3

Streaming Algorithms

Great Ideas in Theoretical Computer Science
Saarland University, Summer 2014

Some Admin:

- Deadline of Problem Set 1 is 23:59, May 14 (today)!
- Students are divided into two groups for tutorials. Visit our course website and use Doodle to to choose your free slots.
- If you find some typos or like to improve the lecture notes (e.g. add more details, intuitions), send me an email to get .tex file :-)

We live in an era of "Big Data": science, engineering, and technology are producing increasingly large data streams every second. Looking at social networks, electronic commerce, astronomical and biological data, the big data phenomenon appears everywhere nowadays, and presents opportunities and challenges for computer scientists and mathematicians. Comparing with traditional algorithms, several issues need to be considered: A massive data set is too big to be stored; even an $O(n^2)$-time algorithm is too slow; data may change over time, and algorithms need to cope with dynamic changes of the data. Hence streaming, dynamic and distributed algorithms are needed for analyzing big data.

To illuminate the study of streaming algorithms, let us look at a few examples. Our first example comes from network analysis. Any big website nowadays receives millions of visits every minute, and very frequent visit per second from the same IP address are probably generated automatically by computers and due to hackers. In this scenario, a webmaster needs to efficiently detect and block those IPs which generate very frequent visits per second. Here, instead of storing all IP record history which are expensive, we need an *efficient* algorithm that detect *most* such IPs.

The second example is to quickly obtain statistical information of a massive data set as a preprocessing step to speed-up the runtime of algorithms. Assume that you want to solve a problem in a massive data set (database, graph, etc.), and solving the problem is quite time-consuming. However, you can get some information by quickly scanning the whole data. For instance, for graph algorithms you want to roughly know if the graph is dense or sparse, if the graph is connected, and so on. This information usually provides statistics of the massive data set, e.g. the frequency of items appearing in the data set, or properties of the set (typically

graphs), e.g. how well connected the graph is. Although such statistics is not precise for most cases, this approximate information is usually used to speedup subsequent processes.

Thees examples motivate the study of streaming algorithms, in which algorithms have access to the input in a certain order, algorithm's space is limited, and fast approximation is required.

3.1 Model & Basic Techniques

A data stream is a sequence of data

$$\mathcal{S} = s_1, s_2, \ldots, s_m, \ldots,$$

where each item s_i belongs to the universe U, where $|U| = n$. A data streaming algorithm $\mathcal{A}$ takes $\mathcal{S}$ as input and computes some function f of stream $\mathcal{S}$. Moreover, algorithm $\mathcal{A}$ has access the input in a "streaming fashion", i.e. algorithm $\mathcal{A}$ cannot read the input in another order and for most cases $\mathcal{A}$ can only read the data once.

Depending on how items in U are expressed in S, there are two typical models [20]:

1. **Cash Register Model:** Each item s_i in stream S is an item of U. Different items come in an arbitrary order.
2. **Turnstile Model:** In this model we have a multi-set D, and $D = \emptyset$ initially. Every coming item is associated with one of two special symbols in order to indicate the dynamic changes of the data set. For instance, every item in S can be a pair (x, U), and x is added into D if U is "$+$", and x is deleted from D if U is "$-$". The turnstile model captures most practical situations that the dataset may change over time. Recent references also call this model *dynamic streams*.

Typically, the size of the universe U is a huge number, and storing the whole data and computing the exact value of f is computationally expensive. Hence the goal is to design *sublinear*-space algorithms which give a good approximation of f for most cases. Formally, our goal is to design an algorithm $\mathcal{A}$ with the following two constraints:

- *Space:* Space of algorithm $\mathcal{A}$ is $O(\mathrm{poly}\log(n))$.
- *Quick update time*: For every coming item in the stream, quick update time is desired.
- *Approximate:* For confidence parameter $\varepsilon > 0$ and approximation parameter $\delta > 0$, the output of $\mathcal{A}$ achieves a $(1 \pm \varepsilon)$-approximation of the exact value $f(\mathcal{S})$ with probability at least $1 - \delta$. That is, the output $f^\star(\mathcal{S})$ satisfies

$$\mathbf{Pr}\left[\, f^\star(\mathcal{S}) \in [(1-\varepsilon)f(\mathcal{S}), (1+\varepsilon)f(\mathcal{S})]\,\right] \geq 1 - \delta.$$

Remark 3.1. *Another widely studied model is called* the semi-streaming model. *In the semi-streaming model algorithms run in $O(n \cdot \mathrm{poly}\log n)$ space, and typically graph streaming algorithms are studied in the semi-streaming model. Note that $O(n \cdot \mathrm{poly}\log n)$ space allows an algorithm to store all nodes which take $O(n \log n)$ bits of space, but storing the whole graph is impossible for dense graphs.*

Basic Techniques. Sampling and sketching are two basic techniques for designing streaming algorithms.

