} \log N \log \frac{1}{\delta})$.

The same overall framework also gives other kinds of deltoids [54] where the differences may be relative or variational. The particular tests performed and statistics kept per group differ. For intuition, consider the following. We could not directly apply the (ϕ, ε)-heavy hitters solutions discussed earlier on the signal $\mathbf{A} - \mathbf{B}$ to detect $1/\phi$ deltoids since they work for the strict Turnstile model and not the non-strict Turnstile model; hence they err in presence of negative $\mathbf{A}[i] - \mathbf{B}[i]$ and can not identify heavy hitters in $|\mathbf{A}[i] - \mathbf{B}[i]|$. The theorem above for deltoids also gives a different (ϕ, ε)-heavy-hitters solution on a single signal, say $\mathbf{A}$.

Problem 6. Paul sees data stream $\mathbf{A}$ followed by stream $\mathbf{B}$. Design a streaming algorithm to determine a certain number of i's with the largest $\frac{\mathbf{A}[i]}{\max 1, \mathbf{B}[i]}$.

This has a strong intuitive appeal: compare today's data with yesterday's and find the ones that changed the most. Certain relative norms similar to this problem are provably hard [53].

Deterministic group testing on streams. All heavy hitter or deltoids solutions known thus far in the Turnstile model use randomization (unlike their counterparts in the cash register model). A natural open problem is if these problems could be solved deterministically in the Turnstile model, strict or non-strict.

We can solve the problem partially. In particular, we consider the special case when there are at most k nonzero $\mathbf{A}[i]$'s. We present a deterministic streaming algorithm to identify the i's with top k $\mathbf{A}[i]$'s which we call *toppers*.

The solution is to construct a group testing procedure of small number of groups in small space, i.e., within constraints of the Turnstile model. Let $k < p_1 < p_2 < .. < p_x = m$ be the sequence of consecutive primes larger than k and with the largest index $x = (k-1) \log_k N + 1$. For each p_i, form groups $0 \bmod p_i, \ldots, p_i - 1 \bmod p_i$, called *$p_i$-groups*. Each of the items in the domain now belong to one p_i group for each i. The claim is, for any k toppers, each one of them will be isolated in at least one of these groups, away from the other $k-1$ toppers. This is true because any such topper α can collide with a different topper β in at most $\log_k N$ p_i'-groups with different i''s. Otherwise, the difference $|\alpha - \beta| < N$ would be divisible by $\log_k N + 1$ different primes $> k$ which is a contradiction. Thus, doing $\log N$ non-adaptive subgrouping within each other groups above will solve the problem of identifying all toppers without any error. Since $m = O(k \log_k N(\log k + \log \log N))$, the total space cost is $\mathrm{poly}(k, \log N)$. To summarize:

Theorem 15. [98] There exists a deterministic algorithm in the Turnstile model to identify the k-toppers, using space, per item processing time and compute time all $\mathrm{poly}(k, \log N)$.

This is the first-known deterministic algorithm in the Turnstile data stream model for any nontrivial problem. It can be extended to the *small tail* situation where sum of the tail $N - k$ of the $\mathbf{A}[i]$'s in sorted order sum to less than the kth largest $\mathbf{A}[i]$. The approach above can be thought of as constructing a certain separating code in small space. This leads to:

Problem 7. Present polylog space and time construction algorithms for constructing various codes.

0.6.2.2 Tree Method

This method applies nicely to the Time Series model. Here, we have a (typically balanced) tree atop the data stream. As the updates come in, the leaves of the tree are revealed in the left to right order. In particular, the path from the root to the most currently seen item is the right boundary of the tree we have seen. We maintain a small-space data structure on the portion of the tree seen thus far, typically, some storage per level; this can be updated as the leaves get revealed by updating along the right boundary. This overall algorithmic scheme finds many applications:

- Computing the B largest Haar wavelet coefficients of a Time Series data stream [9].
- Building a histogram on the Time Series data stream [118]. This also has applications to finding certain outliers called the *deviants* [177].
- Building a parse tree atop the Time Series data stream seen as a string [50]. This has applications to estimating string edit distances as well as estimating size of the smallest grammar to encode the string.

In what follows, we provide more insight into each of these categories of algorithms.

Estimating wavelet coefficients. There are many kinds of wavelets. We work with one of the simplest, most popular, i.e., Haar wavelets [124].

Definition 3. Let N be a power of 2. Define $N - 1$ proper wavelet functions on $[0, N - 1)$ as follows. For integer $j, 0 \leq j < \log(N)$ and integer $k, 0 \leq k < 2^j$, we define a *proper wavelet* $\phi(x)[j, k]$ by $-\sqrt{2^j/N}$ for $x \in [kN/2^j, kN/2^j + N/2^{j+1})$, $+\sqrt{2^j/N}$ for $x \in [kN/2^j + N/2^{j+1}, (k + 1)N/2^j)$ and 0 otherwise. Additionally, define a wavelet ϕ, also known as a *scaling function*, that takes the value $+1/\sqrt{N}$ over the entire domain $[0, N)$. Number the $\phi(x)[j, k]$'s as $\psi_0, \psi_1, \ldots, \psi_{N-2}$ in some arbitrary order and ϕ as ψ_{N-1}.

Wavelets can be used to represent signals. Assume N is a power of 2. Any signal $\mathbf{A}$ is exactly recoverable using the wavelet basis, i.e.,

$$\mathbf{A} = \sum_i \langle \mathbf{A}, \psi_i \rangle \, \psi_i.$$

Typically, we are not interested in recovering the signal exactly using all the N *wavelet coefficients* $\mathfrak{c}_i = \langle \mathbf{A}, \psi_i \rangle$; instead, we want to represent the signal using no more than B wavelet coefficients for some $B \ll N$. Say Λ is a set of wavelets of size at most B. Signal $\mathbf{A}$ can be represented as $\mathbf{R}$ using these coefficients as follows:

$$\mathbf{R} = \sum_{i \in \Lambda} \langle \mathbf{A}, \psi_i \rangle \, \psi_i.$$

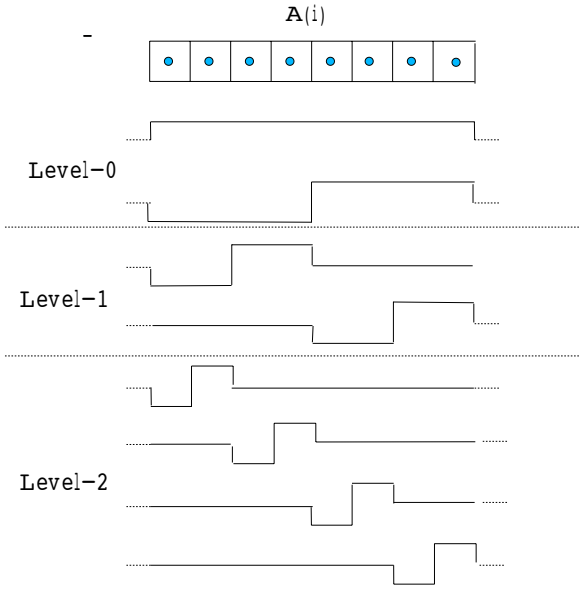

Fig. 2 Wavelet-vectors (N=8)

Clearly $\mathbf{R}$ can only be an approximation of $\mathbf{A}$ in general. The *best B-term representation* (aka *wavelet synopsis*) of $\mathbf{A}$ is the choice of Λ that minimizes the error $\varepsilon_{\mathbf{R}}$ of representing $\mathbf{A}$ which is typically the sum-squared-error, i.e., $\varepsilon_{\mathbf{R}} = ||\mathbf{R} - \mathbf{A}||_2^2 = \sum_i (\mathbf{R}[i] - \mathbf{A}[i])^2$. From a fundamental result of Parseval from 1799 [184, 21], it follows that the best B-term representation for signal $\mathbf{A}$ is $\mathbf{R} = \sum_{i \in \Lambda} c_i \psi_i$, where $c_i = \langle \mathbf{A}, \psi_i \rangle$ and Λ of size k maximizes $\sum_{i \in \Lambda} c_i^2$. This is because the error of $\mathbf{R}$ is

$$\varepsilon_{\mathbf{R}} = \sum_{i \in \Lambda} (c_i - \langle \mathbf{A}, \psi_i \rangle)^2 + \sum_{i \notin \Lambda} \langle \mathbf{A}, \psi_i \rangle^2 .$$

Hence, choosing the B largest $|c_i|$'s suffices to get the best B term representation for any signal.

Consider the problem of estimating the best B term representation for $\mathbf{A}$ given in the time series model. Note that there is a tree representation of the Haar wavelet transformation where a full binary tree is laid atop the $\mathbf{A}$; each internal node corresponds to a 2^j sized interval on the signal. Associate with it the wavelet that is 0 outside that interval and is a difference of the summand for the left half-interval and that of the right half-interval. Thus Haar wavelet vectors can be associated with the nodes of the tree in a one-to-one fashion (except ψ_{N-1}). Recall also that in the Timeseries model, we get the signal values specified as $(i, \mathbf{A}[i])$ in the increasing order of i's. Our algorithm is as follows. Consider reading the data stream and say we are at some position i' in it, that is, we have seen all $\mathbf{A}[i]$'s for $i \leq i'$, for some i'. We maintain the following two sets of items:

(1) Highest B-wavelet basis coefficients for the signal seen thus far.
(2) $\log N$ *straddling* coefficients, one for each level of the Haar wavelet transform tree. At level j, the wavelet basis vector that *straddles* i' is $\psi_{j,k}$ where $k(N/2^j) \leq i' \leq k(N/2^j) + N/2^j - 1$, and there is at most one such vector per level.

When the following data item $(i'+1, \mathbf{A}[i'+1])$ is read, we update each of the straddling coefficients at each level if they get affected. Some of the straddling coefficients may no longer remain straddling. When that happens, we compare them against the highest B-coefficients and retain the B highest ones and discard the remaining. At levels in which a straddling coefficient is no longer straddling, a new straddling coefficient

is initiated. There will be only one such new straddling coefficient for each level. In this manner, at every position on the data stream, we maintain the highest B-wavelet basis coefficients exactly. This gives,

Theorem 16. With at most $O(B+\log N)$ storage, we can compute the highest (best) B-term approximation to a signal exactly in the Timeseries model. Each new data item needs $O(B + \log N)$ time to be processed; by batch processing, we can make update time $O(1)$ in the amortized case.

Histogram Construction. A B-bucket *histogram* is a partition of $[0, N)$ into intervals $[b_0, b_1) \cup [b_1, b_2) \cup \cdots \cup [b_{B-1}, b_B)$, where $b_0 = 0$ and $b_B = N$, together with a collection of B heights h_j, for $0 \leq j < B$, one for each interval (aka bucket). In vector notation, write χ_S for the vector that is 1 on the set S and zero elsewhere. The histogram can be thought of as an approximation to the $\mathbf{A}$ given by

$$\mathbf{R} = \sum_{0 \leq j < B} h_j \chi_{[b_j, b_{j+1})},$$

that is, each point $\mathbf{A}[i]$ is approximated by h_j, where j is the index of the bucket containing i (the unique j with $b_j \leq i < b_{j+1}$). Thus, in building a B-bucket histogram, we want to choose $B-1$ boundaries b_j and B heights h_j that minimize the sum square error

$$\left\| \mathbf{A} - \sum_{0 \leq j < B} h_j \chi_{[b_j, b_{j+1})} \right\|^2.$$

Once we have chosen the boundaries, the best bucket height on an interval I is the average of $\mathbf{A}$ over i. In the data stream context, $\mathbf{R}$ can not be optimally computed. Hence, an approximation is needed. Fix integer B and positive reals ϵ and δ. The (B, ε)-*histogram problem* is to find a B-bucket histogram $\mathbf{R}$ for $\mathbf{A}$ such that with probability at least $1 - \delta$,

$$\|\mathbf{A} - \mathbf{R}\|^2 \leq (1 + \epsilon) \|\mathbf{A} - \mathbf{R}_{\text{opt}}\|^2,$$

where $\mathbf{R}_{\text{opt}}$ is the optimal B-bucket histogram.

There is a simple dynamic programming solution when there are no space constraints. Fix $k < N$ and $\ell \leq B$. Assume that for each $i < k$, we have found the optimal $(\ell - 1)$-bucket histogram on the prefix of data indexed by $[0, i)$. To find the best ℓ-bucket histogram on $[0, k)$, try all possibilities $m < k$ for the final boundary, and form a histogram by joining the best $(\ell - 1)$-bucket histogram on $[0, m)$ with the 1-bucket histogram on $[m, k)$. The time for this step is therefore at most N to try all values of m. Since this has to be repeated for all $k < N$ and all $\ell \leq B$, the total time is $O(N^2 B)$. The space is $O(BN)$, to store a table of a B-bucket histogram for each i. Space $O(N)$ suffices if one is willing to spend more time recomputing best histograms. Recently, generalizing ideas reminiscent of [128], fast algorithms have been designed for reconstructing the histogram with small (near-linear) space [115].

Now consider the problem in the Time Series data stream model. The algorithm in [118] works on B levels simultaneously. In particular, let $\text{sol}[p, x]$ be the optimal cost solution on $\mathbf{A}[1, x]$ with p buckets, $1 \leq p \leq B$. For each p, the algorithm maintains an series of points b_j^p that are successively $1 + \delta$ factor more in error, i.e., roughly, $\text{sol}[p, b_j^p] \leq (1 + \delta)\text{sol}[p, b_{j-1}^p]$. When a new point is seen, the rightmost b_j^p is either set and new candidate b_{j+1}^p begun, or it is updated as needed (this corresponds to the straddling case in the previous example). In particular, use

$$\text{sol}[p + 1, x + 1] = \min_j \text{sol}[p, b_j^p] + \text{var}[b_j^p + 1, \ldots, x + 1]$$

where $\mathrm{var}[]$ is the variance of the signal in the specified interval. This is akin to the dynamic programming solution but only the specific endpoints that delineate significant error growth are involved, at each level $p, 1 \leq p \leq B$. An analysis shows that the number of endpoints that are tracked is $O((B^2 \log N \log(\max_i \mathbf{A}[i]))/\varepsilon)$ and the above algorithm gives $(1+\delta)^p$ approximation to $\mathrm{sol}[p+1, x+1]$. Summarizing:

Theorem 17. [118] There exists an $1+\varepsilon$ approximation to $\mathbf{R}_{\mathrm{opt}}$ in $O((B^2 \log N \log(\max_i \mathbf{A}[i]))/\varepsilon)$ space as well time per update in the Time Series data stream model.

The example above of the tree method of data stream design is more sophisticated than the simple wavelet algorithm. There have been a series of improvements for the histogramming problem in the Time Series model further refining the complexity in terms of per-item processing time or dependence on B [117, 119, 180].

Note. There is an intimate relationship between representing a signal by Haar wavelets and by histograms (which is the piecewise-constant object defined earlier).

- The best $2\log(N)B$-term Haar wavelet representation $\mathbf{R}$ is at least as good as the best B-bucket histogram.
  Any histogram element χ_I on interval I can be expressed as a $2\log(N)$-term Haar wavelet representation. This is because $\langle \chi_I, \psi \rangle = 0$ unless ψ's nonzero parts intersect an endpoint of I.
- The best $(3B+1)$ bucket histogram representation $\mathbf{R}$ is at least as good as the best B-bucket Haar wavelet representation.
  Any Haar wavelet term is itself a 4-bucket histogram (*i.e.*, a histogram with 3 boundaries), so a B-term Haar wavelet representation can be viewed as a $(3B+1)$-bucket histogram.

String parse tree. There are many applications that need to manipulate long strings. Given string S, we will parse it hierarchically as described below and obtain a parse tree T. At each iteration i, we start with a string S_i and *partition* it into *blocks* of length 2 or 3. We replace each such block $S[j \cdots k]$ by its *name* which is any one-to-one hash function on substrings of S. Then S_{i+1} consists of the names for the blocks in the order in which they appear in S_i. So $|S_{i+1}| \leq |S_i|/2$. Initially $S_0 = S$, and the iterations continue until we are left with a string of length 1. The tree T consists of levels such that there is a node at level i for each of the blocks of S_{i-1}; their children are the nodes in level $i-1$ that correspond to the symbols in the block. Each character of $S_0 = S$ is a leaf node.

