Lecture 5: Data Streaming Algorithms

Lecturer: Kurt Mehlhorn & He Sun

In the data stream scenario, the input arrive rapidly in an arbitrary order, and there is only limited space to store some information of the input. Algorithms with limited space need to compute some function of the input data. To illustrate this setting, we give some examples:

- Imagine that you are a webmaster of a big website. You receive thousands of clicks (IPs) every minute, and very frequent visits per minute are probably due to hackers. You need to detect and block these IPs, without storing millions of IPs per hour.
- 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 scan the whole data set quickly and get some information. This information usually gives some statistical data 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. The information obtained by scanning the dataset once can be used to speedup subsequent processes.

1 Introduction

Model. A data stream is a sequence of data

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

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

Depending on how items in U are expressed in S, there are two typical models [12]:

1. **Cash Register Model:** Here each item s_i in stream S represents an element in U. Different items come in an arbitrary order.

2. **Turnstile Model:** In this model we have a dataset D, and $D = \emptyset$ initially. Each item in S is a pair $s_i = (j, U_j)$, and we add j into D if U_j is "+", and delete j from D if U_j is "−". The turnstile model represents dynamic changes of the dataset, and captures most practical situations that the dataset may change over time.

Typically, the size of the universe, n, is a huge number, and storing the whole data set in the memory and computing a function f is infeasible. Hence we need to design *sub-linear*-space algorithms which give a good approximation of f for most cases. Formally, our goal is to design algorithms such that

- The working space is $O(\text{poly}\log(n))$.
- For confidence parameter $\varepsilon > 0$ and approximation parameter $\delta > 0$, the output of the algorithm achieves a $(1 \pm \varepsilon)$-approximation of the exact value $f(U)$ with probability at least $1 - \delta$. That is, the output $f^\star(U)$ satisfies

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

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

Most sampling-based algorithms follow the same framework: Algorithm A chooses every coming item ℓ with a certain probability. If item ℓ is sampled, then algorithm A puts ℓ into the memory $M[A]$, otherwise item ℓ is discarded. Depending on different situations, algorithm A may discard some items from $M[A]$ after item ℓ is added into $M[A]$. For every query of the data set, algorithm A computes some function f only based on the current set $M[A]$.

Sketching is the other basic approach for streaming algorithms. In contrast to sampling techniques, a sketch-based algorithm A uses the working space $M[A]$ to maintain some data structure, e.g. an array, or a table, and we call this data structure a *sketch*. Usually the data recorded in the sketch represents some statistical information of the data set S, and is quite different from what you see in the stream.

