Data Streams: Algorithms and Applications *

S. Muthukrishnan†

Contents

*This writeup will be periodically updated at my website which can be found by typing "adorisms" in Google search.

†Rutgers Univ., and AT&T Labs—Research, muthu@cs.rutgers.edu, muthu@research.att.com.

1 Introduction

I will discuss the emerging area of algorithms for processing data streams and associated applications, as an applied algorithms research agenda. That has its benefits: we can be inspired by any application to study novel problems, and yet be not discouraged by the confines of a particular one. The discussion will be idiosyncratic. My personal agenda is to be a *scientist*, *mathematician* and *engineer*, all in one. That will be reflected, some times one more than the others. Art, Nature, Homeland Security and other elements will make a cameo. The tagline is IMAGINE, THINK and DO: one imagines possibilities and asks questions, one seeks provable solutions and finally, one builds solutions. This writeup will present a little of each in data streaming. (See Barry Mazur's book for the imaginary and Math [65].) The area has many open problems.

Let me begin with two puzzles.

1.1 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 elements from set $\pi_{-1}[i]$ in increasing order i, one after other. Carole's task is to determine the missing integer. This is trivial to do 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 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 $2 \log n$. On the other hand, Carole needs at least $\log n$ bits in the worst case. (In fact, Carole has an optimal algorithm. Say n is a power of 2 for convenience. 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.) 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 [25]. 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. Joel introduced them to me during my thesis work. I used them in a game in which Paul put coins on weighing pans (panned them!) [6]. In the puzzle above, Paul permutes and Carole cumulates. In a little while, they will play other P/C roles.

Generalizing the puzzle a little further, let π_{-2} be π with two elements missing. Most of the students in my graduate Algorithms class suggested Carole now 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 Carole can use far fewer bits tracking

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

In general, what is the smallest number of bits needed to identify the k missing numbers in π_{-k}? Following the approach above, the problem may be thought of as having *power sums*

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

for $p = 1, \ldots, k$ and solving for x_i's. A different but provably equivalent method uses *elementary symmetric polynomials*. 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 maintain σ_i's for the missing k items in field F_q for prime $q \ge n$ (and $\le 2n$ suffices), as Paul presents the numbers one after the other (the details are omitted). 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 this problem, but randomized algorithms take roughly $O(k^2 \log n)$ bits and time [23]. The power sum method is what colleagues typically propose over dinner. The elementary symmetric polynomial approach comes from [24] where the authors solve the *set reconciliation problem* in the communication complexity model. The *subset* reconciliation problem is related to our puzzle.

Readers may have guessed that they may be a different efficient solution for this puzzle using insights from error correcting codes or combinatorial group testing. Indeed true. We will later reference an $O(k \log k \log n)$ bits and time solution; in contrast, no algorithm can use $o(k \log(n/k))$ bits in the worst case.

It is no coincidence that this puzzle contains elements of data stream algorithms. Generalize it: Paul presents a multiset of elements $1, \cdots, n$ with a single missing integer, i.e., he is allowed to *re*-present integers he showed before; Paul presents 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.

1.2 Puzzle 2: Fishing

Doubtless it will be a more inspiring introduction to data streams if the puzzle was derived from nature. So, 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 $o(|S|)$ 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 $\hat{\rho}[t] = \frac{|\{\ X_i[t] \mid X_i[t]=1\ \}|}{k}$ as an estimator for ρ. Since

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

, we have

$$\Pr(\hat{\rho}[t] \in [\rho[t] - \epsilon, \rho[t] + \epsilon]) = \sum_{i \in [k(\rho[t]-\epsilon), k(\rho[t]+\epsilon)]} \binom{k}{i} (\rho[t])^i (1 - \rho[t])^{k-i}.$$

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.

As an exercise in doing mathematics while fishing, this misses an ingredient: 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 $1..u$ and waiting for it to be caught seems an exercise in, well, fishing. We can make it more realistic by redefining rarity wrt 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 small number of bits. Following [88], 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

$$\Pr_{h \in \mathcal{H}}[h(x) = min\ h(X)] = \frac{1}{|X|}$$

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 $h_i^*(t) = min_{a_j,\ j \leq t}\ h_i(a_j)$ 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 the number of times the item with hash value $h_i^*(t)$ appears 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 $\Pr(C_i(t) = 1)$ 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 Paul's algorithm.

It is ironic that Paul has adapted the cumulating task in the solution above, which is traditionally Carole's shtick. But we are not done. Paul needs to pick h_i's. If Paul resorts to his tendency to permute, i.e., picks a randomly chosen permutation π over $[u] = \{1, \ldots, u\}$, then h_i's will be min-wise hash functions. And he will be done. 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 not fit his suitcase!

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

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

Indyk [18] presents a family—set of polynomials over $GF(u)$ of degree $O(\log(1/\epsilon))$—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. Plugging this 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 of f fishes drawn independently from fish types according to the distribution P. Let $c[t]$ be the number of times t appears in S and $s[k]$ be the number of fish types that appear k times in S. Consider estimating the probability of fish type t being the next catch. Elementary reasoning would indicate that this probability is $c[t]s[c[t]]/f$. 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 fish types is infinite. Hence, there are fish types t that do *not* appear in the sample (i.e., $c[t] = 0$) and they would have probability 0 of being caught next, a conundrum in the elementary reasoning if t is present in the ocean. 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) [35], it is shown that m is estimated by $s[1]/f$, provably with small bias; recall that our rarity γ is closely related to $s[1]/f$. Hence, our result here on estimating rarity in data streams is of independent interest in the context of estimating the missing mass. Those interested in convergence properties of the Good-Turing estimator should see David McAllester's work.)

Once you generalize the fishing—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 sea as needed—there are some real data streaming applications [19].

Honestly, the fishing motif is silly: the total number of fish species in the sea is estimated to be roughly 22000 and anyone can afford an array of as many bits. In the reality of data streams which I will describe next, one is confronted with fishing in a far more numerous domain.

1.3 Lessons

I have noticed that once something is called a puzzle, people look upon the discussion less than seriously. The puzzle in Section 1.1 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 algoritms—deterministic and exact—are uncommon in data stream processing. In contrast, the puzzle in Section 1.2 is solved only up to an approximation using

a randomized algorithm in polylog bits. This—randomized and approximate solutions—is more representative of currently known data stream algorithms. Further, the estimation of γ in Section 1.2 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.

2 Map

Section 3 will describe the data stream phenomenon. I have deliberately avoided specific models here because the phenomenon is real, models are the means and may change over time. Section 4 will present currently popular data stream models, motivating scenarios and other applications for algorithms in these models beyond dealing with data streams.

Section 5 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 that is emerging. This section is right now skimpy, and I will add to it over time. Section 6 discusses applied work on data streams. It is drawn from different systems areas, and I have grouped them into three categories which may be useful for a perspective.

Section 7 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. Finally, in my concluding remarks in Section 8, I will invoke Proust, show you streaming Art, history, and some notes on the future of streaming. The most important part of this writeup is Section 9.

3 Data Stream Phenomenon

The web site *http://www.its.bldrdoc.gov/projects/devglossary/_data_stream.html* defines a data stream to be a "sequence of digitally encoded signals used to represent information in transmission". We will be a little more specific. Data stream to me 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 data eventually gets to where it should go. If computing power is limited or the program has high complexity, it takes long (long longs!) time to get the desired response, but in principle, we would get it. Also, we save almost all the data we need. This simplified picture of TCS requirements is reasonable because need has balanced resources: we have produced the amount of data that technology 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 [118, 113], radar derived meteorological data [119][1], continuous large scale astronomical surveys in optical, infrared and radio wavelengths [117], atmospheric radiation measurements [108] etc. 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. 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.

