CSC2421 Topics in Algorithms: Online and Other Myopic Algorithms Fall 2019

Allan Borodin

November 6, 2019

Todays agenda

- I would like the few people we have doing projects to give us updates weekly
  Anyone want to provide an update today?
- I will be going rather quickly over two new topics today and perhaps expand later.
  1. Streaming algorithms
  2. Stochastic analysis

The streaming model

- In the data stream model, the input is a sequence A of input items (or input elements) $a_1, \ldots, a_n$ which is assumed to be too large to store in memory. Each $a_i \in U$ for some universe U with $m = |U|$. .
- We assume that the sequence is generated by an adversary and arrives online. We usually assume that n is not known.
- While the streaming model is similar to the online model in that an algorithm has no control over the order of arrival of input items, that is where the similarity ends. While the online setting focuses on irrevocable immediate decisions for each input arrival, the streaming setting focuses on the amount of memory required to process the input.
- The space available $S(n, m)$ is some sublinear function. The input items stream by and one can only store information in space S.
- In some applications, a streaming algorithm is only required to produce an answer after seeing the entire stream, thus "revocable decisions" are possible provided they can be implemented within the given space requirements.

The streaming model continued

In other applications, a streaming algorithm may have to frequently output various statistics about the input stream.

Although time and space complexity are always an algorithmic issue, from a theoretical perspective, online competitive analysis does not impose any time and space restrictions. Similarly, streaming algorithms do not impose any time restrictions (although in practice, the total time or even time per input item is usually important).

The streaming model is motivated by practical applications. In networking it is quite common to have throughput around 2.56 terabits per second for the top of the rack switch. One needs to compute some statistics about this stream, e.g., the number of distinct IP addresses, skewness of packet sizes, etc. It is not feasible to store all that information locally and run some offline algorithm on it, instead the computation has to be performed with very limited memory compared to the size of the stream.

Similar requirements often arise in experimental physics and astronomy and other applications.

The streaming model continued

- In some papers, space is measured in bits (which is what we will usually do) and sometimes in words, each word being $O(\log n)$ bits.
- As stated, It is also desirable that that each input item is processed efficiently, say $\log(n) + \log(m)$ time, and perhaps even in time $O(1)$ (assuming we are counting operations on words as $O(1)$).
- The initial (and primary) work in streaming algorithms is to approximately compute some function (say a statistic) of the data or identify some particular item(s) in the data stream.
- Lately, the model has been extended to consider "semi-streaming" algorithms for optimization problems. For example, for a graph problem such as matching for a graph $G = (V, E)$, the goal is to obtain a good approximation using space $\tilde{O}(|V|)$ rather than $O(|E|)$.
- Most results concern the space required for a one pass algorithm. But there are results concerning multi-pass algorithms and also results concerning the tradeoff between the space and number of passes.

A simple deterministic streaming algorithm

Chapter 11 now only contains some positive results concerning streaming algorithms. We will be adding negative results. Most negative results are obtained through a beautiful connection between streaming algorithms and two-party communication protocols.

We will start with a simple deterministic streaming algorithm. The problem is to find the unique missing item in a set of integers. I have given this problem just as an exercise without a hint and it is easy to come up with online and offline and online algorths that are much worse than what can be obtained by a simple online streaming algorithm.

Finding the missing item

You are given a sequence/stream of integers $x_1, \ldots, x_n$ such that each $x_i \in \{1, 2, \ldots, n+1\}$. Moreover, you are promised that all the x_i are distinct. This means that exactly one integer in $\{1, 2, \ldots, n+1\}$ is missing from $\{x_1, \ldots, x_n\}$. Your goal is to find this integer making a single pass over the sequence and using as little memory as possible.

A trivial solution is to keep an array A of size $n+1$ initialized to all zeros. When x_i arrives, you can update the corresponding array entry $A[x_i] \leftarrow 1$. After making a single pass over the sequence, you can find the index x of the array entry that is equal to 0, i.e., $A[x] = 0$. Then x is the missing number. This approach requires n bits of memory. We would like to design an algorithm that uses exponentially fewer bits of memory, i.e., $O(\log n)$ bits.