The crux is to partition the string S_i at iteration i. This will be based on designating some local features as "landmarks" of the string. Consider a substring which is not repeating, that is, each symbol differs from its predecessor. (Other cases are simpler.) For each symbol $S_i[j]$ compute a new label as follows. Denote by ℓ the index of the least significant bit in which $S_i[j]$ differs from $S_i[j-1]$, and let $bit(\ell, S_i[j])$ be the value of $S_i[j]$ at the ℓth bit location. Form $label(S_i[j])$ as $2\ell + bit(\ell, S_i[j])$ — in other words, as the index ℓ followed by the value at that index. For any j, if $S_i[j] \neq S_i[j+1]$ then $label(S_i[j]) \neq label(S_i[j+1])$. Generate a new sequence by replacing the positions by their labels. If the original alphabet was size τ, then the new alphabet is sized $2\log|\tau|$. Iterate this process until alphabet size is unchanged; it can be eventually reduced to size 3. Pick landmarks from resulting string by selecting any position j which is a local maximum and in addition selecting any j which is a local minimum and is not adjacent to an already chosen landmark. For any two successive landmark positions j_1 and j_2, this insures that $2 \leq |j_1 - j_2| \leq 3$. Partition the string around the landmarks. An example of the parse tree generated by the process above is shown in Figure 3.

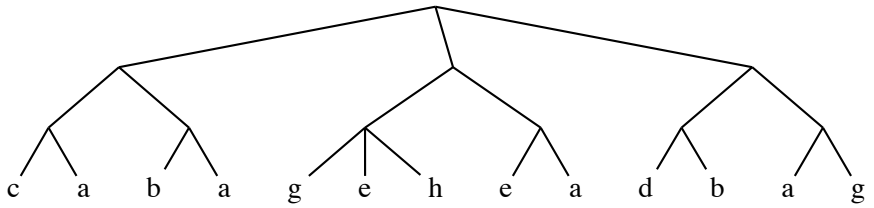

Fig. 3 The hierarchical structure of nodes is represented as a parse tree on the string S.

The chief property of the parsing is that of locality: determining the closest landmark to position j in a string of length τ depends on only $O(\log^* \tau)$ positions to its left and right. As a result, edit operations such as inserts, deletes, changes, and substring moves applied to S have only a local effect on the parsing at any level. Also of interest is that the parsing can be done obliviously at S independent of a different string T and still because of this locality, S and T will get parsed nearly the same way in segments which have small edit distance. Therefore, the parsing gives a hierarchical way to reduce the string sizes of two strings independently while controlling the loss of quality in parsing into similar substrings. The parsing can be done in $O(N \log^* N)$ time. This parsing has found many applications to compression [191], edit distance computation [190], dynamic string matching [192], communication-efficient string exchanges [49], string nearest neighbors [176], streaming embedding [50], etc.

The interest to us here is that the parsing and embedding that underlies many of these applications can be done in the Time Series model by keeping the "straddling" of the parsing at each level of the tree, similar to the wavelet example. The space used is $O(\log N \log^* N)$ because at each of the $O(\log N)$ levels, the algorithm maintains $O(\log^* N)$ information corresponding to the straddling portion.

The construction above is a generalization of the symmetry breaking technique in [40] for permutations to sequences which was pioneered as Locally Consistent Parsing (LCP) starting from [191], and has many variant forms including those in [190, 176, 50] under different names. ∎

All of these results above are in the Time Series Model which is where the tree method finds many applications. Here is a problem of similar ilk, but it needs new ideas.

Problem 8. Given a signal of size N as a Time Series data stream and parameters B and k, the goal is to find k points (*deviants*) to remove so that finding the B largest coefficients for the remaining signal has smallest sum squared error. This is the wavelet version of the problem studied in [177].

There are other applications, where the tree hierarchy is imposed as an artifact of the problem solving approach. The k-center [31] and k-median algorithms [120, 33] on the data stream can be seen as a tree method: building clusters on points, building higher level clusters on their representatives, and so on up the tree. Similarly, recent work on core sets and other geometric problems (see Survey talk by Piotr Indyk [136] and Timothy Chan [29]) fall into this category. Yair Bartal's fundamental result of embedding arbitrary metrics into tree metrics may find applications in data streams context, by reducing difficult problems to ones where the tree method can be applied effectively.

0.6.2.3 Other algorithmic techniques

Here is a brief overview of other techniques in data stream algorithmics.

Exponential Histograms. To algorithms designers, it is natural to think of *exponential histograms*—dividing a line into regions with boundaries at distance 2^i from one end or keep dividers after points of rank 2^i—when one is restricted to use a polylog space data structure. It corresponds to space, rank, or other

statistics-based partitioning of the domain into geometrically increasing sizes. This technique has been used in

- one dimensional nearest neighbor problems and facility location [149];
  Here, usage is quite simple: exponential histograms are kept based on ranks as well as distance.
- maintaining statistics within a window [66];
  Here, a variety of exponential histograms are developed including those that maintain sketches within the buckets. The complication here is that the exponential bucketing has to be maintained on last W items seen at any time and hence this requires merging exponential buckets on the stream.
- problems on geometric streams where space is partitioned in one or higher dimensions into geometrically spaced grids [135, 93].
  Here the partitioning is spatial, or distance-based. The additional complication is one needs to maintain quite sophisticated statistics on points within each of the buckets in small space. This entails using various (non)standard moments and norm estimation methods within the buckets.

Exponential histograms is a simple and natural strategy which is likely to get used seamlessly in data stream algorithms.

Monotone Chain in Random Sequences. Consider a sequence $S[1, \ldots, N]$ of integers. Consider the hashed sequence $T[1, \ldots, N]$ where $T[i] = h(S[i])$ for a random hash function. Define the monotonic chain C to be the minimum element and successive minimums, each to the right of the predecessor. What is the expected length of C? Consider building a *treap* (See [171, 170]) on S: the list C is the right spine of this treap. If the hash values are a fully independent, random permutation, then it is well known that [170, 171, 195]:

Theorem 18. With high probability, $|C| = \Theta(H_N)$, where H_N is the Nth Harmonic number, given by $1 + 1/2 + 1/3 + \cdots + 1/N = \Theta(\log N)$.

However, the hash function that is used in data stream applications are not fully independent; instead, they have limited independence such as a random polynomial of degree $O(\log 1/\varepsilon)$ which is ε-min-wise independent. For a hash function that is chosen randomly from such a family, the expected length of the right spine and hence that of list C is still $\Theta(\log N)$ [171, 195].

In data stream algorithms, C can thus be kept track of entirely since its size fits the space constraints. This approach underlies methods for keeping track of rarity and similarity within recent windows [67], distinct elements and other predicates on the inverse sample on windows [58], multigraph aggregates on windows [57] as well as in keeping moments over network neighborhoods [38]. It is a useful primitive in windowed version of data stream models which is discussed later.

Robust Approximation. This concept is a variation on the local minima/maxima suited for approximations. Consider constructing a near-optimal B bucket histogram for the signal. We use wavelets and histograms interchangeably. For some $T \leq O(1/(\varepsilon^2 \log(1/\varepsilon)))$ and any signal $\mathbf{A}$, $\mathbf{R}_{\rm rob}$, the *robust approximation* is the representation formed by greedily taking $B \log(N)$ wavelet terms at a time until either we get $TB \log(N)$ terms, or taking an additional $B \log(N)$ wavelet terms improves (reduces) the residual $\|\mathbf{A} - \mathbf{R}_{\rm rob}\|_2^2$ by a factor no better than $(1 - \varepsilon')$, where $\varepsilon' \geq \Omega(\varepsilon^2)$. The power of this robust histogram is that it can be computed in data stream models including the Turnstile model [105] and:

Theorem 19. Let $\mathbf{H}$ be the best B-bucket histogram representation to $\mathbf{R}_{\text{rob}}$. Then $\mathbf{H}$ is a $1+\varepsilon$-approximation to the best B-bucket histogram to $\mathbf{A}$.

Thus, it suffices to do the basic dynamic programming method on the robust representation for solving the histogram problem. This approach gives the most general data stream algorithms known for the histogram construction problem in e.g., the Turnstile model.

Theorem 20. [105] There is an algorithm that given parameters B, N, M, ε monitors $\mathbf{A}$ with $\|\mathbf{A}\|^2 \leq M$ in the Turnstile model and answers B-bucket histogram queries with a B bucket $\mathbf{H}$ such that $\|\mathbf{A}-\mathbf{H}\|^2 \leq (1+O(\varepsilon))\|\mathbf{A}-\mathbf{H}_{\text{opt}}\|^2$, where $\mathbf{H}_{\text{opt}}$ is the best possible B-bucket histogram representation to $\mathbf{A}$. The algorithm uses space, update time and computing time of $O(\text{poly}(B, \log N, \log M, 1/\varepsilon))$.

The notion of "robust" approximations has found applications in constructing multidimensional histograms [178], rangesum histograms [179], and more recently, weight-optimal histograms [181], under different data stream models. In particular, in [181] a different—weaker—notion of robustness is identified and used. Robust approximations may find other applications in the future. ∎

As is typical in a nascent area, the techniques above have been rediscovered few times in data stream algorithms.

0.6.3 Lower Bounds

A powerful theory of lower bounds is emerging for data stream models.

Compressibility argument. In Section 0.2.2 there is an example. One argues that if we have already seen a portion S of the data stream and have stored a data structure D on S for solving a problem P, then we can design the subsequent portion of the stream such that solving P on the whole stream will help us recover S precisely from D. Since not every S can be compressed into small space D, the lower bound on size of D will follow. This is a basic and simple lower bound argument, often applied to show that exact computation of various functions is not possible in sublinear space.

Communication Complexity. Communication complexity models have been used to establish lower bounds for data stream problems. Alon, Matias and Szegedy [10] established the relationship between data stream and communication complexity models for lower bounds. In particular, consider the *multiparty set disjointness* problem where each of the t players is given a set from $\{1, \ldots, N\}$ and told the sets are either (a) pairwise disjoint or (b) have intersection of size precisely 1. In [10], a lower bound of $\Omega(N/t^4)$ on the number of bits needed for one designated player to distinguish between the two cases. Via a reduction, this shows a lower bound of $\Omega(N^{1-5/k})$ for space complexity of approximating F_k in the cash register model. The lower bound on communication complexity for the multiparty set disjointness problem was improved in a nice work by Bar-Yossef et al [17] to $\Omega(N/t^2)$ and finally to $\Omega(N/(t\log t))$ in [27]. For special case of one way communication complexity where players each speak once in a given order, the lower bound is $\Omega(\varepsilon^2 N/t^{1+\varepsilon})$ [17] which has been improved to $\Omega(N/t)$ [27]; this complexity immediately implies a lower bound of $\Omega(N^{1-2/k})$ space for F_k estimation the cash register data stream model, nearly optimal with the best known upper bound [138].

Estimating set disjointness and the Index problem are classical, hard problems in Communication Complexity [155] that code the difficulty in estimating some of the basic statistics on data streams. Both have $\Omega(N)$ lower bound on the communication complexity. A useful lower bounding technique is to reduce hard

problems such as these to the data stream problem under study. An example of this is shown in Theorem 11 for the Index problem. For the set disjointness problem, here is another example.

Consider a stream of *multiple* signals given by values $(i, a_{i,j})$ where i's correspond to the domain $[1, N]$, j's index the different signals and $a_{i,j} \geq 0$ give the value of the jth signal at point i. Consider norms that are cumulative of the multiple signals in the data stream, e.g., the *min-dominance* norm is $\sum_i \min_j \{a_{i,j}\}$. Analogously, the *max-dominance* norm is $\sum_i \max_j \{a_{i,j}\}$. These estimate norms of the "upper/lower envelope" of the multiple signals, or alternatively, norms of the "marginal" distribution of tabular data streams. Max-dominance can be approximated well on data streams using generalization of techniques known for distinct count estimation [53]. In contrast, it is not possible to compute a useful approximation to the min-dominance norm.

Theorem 21. Any algorithm that approximates $\sum_i \min_j \{a_{i,j}\}$ to a constant factor requires $\Omega(N)$ bits of storage.

Proof. Consider two parties P and C who each hold a subset (X and Y respectively) of the integers 1 to N. Suppose that there were some algorithm known to both A and B which allowed the approximation of the min-dominance norm. Then A generates data stream $Str(X)$ where $Str(X)_i = (i, 1)$ iff $i \in X$ and $Str(X)_i = (i, 0)$ iff $i \notin X$, and passes this stream to the algorithm. Following this, A takes the complete memory state of the algorithm and communicates it to B. B then proceeds to run the same algorithm based on this memory state applied to the same transformation of Y, that is, $Str(Y)$. The result is an approximation of min-dominance norm for the concatenation of the streams $Str(X)$ and $Str(Y)$ which is $|X \cap Y|$— in other words, the sum of the minimum values is exactly the intersection size of the two sets, which needs $\Omega(N)$ bits in the communication model. Therefore, the size of the memory state of the proposed streaming algorithm must be $\Omega(N)$ bits. ∎

0.6.4 Summary and Data Stream Principles

The algorithmic ideas above have proved powerful for solving a variety of problems in data streams. On the other hand, using the lower bound technology, it follows that many of these problems—finding most frequent items, finding small error histograms, clustering, etc.—have versions that are provably hard to solve exactly or even to approximate on data streams. However, what makes us successful in solving these problems is that in real life, there are two main principles that seem to hold in data stream applications.

- *Signals in real life have "a few good terms" property.*

  Real life signals $\mathbf{A}[0 \cdots N-1]$ have a small number B of coefficients that capture most of the trends in $\mathbf{A}$ even if N is large. For example, Figure 4 shows the sum squared error in reconstructing a distribution with B highest coefficients for various values of B, and two different distributions: the number of bytes of traffic involving an IP address as the source and as the destination. So, here $N = 2^{32}$, total number of IP addresses possible. Still, with $B = 800$ or so, we get the error to drop by more than 50%. For capturing major trends in the IP traffic, few hundred coefficients prove adequate.

  In IP traffic, few flows send a large fraction of the traffic [206]. That is, of the 2^{64} possible $(src, dest)$ IP flows, if one is interested in heavy hitters, one is usually focused on a small number (few hundreds?) of flows. This means that one is typically interested in tracking k most frequent items, for small k, even if N is large.

  This phenomenon arises in other problems as well: one is typically interested in a small number k of facilities or clusters, etc.

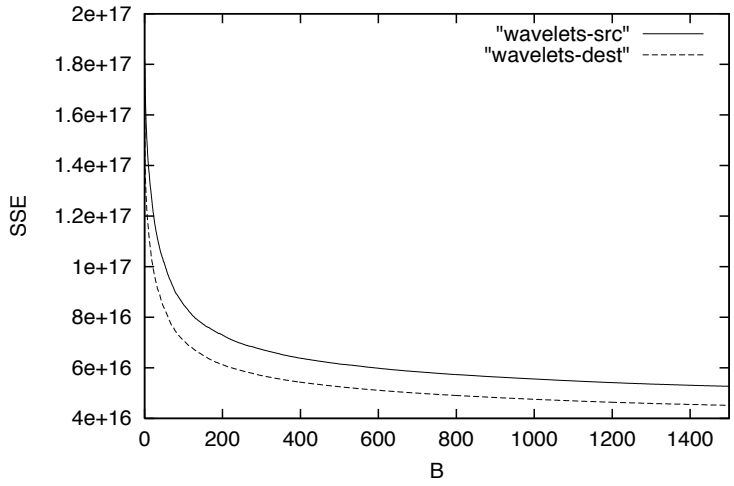

Fig. 4 Decay of SSE of top wavelet coefficients on IP data.

- *During unusual events in data streams, exceptions are significantly large.*

  The number of rare IP flows—flows involving a small number of packets—and the number of distinct IP flows is significantly large during network attacks. When there are data quality problems in data streams—say packets are dropped in measurement platform—the problem is abundant, e.g., the packet drops happen in many places because of correlations.

These two principles are used implicitly in designing many of the useful data stream algorithms. Applied algorithmicists need to keep these principles in mind while abstracting the appropriate problems to study.