- *Need to do sophisticated analyses of update streams in near-real time manner.*

  With traditional datafeeds, 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, forecasting, etc. 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, unusual or anomalous activity, etc., monitor complex correlations, track trends, support exploratory analyses and perform complex tasks such as classification, harmonic analysis etc. These are time critical tasks in each of these applications, more so in emerging applications for homeland security, and they need to be done in near-real time to accurately keep pace with the rate of stream updates and accurately reflect rapidly changing trends in the data.

These two factors uniquely challenge the TCS needs. We in Computer Science community have traditionally focused on scaling wrt to size: how to efficiently manipulate large disk-bound data via suitable data structures [15], how to scale to databases of petabytes [106], synthesize massive datasets [7], 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 [115] 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 [120] will soon produce 40TBytes/s which will be reduced by real time hardware into 800Gb/s data stream: is it careful sampling or carefree shedding? Statisticians have the sampling theory: it is now getting applied to IP network streams [12, 3, 13]. *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 [14]. 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

[1]John Bates, the Chief of Remote Sensing Applications Division of USNOAANDCC, gives a nice exposition at http://www7.nationalacademies.org/bms/BatesPowerPoint.ppt and http://www7.nationalacademies.org/bms/AbstractBATES.html.

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' myopic expectations. 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 (not) we 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. To build data stream applications with ease, and proven performance.

What follows is an introduction to the emerging theory of data streams.

Before that, here is a note. The previous few paragraphs presented a case for data stream research. I could have done it

- Using anecdotics.

  Paul, now a network service Provider, has to convince Carole, his Customer, that IP hosts connecting to her website get high quality real time media service. He needs to monitor IP traffic to her web site and understand per-packet performance for each host. Plotting such statistics in real time is a good way to convince Carole.

- Or using numerics.

  A single OC48 link may transmit few hundred GBytes per hour of packet header information, which is more than 200Mbps. It takes an OC3 link to transfer this streaming log and it is a challenge to write it into tapes or process it by the new 3GHz P4 Intel processor at that rate.

Or I could have used a limerick, haiku or a Socratic dialog. But I chose to describe data stream as a phenomenon in words. Sometimes I think words have become less meaningful to us than greek symbols or numerals. Nevertheless, I hope you would use your imagination and intuit the implications of data streaming. Imagine 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: it should jar us, one way or the other.

4 Data Streaming: Formal Aspects

This section will be more formal: define various models for dealing with data streams and present a motivating application to internalize.

4.1 Data Stream Models

Input stream $a_1, a_2, \ldots$ arrives sequentially, item by item, and describes an underlying *signal* $\mathbf{A}$, a one-dimensional function $\mathbf{A} : [1 \ldots N] \rightarrow R$.[2] Models differ on how 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 at an IP link each 5 minutes, or NASDAQ volume of trades each minute, etc.

[2] Input may comprise of multiple streams or multidimensional signals, but we do not consider those variations here.

- *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. 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 [27] with this name.

- *Turnstile Model.*[3] Here a_i's are updates to $\mathbf{A}[j]$'s. Think of $a_i = (j, U_i)$, to mean $\mathbf{A}_i[j] = \mathbf{A}_{i-1}[j] + U_i$ where $\mathbf{A}_i$ is the signal after seeing the ith item in the stream, and U_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 interesting bounds in this model. This model too has appeared before under different guises, but it gets christened here with its name.

  There is a small detail: in some cases, $\mathbf{A}_i[j] \geq 0$ for all i. We refer to this as the *strict* Turnstile model. Intuitively this corresponds to letting people only exit via the turnstile they entered the system in: it is a unrealistic intuition, but it fits many applications. For example, in a database, you can only delete a record you inserted. On the other hand, there are instances when streams may be *non-strict*, that is, $\mathbf{A}_i[j] < 0$ for some i. For example, when one considers a signal over the difference between two cash register streams, one obtains a non-strict Turnstile model. We will avoid making a distinction between the two Turnstile models unless necessary.

The models in decreasing order of generality are as follows: Turnstile, Cash Register, Time Series. (A more conventional description of models appears in [27].) 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. 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 time)[4]

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

Puzzle	Model	Function	Proc. Time	Storage	Compute Time
Section 1.1	cash register	$k = 1$, $\{j \mid \mathbf{A}[j] = 0\}$	$O(\log n)$	$O(\log n)$	$O(1)$
Section 1.2	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 is generally accepted:

[3] I remember the exhilaration we felt when Martin Farach-Colton and I coined the name *Disk Access Machine* model or the DAM model during a drive from New York city to Bell labs. The DAM model is catching on.

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

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 [10], 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.

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 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 potentially 2^{64} now), or may be the number of time intervals where certain observations were made (which increases rapidly over time), or may be the 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.

Second, why do we use the polylog function? Well, log in 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 appeal to some of the readers (it did, to Mikkel Thorup). 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 data stream processing machines.

4.2 A Motivating Scenario

Let me present a popular scenario for data streaming. 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 analysis of traffic and fault data in real time. Consider 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, etc.

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 attributes such as the source and destination IP addresses and the log contains cumulative information about number of bytes and packets sent, start time, end time, protocol type, etc. per flow.

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. You can collect and store SNMP data. (I store 6 months worth of this data of a large ISP in my laptop for the IPSOFACTO tool I will reference later. I could store data up to a year or two without stressing my laptop.) The arguments we presented for data streaming apply to flow and packet logs which are far more voluminous than the SNMP log. A more detailed description and defense of streaming data analysis in IP network traffic data is presented in [26], in particular, in Section 2.

Here are some 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 on ongoing flow?

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 1.1.

4. How many flows comprised one packet only (i.e., rare flows)? Closely related to this is the question: Find TCP/IP SYN packets without matching SYNACK 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 1.2.

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. The IPSOFACTO system supports such similarity queries on the SNMP logs for an operational network provider [57].

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. Those that rely on signal analysis—wavelet, fourier etc.—of the traffic pattern prove effective. We will later describe algorithms for computing wavelet or fourier representation for data streams.

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 are simple slice-and-dice or aggregate or group by queries. This is but a sparse sample of interesting questions. Imagine the setup, and you will discover many more relevant questions. Some of the more complex queries will involve *joins* between multiple data stream sources. For example, how to correlate

Let me formalize one of the examples above in more detail, say example two.

First, 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. (This signal is more general for reasons that will be clear in the next paragraph.) 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.

Second, now, consider 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 to 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 $\mathbf{A}[s_j]$ if s_j is the source of packet a_i; if it is the end of the flow, subtract one from $\mathbf{A}[s_j]$ if 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.

A note: 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. My own experience is with cellular CDRs and there is a lot you can do to discover engineering problems in a network in near-real time with the live feed of CDRs from the cellular network. 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 and value-added services. 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 [109] and AT&T [30] 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.

At this point, I am going to continue the theme of being imaginative, and suggest a mental exercise. Consider a data streaming scenario from Section 3 that is different from the IP traffic log case. For example,

Exercise 1 *Consider multiple satellites continuously gathering multiple terrestial, 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 if you are teaching a course. 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. Wealth of (useful, fundamental) research remains to be done.

Those who want a mental exercise more related to the Computer Science concepts can consider the streaming text scenario [110].