Missing item continued

A $O(\log n)$ space streaming algorithm is easily obtained since the missing number x can be recovered via the following simple calculation:

$$x = \left(\sum_{i=1}^{n+1} i\right) - \left(\sum_{i=1}^{n} x_i\right) = \frac{(n+1)(n+2)}{2} - \left(\sum_{i=1}^{n} x_i\right).$$

The sum $S = \sum_{i=1}^{n} x_i$ can be computed in a single pass over the sequence using $O(\log S) = O(\log(n+1)^2) = O(\log n)$ bits of space.

Observe that we cannot have an exact deterministic algorithm using $o(\log n)$ bits of space for this problem. If such an algorithm ALG existed, then by the pigeonhole principle there would exist two streams with different missing numbers on which ALG would give the same answer.

Generalizing to k missing elements

Now suppose we are promised a stream A of length $n-k$ whose input elements consist of a permutation of $n-k$ distinct elements in $\{1,\ldots,n\}$. We want to find the missing k elements.

- Generalizing the one missing element solution, to the case that there are k missing elements we can (for example) maintain the sum of j^{th} powers $(1 \leq j \leq k)$ $s_j = \sum_{i \in A}(a_i)^j = c_j(n) - \sum_{i \notin A} x_i^j$. Here $c_j(m)$ is the closed form expression for $\sum_{i=1}^{n} i^j$. This results in k equations in k unknowns using space $k^2 \log n$ but without an efficient way to compute the solution.
- As far as I know there may not be an efficient small space *deterministic* streaming algorithm for this problem.
- Using randomization, much more efficient methods are known; namely, there is a streaming alg with space and time/item $O(k \log k \log n)$; it can be shown that $\Omega(k \log(n/k))$ space is necessary.

Frequent items

You are given a sequence/stream of integers $x_1, \ldots, x_n$ such that $x_i \in \{1, 2, \ldots, m\}$. For this problem we can think of m as being much smaller than n, so certain values have to repeat. For each $i \in \{1, 2, \ldots, m\}$, we define the frequency f_i of integer i as the number of occurrences of i in the stream:

$$f_i = |\{j : x_j = i\}|.$$

Fix an additional parameter $k \in \mathbb{N}$ such that $k \geq 2$. We say that an integer i is k-frequent if $f_i > n/k$. The goal is to identify all k-frequent integers by making a single pass over the sequence and using as little memory as possible. Observe that there can be at most $k - 1$ frequent integers. We are interested in an exact, deterministic, one pass algorithm. Unfortunately, such an algorithm with small memory does not exist, so we need to relax at least one of the conditions. Instead of requiring the algorithm to perform a single pass over the stream, we allow the algorithm to perform 2 passes over the stream.

This problem is also know as the *k heavy hitters* problem.

The majority element

As a special case, lets first consider the case of $k = 2$. That is, we are looking to find (if one exists) an integer i such that $f_i > n/2$; that is a mahjority element.

There is a temptation to solve this problem by divide and conquer; divide the sequence in half, find the heavy hitters in each half and then check.

The streaming model fascilitates thinking about a much better solution. In the case of majority, lets just try to maintain one possible candidate in the first pass and then check to see if the candidate is a true more than majority item in the second pass.

The Misra-Gries algorithm

Aside: This is named for David Gries, father of our own Paul Gries.

Maintain a candidate for the majority element and a counter for that candidate.

When the counter is empty, the next element in the stream becomes the candidate.

Every time the next element in the stream is the candidate increase the counter by 1. If the next element is not the candidate decrease the counter by 1.

Claim: If there is a majority element then it has to be the current candidate.

We can use a second pass over the elements to check if the candidate occurs more than $n/2$ times.

The space used is $O(\log n + \log m)$ and the time is $(\log n)$ (or $O(1)$ if counting element comparisons) per input element.