0.7 Streaming Systems

There are systems that (will) process data streams. They fall into two categories.

First, is the hands-on systems approach to data streams. One uses operating system support to capture streams, and perhaps use special hooks in standard programming languages like C or scripting tools like perl or database functionality to get additional facility in manipulating streams. The work on telephone Call Detail Records analysis using Hancock [60]—a special-purpose language—is such an example. Other researchers work in this framework, and quite successfully process large data sets. Many companies such as NARUS and others can capture a suitable subset of the stream and use standard database technology atop for specific analyses.

Second, there are database systems where the internals are directly modified to deal with data streams. This is an active research area that involves new stream operators, SQL extensions, novel optimization techniques, scheduling methods, the continuous query paradigm, etc., the entire suite of developments needed for a *data stream management system* (DSMS). Projects at various universities of this type include NiagaraCQ [35], Aurora [1], Telegraph [152], Stanford Stream [11] etc. They seem to be under development, and demos are being made at conferences. Another system in this category is Gigascope [63] which is operationally deployed in an IP network. It deal with stream rates generated in IP networks, but it provides only features suitable for IP network data analysis. It is not yet suitable for general purpose data stream management for a variety of data streams. Streambase is a commercial company with its data stream management system [16].

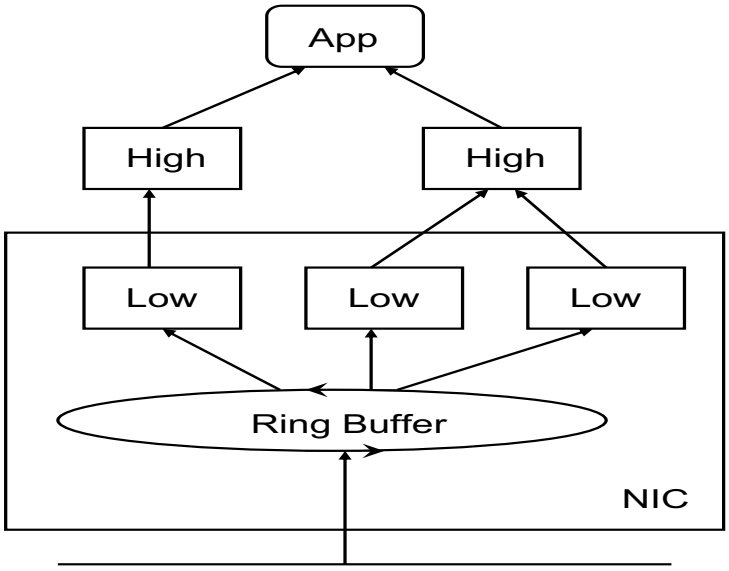

Fig. 5 Gigascope Architecture

One of the outstanding questions with designing and building DSMSs is whether there is a need. One needs multiple applications, a well-defined notion of stream common to these applications, and powerful operators useful across applications in order to justify the effort needed to actualize DSMSs. At this fledgling point, the IP network traffic monitoring is a somewhat well developed application. Financial streams seem to be another developing application. But more work needs to be done—in applications like text and spatial data stream scenarios—to bolster the need for DSMSs.

To what extent have the algorithmic ideas been incorporated into the emerging streaming systems? In the last two years, some significant developments have happened in Gigascope to incorporate data streaming algorithms. Gigascope is a DSMS specialized for monitoring network feeds. It is programmed using an SQL-like language, GSQL, allowing the simple specification of complex monitoring tasks. This takes advantage of the large body of research on query optimization and the specialized application to make many aggressive performance optimizations. As a result, Gigascope queries often run faster than hand-written code. In order to obtain high performance in data reduction, Gigascope has a two level architecture (see Figure 5). Query nodes which are fed by source data streams (e.g., packets sniffed from a network interface) are called *low level* queries, while all other query nodes are called *high level* queries. Data from a source stream is fed to the low level queries from a ring buffer without copying. Early data reduction by low level queries is critical for high performance: hence, low level queries need to be fast enough to deal with input stream rate and yet be rich enough to perform significant reduction in the data rate; the high level queries do more sophisticated query processing and finish up the computation. What is unique about Gigascope is the flexible infrastructure it provides in controlling the low and high level queries independently for optimizing the overall performance better.

Two specific data stream methods have been incorporated into Gigascope.

- *CM Sketch has been incorporated into Gigascope as a user-defined function.*

  In [46], the performance of CM Sketch is shown on live IP traffic streams at close to 2 million packets (updates) a second with minimal computation load on routers. Careful engineering is needed to tradeoff the complexity and richness of low-level vs high-level queries to achieve

this performance and show CM sketch to be usable and effective at these streaming speeds. Since CM sketch is generic, its optimization within Gigascope helps implement many different algorithms directly and easily, including those for rangesums, quantiles, heavy hitters, deltoids, F_2 estimation, etc.

- *A large class of sampling algorithms have been incorporated into Gigascope as an operator.*

  There are many different sampling methods for data streams and it is a challenge to incorporate them all into Gigascope. In particular, the difficulty is not coding them as individual functions which creates its own challenges, but rather optimizing them carefully for the two-level architecture of Gigascope. In [140], authors abstracted a common template for many of these sampling methods and implemented that as an operator. They showed that using this template, sampling methods can be added to Gigascope very easily; further, these methods now work at about 2 million packets (updates) a second with only a small overhead on the processor in the routers.

These two additions are significant: they optimize the performance of the key ingredients on which higher level stream analyses can be performed. They also show real performance on live IP traffic feeds.

There are following two major challenges in building future DSMSs.

Problem 9. Design and build a scalable, *distributed* data stream management system with truly distributed, communication-efficient query processing.

This is the next phase of the challenge in actualizing DSMSs and there is already some emerging work [16].

Another emerging phase is IP content analysis. Increasingly, IP traffic analysis involves looking at the contents (not just the headers) of the IP packets for detecting malicious activity. This involves streaming analysis on the "strings" in packet contents.

Problem 10. Build a scalable DSMS for IP *content* analyses. Notice that the contents are more voluminous than the headers; also, string analyses are richer and more computationally intensive than header analyses. Hence, this is a serious challenge.

0.8 New Directions

This section presents some results and areas that did not get included above. The discussion will reveal open directions and problems: these are not polished gems, they are uncut ideas. Sampath Kannan has an interesting talk on open problems in streaming [143]. Piotr Indyk has many open geometric problems in data stream models [136].

0.8.1 Related Areas

In spirit and techniques, data streaming area seems related to the following areas.

- *Sublinear time algorithms:* This area focuses typically on sublinear *time* algorithms. Typically they rely on powerful oracle access to underlying objects, and devise methods that test objects and separate them based on whether they are far from having a desired property, or not. See the survey on sublinear time methods [154]. Typically these results focus on sampling and processing only sublinear amount of data. As mentioned earlier, sampling algorithms can be simulated by streaming algorithms, and one can typically do more in streaming models.

- *PAC learning:* In [168] authors studied sampling algorithms for clustering in the context of PAC learning. More detailed comparison needs to be done for other problems such as learning fourier or wavelet spectrum of distributions between streaming solutions and PAC learning methods.
- *On line algorithms:* Data stream algorithms have an on line component where input is revealed in steps, but they have resource constraints that are not typically incorporated in competitive analysis of on line algorithms.
- *Markov methods*. Some data streams may be thought of as intermixed states of multiple markov chains. Thus we have to reason about maximum likelihood separation of the markov chains [20] or reasoning about individual chains [145] under resource constraints of data streams. This view needs to be developed a lot further.

0.8.2 Functional Approximation Theory

One of the central problems of modern mathematical approximation theory is to approximate functions concisely, with elements from a large candidate set $\mathcal{D}$ called a *dictionary*; $\mathcal{D} = \{\phi_i\}_{i \in I}$ of unit vectors that span $\mathcal{R}^N$. Our input is a signal $\mathbf{A} \in \mathcal{R}^N$. A representation $\mathbf{R}$ of B terms for input $\mathbf{A} \in \mathcal{R}^N$ is a linear combination of dictionary elements, $\mathbf{R} = \sum_{i \in \Lambda} \alpha_i \phi_i$, for $\phi_i \in \mathcal{D}$ and some $\Lambda, |\Lambda| \leq B$. Typically, $B \ll N$, so that $\mathbf{R}$ is a concise approximation to signal $\mathbf{A}$. The error of the representation indicates by how well it approximates $\mathbf{A}$, and is given by $\|\mathbf{A} - \mathbf{R}\|_2 = \sqrt{\sum_t |\mathbf{A}[t] - \mathbf{R}[t]|^2}$. The problem is to find the best B-term representation, *i.e.*, find a $\mathbf{R}$ that minimizes $\|\mathbf{A} - \mathbf{R}\|_2$. A signal has a $\mathbf{R}$ with error zero if $B = N$ since $\mathcal{D}$ spans $\mathcal{R}^N$.

The popular version of this problem is one where the dictionary is a Fourier basis, i.e., ϕ_i's are appropriate sinusoidal functions. Haar wavelets comprise another special case. Both these special cases are examples of *orthonormal* dictionaries: $||\phi_i|| = 1$ and $\phi_i \perp \phi_j$. In this case, $|\mathcal{D}| = N$. For orthonormal dictionaries when $B = N$, Parseval's theorem holds:

$$\sum_i \mathbf{A}[i]^2 = \sum_i \alpha_i^2.$$

For $B < N$, Parseval's theorem implies that the best B term representation comprises the B largest inner products $| \langle \mathbf{A}, \phi \rangle |$'s over $\phi \in \mathcal{D}$.

In functional approximation theory, we are interested in larger—*redundant*—dictionaries, so called because when $\mathcal{D} > N$, input signals have two or more distinct zero-error representation using $\mathcal{D}$. Different applications have different natural dictionaries that best represent the class of input signals they generate. There is a tradeoff between the dictionary size and the quality of representation when dictionary is properly chosen. Choosing an appropriate dictionary for an application is an art. So, the problem of determining an approximate representation for signals needs to be studied with different redundant dictionaries.

There are three directions: studying specific dictionaries derived from applications, studying the problem for an arbitrary dictionary so as to be completely general, or studying modified norms.

Studying specific dictionaries. One specific dictionary of interest is *Wavelet Packets*. Let $W_0(x) = 1$ for $0 \leq x < 1$. Define $W_1, W_2, \ldots$ as follows.

$$\begin{aligned} W_{2n}(x) &= W_n(2x) + W_n(2x-1) \\ W_{2n+1}(x) &= W_n(2x) - W_n(2x-1) \end{aligned}$$

Wavelet packets comprises vectors defined by $w_{j,n,k}(x) = 2^{j/2} W_n(2^j x - k)$ for different values of j, n, k. They are richer than the well known Haar wavelets, and hence, they potentially give better compression.

As before, the problem is to represent any given function $\mathbf{A}$ as a linear combination of B vectors from the wavelet packets. Two such vectors w and w' are not necessarily orthogonal, hence merely choosing the B largest $|\langle \mathbf{A}, w_{j,n,k} \rangle|$'s does not optimally solve the problem.

A gem of a result on this problem is in [209]: the author proves that the best representation of $\mathbf{A}$ using B terms can be obtained using $O(B^2 \log N)$ *orthogonal* terms. Representing signals using orthogonal wavelet packet vectors is solvable. This result may be improvable by allowing some approximation to the best B term representation.

Problem 11. There are a number of other special dictionaries of interest—beamlets, curvelets, ridgelets, segmentlets—each suitable for classes of applications. Design efficient algorithms to approximate the best representation of a given function using these dictionaries.

Studying general dictionaries. On the other extreme, one may be interested in studying the sparse approximation problem under any general, arbitrary dictionary. A paper that seems to have escaped the attention of approximation theory researchers is [182] which proves this general problem to be NP-Hard. This was reproved in [68].

In what follows, we show a simple inapproximability result. Formally, consider the *general sparse approximation problem* (GSAP). Let $\mathbf{A}$ be the target signal and let M be the matrix in which the columns are the dictionary vectors. As before, the objective is to find the minimum error in approximating $\mathbf{A}$ with at most B vectors from $\mathbf{M}$. The standard satisfiability problem (SAT) is to determine for a given boolean formula in its Conjunctive Norm Form, whether there is an assignment to boolean variables so that the formula evaluates to TRUE (it is a YES instance) or there does not exist such as an assignment (it is a NO instance). This is an NP-Hard problem [97].

Theorem 22. There exists a reduction from SAT to GSAP such that: if SAT is a YES instance, the minimum error in GSAP is zero; if SAT is a NO instance, the minimum error in GSAP is strictly greater than zero.

Proof. Given a collection $S_1, \ldots, S_m$ of sets over some domain $[1, n]$, a *set cover* is a collection S of sets S_{ℓ_i}'s for $\ell_i \in [1, m]$, such that each element $i \in [1, n]$ is contained in some set S_{ℓ_i}. The *exact set cover* is one in which each element $i \in [1, n]$ is contained in *precisely* one of the sets in S. For any given $\eta < 1$, there exists a reduction from SAT to exact set cover, s.t., for a YES instance of SAT, there is an *exact* set cover of size ηd; for a NO instance, there doest not exist *any* set cover of size d. Here, d is a function of the parameters of the SAT instance.

The reduction from SAT to GSAP proceeds via the reduction to set cover above. Given an instance of set cover with elements $[1, n]$ and m sets $S_1, \ldots, S_m$, we define the $n \times m$ matrix $\mathbf{M}$ by setting $M[i][j] = Q$ iff $i \in S_j$, and 0 otherwise. Here Q is some large number. In the target $\mathbf{A}$, all n coordinates are set to Q. Further, set B in the GSAP problem to be the d in the reduction from SAT to the set cover.

If the SAT instance is YES, there is an exact cover of size ηd. The vector $\mathbf{y}$, where $\mathbf{y}[i] = 1$ if and only if S_j is in the set cover and is 0 otherwise, satisfies $|\mathbf{My} - \mathbf{A}| = \mathbf{0}$, giving zero error to the GSAP problem with at most ηd terms. If the SAT is NO, consider any solution to the GSAP problem comprising columns $i_1, i_2, \ldots, i_d$ to represent $\mathbf{A}$ with coefficients $\alpha_1, \alpha_2, \ldots, \alpha_d$, respectively. Define vector $\mathbf{y}$ such that $\mathbf{y}[i_k] = \alpha_k$ for all $k = 1, \ldots, d$, with the rest of the coordinates being zero. Define set S to be collection of sets S_j, s.t. $\mathbf{y}[j] \neq 0$. The cardinality of S is no larger than d. Since in a negative instance there does not exist any set cover of size d, it implies that there exist an element i not covered by S. Then $\mathbf{My}[i] = 0$ and as a result, $\mathbf{My}[i] - \mathbf{A}[i] = -Q$. The minimum error then is greater than zero. ∎

The theorem above immediately gives a number of hardness results. First, it shows that the GSAP problem is NP-hard since there is a polynomial reduction from SAT to the set cover and our reduction above is also in polynomial time. This gives an alternative proof to [182, 68]. Second, it shows that any B vector solution to GSAP will not be able to approximate the error to any multiplicative term since it is NP-hard to even distinguish the zero error case. Further, since there is a polynomial time algorithm to reduce SAT to the set cover with any constant η, the theorem shows that it is NP-hard to design an algorithm for GSAP that produces the minimum error even if allowed to pick cB vectors, for constant $c \geq 1$. Further, using the results in [158, 82], it will follow that unless NP $\in$ DTIME($N^{O(\log \log N)}$), there does not exist a polynomial time algorithm for GSAP that produces a reasonable approximation to the optimal error even if allowed to pick $\Omega(B \log N)$ vectors. Thus, if one considers (α, β) approximations to the GSAP where one uses αB vectors and tries to get an error at most β times the optimal, we can only hope for a narrow set of feasible values for α and β with polynomial algorithms. As a side remark, the hardness above holds for other error norms besides the L_2-norm, including L_1 and L_∞ norms.