The idea behind sampling is easy to state. Every arriving item is retained with a certain probability, and only a subset of the data is retained for further computation. Sampling is easy to implement, and has wide applications.

Sketching is the other technique for designing streaming algorithms. A sketch-based algorithm $\mathcal{A}$ creates a compact synopsis of the data which has been observed, and the size of the synopsis is vastly smaller than the full observed data. Each update observed in the stream potentially causes this synopsis to be updated, so that the synopsis can be used to approximate certain functions of the data seen so far.

Figure 3.1 shows one example of sketches. Durand and Flajolet [12] proposed the LogLog sketch in 2003, which is used to estimate the number of distinct items in a data set. Based on the LogLog sketch, they condense the whole of Shakespear's works to a table of 256 "small bytes" of 4 bits each. The estimate of the number of distinct words by the LogLog sketch here is 30897, while the true answer is 28239. I.e., a relative error is $+9.4\%$.

```
ghfffghfghgghgggggghghheehfhfhhgghghghhfgffffhhhiigfhhffgfiihfhhh
igigighfgihfffghigihghigfhhgeegeghgghhhgghhfhidiigihighihehhhfgg
hfgighigffghdieghhhggghhfghhfiiheffghghihifgggffihgihfggighgiiif
fjgfgjhhjiifhjgehgghfhhfhjhiggghghihigghhihihgiighgfhlgjfgjjjmfl
```

Figure 3.1: The table above condenses the whole of Shakespear's works. The estimate of the number of distinct words by the LogLog sketch here is 30897, while the true answer is 28239. I.e., a relative error is $+9.4\%$.

Notations. We list basic notations used in this lecture. For any integer M, let $[M] \triangleq \{0, \ldots, M-1\}$. For any set X, $x \sim_R X$ stands for choosing x uniformly at random from set X.

3.2 Hash Functions

Most data streaming algorithms rely on constructions of a class of functions, called *hash functions*, that have found a surprising large number of applications. The basic idea behind using hash functions is to make input date have certain independence through some easy-to-compute function. Formally, we want to construct a family of functions $\mathcal{H} = \{h \mid h : N \mapsto M\}$ such that (1) every function $h \in \mathcal{H}$ is easy to represent; (2) for any $x \in N$, $h(x)$ is easy to evaluate; (3) for any set S of small cardinality, hashed values of items in S have small collisions.

Recall that a set of random variables $X_1, \ldots, X_n$ is k-wise independent if, for any index set $J \subset \{1, \ldots, n\}$ with $|J| \leq k$ and for any values $x_i, i \in J$, it holds that

$$\mathbf{Pr}\left[\bigcap_{i \in J} X_i = x_i\right] = \prod_{i \in J} \mathbf{Pr}\left[X_i = x_i\right].$$

In particular, the random variables $X_1, \ldots, X_n$ are said to be pairwise independent if they are

2-wise independent. That is, for any i, j and values x, y, it holds that

$$\mathbf{Pr}\left[\,(X_i = x) \cap (X_j = y)\,\right] = \mathbf{Pr}\left[\,X_i = x\,\right]\mathbf{Pr}\left[\,X_j = y\,\right].$$

Hash functions with similar properties are called k-wise independent and pairwise independent hash functions, respectively.

Definition 3.2 (Pairwise Independent Hash Functions)**.** *A family of functions $\mathcal{H} = \{h \mid h : N \mapsto M\}$ is pairwise independent if the following two conditions hold:*

1. *$\forall x \in N$, the random variable $h(x)$ is uniformly distributed in M, where $h \sim_R \mathcal{H}$,*
2. *$\forall x_1 \neq x_2 \in N$, the random variables $h(x_1)$ and $h(x_2)$ are independent, where $h \sim_R \mathcal{H}$.*

These two conditions state that for any different $x_1 \neq x_2 \in N$, and any $y_1, y_2 \in M$, it holds that

$$\mathbf{Pr}_{h\sim_R\mathcal{H}}\left[\,h(x_1) = y_1 \cap h(x_2) = y_2\,\right] = \frac{1}{|M|^2},$$

where the probability above is over all random choices of a function from $\mathcal{H}$.