The following is an example of sketches. Durand and Flajolet [10] proposed one sketch, called `LogLog`, to count 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
```

2 Counting Distinct Elements

The first problem people study in the streaming setting is to approximate the frequency moments of a data set. Let $\mathcal{S}$ be a sequence of items s_i, where each $s_i \in [n] \triangleq \{0, \ldots, n-1\}$. Let

$$m_i \triangleq |\{j : s_j = i\}|$$

be the number of occurrences of i in $\mathcal{S}$. We define the k-th moment of $\mathcal{S}$ as

$$F_k \triangleq \sum_i m_i^k$$

where 0^0 is defined to be 0. By definition we have:

- F_1 is the number of items in $\mathcal{S}$.
- F_0 is the number of distinct items in $\mathcal{S}$.

2.1 The AMS Algorithm

We recall the following definition of pairwise independent hash functions.

Definition 1 (Family of pairwise independent hash functions)**.** *A set $\mathcal{H}$ of functions $f : D \mapsto R$ is called a family of pairwise independent hash functions if for different x_1, x_2, and (not necessarily distinct) y_1, y_2, it holds that*

$$\mathbf{Pr}_{h\sim_u\mathcal{H}}[h(x_1) = y_1 \wedge h(x_2) = y_2] = \frac{1}{|R|^2},$$

where $h \sim_u \mathcal{H}$ expresses that h is chosen uniformly at random from $\mathcal{H}$.

An easy way to get a pairwise independent hash function is as follows: (1) Choose a prime number p. (2) Choose a randomly from $\{1, \ldots, p-1\}$, and choose b randomly from $\{0, \ldots, p-1\}$. (3) Define $h(x) \triangleq (ax + b) \bmod p$.

Algorithm. The first algorithm for approximating F_0 is due to Alon, Matias, and Szegedy [4], and most references use AMS to name their algorithm. We first define

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

The intuition of the AMS algorithm is that, after applying a hash function h, all items in $\mathcal{S}$ are uniformly distributed, and on average one out of F_0 distinct numbers hit $\rho(h(x)) \geq \log F_0$, and thus 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 1 An Algorithm For Approximating F_0

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

Analysis. Now we analyze Algorithm 1. Our goal is to prove the following statement.

Theorem 2. *By running $\Theta(\log(1/\delta))$ independent copies of Algorithm 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 $Y_r = \sum_{j\in\mathcal{S}} X_{r,j}$, and $z^\star$ be the value of z when the algorithm terminates. Hence, $Y_r > 0$ iff $z^\star \geq r$.

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

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

Hence by the linearity of expectations, 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 α be the smallest integer such that $2^{\alpha+1/2} \geq 3F_0$. Then

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

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 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 F_0 over the stream. □

2.2 The BJKST Algorithm

The following is a simplifier version of the algorithm by Bar-Yossef, Jayram, Kumar, Sivakumar and Trevisan [5]. In contrast to the AMS algorithm, the BJKST algorithm uses a set to maintain 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 number of distinct items in $\mathcal{S}$.

Algorithm 2 presents a simplified version of the BJKST algorithm, where c is a constant. The general idea of the BJKST algorithm is as follows:

1. Use a set B to maintain the sampled items;
2. When the set B becomes full, shrink B by removing about half items and from then on the sample probability becomes smaller.
3. In the end the number of items in B can be used to give a good approximation of the number of distinct items in $\mathcal{S}$.

See [5] for detailed analysis.

Algorithm 2 The BJKST Algorithm (Simplified Version)

Choose a random function $h : [n] \to [n]$ from a family of pairwise independent hash functions;
$z \leftarrow 0$;
$B \leftarrow \emptyset$
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**
            $z \leftarrow z + 1$;
            shrink B by removing all $(x, \rho(h(x)))$ with $\rho(h(x)) < z$;
Return $|B| \cdot 2^z$

2.3 Indyk's Algorithm

We further address the problem of approximating F_0 in the turnstile stream, and present an algorithm by Indyk [11]. Although the sampling-based algorithms are simple, they cannot be applied in turnstile streams, and we need to develop other techniques.

We first formulate the problem of approximating F_0 in the turnstile stream.

- **Input:** A sequence of pairs of the form (s_i, U_i), where $s_i \in [n]$ and $U_i = +/-$.
- **Output:** The F_0 of the data set expressed by the stream. Moreover, for parameters ε and δ, the output of the algorithm achieves an (ε, δ)-approximation.

Stable Distributions. A distribution D over $\mathbb{R}$ is called p-stable, if there is a parameter $p \geq 0$ such that for any n real numbers $a_1, \ldots, a_n$, 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 $\left(\sum_i |a_i|^p\right)^{1/p} X$, where X is a random variable with distribution D.

Definition 3 (Stable Distributions)**.** *For $0 < p \leq 2$ there exists a probability distribution $\mathcal{D}_p$, called the p-stable distribution, with $\mathbf{E}[\mathrm{e}^{\mathrm{i}tX}] = \mathrm{e}^{-|t|^p}$ for $X \sim \mathcal{D}_p$. For any integer $n > 0$ and vector $a \in \mathbb{R}^n$, if $X_1, \ldots, X_n \sim \mathcal{D}_p$ are independent, then $\sum_{i=1}^n a_i X_i \sim \left(\sum_{i=1}^n |a_i|^p\right)^{1/p} X$, where $X \sim \mathcal{D}_p$.*

It is known that stable distributions exist for any $p \in (0, 2]$. In particular:

- A Cauchy distribution, defined by the density function $c(x) = \frac{1}{\pi} \cdot \frac{1}{1+x^2}$, is 1-stable.
- A Gausssian (normal) distribution, defined by the density function $g(x) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-x^2/2}$, is 2-stable.

For general values of p except the cases that $p \in \{1/2, 1, 2\}$, we have not found the closed formula of the density function yet. For detailed discussion of stable distributions, see [3]. Reference [7] gives a short and interesting introduction to stable distributions in designing streaming algorithms.

How To Generate Stable Distributions. A random variable X satisfying a p-stable distribution can be generated as follows [6]: (i) Pick Θ uniformly at random from $[-\pi/2, \pi/2]$, and pick r uniformly at random from $[0, 1]$. (ii) Output

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

Algorithm. We first give the framework of an *idealized* algorithm. Assume that we have a random matrix $\mathbf{M}$ of size $k \times n$, where $k \triangleq \Theta(\varepsilon^{-2}\log(1/\delta))$, and each entry of $\mathbf{M}$ is drawn from the p-stable distribution. Given this, the algorithm only needs to maintain a vector E of size k, and is described in Algorithm 3.

Algorithm 3 Approximating F_0 in a Turnstile Stream (An Idealized Algorithm)

