§6. Streaming Algorithms

Qin Zhang
The model and challenge

- **The data stream model** (Alon, Matias and Szegedy 1996)

- **Why hard?** Cannot store everything.

- **Applications**: Internet router, stock data, ad auction, flight logs on tape, etc.

(next 4 slides, in courtesy of Jeff Phillips)
Network routers

Internet Router

- data per day: at least 1 Terabyte
- packet takes 8 nanoseconds to pass through router
- few million packets per second

What statistics can we keep on data?
For example, want to detect anomalies for security.
Cell phones connect through switches

- each message 1000 Bytes
- 500 million calls / day
- 1 Terabyte per month second

Search for characteristics for dropped calls?
Serving Ads on web Google, Yahoo!, Microsoft

- Yahoo.com viewed 77 trillion times
- 2 million / hour
- Each page serves ads; which ones?

How to update ad delivery model?
All airplane logs over Washington, DC

- About 500 - 1000 flights per day.
- 50 years, total about 9 million flights
- Each flight has trajectory, passenger count, control dialog.

Stored on Tape. Can only make 1 (or $O(1)$) pass! What statistics can be gathered?
§1.1 Sampling

Uniformly sample an item
A toy example: Reservoir Sampling

**Tasks**: Find a *uniform sample* from a stream of unknown length, can we do it in $O(1)$ space?
A toy example: Reservoir Sampling

Tasks: Find a uniform sample from a stream of unknown length, can we do it in $O(1)$ space?

Algorithm: Store 1-st item. When the $i$-th ($i > 1$) item arrives

- With probability $1/i$, replace the current sample;
- With probability $1 - 1/i$, throw it away.
A toy example: Reservoir Sampling

Tasks: Find a uniform sample from a stream of unknown length, can we do it in $O(1)$ space?

Algorithm: Store 1-st item. When the $i$-th ($i > 1$) item arrives

   - With probability $1/i$, replace the current sample;
   - With probability $1 - 1/i$, throw it away.

Correctness: each item is included in the final sample w.p.

\[
\frac{1}{i} \times \left(1 - \frac{1}{i+1}\right) \times \ldots \times \left(1 - \frac{1}{n}\right) = \frac{1}{n}
\]

(n: total # items)

Space: $O(1)$
Maintain a sample for Sliding Windows

Tasks: Find a uniform sample from the last $w$ items.
**Tasks:** Find a uniform sample from the last $w$ items.

**Algorithm:**
- For each $x_i$, we pick a random value $v_i \in (0, 1)$.
- In a window $\langle x_{j-w+1}, \ldots, x_j \rangle$, return value $x_i$ with smallest $v_i$.
- To do this, maintain the set of all $x_i$ in sliding window whose $v_i$ value is minimal among subsequent values.
Tasks: Find a uniform sample from the last $w$ items.

Algorithm:
- For each $x_i$, we pick a random value $v_i \in (0, 1)$.
- In a window $< x_{j-w+1}, \ldots, x_j >$, return value $x_i$ with smallest $v_i$.
- To do this, maintain the set of all $x_i$ in sliding window whose $v_i$ value is minimal among subsequent values.

Correctness: Obvious.

Space (expected): $1/w + 1/(w - 1) + \ldots + 1/1 = \log w$. 
§1.0 Distinct Elements

How many distinct elements?
Approximation needed.
The FM sketch

Denote the stream by $A = a_1, a_2, \ldots, a_m$, where $m$ is the length of the stream, which is unknown at the beginning. Let $n$ be the item universe.
The FM sketch

Denote the stream by $A = a_1, a_2, \ldots, a_m$, where $m$ is the length of the stream, which is unknown at the beginning. Let $n$ be the item universe.

**The algorithm** (Flajoet and Martin ’83)

1. Choose a random hash function $h : [n] \rightarrow [n]$ from a 2-universal family. Set $z = 0$. Let $\text{zeros}(h(e))$ be the \# tailing zeros of the binary representation of $h(e)$.

2. For each new coming item $e$, if $\text{zeros}(h(e)) > z$, then set $z = \text{zeros}(h(e))$;

3. Output $2^{z+0.5}$. 
The FM sketch

Denote the stream by \( A = a_1, a_2, \ldots, a_m \), where \( m \) is the length of the stream, which is unknown at the beginning. Let \( n \) be the item universe.