Returning to the general k-frequent elements

We generalize the majority algorithm as follows: during the first pass, we identify a small set of *candidates* for k-frequent elements. Each k-frequent element is *guaranteed* to appear in this set, but not every element in the set is necessarily a k-frequent element. During the second pass, we maintain an explicit count of the number of times each candidate appears in the stream, so we can simply check which of the candidates are actually k-frequent. Next, we concentrate just on the first pass over the stream.

We present a classical algorithm for this. Since computing f_i exactly for all i requires a lot of space, the main idea behind the algorithm is to *estimate* the frequencies f_i up to an additive error n/k. That is we wish to compute $\widetilde{f}_i$ such that

$$f_i - n/k \leq \widetilde{f}_i \leq f_i.$$

If we can compute $\widetilde{f}_i$, then candidates for k-frequent integers can be identified as those satisfying $\widetilde{f}_i > 0$. We are aiming for the space bound of the form $O(\log m + \log n)$ for constant k.

k-frequent elements continued

The algorithm maintains a counter for at most $k-1$ candidates. When a new integer x_i arrives in the stream, it is processed as follows. If x_i is present among the current candidates then the corresponding counter is incremented. If x_i is not present among the keys, we check to see if it can be added without violating the number of candidates constraint. If it can, then we add x_i to the map and initialize the corresponding counter to 1. If x_i cannot be added to the map, then all the counters corresponding to the current is decremented by one. If a counter gets down to 0, then the corresponding element is removed from the current candidates, potentially making space for new candidates.

The k-frequent element algorithm

procedure FrequentIntegers
    $A \leftarrow$ a map from integers to integers
    $i \leftarrow 1$
    while $i \leq n$ **do**
        if $x_i \in A.keys()$ **then**
            $A[x_i] \leftarrow A[x_i] + 1$
        else if $|A.keys()| < k - 1$ **then**
            $A[x_i] \leftarrow 1$
        else
            for $x \in A.keys()$ **do**
                $A[x] \leftarrow A[x] - 1$
                if $A[x] = 0$ **then**
                    Remove x from $A.keys()$
    for x from 1 to m **do**
        if $x \in A.keys()$ **then**
            $\widetilde{f}_x \leftarrow A[x]$
        else
            $\widetilde{f}_x \leftarrow 0$

Analysis of the k frequent algorithm

Theorem

The Algorithm computes $\widetilde{f}_i$ such that

1. *total space used is $O(k(\log n + \log m))$, and*
2. *for each $x \in \{1, 2, \ldots, m\}$ we have $f_x - n/k \leq \widetilde{f}_x \leq f_x$.*

The only part of the theorem that requires some explanation is that $f_x - n/k \leq \widetilde{f}_x$. To see this, note that a decrement in the approximate count of any element occurs only when $x_i \notin A.keys()$ and $|A.keys()| = k - 1$. Hence when this happens, k counts decrease by 1 simultaneously: $k - 1$ of these come from decrementing each value in A, and 1 more is the virtual count associated with x_i itself, as it can be considered as an implicit sequence of operations $A[x_i] \leftarrow 1$ followed by $A[x_i] \leftarrow A[x_i] - 1$ followed by removing x_i from $A.keys()$. Thus, for any x each decrement of $A[x]$ can be charged to k distinct elements of the stream. Since there are n elements in total, the total number of decrements for a fixed x can be at most n/k. It follows that $\widetilde{f}_x \geq f_x - n/k$.

Approximate Randomized Solutions

We just presented algorithms that use logarithmic space. Moreover, these algorithms gave exact and correct answers and the algorithms were deterministic. Problems admitting exact deterministic solutions are extremely rare in the area of streaming algorithms.

For most streaming problems it is possible to prove that both *approximation* and *randomness* are required for non-trivial algorithms. This results in two extra parameters $\epsilon > 0$ and $\delta > 0$ describing a particular streaming algorithm. The parameter ϵ bounds the approximation guarantee, and the parameter δ bounds the probability of failing to achieve the approximation guarantee.