Theorem 3.3. *Let p be a prime number, and let $h_{a,b} = (ax + b) \bmod p$. Define*

$$\mathcal{H} = \{h_{a,b} \mid 0 \leq a, b \leq p - 1\}.$$

Then $\mathcal{H}$ is a family of pairwise independent hash functions.

Proof. We need to show that, for any two different $x_1, x_2 \in N$ and any two y_1, y_2, it holds that

$$\mathbf{Pr}_{h\sim_R\mathcal{H}}\left[\,h(x_1) = y_1 \cap h(x_2) = y_2\,\right] = \frac{1}{p^2}. \tag{3.1}$$

For any a, b, the conditions that $h_{a,b}(x_1) = y_1$ and $h_{a,b}(x_2) = y_2$ yield two equations $ax_1 + b = y_1 \bmod p$, and $ax_2 + b = y_2 \bmod p$. Such system has a unique solution of a and b, out of p^2 possible pairs of (a, b). Hence (3.1) holds, and $\mathcal{H}$ is a family of pairwise independent hash functions. □

3.3 Counting Distinct Elements

Our first problem is to approximate the F_p-norm of items in a stream. We first define the F_p-norm.

Definition 3.4 (F_p-norm)**.** *Let S be a multi-set, where every item i of S is in $[N]$. Let m_i be the number of occurrences of item i in set S. Then the F_p-norm of set S is defined by*

$$F_p \triangleq \sum_{i\in[N]} |m_i|^p,$$

where 0^p is set to be 0.

By definition, the F_0-norm of set S is the number of distinct items in S, and the F_1-norm of S is the number of items in S. In this lecture we focus on approximating F_0.

Problem 3.5. *Let $\mathcal{S}$ be a data stream representing a multi set S. Items of S arrive consecutively and every item $s_i \in [n]$. Design a streaming algorithm to (ε, δ)-approximate the F_0-norm of set S.*

3.3.1 The AMS Algorithm

Algorithm. The first algorithm for approximating F_0 is by Noga Alon, Yossi Matias, and Mario Szegedy [4], and most references use AMS to name their algorithm. The intuition behind their algorithm is quite simple. Assume that we have seen sufficiently many numbers, and these numbers are uniformly distributed. We look at the binary expression $\text{Binary}(x)$ of every item x, and we expect that for one out of d distinct items $\text{Binary}(x)$ ends with d consecutive zeros. More generally, let

$$\rho(x) \triangleq \max_i \{i : x \bmod 2^i = 0\}$$

be the number of zeros that $\text{Binary}(x)$ ends with, and we have the following observation:

- If $\rho(x) = 1$ for any x, then it is likely that the number of distinct integers is $2^1 = 2$.
- If $\rho(x) = 2$ for any x, then it is likely that the number of distinct integers is $2^2 = 4$.
- If $\rho(x) = 3$ for any x, then it is likely that the number of distinct integers is $2^3 = 8$.
- If $\rho(x) = r$ for any x, then it is likely that the number of distinct integers is 2^r.

To implement this idea, we use a hash function h so that, after applying h, all items in $\mathcal{S}$ are uniformly distributed, and on average one out of F_0 distinct numbers hit $\rho(h(x)) \geq \log F_0$. Hence the maximum value of $\rho(h(x))$ over all items x in the stream could give us a good approximation of the number of distinct items.

Algorithm 3.1 An Algorithm For Approximating F_0

```
Choose a random function h : [n] → [n] from a family of pairwise independent hash functions;
z ← 0;
while an item x arrives do
    if ρ(h(x)) > z then
        z ← ρ(h(x));
Return 2^{z+1/2}
```

Analysis. Now we analyze Algorithm 3.1. Our goal is to prove the following statement.