Exercise 2 *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. This involves text processing on data streams which is quite different from the IP traffic logs or the satellite-based terrestial or stellar observation scenarios.

We need to develop the two 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. As Martin Strauss quips, hope this is not a case of "insurmountable opportunities".

4.3 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.

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.[5]

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. Here, you may have in mind to compute any common function (transitive closure, eigenvalues, etc) and want to do it on massive data in one or more passes. 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. Here is how it maps into the data stream scenario. 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 sream 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.

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. They will not be discussed further, mea culpa. Again data stream algorithms find direct applications in these areas.

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

[5] Anecdotics. *John Bates* of US National Oceanographic and Atmospheric Administration (http://www.etl.noaa.gov/ jbates/) faces the task of copying two decades worth of data from legacy tapes into current tapes, which will take a couple of years of continuous work on multiple tape readers. His question: during this intensive copying process, blocks of data reside on disk for a period of time. In the interim, can we perform much-needed statistical analyses of historic data? This is apt for data stream algorithms.

space and fast compute time aspect of data stream algorithms to avoid rescanning the database for quick monitoring.

5 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. It is sketchy at this point with pointers to papers.

5.1 Basic Mathematical Ideas

5.1.1 Sampling

Many different sampling methods have been proposed: domain sampling, universe sampling, reservoir sampling, distinct sampling etc. Sampling algorithms are known for:

- Find the number of distinct items in a Cash Register data stream. See [85].
- Finding the quantiles on a Cash Register data stream. See [83] for most recent results.
- Finding frequent items in a Cash Register data stream. See [84].

Each of these problems has nice applications (and many other results besides the ones we have cited above). Further, it is quite practical to implement sampling even on high speed streams. (In fact, some of the systems that monitor data streams—specially 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, else, valuable signals may be lost.) Also, keeping a sample helps one estimate many different statistics, and additionally, actually helps one return certain sample answers to non-aggregate queries. Consider:

Problem 3 *Say we have data streams over two observed variables* (x_t, y_t). *An example* correlated aggregate *is* $\{g(y_t) \mid x_t \leq f(x_1 \cdots x_t)\}$, *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 [87] for the motivation.*

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 [86] for some lower bounds. Second, sampling method typically does not work in the Turnstile data stream model: as 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.

5.1.2 Random Projections

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

Building on the influential work [1], Indyk [89] proposed using stable distributions to generate the random variables. Sketches with different stable distributions are useful for estimating various L_p norms on the data stream. In particular, sketches using Gaussian random variables get a good estimate of the L_2 norm of data streams, using Cauchy distributions one gets a good estimate for the L_1 norm etc. I have not found a

lot of motivation for computing the L_1 or L_2 norm of a data stream by itself, although these methods prove useful for computing other functions. For example,

- Using L_p sketches for $p \to 0$, we can estimate the number of distinct elements at any time in the Turnstile data stream model [90].
- Using variants of L_1 sketches, we can estimate the quantiles at any time in the Turnstile data stream model [91].
- Using variants of L_1 sketches and other algorithmic techniques, we can dynamically track most frequent items [92], wavelets and histograms [93], etc. in the Turnstile data stream model.
- Using L_2 sketches, one can estimate the self-join size of database relations [97]. This is related to estimating inner product of vectors, which is provably hard to do in general, but can be estimated to high precision if the inner product is large.

There are many variations of random projections which are of simpler ilk. For example, Random subset sums [27], counting sketches [28] and also Bloom filters [29]. A detailed discussion of the connection between them is needed.

Problem 4 *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 [58].

A powerful concept in generating a sequence of random variables each drawn from a stable distribution is doing so with the property that any given range of them can be summed fast, on demand. Such constructions exist, and in fact, they can be generated fast and using small space. Number of constructions are now known: preliminary ones in [94], Reed-Muller construction in [27], general construction in [93] with L_1 and L_2 sketches, and approach in [95] for stable distributions with $p \to 0$. They work in the Turnstile model and find many applications including histogram computation and graph algorithms on data streams.

5.2 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 powerful.

5.2.1 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 them

truthfully. How many questions does Carole need, in the worst case? There is an entire area of Combinatorial Group Testing that produces solutions for such problems. In the data stream case, each question is a group of items and the algorithm plays the role of Carole. This set up applies to a number of problems in data streams. (It may also be thought of as coding and decoding in small space.) Examples are found in:

- Finding B most frequent items in Turnstile data streams [92].
- Determining the highest B Haar wavelet coefficients in Turnstile data streams [93].
- Estimating top B fourier coefficients by sampling [96].

Problem 5 *Paul sees data stream representing $\mathbf{A}_P$ and Carole sees data stream representing $\mathbf{A}_C$, both on domain $1, \ldots, N$. Design a streaming algorithm to determine certain number of i's with the largest $\frac{\mathbf{A}_P[i]}{\max 1, \mathbf{A}_C[i]}$.*

Monika Henzinger and Jennifer Rexford posed this problem to me at various times. It 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 [95].

5.2.2 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 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 [97].
- Building a histogram on the Time Series data stream [98]. This also has applications to finding certain outliers called the deviants [105].
- Building a parse tree atop the Time Series data stream seen as a string [81]. This has applications to estimating string edit distances as well as estimating size of the smallest grammar to encode the string.

Here is a problem of similar ilk, but it needs new ideas.

Problem 6 *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 [99].*

There are other applications, where the tree hierarchy is imposed as an artifact of the problem solving approach. The k-means algorithm on the data stream [100] can be seen as a tree method: building clusters on points, building higher level clusters on their representatives, and so on up the tree.

Finally, I will speculate that Yair Bartal's fundamental result of embedding arbitrary metrics into tree metrics will find applications in data streams context, by reducing difficult problems to ones where the tree method can be applied effectively.

5.2.3 Robust Approximation

This concept is a variation on the local minima/maxima, but suited for approximations. Consider constructing a near-optimal B bucket histogram for the signal. An approximation H to the optimal histogram H^* is called *robust* if it has the property that when refined further with a few buckets, the resulting histogram is only a little better than H itself as an approximation to H^*. This is a powerful concept for constructing histograms: to get a B bucket optimal histogram, we first pull out a $poly(B, \log N)$ bucket histogram that is robust and then cull a B bucket histogram from it appropriately which is provably $1+\epsilon$ approximate. The details are in [93]. We suspect that robust approximations will find many applications.

In a recent result [101] on an improved algorithm for the k-median problem on data streams, in the first phase, a $O(k \text{ polylog}(n))$ facility solution is obtain from which the algorithm culls the k facilities which is provably $1+\epsilon$ accurate. This is reminiscent of robust approximation, but there is a technical distinction: the $O(k \text{ polylog}(n))$ facility solution does not seem to have the robustness property.

5.2.4 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. This technique has been used in one dimensional nearest neighbor problems and facility location [46], maintaining statistics within a window [36], and from a certain perspective, for estimating the number of distinct items [34]. It is a simple and natural strategy which is likely to get used seamlessly in data stream algorithms.

5.3 Lower Bounds

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

- Compressibility argument.

  In Section 1.2 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.

- Communication Complexity.

  Communication complexity models have been used to establish lower bounds for data stream problems. In particular, see [1]. Estimating set disjointness is a classical, hard problem in Communication Complexity that underlies the difficulty in estimating some of the basic statistics on data streams. See [102] for a few different communication models in distributed stream settings.

- Reduction.

  One reduces problems to known hard ones. Several such results are known. See [95] for some examples.