On the positive side, [182] has the following very nice result. Say to obtain a representation with error ϵ one needs $B(\epsilon)$ terms. Let D be the $N \times |\mathcal{D}|$ matrix obtained by having ϕ_i as the ith column for each i. Let D^+ be the pseudoinverse of D. (The pseudoinverse is a generalization of the inverse and exists for any (m, n) matrix. If $m > n$ and A has full rank n, then $A^+ = (A^T A)^{-1} A^T$.) The author in [182] presents a greedy algorithm that finds a representation with error no more than ϵ but using

$$O(B(\epsilon/2)||D^+||_2^2 \log(||\mathbf{A}||_2))$$

terms. [182] deserves to be revived: many open problems remain.

Problem 12. Improve [182] to use fewer terms, perhaps by relaxing the error achieved. Is there a nontrivial non-greedy algorithm for this problem?

Problem 13. A technical problem is as follows: The algorithm in [182] takes $|\mathcal{D}|$ time for each step of the greedy algorithm. Using dictionary preprocessing, design a faster algorithm for finding an approximate representation for a given signal using the greedy algorithm. Instead of finding the "best", the approach may be to find "near-best" in each greedy step and prove that the overall approximation does not degrade significantly.

Both these questions have been addressed for a fairly general (but not fully general) dictionaries. Dictionary $\mathcal{D}$ has *coherence* μ if $\|\phi_i\| = 1$ for all i and for all distinct i and j, $|\langle \phi_i, \phi_j \rangle| \leq \mu$. For orthogonal dictionaries, $\mu = 0$. Thus coherence is a generalization. Nearly exponentially sized dictionaries can be generated with small coherence. For dictionaries with small coherence, good approximation algorithms have been shown:

Theorem 23. [107, 108] Fix a dictionary $\mathcal{D}$ with coherence μ. Let $\mathbf{A}$ be a signal and suppose it has a B-term representation over $\mathcal{D}$ with error $\|\mathbf{A} - \mathbf{R}_{\text{opt}}\|$, where $B < 1/(2\mu)$. Then, in iterations polynomial in B, we can find a B-term representation $\mathbf{R}$ such that

$$\|\mathbf{A} - \mathbf{R}\| \leq \sqrt{1 + \frac{2\mu B^2}{(1 - 2\mu B)^2}} \, \|\mathbf{A} - \mathbf{R}_{\text{opt}}\| .$$

The concept of coherence has been generalized recently in [205].

Further in [107, 108], authors used approximate nearest neighbor algorithms to implement the iterations in Theorem 23 efficiently, and proved that approximate implementation of the iterations does not degrade the error estimates significantly. This is a powerful framework, and efficient algorithms for other problems in Functional Approximation Theory may use this framework in the future.

Different Norms. An interesting direction is nonuniform sparse approximation. Formally, there is an *importance function* π where $\pi[i]$ is the importance of i and it is assumed to be nonnegative, all normalized to 1. The goal is to minimize

$$\varepsilon^{\pi}_{\mathbf{R}} = \sum_i \pi[i]\ (\mathbf{R}[i] - \mathbf{A}[i])^2.$$

The problem of *nonuniform sparse approximation* is as follows:

Problem 14. Given a signal $\mathbf{A} \in \mathcal{R}^N$, importance function π, and a dictionary $\mathcal{D} = \{\phi_i\}_{i \in I}$ of unit vectors that span $\mathcal{R}^N$, find a representation $\mathbf{R}$ of B terms for $\mathbf{A}$ given by $\mathbf{R} = \sum_{i \in \Lambda} \alpha_i \phi_i$, for $\phi_i \in \mathcal{D}$ and some Λ, $|\Lambda| \leq B$, $B \ll N$, such that $\varepsilon^{\pi}_{\mathbf{R}}$ is minimized.

Nonuniform sparse approximation is well motivated in many applications where sparse approximation finds uses. As an example, consider how Haar wavelet synopses get used in databases and data analysis. A signal $\mathbf{A}$ is represented using a B-term wavelet synopsis. Then its representation $\mathbf{R}$ is used as the (succinct) approximation or the surrogate of $\mathbf{A}$. When the signal is queried using point queries, i.e., queries are of the form "$\mathbf{A}[i] =$?", the approximation $\mathbf{R}[i]$ is returned instead. *If all point queries are equally likely*, then average of the squared error of using $\mathbf{R}[i]$ rather than $\mathbf{A}[i]$ is precisely $\varepsilon_{\mathbf{R}}$, thus, the rationale to minimize $\varepsilon_{\mathbf{R}}$. However, in practice, all point queries are *not* equally likely. In general, there is bias in how points are queried that depends on the application and the nature of the signal. So, there is a workload π where $\pi[i]$ is the *probability* of i being queried. In the general workload case then, one wants to minimize, analogous to the uniform workload case, the total squared-error over all i's, but now normalized for the probability of a point i being queried, or equivalently the probability of incurring the error on i. That gives the nonuniform sparse approximation problem above. Study of nonuniform sparse approximation problems is discussed in [174] where additional motivations are given.

Curiously, no prior algorithmic result is known for these problems. In particular, even for orthonormal basis such as Haar or Fourier, Parseval's theorem no longer applies. Hence new algorithms are needed. This is a rich new area in sparse approximations. Typical heuristics for solving these problems involve modifying the dictionary so it is orthonormal with respect to the error [165], but that does not necessarily provide an optimal solution.

Recently some algorithmic results have been obtained under the assumption that $\alpha_i = \langle \mathbf{A}, \psi_i \rangle$ for $i \in \Lambda$, where ψ's are the Haar wavelet vectors [96]. Under this restriction, a dynamic programming approach gives a quadratic time algorithm. Also, recently, non-euclidean norms including the L_∞-norm for error have been studied. [116] ∎

Sparse approximation theory has in general focused on the class of functions for which the error has a certain decay as $N \to \infty$. See [69] and [203] for many such problems. But from an algorithmicists point of view, the nature of problems discussed above are more clearly more appealing. This is a wonderful area for new algorithmic research; a starting recipe is to study [69] and [203], formulate algorithmic problems, and to solve them. Recently there has been some exciting developments in "compressed sensing" where a small (polylog) number of measurements suffice to reconstruct a sparse signal [71]. See [218] for a webpage of a slew of results of this genre that have emerged. From a Computer Science point of view, this seems to be a collection of results that rely on group testing by adaptive or nonadaptive means. More work needs to be done to relate simple algorithmic principles such as group testing to the mathematical procedures in these papers.

Here are two further, unconventional directions: Can one design new wavelets based on rich *two* dimensional tiling such as hierarchical, $p \times p$ or others? Current wavelet definitions rely on rather restricted set of two dimensional tiling such as quadtrees, but the space of two dimensional tilings are much richer. There

are complexity results known for computing such two dimensional tilings and some can be approximated nicely in near-linear time [175]. Also, can one design new one dimensional wavelets based on the $2-3$ tree decomposition based on LCP parsing described here earlier? In both cases, this gives vectors in the dictionary defined over intervals that are not just dyadic as in Haar wavelets. Exploring the directions means finding if there are classes of functions that are represented compactly using these dictionaries, and how to efficiently find such representations.

0.8.3 Data Structures

Many of us have heard of the puzzle that leaves Paul at some position in a singly linked list, and he needs to determine if the list has a *loop*. He can only use $O(1)$ pointers. The puzzle is a small space exploration of a list, and has been generalized to arbitrary graphs even [22]. One of the solutions relies on leaving a "finger" behind, doing 2^i step exploration from the finger each i, $i = 1, \ldots$; the finger is advanced after each iteration. This puzzle underlies the discussion in Section 0.2.3 and has the flavor of finger search trees where the goal is to leave certain auxiliary pointers in the data structure so that a search for an item helps the subsequent search. Finger search trees is a rich area of research, e.g., see [39].

Recently, a nice result has appeared [23]. The authors construct $O(\log n)$ space finger structure for an n node balanced search tree which can be maintained under insertions and deletions; searching for item of rank j after an item of rank i only takes $O(\log |j-i|)$ time. (Modulo some exceptions, most finger search data structures prior to this work needed $\Omega(n))$ bits.) This is a streaming result and hopefully this result will generate more insights into streaming data structures. In particular, two immediate directions are to extend these results to external memory or to geometric data structures such as segment trees, with appropriate formalization.

Here is a specific data structural traversal problem.

Problem 15. We have a graph $G = (V, E)$ and a memory M, initially empty. Vertices have to be explicitly loaded into memory M; at most k vertices may reside in M at any time. We say an edge $(v_i, v_j) \in E$ is *evaluated* when both v_i and v_j are in the memory M at the same time. What is the minimum number of loads needed to evaluate all the edges of the graph G?

For $k = 2$, a caterpillar graph can be loaded optimally easily. For $k = 3$, Fan Chung pointed out that the dual of the graph obtained by looking at triangles of G may have certain structure for it to be loaded optimally. This problem arises in query optimization for tertiary databases [194]. As stated above, the problem only addresses the small space (synopses) aspect of the issue, but ideally, one would like an algorithm for doing the entire evaluation that uses one or few passes over the tape resident data, connecting it with the streaming algorithms.

0.8.4 Computational Geometry

Computational Geometry is a rich area. Problems in computational geometry arise because there are applications—earth observations for example—that naturally generate spatial data streams and spatial queries. Also, they arise implicitly in modeling other real life scenarios. For example, flows in IP networks may be thought of intervals [start time, end time], text documents may be mapped to geometric intervals via tries, etc.

Consider the problem of estimating the diameter of points presented on a data stream. Indyk [133, 134] considers this problem in the cash register model where points in d dimensions arrive over time. His algorithm uses $O(dn^{1/(c^2-1)})$ space and compute time per item and produces c-approximation to the diameter,

for $c > \sqrt{2}$. The algorithm is natural. Choose ℓ random vectors $v_1, \ldots, v_\ell$ and for each v_i, maintain the two points with largest and smallest projection along v_i over all point p's. For sufficiently large ℓ, computing diameter amongst these points will give a c-approximation. For $d = 2$, a better algorithm is known. Take any point in the stream as the center and draw sectors centered on that point of appropriate angular width. Within each sector, we can keep the farthest point from the center. Then diameter can be estimated from the arcs given by these points. One gets an ϵ-approximation to the diameter with $O(1/\varepsilon)$ space and $O(\log(1/\varepsilon))$ compute time per inserted point [86]. In an improvement, [127] presents an adaptive sampling approach that gives the same approximation with space only $O(\varepsilon^{-1/2})$. All these results work in the cash register model.

Since the appearance of these early results, a number of new streaming algorithms have been developed in the cash register model including convex hulls [127], approximate range counting in d dimensions over rectangle and halfplane ranges [200], ε-nets and its applications to various geometric depth estimates [14], etc. More generally, a number of results have been designed based on various core-sets. See [5] for an excellent survey of these results. In small dimensional applications like $d = 2$ or 3, keeping certain *radial histograms*, i.e., histograms that emanate in sectors from centers and use bucketing within sectors, seems to find many applications in practice [52].

In an earlier version of this survey, it was mentioned that there were only a few nontrivial results for the computational geometry problems in the Turnstile model. To understand the challenge, consider points *on a line* being inserted and deleted, all insertions and deletions coming only at the right end (the minimum point is fixed). Maintaining the diameter reduces to maintaining the maximum value of the points which is impossible with $o(n)$ space when points may be arbitrarily scattered. Instead, if the points are in the range $1 \cdots R$: then, using $O(\log R)$ space, we can approximate the maximum to $1 + \epsilon$ factor. This may be an approach we want to adopt in general, i.e., have a bounding box around the objects and using resources polylog in the area of the bounding box (or in terms of the ratio of min to the max projections of points along suitable set of lines).

This line of reasoning has led to new geometric results on Turnstile data streams for minimum spanning tree weight estimation, bichromatic matching, facility location and k-median clustering [135, 93, 92]. Many open problems remain. Some are discussed in [136]. Also, in some of the geometric problems, making multiple passes gives additional power [30, 127].

0.8.5 Graph Theory

The graph connectivity problem plays an important role in log space complexity. See [189] for history and new development. However, graph problems have not been thoroughly studied in the data stream model where (poly)log space requirement comes with other constraints such as the number of passes.

In [19], authors studied the problem of counting the number of triangles in the cash register model. Graph $G = (V, E)$ is presented as a series of edges $(u, v) \in E$ in no particular order. The problem is to estimate the number of triples (u, v, w) with an edge between each pair of vertices. Let $T_i, = 0, 1, 2, 3$, be the number of triples with i total edges between the pairs of vertices. Consider the signal $\mathbf{A}$ over the triples (u, v, w), $u, v, w \in V$, where $\mathbf{A}[(u, v, w)]$ is the number of edges in the triple (u, v, w). Let $F_i = \sum_{(u,v,w)}(\mathbf{A}[(u, v, w)])^i$. It is simple to observe that

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \end{pmatrix} . \begin{pmatrix} T_1 \\ T_2 \\ T_3 \end{pmatrix}$$

Solving, $T_3 = F_0 - 1.5F_1 + 0.5F_2$. Now, F_1 can be computed precisely. But F_0 and F_2 can only be approximated. This needs a trick of considering the domain of the signal in a specific order so that each

item in the data stream, i.e., an edge, entails updating a constant number of *intervals* in the signal. Using appropriate rangesum variables, this can be done efficiently, so we find a use for the rangesum variables from Section 0.6.1. As a result T_3 can be approximated. In fact, this method works in the Turnstile model as well even though the authors in [19] did not explicitly study that model.

The general problem that is interesting is to count other subgraphs, say constant sized ones. Certain small subgraphs appear in web graphs intriguingly [187], and estimating their number may provide insights into the web graph structure. Web crawlers spew nodes of the web graph in data streams. So, it is a nicely motivated application.

In an interesting paper [25], authors study a data stream graph problem where they consider *indegrees* of nodes, compute various statistics and use that to estimate the density of minors such as small bipartite cliques. When one crawls webpages, the edges leading out of each page can all be seen at once; however, edges directed into a page is distributed over the entire stream. Hence, focusing on indegrees makes it a challenging data stream problem. [25] shows results of using data stream algorithm on the graph obtained from Wikipedia which is an interesting new data source.

Another class of problems that have been studied are on multigraphs. In IP traffic analysis, each node (IP address) represents a communicant, and a (directed or undirected) edge—a packet (s_p, d_p, b_p) with source s_p, destination d_p and number of bytes b_p—connects two communicants s_p and d_p. A stream of packets can therefore be viewed as a series of edges, and this depicts a multigraph since the same pair of communicants may appear many times distributed over the edge stream. Properties of this multigraph are important for network managers: for example, nodes of high degree are those that communicate with a large number of other participants in the network; identifying such nodes (called *superspreaders* [207]) in an IP network can indicate unusual activity on that host, such as port scans or virus activity, even though the overall traffic volume for that host may still be low since each scan requires only a few packets. So, one has to study degrees and moments based on the number of different communicants connected to a node rather than the total volume of communications between hosts. In particular, the frequency moments of interest are

$$M_k = \sum_j (|\{d_k | s_k = j\}|)^k$$

(in contrast to $F_k = \sum_j (\#k \mid s_k = j)^k$). In other words, *degree* of node j is defined to be $|\{d_k | s_k = j\}|$ and not $(\#k \mid s_k = j)$.

Streaming algorithms have been designed for basic problems on multigraph streams such as estimating large degrees, estimating M_k's, and summarizing the degree distribution by quantiles [57]. The idea is to embed one type of stream data structure such as a sample within a different type of data structure such as a sketch. In particular, for the problem of estimating individual degrees (using which solutions can be derived for the other problems), the solution involves the CM Sketch data structure where instead of a count, we maintain a distinct count estimator. This is shown in Figure 6.

Many of the known bounds for multigraph problems are not tight and need to be improved. Further, M_2 estimation can be thought of as a *cascaded* computation $F_2(F_0)$ where F_0 is applied in k's and F_2 is applied on the resulting sums. Of interest are arbitrary cascaded computations $P(Q)$ for different norms P and Q; several open problems remain in understanding the full complexity of $P(Q)$ cascaded computations.

Graph problems need to be explored more extensively in data stream models. But they appear to be difficult in general. Even connectivity is impossible in space constraints of streaming models; there is a fairly general formulation of this difficulty in [88]. One has to find novel motivations and nifty variations of the basic graph problems. Here is a potential direction.