The algorithm (Flajoet and Martin '83)

1. Choose a random hash function \( h : [n] \rightarrow [n] \) from a 2-universal family. Set \( z = 0 \). Let \( \text{zeros}(h(e)) \) be the \# tailing zeros of the binary representation of \( h(e) \).
2. For each new coming item \( e \), if \( \text{zeros}(h(e)) > z \), then set \( z = \text{zeros}(h(e)) \);
3. Output \( 2^{z+0.5} \).

Analysis (on board)
The FM sketch

Denote the stream by $A = a_1, a_2, \ldots, a_m$, where $m$ is the length of the stream, which is unknown at the beginning. Let $n$ be the item universe.

**The algorithm** (Flajolet and Martin '83)

1. Choose a random hash function $h : [n] \rightarrow [n]$ from a 2-universal family. Set $z = 0$. Let zeros($h(e)$) be the number of tailing zeros of the binary representation of $h(e)$.
2. For each new coming item $e$, if zeros($h(e)$) $>$ $z$, then set $z = $ zeros($h(e)$);
3. Output $2^{z + 0.5}$.

**Analysis** (on board)

**Theorem**

The number of distinct elements can be $O(1)$-approximated with probability 2/3 using $O(\log n)$ bits.
Can we boost the success probability to $1 - \delta$?

The idea is to run $k = \Theta(\log(1/\delta))$ copies of this algorithm in parallel, using mutually independent random hash functions, and output the median of the $k$ answers.
An improved algorithm (optional)

Idea: two-level hashing.

**The algorithm** (Bar-Yossef et al. ’02)

1. Choose a random hash function $h : [n] \rightarrow [n]$ from a 2-universal family. Set $z = 0$, $B = \emptyset$.

   Choose a secondary 2-universal hash function $g : [n] \rightarrow [(\log n/\epsilon)^{O(1)}]$.

2. For each new coming item $e$, if $\text{zeros}(h(e)) \geq z$, then
   (a) set $B = B \cup \{g(e)\}$;
   (b) if $|B| > c/\epsilon^2$ then set $z = z + 1$ and rehash $B$.

3. Output $|B|2^z$. 
An improved algorithm (optional)

Idea: two-level hashing.

The algorithm (Bar-Yossef et al. ’02)

1. Choose a random hash function $h : [n] \rightarrow [n]$ from a 2-universal family. Set $z = 0$, $B = \emptyset$.
   Choose a secondary 2-universal hash function $g : [n] \rightarrow [\log n/\epsilon]^{O(1)}$.

2. For each new coming item $e$, if $\text{zeros}(h(e)) \geq z$, then
   (a) set $B = B \cup \{g(e)\}$;
   (b) if $|B| > c/\epsilon^2$ then set $z = z + 1$ and rehash $B$.

3. Output $|B| 2^z$.

Analysis (on board)
An improved algorithm (optional)

Idea: two-level hashing.

The algorithm (Bar-Yossef et al. ’02)

1. Choose a random hash function $h : [n] \rightarrow [n]$ from a 2-universal family. Set $z = 0$, $B = \emptyset$.

   Choose a secondary 2-universal hash function $g : [n] \rightarrow [(\log n/\epsilon)^O(1)]$.

2. For each new coming item $e$, if $\text{zeros}(h(e)) \geq z$, then
   (a) set $B = B \cup \{g(e)\}$;
   (b) if $|B| > c/\epsilon^2$ then set $z = z + 1$ and rehash $B$.

3. Output $|B| 2^z$.

Analysis (on board)

Theorem

The number of distinct elements can be $(1 + \epsilon)$-approximated with probability $2/3$ using $O(\log n + 1/\epsilon^2 \cdot (\log(1/\epsilon) + \log \log n))$ bits.
Nice algorithms, but only work for insertion-only sequences ...
Random linear projection $M : R^n \rightarrow R^k$ that preserves properties of any $v \in R^n$ with high prob. where $k \ll n$. 

$$
\begin{bmatrix}
M \\
\end{bmatrix}
\begin{bmatrix}
v
\end{bmatrix}
= 
\begin{bmatrix}
Mv
\end{bmatrix}
\rightarrow \text{answer}
$$
- **Random linear projection** $M : R^n \rightarrow R^k$ that preserves properties of any $v \in R^n$ with high prob. where $k \ll n$.

\[
\begin{bmatrix}
M \\
\end{bmatrix}
\begin{bmatrix}
v \\
\end{bmatrix}
= 
\begin{bmatrix}
Mv \\
\end{bmatrix} \rightarrow \text{answer}
\]

- **Simple and useful**

Perfect for streaming and distributed computations.