```
1: while 1 ≤ j ≤ k do
2:     E_j ← 0;
3: while a pair (s_i, U_i) arrives do
4:     if U_i = + then
5:         for j ← 1, k do
6:             E_j ← E_j + M[s_i, j];
7:     else
8:         for j ← 1, k do
9:             E_j ← E_j − M[s_i, j];
10: Return medium_{1≤j≤k}{|E_j|^p} · scalefactor(p)
```

The 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

- Every entry of $\mathbf{M}$ is random and generated according to the p-stable distribution.
- Each column can be reconstructed when necessary.

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

The following lemma shows that algorithms for estimating F_p for small p can be used to estimate F_0.

Lemma 4. *Assume that we have an upper bound K of each entry of the vector E. Then the F_0 norm of the set S can be approximated by the F_p norm of the set S.*

Proof. By definition

$$F_0 = \sum_i |m_i|^0 \le \sum_i |m_i|^p = F_p \le \sum_i K^p |m_i|^0,$$

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

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

□

Now we give the space complexity of Algorithm 3. See [8, 11] for the correctness proof.

Theorem 5. *For any parameters ε, δ, there is an algorithm that achieves an (ε, δ)-approximation of 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 $O(\varepsilon^{-2} \log(1/\delta))$.*

3 Frequency Estimation

Consider the following frequency estimation problem: Starting from an empty set S, there is a sequence of update operations of $\texttt{INSERT}(S, x_i)$ (performs $S \leftarrow S \cup \{x_i\}$) or $\texttt{DELETE}(S, x_i)$ (performs $S \leftarrow S \setminus \{x_i\}$). In addition, there is a query operation $\texttt{QUERY}(S, x_i)$ which asks for the number of occurrences of x_i in the multiset S.

We introduce the Count-Min Sketch that is used to solve the problem above. The Count-Min Sketch is introduced by Cormode and Muthukrishnan [9], and consists of a fixed array C of counters of width w and depth d. These counters are all initialized to be zero. Each row is associated to a hash function h_i, where each h_i maps an element from U to $\{1, \ldots, w\}$. For every $\texttt{INSERT}(S, x_i)$ we update $C[j, h(x_i)]$ via

$$C[j, h(x_i)] \leftarrow C[j, h(x_i)] + 1$$

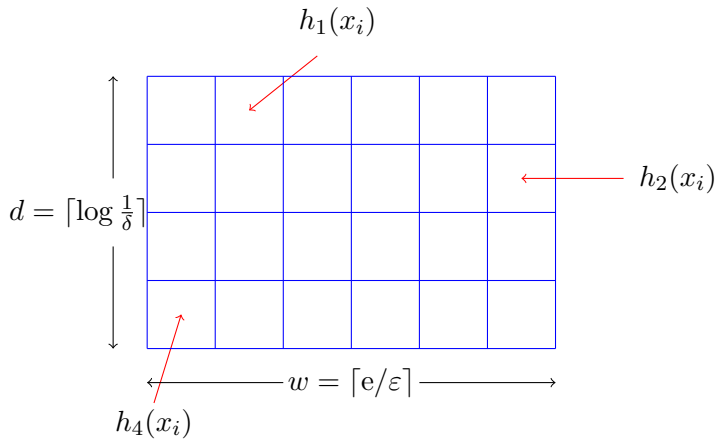
for every row $1 \le i \le d$. For every $\texttt{DELETE}(S, x_i)$ we update $C[j, h(x_i)]$ via

$$C[j, h(x_i)] \leftarrow C[j, h(x_i)] - 1$$

When the number of occurrences of any s_i is asked, we output

$$\widehat{m_i} \triangleq \min_{1 \le j \le d} C[j, h(x_i)].$$

The figure below shows the structure of the Count-Min Sketch.



Hash Functions. The hash functions used here need to be pairwise independent, and it is easy to implement. Let p be a prime number larger than n. For every h_i, we choose two integers $a \in \{1, \ldots, p-1\}$, $b \in \{0, \ldots, p-1\}$, and let $h_i(x) \triangleq (a_i x + b_i) \bmod p \bmod w$.

Choosing w and d. For given parameters ε and δ, set $w \triangleq \lceil \mathrm{e}/\varepsilon \rceil$ and $d \triangleq \lceil \ln(1/\delta) \rceil$. Hence for constant ε and δ, the sketch only consists of constant number of counters.

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

Theorem 6. *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 data 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

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

□

4 Further Reference

[12] is an excellent survey and covers basic algorithmic techniques in designing streaming algorithms. [1] gives a summary, applications, and implementations of the Count-Min Sketch. [2] maintains a list of open questions in data streams which were proposed in the past data streaming workshops.

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 *RANDOM*, pages 1–10, 2002.

[6] 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.

[7] 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.

[8] Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). In *VLDB*, pages 335–345, 2002.

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

[10] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In *ESA*, pages 605–617, 2003.

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

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