An information-based approach to data stream lower bounds is in [103].

5.4 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

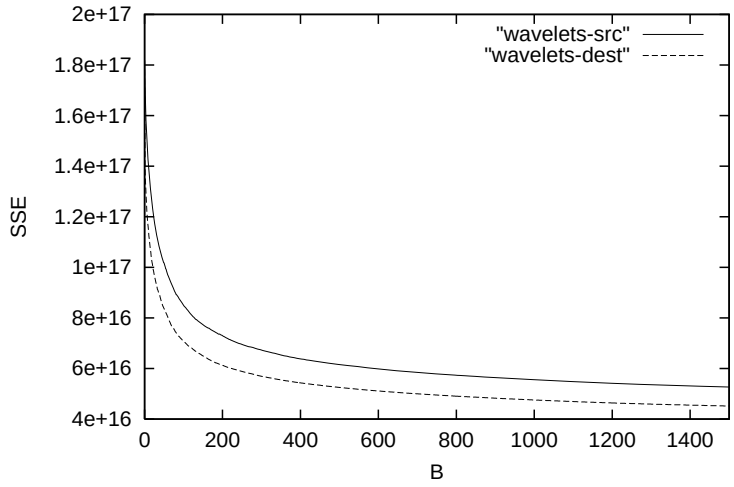

Figure 1: Decay of SSE of top wavelet coefficients on IP data.

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 "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 1 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 large fraction of the traffic [3]. 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 in $\mathbf{A}$.

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

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

  The number of rare flows—flows involving a small number of packets—and the number of distinct flows is significantly large during network attacks.

  When there are data quality problems in data streams—say an interface is polled more often than expected—the problem is abundant, eg., the number of polls may be far more than expected.

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.

6 Streaming Systems

There are systems that (will) process data streams. I think of them in three 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 to get additional facility in manipulating streams. The work at AT&T Research on Call Detail Records analysis falls in this category; Hancock [67] is a special-purpose language. Researchers like Andrew Moore [124] work in this framework, and quite successfully process large data sets. Ultimately however, I do not know of such work that processes data at the stream rate generated by IP network routers.

Second, there are systems that let a high performance database process updates using standard technology like bulk loading, or fast transaction support. Then one builds applications atop the database. IPSOFACTO [57] is such a system that lets Daytona database handle SNMP log updates and provides application level support for visualizing and correlating traffic patterns on links between IP routers. This works well for SNMP logs and is used on production quality datafeed, but will be highly stressed for packet or flow logs. Bellman [66] which monitors data quality problems in databases takes this approach as well, capturing transactions from a generic database and performing statistical analysis on relationships between attributes in various database tables. It needs further work to scale to large transaction rates.

Finally, 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 [70], Aurora [72], Telegraph [73], Stanford Stream [71] etc. They seem to be under development, and demos are being made at conferences (see SIGMOD 2003). Another system I know of in this category is Gigascope [74] which is operationally deployed in an IP network. It does deal with stream rates generated in IP networks, but at this point, it provides only features geared towards IP network data analysis. It is not yet suitable for general purpose data stream management for a variety of data streams.

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. 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? Both Bellman and IPSOFACTO use some of the approximation algorithms including sampling and random projections. Most of the systems in the third category have hooks for sampling. There is discussion of testing random projection based estimations using Gigascope, and reason to believe that simple, random projections technique will be useful in other systems too.

7 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 [17].

7.1 Related Areas

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

- *PAC learning:* In [47] 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.
- *Online algorithms:* Data stream algorithms have an online component where input is revealed in steps, but they have resource constraints that are not typically incorporated in competitive analysis of online algorithms.
- *Property testing:* This area focuses typically on sublinear time algorithms for testing objects and separating them based on whether they are far from having a desired property, or not. Check out Ronitt Rubinfeld's talk on what can be done in sublinear time [123]. 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 do more in streaming models.
- *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 [49], reasoning about individual chains [48], etc. under resource constraints of data streams. This outlook needs to be developed a lot further.

7.2 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 Λ, $|\Lambda| \leq B$. 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 $\|\mathbf{A} - \mathbf{R}\|_2$. I will only focus on the L_2 error here. A signal has a $\mathbf{R}$ with error zero if $B = N$ since $\mathcal{D}$ spans $\mathcal{R}^N$.

Many of us are familiar with a special case of this problem if the dictionary is a Fourier basis, i.e., ϕ_i's are appropriate trigonometric 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 product $| \langle \mathbf{A}, \phi \rangle |$ 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 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 two directions: studying specific dictionaries derived from applications or studying the problem for an arbitrary dictionary so as to be completely general.

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.

$$W_{2n}(x) \quad = \quad W_n(2x) - W_n(2x-1)$$

$$W_{2n+1}(x) \quad = \quad W_n(2x) - W_n(2x-1)$$

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 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.

A gem of a result on this problem is in [80]: 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 doable.) Presumably this result can be improved by allowing some approximation to the best B term representation.

Problem 7 *There are a number of other special dictionaries of interest—beamlets, curvelets, ridgelets, segmentlets, etc.—each suitable for classes of applications. Design efficient algorithms to approximate the best representation of a given function using these dictionaries.*

Studying general dictionaries. A paper that seems to have escaped the attention of approximation theory researchers is [63] which proves the general problem to be NP-Hard. This was reproved in [60]. In addition, [63] contained 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 [63] 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. [63] deserves to be revived: many open problems remain.

Problem 8 *Improve [63] to use fewer terms, perhaps by relaxing the error achieved. Is there a nontrivial non-greedy algorithm for this problem?*

Problem 9 *A technical problem is as follows: The algorithm in [63] 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. This is likely to be not difficult: instead of finding the "best", find the "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 10 *[64] 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 $\|\mathbf{A} - \mathbf{R}_{\text{opt}}\| = \delta$, where $B < 1/(32\mu)$. Then, in iterations polynomial in B, we can find a representation with error at most $\sqrt{(1 + 2064\mu B^2)}\delta$.*

This line of research is just being developed; see [68] for new developments.

Further in [64], authors used approximate nearest neighbor algorithms to implement the iterations in Theorem 10 efficiently, and proved that approximate implementation of the iterations does not degrade the

error estimates significantly. I think this is a powerful framework, and efficient algorithms for other problems in Functional Approximation Theory will use this framework in the future. Recently, Ingrid Daubechies spoke some of these results at the AMS-MAA joint meetings [69].

Functional approximation theory has in general focused on characterizing the class of functions for which error has a certain decay as $N \to \infty$. See [62] and [61] for many such problems. But from an algorithmicists point of view, the nature of problems I discussed above are more clearly more appealing. This is a wonderful area for new algorithmic research; a starting recipe is to study [62] and [61], formulate algorithmic problems, and to solve them.

Let me propose two further, out-there directions: Can we design new wavelets based on general two dimensional tiling (current wavelet definitions rely on rather restricted set of two dimensional tiling)? Can we design new wavelets based on the $2-3$ tree decomposition a la ESP in [81]? 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.

7.3 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 remember $O(\log n)$ bits, where n is the number of items in the list. 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 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. Richard Cole's work on dynamic finger conjecture for splay trees is an example of deep problems to be found [21].

Recently, a nice result has appeared [20]. 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.) I think of this as a streaming result. I believe and hope 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, of course.

Let me present a specific data structural traversal problem.

Problem 11 *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. I think this problem arises in query optimization for tertiary databases from Sunita Sarawagi's thesis work.