Theorem 3.6. *By running $\Theta(\log(1/\delta))$ independent copies of Algorithm 3.1 and returning the medium value, we achieve an $(O(1), \delta)$-approximation of the number of distinct items in $\mathcal{S}$.*

Proof. Let $X_{r,j}$ be an indicator random variable such that $X_{r,j} = 1$ iff $\rho(h(j)) \geq r$. Let $Y_r = \sum_{j \in \mathcal{S}} X_{r,j}$ be the number of items j such that $\rho(h(j)) \geq r$, and $z^\star$ be the value of z when the algorithm terminates. Hence, $Y_r > 0$ iff $z^\star \geq r$.

Since h is pairwise independent, $h(j)$ is uniformly distributed, and

$$\mathbf{E}[X_{r,j}] = \mathbf{Pr}\,[\,\rho(h(j)) \geq r\,] = \mathbf{Pr}\,[\,h(x) \bmod 2^r = 0\,] = \frac{1}{2^r}.$$

By linearity of expectation, we have

$$\mathbf{E}[Y_r] = \sum_{j \in \mathcal{S}} \mathbf{E}[X_{r,j}] = \frac{F_0}{2^r},$$

and

$$\mathbf{Var}[Y_r] = \sum_{j \in \mathcal{S}} \mathbf{Var}[X_{r,j}] \leq \sum_{j \in \mathcal{S}} \mathbf{E}[X_{r,j}^2] = \sum_{j \in \mathcal{S}} \mathbf{E}[X_{r,j}] = \frac{F_0}{2^r}.$$

By using the Markov's Inequality and Chebyshev's Inequality, we have

$$\mathbf{Pr}\,[\,Y_r > 0\,] = \mathbf{Pr}\,[\,Y_r \geq 1\,] \leq \frac{\mathbf{E}[Y_r]}{1} = \frac{F_0}{2^r},$$

and

$$\mathbf{Pr}\,[\,Y_r = 0\,] \leq \mathbf{Pr}\,[\,|Y_r - \mathbf{E}[Y_r]| \geq F_0/2^r\,] \leq \frac{\mathbf{Var}[Y_r]}{(F_0/2^r)^2} \leq \frac{2^r}{F_0}.$$

Let $F^\star$ be the output of the algorithm. Then $F^\star = 2^{z^\star + 1/2}$. Let a be the smallest integer such that $2^{a+1/2} \geq 3F_0$. Then

$$\mathbf{Pr}\,[\,F^\star \geq 3F_0\,] = \mathbf{Pr}\,[\,z^\star \geq a\,] = \mathbf{Pr}\,[\,Y_a > 0\,] \leq \frac{F_0}{2^a} \leq \frac{\sqrt{2}}{3}.$$

Similarly, let b be the largest integer such that $2^{b+1/2} \leq F_0/3$. We have

$$\mathbf{Pr}\,[\,F^\star \leq F_0/3\,] = \mathbf{Pr}\,[\,z^\star \leq b\,] = \mathbf{Pr}\,[\,Y_{b+1} = 0\,] \leq \frac{2^{b+1}}{F_0} \leq \frac{\sqrt{2}}{3}.$$

Running $k = \Theta(\log(1/\delta))$ independent copies of Algorithm 3.1 above and returning the median value, we can make the two probabilities above at most δ. This gives an $(O(1), \delta)$-approximation of the number of distinct items over the stream. □

3.3.2 The BJKST Algorithm

Our second algorithm for approximating F_0 is a simplified version of the algorithm by Bar-Yossef et al. [5]. In contrast to the AMS algorithm, the BJKST algorithm uses a set to keep the sampled items. By running $\Theta(\log(1/\delta))$ independent copies in parallel and returning the medium of these outputs, the BJKST algorithm (ε, δ)-approximates the F_0-norm of the multiset S.

The basic idea behind the sampling scheme of the BJKST algorithm is as follows:

1. Let B be a set that is used to retain sampled items, and $B = \emptyset$ initially. The size of B is $O(1/\varepsilon^2)$ and only depends on approximation parameter ε.

2. The initial sampling probability is 1, i.e. the algorithm keeps all items seen so far in B.
3. When the set B becomes full, shrink B by removing about half items and from then on the sample probability becomes smaller.
4. In the end the number of items in B and the current sampling probability are used to approximate the F_0-norm.

A simplified version of the BJKST algorithm is presented in Algorithm 3.2, where c is a constant.