Problem 16. Consider the *semi-streaming* model, i.e., one in which we have space to store vertices, say

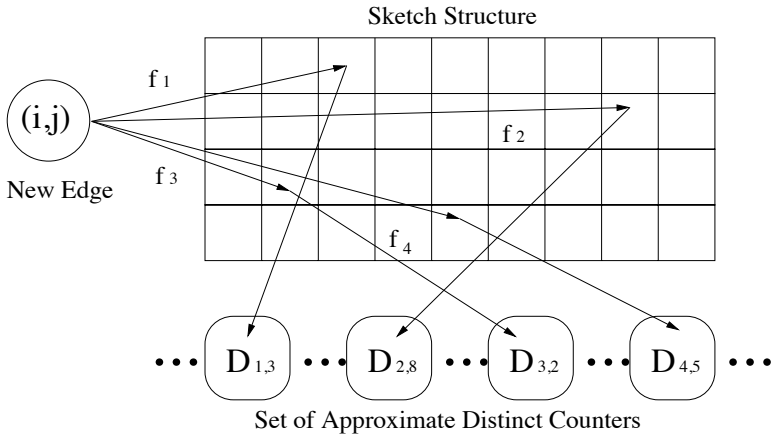

Fig. 6 Sketch Structure for Multigraph Degree Estimation

$O(|V|polylog(|V|)$ bits, but not enough to store all the edges. So we have $o(|E|)$ bits. Solve interesting (in particular, dense) graph problems in this model.

In the past one year, many nice algorithmic results have been shown in the semi-streaming model. This includes approximations to matching problems [87], spanners, the diameter and the shortest path problems [87, 88, 78]. While many problems still remain open, these papers have developed some nice ideas for dealing with graph problems. These papers also study tradeoffs between the number of passes and the space complexity of each pass. Here is an example of a problem of interest on massive graphs.

Problem 17. Formulate a notion of a *community* in a graph where edges arrive with timestamps. Design streaming algorithms to detect or count the communities or estimate their statistics. An example of a community is a subgraph that is more densely connected within itself. An intriguing new notion of community is that it is a subset of vertices that are connected in each time slice for a suitable notion of a time slice [159]. Other definitions may be useful too.

0.8.6 Databases

Databases research community has considered streaming extensively, far too much to be summarized here. Here are a few interesting directions.

Set Histograms. In approximate query processing, the following problem is relevant.

Problem 18. Consider a signal $\mathbf{A}$ where $\mathbf{A}[i]$ is a subset of $1, \ldots, U$. Each query is a *range query* $[i..j]$ for which the response is $|\bigcup_{i \leq k \leq j} \mathbf{A}[k]|$. Build a histogram of B buckets that is (near-)optimal for this task in the data streaming models.

The problem above has to be carefully formalized. Histograms have been studied extensively in Statistics and find many applications in commercial databases. In general they study signals where $\mathbf{A}[i]$ is the number of tuples in a database with value i. Instead, if we interpret $\mathbf{A}[i]$ as the set of memory pages that contain records with value i, histogram described in the problem above is relevant. In database parlance, this is an attempt at modeling the *page selectivity* of queries.

Multidimensional Histograms. In databases, signals are often multidimensional and the following problem is relevant.

Problem 19. Consider two dimensional signal $\mathbf{A}_t[i][j], i, j \in [1, N]$. Design practical algorithms for building (near) optimal two dimensional histogram with B partitions in data stream models.

Readers should not think of this as a straightforward generalization of one dimensional problem to multiple dimensions. There are many ways to partition two dimensional arrays. While the one dimensional problem is polynomial time solvable, two dimensional problems are NP-hard for most partitions. Further, as argued earlier, even if one-dimensional domains are small enough to fit in to given memory, streaming algorithms may well be appropriate for the multidimensional version.

In [175], authors proposed efficient approximation algorithms for a variety of two dimensional histograms for the non-streaming case. In the Turnstile case, [204] proposed a polylog space, $1+\varepsilon$ approximation algorithm with $O(B \log N)$ buckets, taking $\Omega(N^2)$ time. Using the ideas in [175] and robust histograms from [105], a truly streaming algorithm was presented in [179] i.e., one that is a B bucket, $1 + \varepsilon$ approximation using *both* polylog space as well as polylog time. Still, the algorithms are quite complicated; simple and practical algorithms are of great interest for dimensions $d \geq 2$.

Rangesum Histograms. Say a query is a pair of points (ℓ, r) and the ideal answer is $\mathbf{A}[\ell, r) = \sum_{\ell \leq i < r} \mathbf{A}[i]$. An answer from a standard histogram $\mathbf{H}$ would be $\mathbf{H}[\ell, r) = \sum_{\ell \leq i < r} \mathbf{H}[i]$. The sum square error of a histogram is

$$\sum_{\ell, r} |\mathbf{A}[\ell, r) - \mathbf{H}[\ell, r)|^2 .$$

A *rangesum histogram* minimizes the error above. Rangesum histograms are harder to optimally construct since even given bucket boundaries, it is challenging to find heights for a bucket without considering other buckets. This dependence defeats standard dynamic programming approaches.

Transform $\mathbf{A}$ to the prefix array, $\mathbf{P}(\mathbf{A})$. A piecewise-constant representation for $\mathbf{A}$ becomes a piecewise linear *connected* representation for $\mathbf{P}(\mathbf{A})$ since the latter is an integral of the former. While the piecewise constant function may have "jumps" between successive constant pieces, the linear pieces join without gaps in the piecewise linear connected representation since it is the area sum underneath the piecewise constant representation. Also, the square error summed over all range queries (ℓ, r) for $\mathbf{A}$ transforms to N times the sum square error over all point queries in $\mathbf{P}(\mathbf{A})$. As a result, to get a piecewise-constant representation that minimizes the sum square error of range queries to $\mathbf{A}$, we only need to find a piecewise linear connected representation $\mathbf{H}$ that minimizes the sum square error of point queries to $\mathbf{P}(\mathbf{A})$, then output $\Delta(\mathbf{H})$, where $\Delta(\mathbf{H})[i] = \mathbf{H}[i+1] - \mathbf{H}[i]$. Unfortunately, we do not know how to find optimal piecewise linear *connected* representations. But we do know how to find nearly optimal representations from the larger class general piecewise-linear B-bucket representations, not necessarily connected, by generalizing the machinery of robust histograms described earlier. Since $\mathbf{H}$ is not connected, $\Delta(\mathbf{H})$ obtained this way is not necessarily a B-bucket piecewise-constant representation, but it is a $(2B-1)$-bucket piecewise-constant representation, where the discontinuities in $\mathbf{H}$ each give rise to a bucket of size one in $\Delta(\mathbf{H})$. Using this approach $(2, 1+\varepsilon)$ approximation to rangesum histograms were obtained in streaming models [178]. (An (α, β) approximation using α times more buckets and β times more error.) A different algorithm gives $(1, 1+\varepsilon)$ approximation in the non-streaming case.

Problem 20. Is there a polynomial time algorithm for constructing optimal rangesum histograms?

Problem 21. Devise a streaming algorithm that finds B-term Haar wavelet representation $\mathbf{R}$ for the $\mathbf{A}$ such that $\sum_{\ell,r} |\mathbf{A}[\ell, r) - \mathbf{R}[\ell, r)|^2$ is minimized where $\mathbf{R}[\ell, r]$ is the estimation for rangesum query $[\ell, r)$ using $\mathbf{R}$.

Despite the progress in [178], several fundamental questions such as those above remain open about rangesum histograms (and wavelets). ∎

The three different histograms discussed above led to technical questions. From a database point of view, there are many conceptual questions to be resolved: how to scale continuous queries, how to develop a notion of stream operator that is composable so complex stream queries can be expressed and managed, etc. Here is a direction that is likely to be of interest.

Problem 22. A *Stream In, Stream Out* (SISO) query is one in which both the inputs as well as the outputs are streams—in contrast, in much of the problems studies so far, the output is an aggregate value or a (multi)set. An example of a SISO query is what is known as the "join" (related to the cartesian product) of two streams in local time windows. What is an approximation for a *Stream In, Stream Out* (SISO) query? Can one develop a theory of rate of input stream and rate of output stream for various SISO queries? Both probabilistic and adversarial rate theories are of relevance.

0.8.7 Hardware

An important question in data streaming is how to deal with the rate of updates. Ultimately, the rate of updates may be so high that it is not feasible to capture them on storage media, or to process them in software. Hardware solutions may be needed where updates, as they are generated, are fed to hardware units for per-item processing. This has been explored in the networking context for a variety of per-packet processing tasks (see e.g. [196]) previously, but more needs to be done. Consider:

Problem 23. Develop hardware implementation of the projection and sampling based algorithms described in Section 0.6 for various data stream analyses.

Problem 24. There are certain tasks such as IP packet lookup and packet classification that are optimized on hardware and high speed memory within a router. Design algorithms for traffic analyses that use these primitives. In particular, the heavy hitters and quantile estimation problems seem suitable candidates. See [215] for some recent work.

Here is a related topic. The trend in the graphics hardware is to provide a programmable pipeline. Thus, graphics hardware that will be found in computing systems may be thought of as implementing a stream processing programming model. Tasks will be accomplished in multiple passes through a highly parallel stream processing machine with certain limitations on what instructions can be performed on each item at each pixel in each pass. See [130] for an example and [121] for stream-related results. Generic graphics hardware may not be suitable for processing data streams coming in at a high rate, but stream algorithms may find applications in using graphics hardware as a computing platform for solving problems. A lot remains to be explored here; see the thoughtful perspective on the theory and systems of the graphics card as a stream processor by Suresh Venkatsubramanian [208], and the graphics perspective in [164].

0.8.8 Streaming Models

Models make or mar an area of foundational study. We have a thriving set of streaming models already, but some more are likely, and are needed.

0.8.8.1 Permutation Streaming

This is a special case of the cash register model in which items do not repeat. That is, the input stream is a permutation of some set, and items may arrive in a unordered way. (This fits Paul's prediliction of permuting from Section 0.2.1.) There is also some theoretical justification: permutations are special cases of sequences and studying permutation edit distance may well shed light on the hard problems in string processing on streams.

A few problems have been studied in this model. In [59], authors studied how to estimate various permutation edit distances. The problem of estimating the number of inversions in a permutation was studied in [7].

Here is an outline of a simple algorithm to estimate the number of inversions [122]. Let $\mathbf{A}_t$ be the indicator array of the seen items before seeing the tth item, and I_t be the number of inversions so far. Say the tth item is i. Then

$$I_{t+1} = I_t + |\{j \mid j > i \ \& \ \mathbf{A}_t[j] = 1\}|.$$

Estimating $|\{j \mid j > i \ \& \ \mathbf{A}_t[j] = 1\}|$ for any i, up to $1 + \varepsilon$ accuracy can be thought of as the following problem.

Problem 25. Let $\mathbf{PA}$ be the prefix sum version of $\mathbf{A}$ and $0 < \phi < 1$. The *low-biased quantiles* of $\mathbf{A}$ are the set of values $\mathbf{PA}[\lceil \phi^j n \rceil]$ for $j = 1, \ldots, \log_{1/\phi} n$. The *$\varepsilon$-approximate low-biased quantiles* is a set of some k items $q_1, \ldots, q_k$ which satisfy

$$\mathbf{PA}[(1-\varepsilon)\phi^j n] \leq q_j \leq \mathbf{PA}[(1+\varepsilon)\phi^j n]$$

for any $\varepsilon < 1$. The problem is to determine approximate low-biased quantiles in the data stream models.

Directly using known quantile algorithms would mean we would use accuracy $\phi^{\log_{1/\phi} n}$ taking total space $\phi^{-\log_{1/\phi} n} \log(\varepsilon||\mathbf{A}||_1)$ which would be highly space inefficient. The authors in [122] use randomization and an elegant idea of oversampling to retain certain smallest number of them; they obtain an $O(\frac{1}{\varepsilon^3} \log^2(\varepsilon||\mathbf{A}||_1))$ space randomized algorithm for $\varepsilon < \phi$, in the cash register model. An open problem here is what is the best we can do deterministically, or in the Turnstile model.

Here is a problem that arises in practice, in permutation-like streaming.

Problem 26. Each IP flow comprises multiple consecutively numbered packets. We see the packets of the various flows in the Cash Register model. Sometimes packets may get transmitted out of order because of retransmissions in presence of errors, i.e., packets may repeat in the stream. The problem is to estimate the number of flows that have (significant number of) out of order packets at any time. Space used should be smaller than the number of distinct IP flows. This problem may be thought of as estimating the distance of an IP packet stream from the ideal permutation of consecutively numbered packets.

0.8.8.2 Windowed Streaming

It is natural to imagine that the recent past in a data stream is more significant than distant past. How to modify the streaming models to emphasize the data from recent past? There are currently two approaches.

First is *combinatorial*. Here, one specifies a window size w, and explicitly focuses only on the most recent stream of size w, i.e., at time t, only consider a sliding window of updates $a_{t-w+1}, \ldots, a_t$. Items outside this window fall out of consideration for analysis, as the window slides over time. The difficulty of course is that we can not store the entire window, only $o(w)$, or typically only $o(polylog(w))$ bits are

allowed. This model was proposed in [66] and is natural, but it is somewhat synthetic to put a hard bound of w on the window size, for example, irrespective of the rate of arrival of the stream. Still, this model is convenient to work with and a number of results are currently known, e.g. for quantile and heavy hitter estimations [12].

The other model is *telescopic*. Here, one considers the signal as fixed size blocks of size w and λ-ages the signal. Let $\mathbf{A}_i$ represent the signal from block i. We (inductively) maintain β_i as the meta-signal after seeing i blocks. When the $i+1$th block is seen, we obtain

$$\beta_{i+1} = (1 - \lambda_{i+1})\beta_i + \lambda_{i+1}\mathbf{A}_{i+1}.$$

If we unravel the inductive definition, we can see that the signal from a block has influence on the meta-signal in the future that decreases exponentially. This model has certain linear algebraic appeal, and it also leverages the notion of blocks that is inherent in many real life data streams. The original suggestion is in [61] where the block amounted to a days worth of data, and λ_i's were kept mostly fixed. The drawback of this model is clearly that it is difficult to interpret the results in this model in an intuitive manner. For example, if we computed the rangesum of the metasignal $\beta_i[a \cdots b]$, what does the estimate mean for the data stream at any given time?

Another natural model is the *hierarchical block model* described by Divesh Srivastava. Informally, we would like to analyze the signal for the current day at the granularity of a few minutes, the past week at the granularity of hours, the past month at the granularity of days, the past year at the granularity of weeks, etc. That is, there is a natural hierarchy in many of the data streams, and we can study the signals at progressively higher level of aggregation as we look back in to the past. There are very interesting research issues here, in particular, how to allocate a fixed amount of space one has amongst the different signal granularities for samples, sketches or other synopses.

0.8.8.3 The Reset Model

In the *reset* data stream model, we have the signal $\mathbf{A}$ as before. Each update that appears on a stream (i, x) implies a reset, that is, $\mathbf{A}[i] = x$; in contrast, in standard data stream models, each update would have implied $\mathbf{A}[i] = \mathbf{A}[i] + x$. The reset model was introduced in [129].

Motivation for this model arises from the self-tuning approach to selectivity estimation in databases [2]. The query processor has a "summary" that describes some (not necessarily the current) state of the data distribution in the database. It selects query plans for execution using the summary; but when a query is executed, as a by-product, it gets to know the *actual* values the query intersects. The query optimizer needs to use this feedback to update its knowledge of the data distribution. Each time it knows an actual value, it corresponds to a reset. The goal is for the query optimizer to estimate various parameters of interest on the data distribution under this stream of resets. This precisely corresponds to the reset model of data streams.

The reset model is more challenging than the standard data stream models. For example, estimating F_1 is trivial to calculate precisely in standard data stream models. In contrast, it can not be calculated precisely in the reset model. Consider any set S provided as the input to an algorithm in the reset model. We can then reset $\mathbf{A}[1] = 1$ for each i in turn: F_1 will not change after ith reset iff $\mathbf{A}[i] = 1$ prior to the reset. So, we can recover the entire set S which means that the state of the algorithm after seeing S could not have used $o(|S|)$ space. Hence, we seek approximate algorithms when restricted to small space.