Approximate randomized solutions continued

We have two definitions for randomized approximations:
Fix $\epsilon \geq 0$ and $0 \leq \delta < 1$. A randomized algorithm ALG is said to (ϵ, δ) *multiplicatively approximate* f_n if for all inputs $x_1, \ldots, x_n$ we have

$$\Pr\left(\left|\frac{ALG(x_1, \ldots, x_n)}{f_n(x_1, \ldots, x_n)} - 1\right| > \epsilon\right) \leq \delta,$$

where the probability is taken over the randomness of the algorithm.
A randomized algorithm ALG is said to (ϵ, δ) *additively approximate* f_n if for all inputs $x_1, \ldots, x_n$ we have

$$\Pr\left(|ALG(x_1, \ldots, x_n) - f_n(x_1, \ldots, x_n)| > \epsilon\right) \leq \delta,$$

where again the probability is taken over the randomness of the algorithm.

Approximate randomized solutions continued

The condition for multiplicative approximation is equivalent to $P((1-\epsilon)f_n(x_1,\ldots,x_n) \leq ALG(x_1,\ldots,x_n) \leq (1+\epsilon)f_n) \geq 1-\delta$. We will mainly use multiplicative approximation more often than additive, so we shall refer to multiplicative approximation simply as approximation. Additive approximation is often used when the objective f_n is a small constant, since multiplicative approximation with small ϵ might be too much to hope for.

We are only interested in the space used and the approximation guarantee. In particular, we do not put any computational restrictions on processing x_i, although (like online algorithms) algorithms designed in this model are usually efficient in that sense. If we have $k \geq n \log |U|$ (for finite U) then an algorithm can store the entire stream. Note that with $k \geq |U| \log n$ we can record frequencies of all elements of the universe, which is often (but not always!) enough to solve a streaming problem optimally. Thus, we are interested in designing algorithms that have memory requirements sublinear in n and $|U|$, and ideally, even polylogarithmic in n and $|U|$.

Read once random bits

We have to be a little careful in the use of the random bits in our randomized algorithms since we are concerned about the amount of space.

Imagine that an algorithm has access to an infinite tape populated with random bits, but the reading head only moves in one direction. The tape is not counted towards space requirements. Thus, if an algorithm wants to generate a number of random bits, it can simply read from the random tape and this does not cost anything. However, if an algorithm needs to reuse those bits in a later computation, the algorithm has to store those bits in its memory, since it cannot rewind the tape.

Computing the ℓ^{th} Frequency Moment

Recall that given the stream $x_1, \ldots, x_n$ where each $x_i \in [m]$ the i^{th} frequency is defined as $f_i := |\{j \mid x_j = i\}|$. $F_\ell = \sum_{j=1}^{n} f_i^\ell$.

Given an error bound ϵ and confidence bound δ, the goal in the frequency moment problem is to compute an estimate f_i' such that $Prob[|f_i - f_i'| > \epsilon f_i] \leq \delta$.

- The seminal paper in this regard is by Alon, Matias and Szegedy (AMS) [1999]. AMS establish a number of results:
  1. For $\ell \geq 3$, there is an $\tilde{O}(m^{1-1/\ell})$ space algorithm. The $\tilde{O}$ notation hides factors that are polynomial in $\frac{1}{\epsilon}$ and polylogarithmic in $m, n, \frac{1}{\delta}$.
  2. For $\ell = 0$ and every $c > 2$, there is an $O(\log n)$ space algorithm computing F_0' such that $Prob[(1/c)F_0 \leq F_0' \leq cF_0$ does *not* hold$] \leq 2/c$.
  3. For $\ell = 1$, $\log n$ is obvious to exactly compute the length but an estimate can be obtained with space $O(\log \log n + 1/\epsilon)$
  4. For $\ell = 2$, they obtain space $\tilde{O}(1) = O(\frac{\log(1/\delta}{\epsilon^2})(\log n + \log m))$
  5. They also show that for all $\ell > 5$, there is a (space) lower bound of $\Omega(m^{1-5/\ell})$.