Algorithm 3.2 The BJKST Algorithm (Simplified Version)

Choose a random function $h : [n] \to [n]$ from a family of pairwise independent hash functions
$z \leftarrow 0$ ▷ z is the index of the current level
$B \leftarrow \emptyset$ ▷ Set B keeps sampled items
while an item x arrives **do**
  if $\rho(h(x)) \geq z$ **then**
    $B \leftarrow B \cup \{(x, \rho(h(x)))\}$
    while $|B| \geq c/\varepsilon^2$ **do** ▷ Set B becomes full
      $z \leftarrow z + 1$ ▷ Increase the level
      shrink B by removing all $(x, \rho(h(x)))$ with $\rho(h(x)) < z$
Return $|B| \cdot 2^z$

3.3.3 Indyk Algorithm

Algorithm 3.1 and Algorithm 3.2 work in the cash register model, and cannot handle deletions of items. We next show that F_0-norm of a set S can be estimated in dynamic streams. This algorithm, due to Piotr Indyk [13], presents beautiful applications of the so-called *stable distributions* in designing streaming algorithms.

> **Problem 3.7.** *Let $\mathcal{S}$ be a stream consisting of pairs of the form (s_i, U_i), where $s_i \in [n]$ and $U_i = +/-$ represents dynamic changes of s_i. Design a data streaming algorithm that, for any ε and δ, (ε, δ)-approximates the F_0-norm of $\mathcal{S}$.*

Stable Distributions. An important property of Gaussian random variables is that the sum of two of them is again a Gaussian random variable. One consequence of this fact is that if X is Gaussian, then for independent copies of X, expressed by X_1 and X_2, and any $a, b \in \mathbb{N}$, $aX_1 + bX_2$ has the same distribution as $cX + d$ for some positive c and some $d \in \mathbb{R}$, i.e. the shape of X is preserved. One class of distributions with similar properties was characterized by Paul Lévy in 1920s when he studies sums of independent identically distributed terms.

Informally, a random variable is *stable* if for independent copies of a random variable X, expressed by X_1, X_2, there are positive constants c, d such that $aX_1 + bX_2$ has the same

distribution as $cX + d$. A generalization of this stable property allows us to define p-stable distributions for parameter $p \in (0, 2]$.

Definition 3.8 (Stable Distribution)**.** *Let $p \in (0, 2]$. A distribution D over $\mathbb{R}$ is called p-stable, if for any $a_1, \ldots, a_n \in \mathbb{R}$, and independent and identically distributed random variables $X_1, \ldots, X_n$ with distribution D, the random variable $\sum_i a_i X_i$ has the same distribution as the random variable $(\sum_i |a_i|^p)^{1/p} X$, where X is a random variable with distribution D.*

Remark 3.9. *Such distributions exist for any $p \in (0, 2]$. However, except for $p = 1/2, 1, 2$, we have not found closed formulae of their density functions and their distributions are defined with respect to their characteristic functions. That is, distribution $\mathcal{D}$ is p-stable if a random variable X with distribution $\mathcal{D}_p$ satisfies $\mathbf{E}[e^{itX}] = e^{-|t|^p}$. When $p = 1/2, 1$ or 2, closed formulae of the density functions are known. Actually, they are Lévy distribution, Cauchy distribution, and Gaussian distributions, respectively.*

Although closed formulae of density functions of general p-stable distributions are unknown, such distribution for any $p \in (0, 2]$ can be generated easily as follows [7]: (i) Pick Θ uniformly at random from $[-\pi/2, \pi/2]$, and pick r uniformly at random from $[0, 1]$. (ii) Output

$$\frac{\sin(p\Theta)}{\cos^{1/p}(\Theta)} \cdot \left(\frac{\cos(\Theta(1-p))}{-\ln r} \right)^{(1-p)/p}. \tag{3.2}$$

For detailed discussion of stable distributions, see [3]. Reference [8] gives a short and interesting introduction to stable distributions in designing streaming algorithms.

Indyk Algorithm. Algorithm 3.3 gives an *idealized* algorithm for estimating F_p-norm for any $p \in (0, 2]$. The basic setting is as follows: Assume that every item in the stream is in $[n]$, and we want to achieve an (ε, δ)-approximation of the F_p-norm. Let us further assume that we have matrix $\mathbf{M}$ of $k \triangleq \Theta(\varepsilon^{-2} \log(1/\delta))$ rows and n columns, where every item in $\mathbf{M}$ is a random variable drawn from a p-stable distribution, generated by (3.2). Given matrix $\mathbf{M}$, Algorithm 3.3 keeps a vector $\mathbf{z} \in \mathbb{R}^k$ which can be expressed by a linear combination of columns of matrix $\mathbf{M}$.