Work for **insertion**+**deletion** sequences (that is, $\Delta$ can be either positive or negative).
Search version $\Rightarrow$ Decision version

Let $D$ be $\#$ distinct elements:

- If $D \geq T(1 + \epsilon)$, then answer YES.
- If $D \leq T/(1 + \epsilon)$, then answer NO.

Try $T = 1, (1 + \epsilon), (1 + \epsilon)^2, \ldots$
Now, the decision problem

The algorithm

1. Select a random set $S \subseteq \{1, 2, \ldots, n\}$, s.t. for each $i$, independently, we have $\Pr[i \in S] = 1/T$

2. Make a pass over the stream, maintaining $\text{Sum}_S(x) = \sum_{i \in S} x_i$
   Note: this is a linear sketch.

3. If $\text{Sum}_S(x) > 0$, return YES, otherwise return NO.
Now, the decision problem

The algorithm

1. Select a random set $S \subseteq \{1, 2, \ldots, n\}$, s.t. for each $i$, independently, we have $\text{Pr}[i \in S] = 1/T$

2. Make a pass over the stream, maintaining $\text{Sum}_S(x) = \sum_{i \in S} x_i$
   Note: this is a linear sketch.

3. If $\text{Sum}_S(x) > 0$, return YES, otherwise return NO.

Lemma

Let $P = \text{Pr}[\text{Sum}_S(x) = 0]$. If $T$ is large enough, and $\epsilon$ is small enough, then

- If $D \geq T(1 + \epsilon)$, then $P < 1/e - \epsilon/3$.
- If $D \leq T/(1 + \epsilon)$, then $P > 1/e + \epsilon/3$.

Proof (on board)
Repeat to amplify the success probability

1. Select $k$ sets $S_1, \ldots, S_k$ as in previous algorithm, for $k = C \log(1/\delta)/\epsilon^2$, $C > 0$

2. Let $Z$ be the number of values of $Sum_{S_j}(x)$ that are equal to 0, $j = 1, \ldots, k$.

3. If $Z < k/e$ then report YES, otherwise report NO.
Repeat to amplify the success probability

1. Select $k$ sets $S_1, \ldots, S_k$ as in previous algorithm, for $k = C \log(1/\delta)/\epsilon^2$, $C > 0$

2. Let $Z$ be the number of values of $\text{Sum}_{S_j}(x)$ that are equal to 0, $j = 1, \ldots, k$.

3. If $Z < k/e$ then report YES, otherwise report NO.

Lemma

If the constant $C$ is large enough, then this algorithm reports a correct answer with probability $1 - \delta$. 
Repeat to amplify the success probability

1. Select $k$ sets $S_1, \ldots, S_k$ as in previous algorithm, for $k = C \log(1/\delta)/\epsilon^2$, $C > 0$

2. Let $Z$ be the number of values of $\text{Sum}_{S_j}(x)$ that are equal to 0, $j = 1, \ldots, k$.

3. If $Z < k/e$ then report YES, otherwise report NO.

Lemma

If the constant $C$ is large enough, then this algorithm reports a correct answer with probability $1 - \delta$.

Proof (on board)
Repeat to amplify the success probability

1. Select \( k \) sets \( S_1, \ldots, S_k \) as in previous algorithm, for \( k = C \log(1/\delta)/\epsilon^2, \ C > 0 \)

2. Let \( Z \) be the number of values of \( \text{Sum}_{S_j}(x) \) that are equal to 0, \( j = 1, \ldots, k \).

3. If \( Z < k/e \) then report YES, otherwise report NO.

Lemma

If the constant \( C \) is large enough, then this algorithm reports a correct answer with probability \( 1 - \delta \).

Proof (on board)

Theorem

The number of distinct elements can be \( (1 \pm \epsilon) \)-approximated with probability \( 1 - \delta \) using \( O(\log^2 n \log(1/\delta)/\epsilon^3) \) bits.
Question: can we make FM sketch linear?
1.3 Heavy Hitters

What are most frequent items?
Heavy hitters

- **Heavy Hitters:**

\[
HH_\phi = \{ i : f(i) \geq \phi m \}
\]
Heavy Hitters:

\[ HH_\phi = \{ i : f(i) \geq \phi m \} \]

Heavy Hitter Problem:

Given \( \phi, \phi', \) (often \( \phi' = \phi - \epsilon \)), return a set \( S \) such that

\[ HH_\phi \subseteq S \subseteq HH_{\phi'} \]
Algorithm Misra-Gries [Misra-Gries ’82]

Maintain an array of size $1/\epsilon$, each entry is a tuple in the form of $(e, f(e))$. When a new item $e$ comes, we have two cases.