Results following AMS

- A considerable line of research followed this seminal paper. Notably settling conjectures in AMS:
- The following results apply to real as well as integral ℓ.
  1. An $\tilde{\Omega}(m^{1-2/\ell})$ space lower bound for all $\ell > 2$ (Bar Yossef et al [2002]).
  2. Indyk and Woodruff [2005] settle the space bound for $\ell > 2$ with a matching upper bound of $\tilde{O}(m^{1-2/\ell})$
- The basic idea behind these randomized approximation algorithms is to define a random variable Y whose expected value is close or equal to F_ℓ and variance is sufficiently small such that this r.v. can be calculated under the space constraint.
- We will mainly present the result for F_2.
- The quantity $F_2 := \sum_{i=1}^{n} f_i^2$ has important applications in many areas including databases and statistical analysis. In databases, F_2 is sometimes called "the skew" of the data and it is equal to the size of the result of applying a SELF-JOIN operation on a table. In statistical analysis, F_2 is used for computing the Gini coefficients.

Estimating the 2nd frequency moment

We present an algorithm that computes a $(1+\epsilon)$-approximation to F_2 with probability at least $1-\delta$. The approach is as follows:

- Define an unbiased estimator Z^2 for F_2, i.e., Z is a random variable such that $\mathbb{E}(Z^2) = F_2$.
- Compute the variance of Z^2 to bound the probability of significant deviation of Z^2 from $\mathbb{E}(Z^2)$ via a Chebychev inequality.
- Show how to compute Z^2 in a single pass using a 4-wise independent family of hash functions to reduce memory requirements.
- The variance of Z^2 turns out to be too large, so we reduce the variance by averaging several independent copies of Z: $Z_1, \ldots, Z_k$.
- Our final estimator is $Y = \frac{1}{k}\sum_{1\leq i\leq k} Z_i^2$ for a suitably chosen k that depends on the parameters ϵ and δ.

An idealized algorithm for estimating F_2

We first present an idealized algorithm for the desired estimator. The algorithm begins by sampling a random function $h : [m] \to \{-1, 1\}$. It then initializes $Z \leftarrow 0$. When x_i arrives the value of Z is updated $Z \leftarrow Z + h(x_i)$. The returned estimator is Z^2. Note that h is sampled once and for all at the beginning of the algorithm and then it is fixed for the duration of the entire stream: for example, if $h(5) = -1$ then 1 is subtracted from Z every time 5 appears in the stream.

The reason that this algorithm is idealized is because storing random h requires m bits of memory and our goal is to use only $O_{\epsilon,\delta}(\log n + \log m)$ bits of memory eventually. Later we will show how to reduce memory requirements for computing Z^2. For now, we analyze the idealized algorithm.

The idealized algorithm for estimating Z^2

Algorithm 32 The idealized algorithm for computing an unbiased estimator for F_2.

procedure IDEALIZEDUNBIASEDESTIMATOR
  $h : [m] \to \{-1, 1\}$ – a random hash function
  $Z \leftarrow 0$
  $i \leftarrow 1$
  while $i \leq n$ **do**
    $Z \leftarrow Z + h(x_i)$
  return Z^2

Lemma

$$\mathbb{E}[Z^2] = F_2.$$

Analysis of idealized Z^2 alhorithm

Proof.

The total number of appearances of i is f_i, therefore $Z = \sum_{1 \le i \le m} h(i) f_i$.

Using linearity of expectation we can show that $\mathbb{E}(Z^2) = F_2$ as follows:

$$\mathbb{E}(Z^2) = \mathbb{E}\left(\left(\sum_{1 \le i \le m} h(i) f_i\right)^2\right)$$

$$= \mathbb{E}\left(\sum_{1 \le i \le m} h(i)^2 f_i^2\right) + \mathbb{E}\left(\sum_{i \neq j} h(i) h(j) f_i f_j\right)$$

$= \sum_{1 \le i \le m} f_i^2 + \sum_{i \neq j} \mathbb{E}(h(i))\mathbb{E}(h(j)) f_i f_j = F_2$ where the third equality follows because $h(i)^2 = 1$ and $h(i), h(j)$ are independent for $i \neq j$. The last equality follows since $\mathbb{E}(h(i)) = 0$. □

Bounding the variuance of Z^2

In following the approach that has been laid out, we now to bound the variance of Z^2. This will follow from the definition of variance, and some obersvation concerning the expectations of products of the $\{h_i\}$.