Let us prove that this idealized algorithm gives an (ε, δ)-approximation of the F_p-norm.

Lemma 3.10. *The F_0 norm of multi-set S can be approximated by Algorithm 3.3 for choosing sufficiently small p, assuming that we have an upper bound K of the number of occurrences of every item in the stream.*

Proof. Let $\mathbf{M}_{\cdot,j}$ be the jth column of matrix $\mathbf{M}$. Then the vector $\mathbf{z}$ in Algorithm 3.3 can be written as

$$\mathbf{z} = \sum_{j \in S} m_j \cdot \mathbf{M}_{\cdot,j},$$

where m_j is the number of occurrences of item j. Hence $\mathbf{z}_i = \sum_{j \in S} m_j \cdot \mathbf{M}_{i,j}$. Since $\mathbf{M}_{i,j}$ is drawn from the p-stable distribution, $\mathbf{z}_i = \sum_{j \in S} m_j \cdot \mathbf{M}_{i,j}$ has the same distribution as

Algorithm 3.3 Approximating F_p-norm in a Turnstile Stream (An Idealized Algorithm)

```
while 1 ≤ i ≤ k do
    z_i ← 0
while an operation arrives do
    if item j is added then
        for i ← 1, k do
            z_i ← z_i + M[i, j]
    if item j is deleted then
        for i ← 1, k do
            z_i ← z_i − M[i, j]
    if F_p-norm is asked then
        Return medium_{1≤i≤k}{|z_i|^p} · scalefactor(p)
```

$\left(\sum_{j\in S}|m_j|^p\right)^{1/p}$. Note that $\sum_{j\in S}|m_j|^p = F_p$, and

$$F_0 = \sum_{j\in S}|m_j|^0 \le \sum_{j\in S}|m_j|^p = F_p \le K^p \cdot \sum_{j\in S}|m_j|^0,$$

where the last inequality holds by the fact that $|m_j| \le K$ for all j. Hence by setting $p \le \log(1+\varepsilon)/\log K \approx \varepsilon/\log K$, we have that

$$F_p \le (1+\varepsilon)\sum_i |m_i|^0 = (1+\varepsilon)\cdot F_0. \qquad \square$$

This *idealized* algorithm relies on matrix $\mathbf{M}$ of size $k \times n$, and for every occurrence of item i, the algorithm needs the ith column of matrix $\mathbf{M}$. However, sublinear space cannot store the whole matrix! So we need an effective way to generate this random matrix such that (1) every entry of $\mathbf{M}$ is random and generated according to the p-stable distribution, and (2) each column can be reconstructed when necessary.

To construct such matrix $\mathbf{M}$, we use Nisan's pseudorandom generators[1]. Specifically, when the column indexed by i is required, Nisan's generator takes i as the input and, together with the original see, outputs a sequence of pseudorandom numbers. Based on two consecutive pseudorandom numbers, we use (3.2) to generate one random variable.

The theorem below gives the space complexity of Algorithm 3.3. See [9, 13] for the correctness proof.

Theorem 3.11. *For any parameters ε, δ, there is an algorithm (ε, δ)-approximates the number of distinct elements in a turnstile stream. The algorithm needs $O(\varepsilon^{-2}\log n \log(1/\delta))$ bits of space. The update time for every coming item is $O(\varepsilon^{-2}\log(1/\delta))$.*

[1] We will discuss Nisan's pseudorandom generators (PRG) in later lectures, but at the moment you can consider a PRG as a deterministic function $f : \{0,1\}^n \mapsto \{0,1\}^m$, where m is much larger than n, such that if we use a random number x as input, $f(x)$ looks almost random. I.e., f extends a truly random binary sequence to an "almost" random binary sequence.

3.4 Frequency Estimation

A second problem that we study is to estimate the frequency of any item x, i.e. the number of occurrences of any item x. The basic setting is as follows: Let S be a multi-set, and is empty initially. The data stream consists of a sequence of update operations to set S, and each operation is one of the following three forms:

- INSERT(S, x), performing the operation $S \leftarrow S \cup \{x\}$;
- DELETE(S, x), performing the operation $S \leftarrow S \setminus \{x\}$;
- QUERY(S, x), querying the number of occurrences of x in the multiset S.