7.4 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 [state time, end time], text documents get mapped to high dimensional vector spaces, etc.

Consider the problem of estimating the diameter of points presented on a data stream. Two results are interesting.

Indyk 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 $v_1, \ldots, v_l$ and for each v_i, maintain the two points with largest and smallest $v_i p$ 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 $O(1/\epsilon)$ space and $O(\log(1/\epsilon))$ compute time per inserted point [45].

I know of other results in progress, so more computational geometry problems will get solved in the data stream model in the near future.

Let me add a couple of notes. First, 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, will find many applications. This needs to be explored. Second, I do not know of many 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, let me say 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). Finally:

Problem 12 *(Facility location) Say Paul tracks n potential sites on the plane. Carole continuously either adds new client points or removes an existing client point from the plane. Paul can use space $n\ polylog(n)$, but only $o(m)$, preferably $polylog(m)$, where m is the total number of points at any time. Solve the k-means or k-medians facility location problem on the set of n sites.*

This problem arises in a study of sensors on highways [46].

7.5 Graph Theory

The graph connectivity problem plays an important role in log space complexity. See [41] for some details. However, hardly any graph problem has been studied in the data stream model where (poly)log space requirement comes with other constraints.

In [42], 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

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \end{pmatrix} . \begin{pmatrix} T_0 \\ T_1 \\ T_2 \end{pmatrix}$$

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, ie., 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 5.1. As a result T_3 can be approximated. In fact, this method works in the Turnstile model as well even though the authors in [42] did not explicitly study it.

The general problem that is interesting is to count other subgraphs, say constant sized ones. Certain small subgraphs appear in web graphs intriguingly [43], 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.

Many graph problems need to be explored in data stream models. But they appear to be difficult in general. One has to find novel motivations and nifty variations of the basic graph problems.

Let me propose a direction.

Problem 13 *Consider the* semi-streaming *model, ie., one in which we have space to store vertices, say $O(|V|polylog(|V|)$ bits, but not enough to store the edges. So we have $o(|E|)$ bits. Solve interesting (in particular, dense) graph problems in this model.*

7.6 Databases

Databases research has considered streaming extensively, far too much to be summarized here. I will highlight a few interesting directions.

Consider approximate query processing.

Problem 14 *Consider a signal $\mathbf{A}$ where $\mathbf{A}[i]$ is a subset of $1, \cdots U$. 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 "optimal" for this task. First consider a static $\mathbf{A}$ and later streaming signals.*

A lot has to be formalized in the problem above (See [79] for some related details). 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 pages that contain tuples with value i, histogram described in the problem above is relevant. To those with the background, I can say, this is an attempt at modeling the *page selectivity* of queries.

A somewhat related problem concerns multidimensional signals.

Problem 15 *Consider two dimensional signal $\mathbf{A}_t[i][j]$, $i, j \in [1..n]$. Design algorithms for building (near) optimal two dimensional histogram with B partitions.*

Readers should not think of this as a straightforward generalization of one dimensional problem to multiple dimensions. The problem is, but the details are quite different. There are many ways to partition two dimensional arrays. While one dimensional problem is polynomial time solvable, two dimensional problems are NP-hard for most partitions. Further, as I 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 [75], authors proposed efficient approximation algorithms for a variety of two dimensional histograms for a static signal. Some preliminary results were presented in [76] for the streaming case: specifically, the authors proposed a polylog space, $1+\epsilon$ approximation algorithm using $O(B \log N)$ partitions, taking $\Omega(N^2)$ time. Using the ideas in [75] and robustness, I believe that a truly streaming algorithm can be obtained, i.e., one that is a B partitions, $1 + \epsilon$ approximation using *both* polylog space as well as polylog time, but details will be published soon.

Both the questions above are rather technical. 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. Let me propose a direction that is likely to be interesting.

Problem 16 *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 adversial rate theories are of relevance.*

7.7 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 eg. [5]) previously, but more needs to be done. There is commercial potential in such hardware machines. Consider:

Problem 17 *Develop hardware implementation of the inner product based algorithms described in Section 5 for various data stream analyses.*

Here is a related topic. The trend in 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 [38] for an example, [39] for a suitable model, and [82] 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. Lot remains to be explored here; see overview [40].

7.8 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.

7.8.1 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 avocation of permuting from Section 1.1.)

A number of problems have already been studied in this model. In [37], authors studied how to estimate various permutation edit distances. The problem of estimating the number of inversions in a permutation was studied in [33]. Here is an outline of a simple algorithm to estimate the number of inversions [31]. Let $\mathbf{A}_t$ is 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

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

The authors in [31] show how to estimate $|\{j \mid j > i \ \& \ \mathbf{A}_t[j] = 1\}|$ for any i, up to $1 + \epsilon$ accuracy using exponentially separated quantiles. They use randomization, and an elegant idea of oversampling (and

retaining certain smallest number of them) for identifying the exponentially separated quantiles. An open problem here is what is the best we can do deterministically, or in the Turnstile model.

A deeper question is whether there is a compelling motivation to study this model, or the specific problems. There is some theoretical justification: permutations are special cases of sequences and studying permutation edit distance may well shed light on the notoriously hard problem of estimating the edit distance between strings. However, I have not been able to find an overwhelming inspiration for these problems and this model. Yet, here is a related problem that does arise in practice.

Problem 18 *Each TCP flow comprises multiple consecutively numbered packets. We see the packets of the various flows in the Cash Register model. Packets get transmitted out of order because of retransmissions in presence of errors, ie., packets may repeat in the stream. 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 TCP flows.*

7.8.2 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 reemphasize 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 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 [36] 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.

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

$$\beta_{i+1} = (1 - \lambda_{i+1})\beta_i + \lambda_{i+1}\sigma_{i+1}.$$

If we unravel the inductive definition, we can see that the signal from a block affects the meta-signal exponentially less as new blocks get seen. 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 [32] 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?

Let me propose another natural model, a *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, etc. that is being investigated now.

7.8.3 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!"?

Answer of course depends on n, and not by a formula. See [116] for some answers.

This puzzle contains the elements of what I 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.

In the synchronized streaming model, one can do whatever can be done in the generic streaming model in which one of the signals is presented before the other, or the tth updates of the two signals are arbitrarily separated. The interest is if synchronized model can accomplish more. We believe that to be the case. 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. Synchronized streaming is quite natural; more research is needed on this model.

7.9 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 [77].

In traditional view of databases, one sets *integrity constraints* and any violation is considered a data quality problem and exceptions are raised. Bad quality data (eg., 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.

In emerging 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 is simply not a scalable or graceful approach, one needs a different principled approach.

I am a believer in *data quality monitoring* tools. They operate as database applications, monitoring its state by measuring statistics: strong deviation from expected statistics may be projected as a ping for the database administrator or the user to react to. 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 my choice. See [78] for related discussions.

Bellman [66] is such a tool for traditional database systems; it monitors the structure in the database tables using various statistics on the value distribution in the tables. PACMAN [78] is another tool; it uses *probabilistic, approximate constraints* (PACs) to monitor SNMP data streams and works operationally for

a large ISP. PACs are also a principled way to determine what are data quality problems. More needs to be done.

In general, our communities have approached data quality problems as "details" and dealt with individual problems as the need arises. (In Computational Biology for example, one deals with noisy data by redefining a particular problem.) I think 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 19 *Given a set S of strings $s_1, \ldots, s_n$ and set T of strings $t_1, \ldots, t_n$, find a* matching *(ie., one-to-one mapping) $f : S \rightarrow T$ 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.*

7.10 Fish-eye View

Let me do a fish-eye view of other areas where streaming problems abound. The discussion will be elliptical: if you mull over these discussions, you can formulate interesting technical open problems.