Say the update stream is *monotone* if an update always causes $\mathbf{A}[i]$ to increase. If the input is *not* monotone, it provably impossible to estimate F_1 under streaming constraints. Algorithms have been presented in [129] for F_1 estimation, sampling with probability proportional to F_1, and property testing for (non-)monotonicity of a reset stream. While this is a good start, much remains to be done in this model.

0.8.8.4 Distributed Continuous Computation

Consider the following simple game. Paul and Carole see a set of items, denoted P_t and C_t respectively at time t, that continously changes over time t. There is a central monitor M who needs to estimate $m_t = |P_t \cup C_t|$, at all times t. Thus the query "$|P_t \cup C_t| =$?" is a *continuous* query that needs an answer at all times t; in contrast, problems we have studied so far correspond to *one-shot* queries that need an answer at time t when they are posed and not subsequently. The goal is to design a distributed protocol for the continous query above that minimizes the total number of bits communicated until time t, for each t. Paul and Carole do not communicate with each other directly; instead, they communicate with M over private channels.

Obviously if m_t has to be exact during each t, then Paul and Carole have to send their new updates each instant to M and this is inefficient. However, if m_t is allowed to be approximate, then some communication can be eliminated.

This game captures an emerging perspective that streaming systems are being employed in monitoring applications such as tracking events and exceptions in IP traffic networks or sensor networks, hence, they need to support *continuous* queries over distributed networks. A number of heuristic solutions have been proposed recently for set unions such as the game above and other set expressions, quantiles, heavy hitters and sketch-maintenance [161, 95, 45, 44].

However, a solid theory is missing. Formulating the appropriate measure of complexity of these distributed continuous protocols is an interesting challenge—note that standard communication complexity only measures the cost of a distributed one-shot computation and does not apply to the distributed, continuous case. Also, novel distributed continuous protocols are needed for sophisticated monitoring problems. This is a great opportunity to develop an influential theory.

0.8.8.5 Synchronized Streaming

A puzzle, due to Howard Bergerson, is as follows. Imagine the first one thousand vigintillion minus one natural numbers arranged in two lists, one in numerical order and the other in lexicographic order. How many (if any) numbers have their positions same in both lists? There is nothing special about vigintillion, any n will do.

This has a Paul-Carole North American version. Carole counts up $1, \ldots, n$. Paul counts too, but in permuted order given by the lexicographic position of numbers when written in English. For example, if $n = 4$, Carole goes $1, 2, 3, 4$ but Paul goes `Four,One,Three,Two`. Both count in lock step. When, if ever, do they say "Jinx!"?

This puzzle contains the elements of what we call the *synchronized streaming model*. Say we wish to compute a function on two signals $\mathbf{A}_1$ and $\mathbf{A}_2$ given by a data stream. All updates to both the signals are simultaneous and identical except possibly for the update values. That is, if the tth item in the data stream that specifies $\mathbf{A}_1$ is $(i, C_1(i))$, then the tth item in the data stream that specifies $\mathbf{A}_1$ is $(i, C_2(i))$, too. Both these updates are seen one after the other in the data stream. Our goal as before is to compute various functions of interest on $\mathbf{A}_1$ and $\mathbf{A}_2$ satisfying the desiderata of streaming algorithms.

The interest is in if the synchronized model can accomplish more than some of the standard stream models. For example, if the two signals are two strings read left to right in synchronized streaming, one can estimate if their edit distance if at most k, using $O(k)$ space. In contrast, this is difficult to do in a generic streaming model. Recently, improved bounds were presented in synchronized streams model for weighted histogram construction [181]. Synchronized streaming is quite natural; more research is needed on this model.

0.8.9 Data Stream Quality Monitoring.

Any engineer having field experience with data sets will confirm that one of the difficult problems in reality is dealing with poor quality data. Data sets have missing values, repeated values, default values in place of legitimate ones, etc. Researchers study ways to detect such problems (data quality detection) and fixing them (data cleaning). This is a large area of research, see the book [64].

In the traditional view of databases, one sets integrity constraints and any violation is considered a data quality problem and exceptions are raised. Bad quality data (e.g., age of a person being more than 200) is never loaded into the database. This is a suitable paradigm for databases that are manually updated by an operator. Recently, the attention has shifted to data cleaning within a database such as correlating different schemas and finding related attributes.

In the area of data streams, data quality problems are likely to be manifold. For example, in network databases, data quality problems have to do with missing polls, double polls, irregular polls, disparate traffic at two ends of a link due to unsynchronized measurements, out of order values, etc. Now it is unrealistic to set integrity constraints and stop processing a high speed data feed for each such violation; furthermore, appearance of such problems in the feed might by itself indicate an abnormal network phenomena and cleaning it off in the database may hide valuable evidence for detecting such phenomena. Developing algorithms to detect one data quality problem after another after the stream is captured in databases is simply not a scalable or graceful approach.

One needs a principled approach to dealing with data quality problems on streams. In particular, the data quality problems immediately impact one's analyses since the analyses are being done real time. For starters, one needs data quality monitoring tools. They monitor the state of the data stream using probabilistic, approximate constraints and measuring statistics: strong deviation from expected statistics may be projected as a ping for the database administrator or the user to react. To be useful, the tool has to be configured to monitor most suitable statistics and thresholds need to be set to release suitable number of pings while suppressing false alarms. This is an engineering challenge. There are many ways the database and users may deal with these pings: writing their queries in an informed way is a choice for stored database applications. For data streams, one needs a way to automatically rewrite ongoing queries based on data quality problems. There has been some recent progress on data quality monitoring tools [65, 151], but these do not scale to IP traffic speeds yet.

In general, our communities have approached data quality problems as "details" and dealt with individual problems as the need arises. There is a need to develop more principled methods—theory and systems—for dealing with poor quality data.

Here is a specific technical problem not restricted to streams.

Problem 27. Given a set S of strings $s_1, \ldots, s_n$ and set T of strings $t_1, \ldots, t_n$, find a *matching* (i.e., one-to-one mapping) $f : S \to T$ such that $\sum_i d(s_i, f(s_i))$ is (approximately) minimized. Let $d(x, y)$ be the edit distance between strings x and y. This problem can be done by computing $d(s_i, t_j)$ for all pairs i, j and finding min cost matching, but the goal is to get a substantially subquadratic (say near linear) time approximate algorithm. The underlying scenario is S and T are identical lists of names of people, but with some errors; f is our posited (likely) mapping of names of people in one list to the other.

0.8.10 Fish-eye View

Here is a fish-eye view of other areas where streaming problems abound. The discussion will be elliptical: one needs to mull over these discussions to formulate interesting technical open problems.

0.8.10.1 Linear Algebra

Matrix functions need to be approximated in data stream model. For example:

Problem 28. Given a matrix $\mathbf{A}[1 \cdots n, 1 \cdots n]$ in the Turnstile Model (i.e., via updates to $\mathbf{A}$), find an approximation to the best rank-k representation to $\mathbf{A}_t$ at any time t. More precisely, find D^* such that

$$||\mathbf{A}_t - D^*|| \leq f(\min_{D,\, rank(D) \leq k} ||\mathbf{A}_t - D||)$$

using suitable norm $||.||$ and function f.

A result of this type has been proved in [3, 72] using appropriate sampling for a fixed $\mathbf{A}$, and recent progress is in [72] for a similar problem using a few passes, but there are no results in the Turnstile Model. A lot of interesting technical issues lurk behind this problem. One may have to be innovative in seeking appropriate $||.||$ and f. Other linear algebraic functions are similarly of interest such as estimating eigenvalues, determinants, inverses, matrix multiplication and its applications to maxcut, clustering and other graph problems; see the tomes [73, 74, 75] for sampling solutions to many of these problems for the case when $\mathbf{A}$ is fully stored.

0.8.10.2 Statistics

Prior work in data streams has shown how to estimate simple statistical parameters on data streams. We need vastly more sophisticated statistical analyses in data stream models. In statistics, researchers refer to "recursive computing" which resonates with the cash register model of computation. There is an inspiring article by Donoho [70] which is a treasure trove of statistical analyses of interest with massive data. Here is a general task:

Problem 29. Assume a model for the signal $\mathbf{A}$ and estimate its parameters using one of well known methods such as regression fitting or maximum likelihood estimation or expectation maximization and Bayesian approach on the data stream models.

As an example, consider the $\mathbf{A}$ where $\mathbf{A}[i]$ is the number of bytes sent by IP address i. This signal was observed to have self-similarity [147]. For each $p = 0, 1, \ldots, \log N - 1$, define

$$N(p) = \sum_j (\sum_{k=0}^{k=2^p-1} \mathbf{A}[j2^p + k])^2.$$

In principle, the *fractal dimension* of the signal over an infinite domain is defined as

$$D_2 \equiv \lim_{p \to \infty} \frac{\log N(p)}{p} \tag{1}$$

In practice, for finite domains, a plot of $\log N(p)$ vs p is used: if the signal is self-similar, the plot will be a straight line and its slope will be D_2. In general, estimating the slope of the best line fit by the least squares method gives an estimate for D_2.

Let us consider the problem of estimating D_2 on a data stream. A suitable algorithm is to (a) use one of the sketches to estimate $N(p)$ accurately to $1 \pm \varepsilon$ by $\hat{N}(p)$ for each p with probability at least $1 - \delta$, (b) use the least squares method with pairs $(p, \hat{N}(p))$ for all p and (c) get an estimate $\hat{D}_2$. This has been used previously as a heuristic [212]. In fact, this algorithm provably approximates the fractal dimension to desired accuracy as shown below.

Theorem 24. [150] With probability at least $1 - \delta \log N$,

$$|\hat{D_2} - D_2| \leq \frac{3\varepsilon}{(1 + \log N)(1 - \varepsilon)}.$$

Hence, fixing δ and ε appropriately, one gets a highly precise estimate for D_2. Similar approach will work for other statistical parameter estimators that rely on least squares for straight line fit. It will be of interest to design such algorithms with provable accuracy bounds for estimating more sophisticated statistical parameters on the data stream model. Some preliminary work on Bayesian methods for massive data sets appears in [15].

0.8.10.3 Complexity Theory

Complexity theory has already had a profound impact on streaming. Space-bounded pseudorandom generators—developed chiefly in the complexity theory community—play an important role in streaming algorithms. No doubt more of the tools developed for small space computations will find applications in data streaming. Lower bounds in read-once branching programs [188] for example may have applications.

In a recent talk with David Karger, the question arose whether quantum phenomenon can compress computations into much smaller space than conventional computations, i.e., whether quantum memory is more plentiful than conventional memory.

A question likely to have been in researchers' minds is the following: D. Sivakumar has some notes.

Problem 30. Characterize the complexity class given by a deterministic logspace verifier with one-way access to the proof.

0.8.10.4 Privacy Preserving Data Mining

Peter Winkler gives an interesting talk on the result in [81] which is a delightful read. Paul and Carole each have a secret name in mind, and the problem is for them to determine if their secrets are the same. If not, neither should learn the other's secret. The paper [81] presents many solutions, and attempts to formalize the setting. (In particular, there are solutions that involve both Paul and Carole permuting the domain, and those that involve small space pseudorandom generators.) Yao's "two millionaires" problem [214] is related in which Paul and Carole each have a secret number and the problem is to determine whose secret is larger without revealing their secrets.

These problems show the challenge in the emerging area of privacy preserving data mining. We have multiple databases (sets or multisets). Owners of these databases are willing to cooperate on a particular data mining task such as determining if they have a common secret, say for security purposes or because it is mandated. However, they are not willing to divulge their database contents in the process. This may be due to regulatory or proprietary reasons. They need privacy-preserving methods for data mining.

This is by now a well researched topic with positive results in very general settings [99]. However, these protocols have high complexity. There is a demand for efficient solutions, perhaps with provable approximations, in practice. In the past two years, a number of such protocols have been designed and analyzed in theory and database community; see [185] and examples of [6, 94, 139, 199] for a sample of recent developments.

An interesting direction is to formalize approximate notions of privacy and security in distributed settings. In [83] authors formalized the notion of approximate privacy preserving data mining and presented some solutions, using techniques similar to ones used in data stream algorithms. A lot remains to be done. For example, a problem of interest in databases is the following.

Problem 31. Paul has m secrets, and Carole has n secrets. Find an approximate, provably privacy-preserving protocol to find the common secrets. As before, the unique secrets of Paul or Carole should not be revealed to each other.

Other problems arise in the context of banking, medical databases or credit transactions. This gives new problems, for example, building decision trees and detecting outliers, in a privacy-preserving way. For example:

Problem 32. Paul, Carole and others have a list of banking transactions (deposits, withdrawal, transfers, wires etc.) for each of *their* customers. Say the customers have common IDs across the lists. Design an approximate, provably privacy-preserving protocol to find the "heavy hitters", i.e., customers who executed the largest amount in transactions in the *combined* list of all transactions.

0.9 Historic Notes

Data stream algorithms as an active research agenda emerged only over the past few years, even though the concept of making few passes over the data for performing computations has been around since the early days of the Automata Theory. Making one or few passes for selection and sorting [146] got early attention. There was a gem of a paper by Munro and Paterson [172] in 1980 that specifically focused on multi-pass algorithms; it presented one pass algorithms and multi-pass lower bounds on approximately finding quantiles of a signal. Gibbons and Matias at Bell Labs synthesized the idea of Synopsis Data Structures [207] that specifically embodied the idea of a small space, approximate solution to massive data set problems. The influential paper of Alon, Matias and Szegedy [10] used limited independence for small space simulation of sophisticated, one-pass norm estimation algorithms. The paper by Henzinger, Raghavan and Rajagoplan [125] formulated one (and multiple) pass model of a data stream and presented a complexity-theoretic perspective; this is an insightful paper with several nuggets of observations some of which are yet to be developed. Feigenbaum at AT& T Research identified and articulated the case for the network traffic data management as a data stream application [84]; a detailed, and pivotal case was made for this application by Estan and Varghese [80]. From these beginnings, the area has flourished. This survey has focused on algorithmic foundations. A database perspective is at [13].

0.10 Concluding Remarks

In *How Proust can Change Your Life*, Alain de Botton wrote about "the All-England Summarise Proust Competition hosted by Monty Python ... that required contestants to précis the seven volumes of Proust's work in fifteen seconds or less, and to deliver the results first in a swimsuit and then in evening dress." I have done something similar with data stream algorithms in what amounts to an academic 15 seconds sans a change of clothes.

Data streams are more than the topic de jour in Computer Science. Data sources that are massive, automatic (scientific and atmospheric observations, telecommunication logs, text) data feeds with rapid updates are here to stay. We need the TCS infrastructure to manage and process them. That presents challenges to algorithms, databases, networking, and systems. Ultimately, that translates into new questions and methods in Mathematics: approximation theory, statistics and probability. A new mindset—say, seeking only the strong signals, working with distribution summaries—is needed, and that means a chance to reexamine some of the fundamentals. Fittingly, the seminal work of Alon, Matias and Szegedy [10] on data stream processing has been awarded the Godel Prize recently.

Several tutorials, workshops and special journal issues focused on data streams have occurred already,

and many more academic adventures will happen over time. The data stream agenda now pervades many branches of Computer Science including databases, networking, knowledge discovery and data mining, and hardware systems. Industry is in synch too, with DSMSs and special hardware to deal with data speeds. In a recent National Academy of Sciences meeting [219], data stream concerns emerged from physicists, atmospheric scientists and statisticians, so it is spreading beyond Computer Science. Unlike a lot of other topics we—algorithmicists—focused on, data streams is an already accepted paradigm in many areas of Computer Science and beyond. (As a bonus, data streams may be the vehicle that induces people outside theoretical computer science to accept "approximation" as a necessary strategem and seek principled ways of approximating computations, an acceptance that is far from being universal now.) If we keep the spirit of stream data phenomenon in mind and be imaginative in seeking applications and models, potential for new problems, algorithms and mathematics is considerable.

0.11 Acknowledgement

My sincere thanks to the referees for correcting the mistakes without curbing my style. Also, thanks to the many coauthors and co-researchers who have led the way. Despite all the guidance, it is likely that I introduced some errors. Mea culpa.