Problem 3.12 (Frequency Estimation)**.** *Design a streaming algorithm to support* INSERT*,*DELETE*, and* QUERY *operations, and the algorithm approximates the frequency of every item in* S*.*

Cormode and Muthukrishnan [10] introduced the Count-Min Sketch for this frequency estimation problem. Count-Min Sketch consists of a fixed array C of counters of width w and depth d, as shown in Figure 3.2. These counters are all initialized to be zero. Each row is associated to a pairwise hash function h_i, where each h_i maps an element from U to $\{1, \ldots, w\}$. Algorithm 3.4 gives a formal description of update procedures of the Count-Min sketch.

Algorithm 3.4 Frequency Estimation

$d = \lceil \log(1/\delta) \rceil$
$w = \lceil \mathrm{e}/\varepsilon \rceil$
while an operation arrives **do**
  if Insert(S, x) **then**
    for $j \leftarrow 1, d$ **do**
      $C[j, h_j(x)] \leftarrow C[j, h_j(x)] + 1$
  if Delete(S, x) **then**
    for $j \leftarrow 1, d$ **do**
      $C[j, h_j(x)] \leftarrow C[j, h_j(x)] - 1$
  if the number of occurrence of x is asked **then**
    Return $\widehat{m_x} \triangleq \min_{1 \le j \le d} C[j, h_j(x)]$

Choosing w and d. For given parameters ε and δ, the width and height of Count-Min sketch is set to be $w \triangleq \lceil \mathrm{e}/\varepsilon \rceil$ and $d \triangleq \lceil \ln(1/\delta) \rceil$. Hence for constant ε and δ, the sketch only consists of constant number of counters. Note that the size of the Count-Min sketch only depends on the accuracy of the approximation, and independent of the size of the universe.

Analysis. The following theorem shows that the Count-Min Sketch can approximate the number of occurrences of any item with high probability.

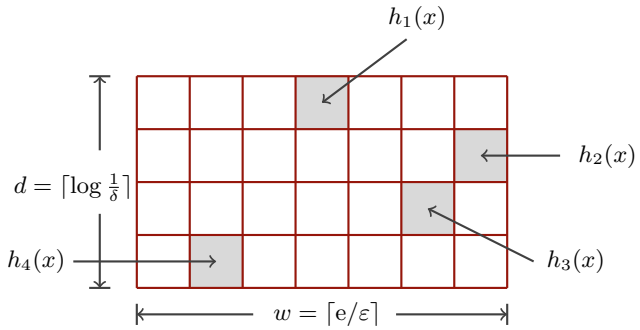

Figure 3.2: Count-Min Sketch.

Theorem 3.13. *The estimator $\widehat{m_i}$ has the following property: $\widehat{m_i} \geq m_i$, and with probability at least $1 - \delta$, $\widehat{m_i} \leq m_i + \varepsilon \cdot F_1$, where F_1 is the first-moment of the multi-set S.*

Proof. Clearly for any $i \in [n]$ and $1 \leq j \leq d$, it holds that $h_j(i) \geq m_i$ and hence $\widehat{m_i} \geq m_i$. So it suffices to prove the second statement. Let $X_{i,j}$ be the number of items $y \in [n] \setminus \{i\}$ satisfying $h_j(i) = h_j(y)$. Then $C[j, h_j(i)] = m_i + X_{i,j}$. Since different hash functions are pairwise independent, we have

$$\mathbf{Pr}\left[\, h_j(i) = h_j(y) \,\right] \leq \frac{1}{w} \leq \frac{\varepsilon}{\mathrm{e}},$$

and $\mathbf{E}[X_{i,j}] \leq \frac{\varepsilon}{\mathrm{e}} \cdot F_1$. By Markov's inequality we have

$$\begin{aligned}
\mathbf{Pr}\left[\, \widehat{m_i} > m_i + \varepsilon \cdot F_1 \,\right] &= \mathbf{Pr}\left[\, \forall j : C[j, h_j(i)] > m_i + \varepsilon \cdot F_1 \,\right] \\
&= \mathbf{Pr}\left[\, \forall j : m_i + X_{i,j} > m_i + \varepsilon \cdot F_1 \,\right] \\
&= \mathbf{Pr}\left[\, \forall j : X_{i,j} > \varepsilon \cdot F_1 \,\right] \\
&\leq \mathbf{Pr}\left[\, \forall j : X_{i,j} > \mathrm{e} \cdot \mathbf{E}[X_{i,j}] \,\right] \leq \mathrm{e}^{-d} \leq \delta.
\end{aligned}$$