Lemma

$$Var(Z^2) \leq 2F_2^2.$$

By definition,

$$Var(Z^2) = \mathbb{E}(Z^4) - \left(\mathbb{E}(Z^2)\right)^2 = \mathbb{E}\left(\left(\sum_{1 \leq i \leq m} h(i) f_i\right)^4\right) - F_2^2,$$

Bounding the variance of Z^2 continued

The main technical thing is to expand $\left(\sum_{1\le i\le m} h(i)f_i\right)^4$.

Upon expanding We get the following terms:

- $\sum_i h(i)^4 f_i^4$ only 1 such summation,
- $\sum_{i\neq j} h(i)^2 h(j)^2 f_i^2 f_j^2$ $\binom{4}{2} = 6$ such summations,
- $\sum_{i\neq j} h(i)h(j)^3 f_i f_j^3$,
- $\sum_{i\neq j\neq k} h(i)^2 h(j)h(k) f_i^2 f_j f_k$, and
- $\sum_{i\neq j\neq k\neq \ell} h(i)h(j)h(k)h(\ell) f_i f_j f_k f_\ell$.

All summations that have an odd power of $h(i), h(j), h(k)$, or $h(\ell)$ will turn into 0 after applying the expectation, because for any odd power p we have $\mathbb{E}(h(i)^p) = \mathbb{E}(h(i)) = 0$ and the expectation distributes over $+$ and $\cdot$ since $h(i), h(j), h(k), h(\ell)$ are independent provided $i \neq j \neq k \neq \ell$. Thus the only surviving terms are those that only contain even powers of $h(i), h(j), h(k), h(\ell)$; that is, the 6 quadratic terms and the one term with homogeneous degree 4.

Finishing the probabilistic analysis

We first complete the bound on the variance.

$$Var(Z^2) = \sum_i f_i^4 + 6\sum_{i<j} f_i^2 f_j^2 - F_2^2 \leq 3\sum_i f_i^4 + 6\sum_{i<j} f_i^2 f_j^2 - F_2^2$$
$$= 3\left(\sum_i f_i^4 + 2\sum_{i<j} f_i^2 f_j^2\right) - F_2^2 = 3F_2^2 - F_2^2 = 2F_2^2.$$

We can now complete the probabilistic analysis of the streaming algorithm by combining (in parallel) sufficiently many independent copies of the Z^2 estimator.

Theorem

Fix $\epsilon > 0$ and $\delta > 0$. Define $k = 2/(\epsilon^2\delta)$. Let $Z_1, \ldots, Z_k$ be i.i.d. copies of Z, as computed by the idealized Algorithm for the F_2 estimator. Define $Y = (1/k)\sum_{1\leq i\leq k} Z_i^2$. Then

$$P((1-\epsilon)F_2 \leq Y \leq (1+\epsilon)F_2) \geq 1-\delta.$$

The proof of the theorem is just an application of the Chebyshev inequality, applied to Y resulting in:

$$P(|Y - \mathbb{E}(Y)| > \epsilon F_2) \leq \frac{Var(Y)}{\epsilon^2 F_2^2} \leq \frac{\epsilon^2\delta F_2^2}{\epsilon^2 F_2^2} = \delta$$

The space used by the streaming algorithm for approximating F_2

The requirement for the random function h is that in the analysis of $\mathbb{E}[F^2]$ and $Var[F^2]$, the expectations that are needed will distribute over addition (which always holds) and multiplication (where we need independence for $h(i), h(j), h(k)$ and $h(\ell)$).