7.10.1 Linear Algebra

Many matrix functions need to be approximated in data stream model. Let me propose a specific problem.

Problem 20 *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 k-rank 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.

Similar result has been proved in [51] using appropriate sampling for a fixed $\mathbf{A}$, and recent progress is in [50] for 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: estimating eigenvalues, determinants, inverses, matrix multiplication, etc.

7.10.2 Statistics

We saw how to estimate simple statistical parameters on data streams. We need vastly more sophisticated statistical analyses in data stream models, for example, kernel methods, scan statistics, kurtosis parameters, data squashing, etc., the whole works. In statistics, researchers seem to refer to "recursive computing" which resonates with the cash register model of computation. There is an inspiring article by Donoho [52] which is a treasure-tove of statistical analyses of interest with massive data. Another resource is http://www.kernel-machines.org/. Any of the problems from these resources will be interesting in data stream models. Let me propose a general task:

Problem 21 *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, etc. on the data stream.*

7.10.3 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.

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

Let me propose a question, which is likely to have been in researchers' minds; Sivakumar has some notes.

Problem 22 *Characterize the complexity class given by a deterministic logspace verifier with one-way access to the proof.*

7.10.4 Privacy Preserving Data Mining

Peter Winkler gives an interesting talk on the result in [53] 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 [53] 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 [54] 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 [56]. However, these protocols have high complexity. But there is a demand for efficient solutions, perhaps with provable approximations, in practice. In [55] authors formalized the notion of approximate privacy preserving data mining and presented some solutions, using techniques similar to ones we use in data stream algorithms. Lot remains to be done.

The database community is researching general, efficient methods to make databases privacy-preserving. Let me propose a basic problem.

Problem 23 *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, detecting outliers, etc. For example:

Problem 24 *Paul, Carole and others have a list of banking transactions (deposits, withdrawal, transfers, wires etc.), 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.*

8 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.

I think 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, systems and languages. Ultimately, that translates into new questions and methods in Mathematics: approximation theory, statistics and probability. New mindset—say, seeking only the strong signals, working with distribution summaries—are needed, and that means a chance to reexamine some of the fundamentals.

8.1 Data Stream Art

Trend or not, data streams are now Art.

- There are ambient orbs [121] dubbed "News that Glows" by the New York Times Magazine, Dec 15 2002, pages 104–105, that indicate fluctuations in Dow Jones Industrial Average using continuously modulated glow. Clearly they are useful for more than vetting financial obsessions.
- *Dangling String* created by (wonderful techno-)artist Natalie Jeremijenko, is a live wire connected to a Ethernet cable via a motor; traffic level in the cable is shown by the range of motions from the tiny twitch to a wild whirl, with associated sounds [111].
- Mark Hansen and Ben Rubin have the *Listening Post* exhibit [114] at various locations including the Brooklyn Academy of Music and the Whitney Museum of Contemporary Art in New York where they convert the live text information in Internet chat rooms and message boards into light and sound, described by NY Times as a "computer-generated opera".

Besides being Art, ambient information displays like the ones above are typically seen as Calming Technology [2]; they are also an attempt to transcode streaming data into a processible multi-sensory flow.

8.2 Short Data Stream History

Data stream algorithms as an active research agenda has emerged only over the past few years. The concept of making few passes over the data for performing computations has been around since the early days of Automata Theory. Making one or few passes for selection [8] or sorting [9] got early attention, but the area seems to have evolved slowly. Computer architecture research has long considered data flow architectures which may be thought of as an approach to data streaming, but the area did not address complex operations on each data item.

There was a gem of a paper by Munro and Paterson [8] in 1980 that specifically defined multi-pass algorithms and presented one pass algorithms and multi-pass lower bounds on approximately finding quantiles of a signal.

In early 90's, I remember Raghu Ramakrishnan of U. Wisconsin, Madison, asking me what I can do if I was allowed to make only one pass over the data. Presumably others had this question in their mind too. "Not much", I told Raghu then, but that has changed in the past 6 years.

Phil Gibbons and Yossi Matias at Bell Labs synthesized the idea of Synopsis Data Structures [59] that specifically embodied the idea of small space, approximate solution to massive data set problems. The influential paper [1] used limited independence for small space simulation of sophisticated, one-pass norm estimation algorithms. This is a great example of ideas that emerged from complexity-theoretic point of view—pseudo random generators for space-bounded computations—getting applied to algorithmic problems that are motivated by emerging systems. The paper by Henzinger, Raghavan and Rajagoplan [10] formulated one (and multiple) pass model of a data stream and presented a complexity-theoretic perspective; this is also an insightful paper with several nuggets of observations some of which are yet to be developed. Joan Feigenbaum, working with researchers at AT& T Research, identified, developed and articulated the case for the network traffic data management as a data stream application. This was a great achievement and it is now widely accepted as one of the (chief?) inspiring applications for the area. Significant work was done at research labs—IBM Research and Bell Labs— and select universities about the same time.

Since these early developments, a lot has been done in Theoretical Computer Science community and in others including programming languages, KDD, Databases, Networking, etc. Hancock, a special purpose C based programming language that supports stream handling, got the best paper award in KDD 2000. There is focus on decision trees on data streams in KDD community. Rajeev Motwani gave a thoughtful plenary talk at PODS 2002 on data stream systems focusing on the fundamental challenges of building a general-purpose data stream management system. The associated paper [11] is well worth reading, in particular, for a database perspective. There have been tutorials in both SIGMOD and VLDB in year 2002. [107] DIMACS gathered working groups on data streams. George Varghese addressed the problem of computing at link speed in router line card and focused on simple data stream problems at a SIGCOMM 2002 tutorial. Sprint Labs work on IP monitoring was presented at the SIGMETRICS 2002 tutorial [4]. Jiawei Han has tutorials and talks on data mining problems in data streams [104].

The wonderful website [122] has a lot of information.

8.3 Perspectives

Doubtless more tutorials, workshops and other academic adventures will happen over time; the main point is that data stream agenda now pervades many branches of Computer Science. Industry is in synch too: several companies are promising to build special hardware to deal with incoming data at high speed. I was at a National Academy of Sciences meeting recently [112], where 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—poured energy into, data streams is an already accepted paradigm in many areas of CS and beyond.[6] I think 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. I hope the perspective I have presented in this writeup helps you ideate.

I have mixed exposition with reflections. Thanks to the SODA 2003 PC for giving me the opportunity. I have left out image, audio and video streams, XML streams, etc.

9 Acknowledgements

Greg Fredrickson said we—TCS researchers—are much too timid in our technical writing. I was inspired by an afternoon of discussion with him to be more alpha. Thanks to Mike Waterman for sharing his writings from his camps under the western sky. Joel Spencer enthused me for Paul and Carole, all through my thesis

[6] As a unexpected topping, 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.

days. I am happy to find new roles for them. Sampath Kannan and I shared a new boyhood in Madison, NJ; he did crossword puzzles, I learned about streaming, by osmosis. Mike Paterson at U. Warwick, UK, is an inspiration, for puzzles, problem solving and classic papers.