□

3.5 Other Problems in Data Streams

We discussed two basic problems in the lecture, these two problems appeared in early references in the community of streaming algorithms. This line of study over the past years has achieved a number of exciting results. Here we select a few problems for which data streaming algorithms work well:

- Approximate the number of triangles in a graph [6, 16]. The stream consists of edges of an underlying graph G, the goal is to approximately count the number T_3 of triangles in G. This number relates the clustering coefficient, and the connectivity coefficient in a network, and can be used to analyze topologies of networks. Reference [14, 17] give further generalization of this problem for counting arbitrary subgraphs.

- Spectral Sparsification of a graph. The stream consists of edges of an underlying undirected graph G, and the goal is to output a subgraph H of G with suitable weights on edges of H, such that all spectral properties between graph G and H are preserved. Another notable property of graph H is that, for any vertex subset S, the cut value between S and $V \setminus S$ in H and G are approximately preserved. Spectral sparsification is widely used as a subroutine to solve other graph problems, and problems in solving linear systems. Kelner and Levin [15] give the first data streaming algorithm for constructing spectral sparsification, and their algorithm works in the semi-streaming setting. It is open for constructing spectral sparsifications in the dynamic semi-streaming setting.
- Estimating PageRank in graph streams [21].

3.6 Further Reference

There are excellent surveys in data streams. Muthukrishnan [20] gives a general introduction of streaming algorithms and discusses basic algorithmic techniques in designing streaming algorithms. Cormode and Muthukrishnan [11] give the survey of various applications of the Count-Min sketch. McGregor [18] gives a survey for graph stream algorithms.

There are also some online resources. The website [1] gives a summary, applications, and implementations of the Count-Min Sketch. The website [2] maintains a list of open questions in data streams which were proposed in the past workshops on streaming algorithms.

For a general introduction to Hash functions, we refer the reader to [19].

References

[1] Count-Min Sketch & Its Applications. https://sites.google.com/site/countminsketch/.

[2] Open questions in data streams. http://sublinear.info/index.php?title=Main_Page.

[3] Stable distributions. http://academic2.american.edu/~jpnolan/stable/stable.html.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[5] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *6th Annual European Symposium (ESA'02)*, pages 1–10, 2002.

[6] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *25th Symposium on Principles of Database Systems (PODS'06)*, pages 253–262, 2006.

[7] J. M. Chambers, C. L. Mallows, and B. W. Stuck. A method for simulating stable random varialbes. *J. Amer. Statist. Assoc.*, 71:340–344, 1976.

[8] Graham Cormode. Stable distributions for stream computations: It is as easy as 0,1,2. In *In Workshop on Management and Processing of Massive Data Streams, at FCRC*, 2003.

[9] Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). In *28th International Conference on Very Large Databases (VLDB'02)*, pages 335–345, 2002.

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

[11] Graham Cormode and S. Muthu Muthukrishnan. Approximating data with the count-min sketch. *IEEE Software*, 29(1):64–69, 2012.

[12] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In *11th International Workshop on Randomization and Computation (RANDOM'03)*, pages 605–617, 2003.

[13] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[14] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *39th International Colloquium on Automata, Languages, and Programming (ICALP'12)*, pages 598–609, 2012.

[15] Jonathan A. Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. *Theory Comput. Syst.*, 53(2):243–262, 2013.

[16] Konstantin Kutzkov and Rasmus Pagh. Triangle counting in dynamic graph streams. *CoRR*, abs/1404.4696, 2014.

[17] Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. Approximate counting of cycles in streams. In *19th International Workshop on Randomization and Computation (RANDOM'11)*, pages 677–688, 2011.

[18] Andrew McGregor. Graph stream algorithms: A survey. http://people.cs.umass.edu/~mcgregor/papers/13-graphsurvey.pdf.

[19] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[20] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[21] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. *J. ACM*, 58(3):13, 2011.