References

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal* 12(2), 120-139, 2003. See also: Aurora: A data stream management system. *Proc. ACM SIGMOD* 2003, Demo.

[2] A. Aboulnaga and S. Chaudhuri. Self-tuning Histograms: Building Histograms Without Looking at Data. *Proc. ACM SIGMOD*, 181-192, 1998.

[3] D. Achlioptas and F. McSherry. Fast computation of low rank approximation. *Proc. ACM STOC*, 611-618, 2001.

[4] L. Adamic. Zipf, power-law, pareto - a ranking tutorial. http://www.hpl.hp.com/research/idl/papers/ranking/, 2000.

[5] P. K. Agarwal, S. Har-Peled and K. Varadarajan. Geometric Approximation via Coresets. Survey. Available at http://valis.cs.uiuc.edu/~sariel/papers/04/survey/

[6] G. Aggarwal, N. Mishra and B. Pinkas Secure computation of the K'th-ranked element. *Advances in Cryptology – Eurocrypt*, LNCS 3027, Springer-Verlag, 40-55, May 2004.

[7] M. Ajtai, T. Jayram, S. R. Kumar and D. Sivakumar. Counting inversions in a data stream. *Proc. ACM STOC*, 370–379, 2002.

[8] N. Alon, N. Duffield, C. Lund and M. Thorup. Estimating sums of arbitrary selections with few probes. *Proc. ACM PODS*, 2005.

[9] N. Alon, P. Gibbons, Y. Matias and M. Szegedy. Tracking join and self-join sizes in limited storage. *Proc. ACM PODS*, 10–20, 1999.

[10] N. Alon, Y. Matias and M. Szegedy. The space complexity of approximating the frequency moments. *Proc. ACM STOC*, 20–29, 1996.

[11] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein and J. Widom. STREAM: The stanford stream data manager. *Proc. ACM SIGMOD* 2003, Demo.

[12] A. Arasu and G. Manku. Approximate Counts and Quantiles over Sliding Windows. *Proc. ACM PODS*, 286–296, 2004.

[13] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom. Models and issues in data stream systems. *Proc. ACM PODS*, 1–16, 2002.

[14] A. Bagchi, A. Chaudhary, D. Eppstein and M. Goodrich. Deterministic Sampling and Range Counting in Geometric Data Streams. *Proc. ACM SOCG*, 144–151, 2004.

[15] S. Balakrishnan and D. Madigan A one-pass sequential Monte Carlo method for Bayesian analysis of massive datasets. *Manuscript*, 2004.

[16] M. Balazinska, H. Balakrishnan, M. Stonebraker. Load management and high availability in the Medusa distributed stream processing system. *Proc. ACM SIGMOD*, 929-930, 2004.

[17] Z. Bar-yossef, T. Jayram, R. Kumar and D. Sivakumar. Information statistics approach to data stream and communication complexity. *Proc. IEEE FOCS*, 209–218, 2002.

[18] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar and L. Trevisan. Counting Distinct Elements in a Data Stream. *Proc. RANDOM*, 1-10, 2000.

[19] Z. Bar-Yossef, R. Kumar and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs, *Proc. ACM-SIAM SODA*, 623–632, 2002.

[20] T. Batu, S. Guha and S. Kannan. Inferring mixtures of markov chains. *Proc. COLT*, 186–199, 2004.

[21] K. Beauchamp. *Walsh functions and their applications*. 1975.

[22] M. Bender, A. Fernandez, D. Ron, A. Sahai and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Proc. ACM STOC*, 269–278, 1998.
[23] G. Blelloch, B. Maggs, S. Leung and M. Woo. Space efficient finger search on degree-balanced search trees. *Proc. ACM-SIAM SODA*, 374–383, 2003.
[24] A. Broder, M. Charikar, A. Freize and M. Mitzenmacher. Min-wise independent permutations. *Proc. ACM STOC*, 327–336, 1998.
[25] L. Buriol, D. Donato, S. Leonardi and T. Matzner. Using data stream algorithms for computing properties of large graphs. *Workshop on Massive Geometric Datasets*, With *ACM SoCG* 2005, Pisa.
[26] A. R. Calderbank, A. Gilbert, K. Levchenko, S. Muthukrishnan, and M. Strauss. Improved range-summable random variable construction algorithms. *Proc. ACM-SIAM SODA*, 840–849, 2005.
[27] A. Chakrabarti, S. Khot and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. *IEEE Conference on Computational Complexity*, 107–117, 2003.
[28] J. Chambers, C. Mallows, and B. Stuck. A method for simulating stable random variables. *Journal of the American Statistical Association*, 71(354), 340–344, 1976.
[29] T. Chan. Data Stream Algorithms in Computational Geometry. *Workshop on New Horizons in Computing*, Kyoto, 2005.
[30] T. Chan and E. Chen. Multi-pass geometric algorithms. *Proc. ACM SoCG*, 180–189, 2005.
[31] M. Charikar, C. Chekuri, T. Feder and R. Motwani. Incremental Clustering and Dynamic Information Retrieval. *Proc. ACM STOC*, 626–635, 1997.
[32] M. Charikar, K. Chen and M. Farach-Colton. Finding frequent items in data streams. *Proc. ICALP*, 693–703, 2002.
[33] M. Charikar, L. O'Callaghan and R. Panigrahy. Better streaming algorithms for clustering problems. *Proc. ACM STOC*, 693–703, 2003.
[34] S. Chaudhuri, R. Motwani and V. Narasayya. Random sampling for histogram construction: How much is enough? *Proc. SIGMOD*, 436–447, 1998.
[35] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. *Proc. ACM SIGMOD*, 379–390, 2000.
[36] S. Chen, A. Gaur, S. Muthukrishnan and D. Rosenbluth. Wireless in loco sensor data collection and applications. *Workshop on Mobile Data Access* (MOBEA) II, Held with *WWW Conf*, 2004.
[37] S. Chien, L. Rasmussen and A. Sinclair. Clifford algebras and approximating the permanent. *Proc. ACM STOC*, 222–231, 2002.
[38] E. Cohen and H. Kaplan. Spatially-decaying aggregation over a network: model and algorithms. *Proc. ACM SIGMOD*, 707–718, 2004.
[39] R. Cole. On the dynamic finger conjecture for splay trees, part II. The proof. *Technical report TR1995-701*, Courant Institute, NYU, 1995.
[40] R. Cole and U. Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. *Proc. ACM STOC*, 206–219, 1986.
[41] D. Coppersmith and R. Kumar. An improved data stream algorithm for frequency moments. *Proc. ACM-SIAM SODA* 151–156, 2004.
[42] G. Cormode. Stable distributions for stream computations: it's as easy as 0,1,2. *Workshop on Management and Processing of Massive Data Streams* (MPDS) at FCRC, 2003.
[43] G. Cormode, M. Datar, P. Indyk and S. Muthukrishnan. Comparing data streams using hamming norms (How to zero in). *Proc. VLDB*, 335–345, 2002.
[44] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. *Proc. VLDB*, 13–24, 2005.
[45] G. Cormode, M. Garofalakis, S. Muthukrishnan, R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. *Proc. ACM SIGMOD*, 25–36, 2005.
[46] G. Cormode, T. Johnson, F. Korn, S. Muthukrishnan, O. Spatscheck, and D. Srivastava. Holistic UDAFs at streaming speeds. *Proc. ACM SIGMOD*, 35–46, 2004.
[47] G. Cormode, F. Korn, S. Muthukrishnan and D. Srivastava. Finding hierarchical heavy hitters in data streams. *Proc. VLDB* 464–475, 2003.
[48] G. Cormode, F. Korn, S. Muthukrishnan and D. Srivastava. Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data. *Proc. ACM SIGMOD*, 155–166, 2004.
[49] G. Cormode, M. Paterson, S. Ṣahinalp, and U. Vishkin. Communication complexity of document exchange. *Proc. ACM-SIAM SODA*, 197–206, 2000.
[50] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *Proc. ACM-SIAM SODA*, 667–676, 2002.
[51] G. Cormode and S. Muthukrishnan. What is hot and what is not: Tracking most frequent items dynamically. *Proc. ACM PODS*, 296–306, 2003.
[52] G. Cormode and S. Muthukrishnan. Radial histograms for spatial streams. *DIMACS Technical Report*, 2003-11.
[53] G. Cormode and S. Muthukrishnan. Estimating dominance norms on multiple data streams. *Proc. ESA*, 148–160, 2003.

[54] G. Cormode and S. Muthukrishnan. What is new: Finding significant differences in network data streams. *Proc. INFOCOM*, 2004.

[55] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1), 58–75, April 2005.

[56] G. Cormode and S. Muthukrishnan. Summarizing and Mining Skewed Data Streams. *Proc. SIAM SDM*, 2005.

[57] G. Cormode and S. Muthukrishnan. Space Efficient Mining of Multigraph Streams. *Proc. ACM PODS* 2005.

[58] G. Cormode, S. Muthukrishnan and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. *Proc. VLDB*, 25–36, 2005.

[59] G. Cormode, S. Muthukrishnan and C. Sahinalp. Permutation editing and matching via embeddings. *Proc. ICALP*, 481–492, 2001.

[60] C. Cortes, K. Fisher, D. Pregibon and A. Rogers. Hancock: A language for extracting signatures from data streams. *Proc. KDD*, 9–17, 2000.

[61] C. Cortes and D. Pregibon. Signature-based methods for data streams. *Data Mining and Knowledge Discovery* 5(3), 167–182, 2001.

[62] C. Cortes, D. Pregibon and C. Volinsky. Communities of interest. *Proc. of Intelligent Data Analysis*, 105–114, 2001.

[63] C. Cranor, T. Johnson, V. Shkapenyuk and O. Spatscheck. The Gigascope Stream Database. *IEEE Data Engineering Bulletin*, 26(1), 27–32, 2003. See also: C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk and O. Spatscheck. Gigsacope: High performance network monitoring with an SQL interface. *ACM SIGMOD* Demo 2002.

[64] T. Dasu and T. Johnson. *Exploratory data mining and data quality*. ISBN: 0-471-26851-8. Wiley, May 2003.

[65] T. Dasu, T. Johnson, S. Muthukrishnan and V. Shkapenyuk. Mining database structure or how to build a data quality browser. *Proc. ACM SIGMOD*, 240—251, 2002.

[66] M. Datar, A. Gionis, P. Indyk and R. Motwani. Maintaining stream statistics over sliding windows. *Proc. ACM-SIAM SODA*, 635–644, 2002.

[67] M. Datar and S. Muthukrishnan. Estimating rarity and similarity in window streams. *Proc. ESA*, 323–334, 2002.

[68] G. Davis, S. Mallat, and M. Avellaneda. Greedy adaptive approximation. *Journal of Constructive Approximation*, 13, 57–98, 1997.

[69] R. DeVore and G. Lorentz. *Constructive Approximation*, Springer-Verlag, New York, 1993.

[70] D. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. *Manuscript*, 2000. http://www-stat.stanford.edu/˜donoho/

[71] D. Donoho. Compressed sensing. *Manuscript*, 2004. http://www-stat.stanford.edu/˜donoho/Reports/2004/CompressedSensing091604.pdf

[72] P. Drineas and R. Kannan. Pass efficient algorithms for approximating large matrices. *Proc. ACM-SIAM SODA*, 223–232, 2003.

[73] P. Drineas, R. Kannan and M. Mahoney. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication *Yale Technical Report* YALEU/DCS/TR-1269. 2004. To appear in *SIAM J. Computing*.

[74] P. Drineas, R. Kannan and M. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing low-rank approximations to a matrix. *Yale Technical Report* YALEU/DCS/TR-1270. 2004. To appear in *SIAM J. Computing*.

[75] P. Drineas, R. Kannan and M. Mahoney. Fast Monte Carlo algorithms for matrices III: Computing an efficient approximate decomposition of a matrix. *Yale Technical Report* YALEU/DCS/TR-1271. 2004. To appear in *SIAM J. Computing*.

[76] D. Du and F. Hwang. *Combinatorial Group Testing and Its Applications*, 2nd ed., World Scientific Singapore, 2000.

[77] N. Duffield, C. Lund and M. Thorup. Flow Sampling Under Hard Resource Constraints. *Sigmetrics*, 85–96, 2004.

[78] M. Elkin and J. Zhang. Efficient algorithms for constructing $(1+\varepsilon,\beta)$-spanners in the distributed and streaming models. *Proc. ACM PODC*, 160–168, 2004.

[79] C. Estan, S. Savage and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. *Proc. SIGCOMM*, 137–148, 2003.

[80] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3), 270–313, 2003.

[81] R. Fagin, M. Naor and P. Winkler. Comparing information without leaking it: Simple solutions. *Communications of the ACM*, 39:5, 77–85, 1996.

[82] U. Feige. A Threshold of ln n for Approximating Set Cover. *Journal of ACM*, 45(4), 634–652, 1998.

[83] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss and R. Wright. Secure multiparty computation of approximations. *Proc. ICALP*, 927–938, 2001.

[84] J. Feigenbaum. Massive graphs: algorithms, applications, and open problems. Invited Lecture, *Combinatorial Pattern Matching*, 1999.

[85] J. Feigenbaum, S. Kannan, M. Strauss and M. Viswanathan. An approximate L_1 difference algorithm for massive data streams. *Proc. IEEE FOCS*, 501–511, 1999.

[86] J. Feigenbaum, S. Kannan and J. Ziang. Computing diameter in the streaming and sliding window models. *Algorithmica*, 41(1), 25–41, 2004.

[87] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri and J. Zhang. On Graph Problems in a Semi-Streaming Model *Proc of ICALP*, 531–543, 2004.