1. If $e$ is already in the array, increment $f(e)$ in $(e, f(e))$ by 1.

2. If $e$ is not in the array, create a new tuple $(e, 1)$ and try to insert it into the array.

In the case when there are already $1/\epsilon$ items in the array (i.e., the array is full), we decrement the $f(e)$ of all items $e$ in the array, and repeat doing it, until the counts of some items becomes 0, and then we delete those tuples to find space for the new tuple.
Algorithm Misra-Gries [Misra-Gries ’82]

Maintain an array of size $1/\epsilon$, each entry is a tuple in the form of $(e, f(e))$. When a new item $e$ comes, we have two cases.

1. If $e$ is already in the array, increment $f(e)$ in $(e, f(e))$ by 1.

2. If $e$ is not in the array, create a new tuple $(e, 1)$ and try to insert it into the array.

In the case when there are already $1/\epsilon$ items in the array (i.e., the array is full), we decrement the $f(e)$ of all items $e$ in the array, and repeat doing it, until the counts of some items becomes 0, and then we delete those tuples to find space for the new tuple.

Example, Proof (on board)
Space-saving: a variant of Misra-Gries

**Algorithm Space-saving** [Metwally et al. ’05]

When a new item $e$ comes, we have two cases.

1. If $e$ is already in the array. We just increment $f(e)$ by 1 and reinsert the $(e, f(e))$ into the array.

2. If $e$ is not in the array, we create a new tuple $(e, \text{MIN} + 1)$ where $\text{MIN} = \min\{f(e_i) : e_i \text{ is in the array}\}$.

   We always keep the array sorted according to $f(e_i)$, and then $\text{MIN}$ is just the estimated frequency of the last item. If the length array is larger than $1/\epsilon$, we delete the last tuple.

Cost: $O(1/\epsilon)$ words.
Algorithm Space-saving [Metwally et al. ’05]

When a new item $e$ comes, we have two cases.

1. If $e$ is already in the array. We just increment $f(e)$ by 1 and reinsert the $(e, f(e))$ into the array.

2. If $e$ is not in the array, we create a new tuple $(e, \text{MIN} + 1)$ where $\text{MIN} = \min\{f(e_i) : e_i \text{ is in the array}\}$.

   We always keep the array sorted according to $f(e_i)$, and then $\text{MIN}$ is just the estimated frequency of the last item. If the length array is larger than $1/\epsilon$, we delete the last tuple.

Cost: $O(1/\epsilon)$ words.

Example (on board)
Algorithm **Count-Min** [Cormode and Muthu ’05]

- Pick $d$ ($d = \log(1/\delta)$) independent hash functions $h_1, \ldots, h_d$ where $h_i : \{1, \ldots, n\} \rightarrow \{1, \ldots, w\}$ ($w = 2/\epsilon$) from a 2-universal family.

- Maintain $d$ vectors $Z^1, \ldots, Z^d$ where $Z^t = \{Z^t_1, \ldots, Z^t_w\}$ such that $Z^t_j = \sum_{i: h_t(i) = j} f(i)$

- Estimator: $\tilde{f}(i) = \min_t Z^t_{h_t(i)}$
Algorithm Count-Min [Cormode and Muthu '05]

- Pick $d$ ($d = \log(1/\delta)$) independent hash functions $h_1, \ldots, h_d$ where
  $h_i : \{1, \ldots, n\} \rightarrow \{1, \ldots, w\}$ ($w = 2/\epsilon$) from a 2-universal family.

- Maintain $d$ vectors $Z^1, \ldots, Z^d$ where $Z^t = \{Z^t_1, \ldots, Z^t_w\}$ such that
  $Z^t_j = \sum_{i: h_t(i) = j} f(i)$

- Estimator: $\tilde{f}(i) = \min_t Z^t_{h_t(i)}$

Theorem

We can solve heavy hitter with additive approximation $\epsilon m$ ($m$ is the length of the stream) and failure probability $\delta$ by storing $O(\frac{1}{\epsilon} \log(1/\delta))$ words.
Which one is better?
Some of the contents are borrowed from Pitor Indyk’s course http://stellar.mit.edu/S/course/6/fa07/6.895/,