Rob Calderbank, my boss (many times removed), supported me tremendously as I conceived of one project after another at AT&T, and inspired me with Math and Management. As did my immediate boss David Johnson. Joan Feigenbaum recruited and mentored me. AT&T gave me the opportunity to build systems with SNMP data, AWS cellular CDRs, and Intellectual property—patent data, and a national system for location-based services in wireless networks. That is a lot of toys!

Ted Johnson at AT& T and I have built many systems together. We have shared Tall Boys and commutes. Half of what I know about systems, I learned attacking Ted's work; the other half by building them with him.

I have had many colleagues in databases area, I have tried to learn from them, despite my moody self. Thanks to Vishy Poosala, Divesh Srivastava, Flip Korn, Rajeev Rastogi, Nick Koudas, Swarup Acharya and Minos Garofalakis, for teaching me selflessly. At Rutgers, Tomasz Imielinski has been a guiding colleague, sharing passion for unusual music and theater as well as his invaluable insight into impactful database systems research.

Graham Cormode and I have worked on several problems in data streaming that has shaped my insights. I make his English side sigh with my passion for coining new words. He retaliates with limericks:

There was a researcher called Muthu
Who one day thought it would be cute to
Live out his dreams
By fishing in streams
And if you could, wouldn't you too?

Talking about fishing: that puzzle is from my work with Mayur Datar, who helped me with comments and suggestions on this writeup.

Sometime in 1999, Anna Gilbert, Yannis Kotidis, Martin Strauss and I formed a (cross-functional!) team and studied data streams. I am very thankful to them. My work on histograms is joint with them as well as Sudipto Guha and Piotr Indyk, who have independently made very significant contributions to data streaming.

I am grateful to the many researchers who helped me with pointers: Yaron Minsky, Rajeev Motwani, Amin Shakrallohi, Sasha Barg, Sudipto Guha, Monika Henzinger, Piotr Indyk, D. Sivakumar, George Varghese, Ken Clarkson, David Madigan, Ravi Kumar, Eric Bach, Joachim von zur Gathen and Moses Charikar.

Finally, on professional and personal issues, Martin Farach-Colton has been my guide and inspiration, and I owe him lot more than can be acknowledged in this writeup.

References

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

[2] M. Weiser and J.S. Brown. The coming age of calm technology. Chapter 6. *Beyond calculation: The next fifty years of computing*, by P. J. Denning and R. M. Metcalfe. Springer Verlag. 1996. Versions of this article exist on the web.

[3] G. Varghese. Detecting packet patterns at high speeds. Tutorial at SIGCOMM 2002.

[4] S. Bhattacharya and S. Moon. Network performance monitoring and measurement: techniques and experience. ACM SIGMETRICS tutorial, 2002.

[5] D. Shah, S. Iyer, B. Prabhakar and N. McKeown. Maintaining statistics counters in router line cards. *IEEE Micro*, 2002, 76–81.

[6] S. Muthukrishnan. On optimal strategies for searching in presence of errors. *Proc. SODA*, 1994, 680–689.

[7] J. Gray, P. Sundaresan, S. Eggert, K. Baclawski and P. Weinberger. Quickly generating billion-record synthetic databases. *Proc. ACM SIGMOD*, 1994, 243–252.

[8] I. Munro and M. Paterson. Selection and sorting with limited storage. *Proc. IEEE FOCS*, 1978, 253–258. Also, *Theoretical Computer Science* 12: 315–323, 1980.

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

[10] M. Henzinger, P. Raghavan and S. Rajagopalan. Computing on data stream. Technical Note 1998-011. Digital systems research center, Palo Alto, May 1998.

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

[12] N. Duffield, C. Lund and M. Thorup. Learn more, sample less: control of volume and variance in network measurement. SIGCOMM 2001 Measurement workshop.

[13] N. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking*, Vol 9, No 3, 2001.

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

[15] J. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2), 2001, 209–271.

[16] A. Szalay, J. Gray and J. vandenBerg. Petabyte scale data mining: dream or reality? MSR-TR-2002-84, 2002.

[17] S. Kannan. Open problems in streaming. Ppt slides on request from the source.

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

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

[20] G. Blelloch, B. Maggs, S. Leung and M. Woo. Space efficient finger search on degree-balanced search trees. *Proc. ACM-SIAM SODA*, 2003.

[21] R. Cole. On the dynamic finger conjecture for splay trees, part II. The proof. Technical report TR1995-701, Courant Institute, NYU, 1995.

[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*, 1998, 269–278.

[23] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[24] Y. Minsky, A. Trachtenberg and R. Zippel. Set reconciliation with nearly optimal communication complexity. Technical Report 2000-1796, Cornell Univ.

[25] J. Spencer and P. Winkler. Three thresholds for a liar. *Combinatorics, Probability and Computing*, 1:1 (1992), 81–93.

[26] A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss. QuickSAND: Quick summary and analysis of network data. DIMACS Technical Report, 2001-43.

[27] 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.

[28] M. Charikar, K. Chen and M. Farach-Colton. Finding frequent items in data streams. *Proc. ICALP*, 2002, 693–703.

[29] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. Allerton Conference, 2002.
http://www.eecs.harvard.edu/m̃ichaelm/NEWWORK/papers.html.

[30] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford and F. True. Deriving traffic demands for operational IP networks: methodology and experience. *IEEE/ACM Transactions on Networkin*, 9(3), 265–280, 2001.

[31] A. Gupta and F. Zane. Counting inversions in lists. *Proc. SODA*, 2003, 253–254.

[32] C. Cortes and D. Pregibon. Signature-based Methods for Data Streams. *Data Mining and Knowledge Discovery* 5(3): 167-182, 2001.

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

[34] P. Flajolet and G. Nigel Martin. Probabilistic counting. *Proc. FOCS*, 1983, 76–82.

[35] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(16): 237–264, 1953.

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

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

[38] 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. SIGGRAPH 2002.

[39] S. Krishnan, N. Mustafa, S. Muthukrishnan and S. Venkatasubramanian. Extended range search queries on geometric SIMD machine. *Manuscript*, 2002.

[40] Interactive geometric computations using graphics hardware. Course. Organizer: D. Manocha. SIGGRAPH 2002.
http://www.siggraph.org/s2002/conference/courses/crs31.html

[41] R. Armoni, A. Ta-Shma, A. Wigderson and S. Zhou. A $(\log n)^{4/3}$ space algorithm for (s, t) connectivity in undirected graphs. *JACM*, 47(2), 294–311, 2000.

[42] 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.

[43] P. Raghavan. Graph structure of the web: A survey. *LATIN 2000*, 123–125.

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

[45] J. Feigenbaum, S. Kannan and J. Ziang. Computing diameter in the streaming and sliding window models. *Manuscript*, 2002.

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

[47] N. Mishra, D. Oblinger and L. Pitt. Sublinar time approximate clustering. *Proc. ACM-SIAM SODA*, 2001.

[48] J. Kleinberg. Bursty and hierarchical structure in streams. *Proc. 8th ACM SIGKDD KDD conf.*, 2002.

[49] T. Batu, S. Guha and S. Kannan. Inferring mixtures of markov chains. *Manuscript*, 2002.

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

[51] D. Achlioptas and F. McSherry. Fast computation of low rank approximation. *STOC*, 2001.

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

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

[54] A. Yao. Protocols for secure computations. *Proc. FOCS*, 1982, 160–164.

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

[56] O. Goldreich. Secure multiparty computation. Book at http://philby.ucsd.edu/cryptolib/BOOKS/oded-sc.html. 1998.