[88] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri and J. Zhang. Graph Distances in the Streaming Model: The Value of Space. *Proc. ACM-SIAM SODA*, 745–754, 2005.
[89] P. Flajolet and G. Martin. Probabilistic counting. *Proc. FOCS*, 76–82, 1983.
[90] M. Fischer and S. Salzberg. Finding a majority among N votes: Solution to problem 81-5. *J. Algorithms*, 3, 376–379, 1982.
[91] Lance Fortnow. Blog on 03/2005. http://weblog.fortnow.com/2005/03/finding-duplicates.html
[92] G. Frahling, P. Indyk and C. Sohler. Sampling in dynamic data streams and applications. *Proc. ACM SoCG*, 142–149, 2005.
[93] G. Frahling and C. Sohler Coresets in dynamic geometric data streams. *Proc. ACM STOC*, 209–217, 2005.
[94] M. Freedman, K. Nissim and B. Pinkas. Efficient Private Matching and Set Intersection *Advances in Cryptology – Eurocrypt*, LNCS 3027, Springer-Verlag, 1–19, May 2004.
[95] S. Ganguly, M. Garofalakis and R. Rastogi. Tracking set-expression cardinalities over continuous update streams. *VLDB Journal*, 13(4), 354–369, 2004.
[96] M. Garofalakis and A. Kumar. Deterministic Wavelet Thresholding for Maximum-Error Metrics. *Proc. ACM PODS*, 166-176, 2004.
[97] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
[98] L. Gasieniec and S. Muthukrishnan. Determinsitic algorithms for estimating heavy-hitters on Turnstile data streams. *Manuscript*, 2005.
[99] O. Goldreich. Secure multiparty computation. Book at http://philby.ucsd.edu/cryptolib/BOOKS/oded-sc.html. 1998.
[100] P. Gibbons and Y. Matias. Synopsis data structures. *Proc. ACM-SIAM SODA*, 909–910, 1999.
[101] P. Gibbons and S. Trithapura. Estimating simple functions on the union of data streams. *Proc. ACM SPAA*, 281–291, 2001.
[102] A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss. Surfing wavelets on streams: One pass summaries for approximate aggregate queries. *VLDB Journal*, 79–88, 2001.
[103] A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss. QuickSAND: Quick summary and analysis of network data. DIMACS Technical Report, 2001-43.
[104] A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. *Proc. VLDB*, 454–465, 2002.
[105] A. Gilbert, S. Guha, Y. Kotidis, P. Indyk, S. Muthukrishnan and M. Strauss. Fast, small space algorithm for approximate histogram maintenance. *Proc. ACM STOC*, 389–398, 2002.
[106] A. Gilbert, S. Muthukrishnan and M. Strauss. Improved time bounds for near-optimal sparse Fourier representations. *SPIE Conf*, Wavelets, 2005. See also: A. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan and M. Strauss. Near-optimal sparse fourier estimation via sampling. *Proc. ACM STOC*, 152–161, 2002.
[107] A. Gilbert, S. Muthukrishnan and M. Strauss. Approximation of Functions over Redundant Dictionaries Using Coherence. *Proc. ACM-SIAM SODA*, 243–252, 2003.
[108] A. Gilbert, S. Muthukrishnan, M. Strauss and J. Tropp. mproved sparse approximation over quasi-coherent dictionaries. *Intl Conf on Image processing* (ICIP), 37–40, 2003.
[109] D. Geiger, V. Karamcheti, Z. Kedem and S. Muthukrishnan. Detecting malicious network traffic using inverse distributions of packet contents. *Proc. MineNet*, 2005. Held with *Proc. ACM SIGCOMM*, 2005.
[110] Identifying frequent items in sliding windows over on-line packet streams. L. Golab, D. DeHaan, E. Demaine, A. Lopez-Ortiz, and I. Munro. *Internet Measurement Conference*, 173–178, 2003.
[111] I. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(16), 237–264, 1953.
[112] J. Gray and T. Hey. In search of petabyte databases. http://www.research.microsoft.com/~Gray/talks/
[113] J. Gray, P. Sundaresan, S. Eggert, K. Baclawski and P. Weinberger. Quickly generating billion-record synthetic databases. *Proc. ACM SIGMOD*, 1994, 243–252.
[114] M Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. *Proc. ACM SIGMOD*, 2001.
[115] S. Guha. Space efficiency in synopsis construction algorithms. *Proc. VLDB*, 2005.
[116] S. Guha and B. Harb. Wavelets Synopsis for data Streams: Minimizing non-Euclidean error. *Proc. ACM KDD*, 2005.
[117] S. Guha, P. Indyk, S. Muthukrishnan and M. Strauss. Histogramming data streams with fast per-item processing. *Proc. ICALP*, 681-692, 2002.
[118] S. Guha, N. Koudas and K. Shim. Data streams and histograms. *Proc. ACM STOC*, 471–475, 2001.
[119] S. Guha, N. Koudas and K. Shim. Approximation and streaming algorithms for histogram construction problems. Journal version.
[120] S. Guha, N. Mishra, R. Motwani and L. O'Callaghan. Clustering data streams. *Proc. IEEE FOCS*, 359–366, 2000.
[121] S. Guha, K. Mungala, K. Shankar and S. Venkatasubramanian. Application of the two-sided depth test to CSG rendering. *Proc. I3d, ACM Interactive 3D graphics*, 2003.
[122] A. Gupta and F. Zane. Counting inversions in lists. *Proc. ACM-SIAM SODA*, 253–254, 2003.
[123] M. Hansen. *Slogging*. Keynote plenary talk at *SIAM Conf. Data Mining*, 2005.
[124] A. Haar. "Zur theorie der orthogonalen functionsysteme". *Math Annal.*, Vol 69, 331–371, 1910.
[125] M. Henzinger, P. Raghavan and S. Rajagopalan. Computing on data stream. *Technical Note 1998-011*. Digital systems research center, Palo Alto, May 1998.

[126] J. Hershberger, N. Shrivastava, S. Suri and C. Toth. Space complexity of hierarchical heavy hitters in multi-dimensional data streams. *Proc. ACM PODS*, 2005.

[127] J. Hershberger and S. Suri. Adaptive Sampling for Geometric Problems over Data Streams. *Proc. ACM PODS*, 252–262, 2004.

[128] D. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Comm. ACM*, 18(6), 341–343, 1975.

[129] M. Hoffman, S. Muthukrishnan and R. Raman. Location streams: Models and Algorithms. *DIMACS TR*, 2004-28.

[130] G. Humphreys, M. Houston, Y. Ng, R. Frank, S. Ahern, P. Kirchner and J. Klosowski. Chromium: A stream processing framework for interactive rendering on clusters. *Proc. ACM SIGGRAPH*, 693–702, 2002.

[131] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. *Proc. IEEE FOCS*, 189–197, 2000.

[132] P. Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1), 84–90, 2001.

[133] P. Indyk. Stream-based geometric algorithms. *Proc. ACM/DIMACS Workshop on Management and Processing of Data Streams* (MPDS), 2003.

[134] P. Indyk. Better algorithms for high dimensional proximity problems via asymmetric embeddings. *Proc. ACM-SIAM SODA*, 539–545, 2003.

[135] P. Indyk. Algorithms for dynamic geometric problems over data streams. *Proc. ACM STOC*, 373–380, 2004.

[136] P. Indyk. *Streaming Algorithms for Geometric Problems*. Invited talk at *CCCG'04*.

[137] P. Indyk and D. Woodruff. Tight Lower Bounds for the Distinct Elements Problem. *Proc. IEEE FOCS*, 2003.

[138] P. Indyk and D. Woodruff. Optimal approximations of the frequency moments of data streams. *Proc. ACM STOC*, 202–208, 2005.

[139] G. Jagannathan and R. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. *Proc. ACM KDD*, 2005.

[140] T. Johnson, S. Muthukrishnan and I. Rozenbaum. Sampling algorithms in a stream operator. *Proc. ACM SIGMOD*, 1–12, 2005.

[141] T. Johnson, S. Muthukrishnan, O. Spatscheck and D. Srivastava. Streams, security and scalability. Keynote talk, appears in *Proc.of 19th Annual IFIP Conference on Data and Applications Security*, Lecture Notes in Computer Science 3654, Springer-Verlag, 1–15, 2005.

[142] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet. *ASPLOS-X conference*, 96–107, 2002.

[143] S. Kannan. Open problems in streaming. Ppt slides. Personal communication.

[144] R. Karp, C. Papadimitriou, and S. Shenker. A simple algorithm for finding frequent elements in sets and bags. *ACM Transactions on Database Systems*, 28, 51–55, 2003.

[145] J. Kleinberg. Bursty and hierarchical structure in streams. *Proc. ACM KDD*, 91–101, 2002.

[146] D. Knuth. *The art of computer programming, Volume III: Sorting and searching*. Addison-Wesley, 1973.

[147] E. Kohler, J. Li, V. Paxson and S. Shenker. Observed structure of addresses in IP traffic. *Internet Measurement Workshop*, 253-266, 2002.

[148] F. Korn, J. Gehrke and D. Srivastava. On computing correlated aggregates over continual data streams. *Proc. ACM SIGMOD*, 13–24, 2001.

[149] F. Korn, S. Muthukrishnan and D. Srivastava. Reverse nearest neighbor aggregates over data streams. *Proc. VLDB*, 814–825, 2002.

[150] F. Korn, S. Muthukrishnan and Y. Wu. Model fitting of IP network traffic at streaming speeds. *Manuscript*, 2005.

[151] F. Korn, S. Muthukrishnan and Y.Zhu. Checks and balances: Monitoring data quality in network traffic databases. *Proc. VLDB*, 536-547, 2003.

[152] S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Madden, F. Reiss and M. Shah. TelegraphCQ: An Architectural Status Report. *IEEE Data Engineering Bulletin*, 26 (1), 11–18, 2003.

[153] B. Krishnamurthy, S. Sen, Y. Zhang and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications *Proc. ACM SIGCOMM Internet Measurement Conference*, 234–247, 2003.

[154] R. Kumar and R. Rubinfeld. *Sublinear time algorithms*. Algorithms column in SIGACT News 2003.

[155] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[156] K. Levchenko and Y. Liu. Counting solutions of polynomial equations. *Manuscript*, 2005.

[157] K. Levchenko, R. Paturi and G. Varghese. On the difficulty of scalably detecting network attacks. ACM Conference on Computer and Communications Security, 12–20, 2004.

[158] C. Lund and M. Yannakakis. On the Hardness of Approximating Minimization Problems. *Journal of ACM*, 41, 960–981, 1994.

[159] M. Magdon-Ismail, M. Goldberg, W. Wallace and D. Siebecker. Locating hidden groups in communication networks using hidden Markov models. *Proc. ISI*, 126–137, 2003.

[160] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler and J. Anderson. Wireless sensor networks for habitat monitoring. *Proc. WSNA*, 88–97, 2002.

[161] A. Manjhi, V. Shkapenyuk, K. Dhamdhere and C. Olston. Finding (recently) frequent items in distributed data streams. *Proc. ICDE*, 767–778, 2005.
[162] G. Manku and R. Motwani. Approximate frequency counts over data streams. *Proc. VLDB*, 346–357, 2002.
[163] G. Manku, S. Rajagopalan and B. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. *Proc. ACM SIGMOD*, 251–262, 1999.
[164] D. Manocha. Interactive geometric computations using graphics hardware. Course. *ACM SIGGRAPH*, 2002.
[165] Y. Matias and D. Urieli. Optimal workload-based wavelet synopses. *Proc. ICDT*, 368–382, 2005.
[166] A. Metwally, D. Agrawal and A. El Abbadi. Efficient computation of frequent and top-k elements in data stream. *Proc. ICDT*, 398–412, 2005.
[167] Y. Minsky, A. Trachtenberg and R. Zippel. Set reconciliation with nearly optimal communication complexity. *Technical Report* 2000-1796, Cornell Univ.
[168] N. Mishra, D. Oblinger and L. Pitt. Sublinear time approximate clustering. *Proc. ACM-SIAM SODA*, 439–447, 2001.
[169] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2:143–152, 1982.
[170] R. Motwani, P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
[171] K. Mulmuley *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice Hall, 1993.
[172] I. Munro and M. Paterson. Selection and sorting with limited storage. *Proc. IEEE FOCS*, 253–258, 1978. Also, *Theoretical Computer Science* 12: 315–323, 1980.
[173] S. Muthukrishnan Data streams: Algorithms and Applications. `http://www.cs.rutgers.edu/~muthu/stream-1-1.ps`.
[174] S. Muthukrishnan. Nonuniform sparse approximation with Haar wavelet basis. *DIMACS TR*, 2004-42.
[175] S. Muthukrishnan, V. Poosala and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity and applications. *Proc. ICDT*, 236–256, 1999.
[176] S. Muthukrishnan and S. Ṣahinalp. Approximate nearest neighbors and sequence comparison with block operations. *Proc. STOC*, 416–424, 2000.
[177] S. Muthukrishnan, R. Shah and J. Vitter. Finding deviants on data streams. *Proc. SSDBM*, 41–50, 2004.
[178] S. Muthukrishnan and M. Strauss. Rangesum histograms. *ACM-SIAM SODA*, 233–242, 2003.
[179] S. Muthukrishnan and M. Strauss. Maintenance of multidimensional histograms. *Proc. FSTTCS*, 352–362, 2003.
[180] S. Muthukrishnan and M. Strauss. Approximate histogram and wavelet summaries of streaming data. *DIMACS TR* 2004-52. Survey.
[181] S. Muthukrishnan, M. Strauss, and X. Zheng. Workload-Optimal histograms on streams. *Proc. ESA*, 2005.
[182] B. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Computing*, 25(2), 227–234, 1995.
[183] A. Orlitsky, N. Santhanam, J. Zhang. Always Good Turing: Asymptotically optimal probability estimation. *Proc. IEEE FOCS*, 179–188, 2003.
[184] M. Parseval. http://encyclopedia.thefreedictionary.com/Parseval's+theorem 1799.
[185] B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *SIGKDD Explorations*, the newsletter of the ACM Special Interest Group on Knowledge Discovery and Data Mining, January 2003.
[186] I. Pohl. A minimum storage algorithm for computing the median. *IBM TR* 12713, 1969.
[187] P. Raghavan. Graph structure of the web: A survey. *Proc. LATIN*, 123–125, 2000.
[188] A. Razborov, A. Wigderson and A. Yao. Read-once branching programs, rectangular proofs of the pigeonhole principle and the transversal calculus. *Proc. STOC*, 739–748, 1997.
[189] O. Reingold. Undirected ST-Connectivity in Logspace. *Proc. STOC*, 376–385, 2005.
[190] S. Ṣahinalp and U. Vishkin. Symmetry breaking for suffix tree construction. *Proc of 26th Symposium on Theory of Computing*, 300–309, 1994.
[191] S. Ṣahinalp and U. Vishkin. Data compression using locally consistent parsing. Technical report, University of Maryland Department of Computer Science, 1995.
[192] S. Ṣahinalp and U. Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm. *Proc. IEEE FOCS*, 320–328, 1996.
[193] M. Saks and X. Sun. Space lower bounds for distance approximation in the data stream model. *Proc. ACM STOC*, 360–369, 2002.
[194] S. Sarawagi. Query Processing in Tertiary Memory Databases. *Proc. VLDB*, 585–596, 1995.
[195] R. Seidel, and C. Aragon. Randomized Search Trees. *Algorithmica*, 16, 464–497, 1996.
[196] D. Shah, S. Iyer, B. Prabhakar and N. McKeown. Maintaining statistics counters in router line cards. *IEEE Micro*, 76–81, 2002.
[197] N. Shrivastava, C. Buragohain, D. Agrawal, S. Suri. Medians and beyond: New aggregation techniques for sensor networks. *Proc. ACM SenSys*, 2004.
[198] J. Spencer and P. Winkler. Three thresholds for a liar. *Combinatorics, Probability and Computing*, 1:1, 81–93, 1992.
[199] H. Subramaniam, R. Wright and Z. Yang. Experimental analysis of privacy-preserving statistics computation. *Proc. of the Workshop on Secure Data Management* (held in conjunction with VLDB), Springer LNCS 3178, 2004.
[200] S. Suri, C. Toth and Y. Zhou. Range Counting over Multidimensional Data Streams. *Proc. ACM SoCG*, 160–169, 2004.

[201] M. Szegedy. Near optimality of the priority sampling procedure. *ECCC TR05-001*, 2005.
[202] J. Tarui. Finding duplicates in passes. *Personal Communication* and http://weblog.fortnow.com/2005/03/finding-duplicates.html#comments
[203] V. Temlyakov. The best m-term approximation and greedy algorithms. *Advances in Computational Math.*, 8:249–265, 1998.
[204] N. Thaper, S. Guha, P. Indyk and N. Koudas. Dynamic multidimensional histograms. *Proc. ACM SIGMOD*, 428–439, 2002.
[205] J. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Trans. Inform. Theory*, 50 (10), 2231–2242, 2004.
[206] G. Varghese. Detecting packet patterns at high speeds. Tutorial at *ACM SIGCOMM* 2002.
[207] S. Venkataraman, D. Song, P. Gibbons, and A. Blum. New streaming algorithms for superspreader detection *Network and Distributed Systems Security Symposium*, 2005.
[208] S. Venkatasubramanian. The graphics card as a stream computer. *Proc. ACM/DIMACS Workshop on Management and Processing of Data Streams* (MPDS), 2003. See also: http://www.research.att.com/~suresh/papers/mpds/index.html
[209] L. Villemoes. Best approximation with walsh atoms. *Constructive Approximation*, 133, 329–355, 1997.
[210] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.
[211] J. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2), 209–271, 2001.
[212] A. Wong, L. Wu, P. Gibbons and C. Faloutsos. Fast estimation of fractal dimension and correlation integral on stream data. *Inf. Process. Lett.*, 93(2), 91–97, 2005.
[213] D. Woodruff. Optimal space lower bounds for all frequency moments. *Proc. ACM-SIAM SODA*, 167–175, 2004.
[214] A. Yao. Protocols for secure computations. *Proc. IEEE FOCS*, 160–164, 1982.
[215] Y. Zhang, S. Singh, S. Sen, N. Duffield and C. Lund. Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications. *Proc. of the Internet Measurement Conference* (IMC), 101–114, 2004.
[216] G. Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison Wesley, 1949.
[217] http://www.tpc.org/. Details of transactions testing at http://www.tpc.org/tpcc/detail.asp
[218] http://www2.ece.rice.edu/~duarte/compsense/
[219] http://www7.nationalacademies.org/bms/Massive_Data_Workshop.html.