In fact, all we need is that we choose the random h from a *4-wise independent family of functions* $\mathcal{H}$. That is, $\mathcal{H}$ satsifies:
$P[h(x_1) = y_1] \wedge P[h(x_2) = y_2] \wedge P[h(x_3) = y_3] \wedge P[h(x_4) = y_4] = \frac{1}{m^4}$
for disctinct values of $x_1, x_2, x_3, x_4 \in \{1, \ldots, m\}$ and any
$y_1, y_2, y_3, y_4 \in \{-1, 1\}$

Such families $\mathcal{H}$ of functions are known to exist and each h_i can be represented in $O(\log m)$ bits. Storing each Z_i requires $O(\log n)$ bits and we have $k = \frac{2}{\epsilon^2 \delta}$ copies of the Z_i,

Streaming turnstile models

So far, we have only been considering a restricted (but the most prominent) class of streaming problems in which we are interested in some function depending on the frequency counts of elements in $[1, m] = \{1, 2, \ldots, m\}$. For such problems, there are three special models for the nature of the input to the streaming algorithm.

In each of the three models, the universe U of input elements consists of pairs (j, Δ_j) where $j \in [m]$ and $\Delta_j \in \mathbb{R}$. The idea is that there is an underlying vector S of dimension m and the stream describes how this vector changes over time.

Note: We have implicitly been using the turnstill model with $\Delta_j \in \{-1, +1\}$ for all j as determined by the hash functions h applied to j.

Initially, S starts out as an all-zero vector. Each time an element $x_i = (j_i, \Delta_i)$ arrives, the vector S is updated according to the rule $S[j_i] \leftarrow S[j_i] + \Delta_i$. To maintain small space, we will need S to collapse the information about the input; hence we will not be able to recover individual input items or the order of individual input items from S.

The three turnstile models

What distinguishes the special models from each other is the allowable values of the Δ_j.

- **General Turnstile Model.** This is the most general model of the three where Δ is allowed to be an arbitrary real number at all times in the stream. We can just refer to this as the Turnstile model.
- **Strict Turnstile Model.** In this model $S[i]$ has to be non-negative at all times during the execution of the algorithm. Thus you are allowed to decrement component $S[i]$ (corresponding to $\Delta < 0$) provided that $S[i] > 0$ and $|\Delta| \leq S[i]$.
- **Cash Register Model.** In this model Δ is restricted to be non-negative at all times in the stream

Linear sketching

The algorithm for estimating F_2 is an example of a very general class of techniques for achieving small space streaming algorithms. Linear sketching applies in the Turnstile Model, where there is an underlying vector $S \in \mathbb{R}^m$ and the input items x_i describe updates (j, Δ_j) where $j \in [m]$ to the vector S. In the linear sketching approach, the idea is to maintain the result of applying a (random) matrix M to S. The matrix M has m columns and $k \ll m$ rows. The result of applying M to S is a k-dimensional vector R, so R requires much less space to store than S directly.

More explicitly, the approach provided for estimating F_2 is typically how we deal with turnstile model problems. The rows in the sketching matrix correespond to independently drawn haahs functions, drawn from some appropriate family of hash functions. We restate the approach on the next slide to apply more generally to tunrstile problems.

Restating the turnstile approach

Suppose we are (ϵ, δ) computing some function F of the input item frequencies.

- Define an unbiased estimator X for F, i.e., X is a random variable such that $\mathbb{E}(X) = F$.
- Compute the variance of X to bound the probability of significant deviation of X from $\mathbb{E}(F)$.
- Show how to compute X in a single pass using an appropriate hash function drawn from an appropriate family $\mathcal{F}$ of hash finctions.
- If the variance of X turns out to be too large, we reduce the variance by averaging, taking the median, taking the min, etc of k independent copies $X_1, \ldots, X_k$. Each X_i is derived by randomly and independently drawing a hash function from the family $\mathcal{F}$. We compute these X_i in parallel as the elements are streaming by.
- Our final estimator is Y, is derived from these independent $\{X_i\}$ for a suitably chosen k that depends on the parameters ϵ and δ.

Other examples of sketching

In Chakrabarti's lecture notes, there are a variety of sketching based results.

Note: Chakrabarti uses $n = |U|$ whereas we use m since we reserve n to be the length of the input sequence.

Chakratbarti gives a number of sketching results for F_0 (i.e., the number of distinct elements), computing the frequencies of all elements (i.e., being able t report an approximation for f_i upon a query), and frequency moments.

He also presents some streaming algorithms beyond turnstile problems.