[57] F. Korn, S. Muthukrishnan and Y. Zhu. IPSOFACTO: IP stream-oriented fast correlation tool. *SIGMOD 2003 Demo and Manuscript*, 2003.

[58] S. Chien, L. Rasmussen and A. Sinclair. Clifford algebras and approximating the permanent. *ACM STOC* 2002, 222–231.

[59] P. Gibbons and Y. Matias. Synopsis data structures. *Proc. SODA*, 1999, 909–910.

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

[61] V. Temlyakov. The best m-term approximation and greedy algorithms. *Advances in Computational Math.*, 8:249–265, 1998.

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

[63] B. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Computing*, 25(2): 227–234, 1995.

[64] A. Gilbert, S. Muthukrishnan and M. Strauss. Approximation of Functions over Redundant Dictionaries Using Coherence. *Proc. SODA*, 2003.

[65] B. Mazur. *Imagining numbers (Particularly the square root of minus fifteen)*. Farrar, Strauss and Giroux, 2003.

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

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

[68] J. Tropp. Greed is good: Algorithmic results for sparse approximation. Technical Report 2003-04, Texas Institute for Computational and Applied Mathematics.

[69] I. Daubechies. Sublinear algorithms for sparse approximations with excellent odds. http://www.ams.org/amsmtgs/2074_abstracts/983-41-1214.pdf

[70] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. *ACM SIGMOD* 2000.

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

[72] D. Abadi et al. Aurora: A data stream management system. *ACM SIGMOD* 2003, Demo. http://www.cs.brown.edu/research/aurora/demo.pdf

[73] S. Chandrasekharan et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. *CIDR*, 2003.

[74] C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk and O. Spatscheck. Gigsacope: High performance network monitoring with an SQL interface. *ACM SIGMOD* 2002. http://athos.rutgers.edu/ muthu/demo_02.pdf

[75] S. Muthukrishnan, V. Poosala and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity and applications. *Proc. Intl Conf on Database Theory* (ICDT), 1999, 236–256.

[76] N. Thaper, S. Guha, P. Indyk and N. Koudas. Dynamic multidimensional histograms. *ACM SIGMOD*, 2002, 428–439.

[77] T. Dasu and T. Johnson. *Exploratory data mining and data quality.* ISBN: 0-471-26851-8. Wiley, May 2003. http://www.wiley.com/cda/product/0,,0471268518%7Cdesc%7C2927,00.html

[78] F. Korn, S. Muthukrishnan and Y.Zhu. Checks and balances: Monitoring data quality in network traffic databases. *manuscript*, 2003.

[79] S. Muthukrishnan and M. Strauss. Rangesum histograms. *ACM-SIAM SODA*, 2003.

[80] L. Villemoes. Best approximation with walsh atoms. *Constructive Approximation*, 133, 329–355, 1997. See http://www-old.math.kth.se/math/users/larsv/thesis.html

[81] G. Cormode and S. Muthukrishnan The string edit distance matching problem with moves. *ACM-SIAM SODA*, 2002.

[82] S. Guha, K. Mungala, K. Shankar and S. Venkatasubramanian. Application of the two-sided depth test to CSG rendering. *I3d, ACM Interactive 3D graphics*, 2003.

[83] M Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. *Proc. ACM SIGMOD*, 2001.

[84] G. Manku and R. Motwani. Approximate frequency counts over data streams. *Proc. VLDB*, 2002, 346–357.

[85] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. *Proc. VLDB*, 2001, 541–550.

[86] S. Chaudhuri, R. Motwani and V. Narasayya. Random sampling for histogram construction: How much is enough? *Proc. SIGMOD*, 1998, 436–447.

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

[88] A. Broder, M. Charikar, A. Freize and M. Mitzenmacher. Min-wise independent permutations. *Proc. STOC*, 1998,327–336.

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

[90] G. Cormode, M. Datar, P. Indyk and S. Muthukrishnan. Comparing data streams using hamming norms (How to zero in). *Proc. VLDB*, 2002, 335–345.

[91] A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. *Proc. VLDB*, 2002, 454–465.

[92] G. Cormode and S. Muthukrishnan. What is hot and what is not: Tracking most frequent items dynamically. *ACM PODS*, 2003.

[93] A. Gilbert, S. Guha, Y. Kotidis, P. Indyk, S. Muthukrishnan and M. Strauss. Fast, small space algorithm for approximate histogram maintenance. *ACM STOC*, 2002, 389–398.

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

[95] G. Cormode and S. Muthukrishnan. Estimating dominance norms on multiple data streams. *Manuscript*, 2003.

[96] A. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan and M. Strauss. Near-optimal sparse fourier estimation via sampling. *ACM STOC*, 2002, 152–161.

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

[98] S. Guha, N. Koudas and K. Shim. Data streams and histograms. *ACM STOC*, 2001, 471–475.

[99] S. Muthukrishnan, R. Shah and J. Vitter. Finding deviants on data streams. *Manuscript*, 2003.

[100] S. Guha, N. Mishra, R. Motwani and L. O'Callaghan. Clustering data streams. *IEEE FOCS*, 2000, 359–366.

[101] M. Charikar, L. O'Callaghan and R. Panigrahy. Better streaming algorithms for clustering problems. *ACM STOC*, 2003.

[102] P. Gibbons and S. Trithapura. Estimating simple functions on the union of data streams. *ACM SPAA*, 2001.

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

[104] Y. Chen, G. Dong, J. Han, B. Wah, and J. Wang. Multi-Dimensional Regression Analysis of Time-Series Data Streams. *Proc. VLDB*, 2002, 323–334. Also see: http://www-courses.cs.uiuc.edu/ cs497jh/ppt/topics01.ppt.

[105] H. Jagadish, N. Koudas, and S. Muthukrishnan. Mining deviants in a time series database. *Proc. VLDB*, 1999, 102–113.

[106] J. Gray and T. Hey. In search of petabyte databases. http://www.research.microsoft.com/ Gray/talks/

[107] Querying and mining data streams: you only get one look. See http://www.bell-labs.com/user/minos/tutorial.html.

[108] http://www.arm.gov/docs/index.html. Data streams at http://www.archive.arm.gov/docs/catalog/

[109] http://www.sprintlabs.com/Department/IP-Interworking/Monitor/ http://ipmon.sprintlabs.com/

[110] http://stat.rutgers.edu/ madigan/mms

[111] http://cat.nyu.edu/natalie/.

[112] http://www7.nationalacademies.org/bms/Massive_ Data_ Workshop.html.

[113] http://www.unidata.ucar.edu/data/data.general.html

[114] http://cm.bell-labs.com/cm/ms/departments/sia/ear/index.html, and http://earstudio.com/projects/listeningPost.html.

[115] http://www.tpc.org/. Details of transactions testing at http://www.tpc.org/tpcc/detail.asp

[116] http://wordways.com/integers.htm.

[117] http://www.cacr.caltech.edu/SDA/digital_ sky.html and http://archive.stsci.edu/sdss/

[118] http://www.noaa.gov/ and http://www.ngs.noaa.gov/

[119] http://www.unidata.ucar.edu/data/data.detail.html

[120] http://cmsdoc.cern.ch/cms/outreach/html/CMSdocuments/CMSchallenges/CMSchallenges.pdf

[121] http://www.ambientdevices.com/cat/index.html.

[122] http://www-db.stanford.edu/stream/

[123] http://external.nj.nec.com/homepages/ronitt/

[124] http://www.autonlab.org/autonweb/index.jsp and http://www-2.cs.cmu.edu/ awm/