Streaming Algorithms

NARMADA SAMBATURU
SUBHASREE BASU
ALOK KUMAR KESHRI
RAJIV RATN SHAH
VENKATA KIRAN YEDUGUNDLA
VU VINH AN
Overview

- Introduction to Streaming Algorithms
- Sampling Techniques
- Sketching Techniques

Break

- Counting Distinct Numbers
- Q&A
Overview

• Introduction to Streaming Algorithms
• Sampling Techniques
• Sketching Techniques

Break

• Counting Distinct Numbers
• Q&A
What are Streaming algorithms?

- Algorithms for processing data streams
- Input is presented as a sequence of items
- Can be examined in only a few passes (typically just one)
- Limited working memory
Same as Online algorithms?

• **Similarities**
  - decisions to be made before all data are available
  - limited memory

• **Differences**
  - Streaming algorithms – can defer action until a group of points arrive
  - Online algorithms - take action as soon as each point arrives
Why Streaming algorithms

- **Networks**
  - Up to 1 Billion packets per hour per router. Each ISP has hundreds of routers
  - Spot faults, drops, failures

- **Genomics**
  - Whole genome sequences for many species now available, each megabytes to gigabytes in size
  - Analyse genomes, detect functional regions, compare across species

- **Telecommunications**
  - There are 3 Billion Telephone Calls in US each day, 30 Billion emails daily, 1 Billion SMS, IMs
  - Generate call quality stats, number/frequency of dropped calls

- Infeasible to store all this data in random access memory for processing.
- Solution – process the data as a stream – streaming algorithms
Basic setup

• **Data stream**: a sequence $A = <a_1, a_2, ..., a_m>$, where the elements of the sequence (called tokens) are drawn from the universe $[n] = \{1, 2, ..., n\}$

• Aim - compute a function over the stream, eg: median, number of distinct elements, longest increasing sequence, etc.

• **Target Space complexity**
  - Since $m$ and $n$ are “huge,” we want to make $s$ (bits of random access memory) much smaller than these
  - Specifically, we want $s$ to be sublinear in both $m$ and $n.$
    $$s = o(\min\{m, n\})$$
  - The best would be to achieve
    $$s = O(\log m + \log n)$$
Quality of Algorithm

• Let $A(\sigma) = \text{output of a randomized streaming algorithm } A \text{ on input } \sigma$
• Let $\phi = \text{function that } A \text{ is supposed to compute}$
• We say the algorithm $(\varepsilon, \delta)$-approximates $\phi$ if

$$\Pr \left[ \left| \frac{A(\sigma)}{\phi(\sigma)} - 1 \right| > \varepsilon \right] \leq \delta$$

• This is sometimes too strong a condition if the value of $\phi(\sigma)$ is close to 0. Then we relax the rule to expect

$$\Pr \left[ |A(\sigma) - \phi(\sigma)| > \varepsilon \right] \leq \delta$$
Streaming Models - Cash Register Model

• **Time-Series Model**
  Only x-th update is processed
  i.e., $A[x] = c[x]$

• **Cash-Register Model**: Arrivals-Only Streams
  $c[x]$ is always $> 0$
  Typically, $c[x]=1$

  Example: <x, 3>, <y, 2>, <x, 2> encodes the arrival of
  3 copies of item x,
  2 copies of y,
  2 copies of x.
  Could represent, packets in a network, power usage
Streaming Models – Turnstile Model

- **Turnstile Model**: Arrivals and Departures
  - Most general streaming model
  - $c[x]$ can be $>0$ or $<0$

- Example:
  - $<x, 3>, <y, 2>, <x, -2>$ encodes final state of $<x, 1>, <y, 2>$. Can represent fluctuating quantities, or measure differences between two distributions
Overview

- Introduction to Streaming Algorithms
- Sampling Techniques
- Sketching Techniques
- Break
- Counting Distinct Numbers
- Q&A
Sampling

• Idea
  A small random sample $S$ of the data is often enough to represent all the data

• Example
  To compute median packet size
  Sample some packets
  Present median size of sampled packets as true median

• Challenge
  Don’t know how long the stream is
Reservoir Sampling - Idea

• We have a reservoir that can contain $k$ samples
• Initially accept every incoming sample till reservoir fills up
• After reservoir is full, accept sample $k + i$ with probability $\frac{k}{k + i}$

• This means as long as our reservoir has space, we sample every item
• Then we replace items in our reservoir with gradually decreasing probability
Reservoir Sampling - Algorithm

array R[k];       // result
integer i, j;

// fill the reservoir array
for each i in 1 to k do
    R[i] := S[i]
done;

// replace elements with gradually decreasing probability
for each i in k+1 to length(S) do
    j := random(1, i);    // important: inclusive range
    if j <= k then
        R[j] := S[i]
    fi
done
Probability Calculations
Probability of any element to be included at round t

- Let us consider a time $t > N$.
- Let the number of elements that has arrived till now be $N_t$
- Since at each round, all the elements have equal probabilities, the probability of any element being included in the sample is $\frac{N}{N_t}$

**Observation:**
Hence even though at the beginning a lot of elements get replaced, with the increase in the stream size, the probability of a new record evicting the old one drops.
Probability of any element to be chosen for the final Sample

• Let the final stream be of size $N_T$
• **Claim:**
  The probability of any element to be in the sample is $N / N_T$
Probability of survival of the initial N elements

- Let us choose any particular element out of our N initial elements ($e_N$ say)
- The eviction tournament starts after the arrival of the $(N + 1)^{st}$ element
- Probability that $(N + 1)^{st}$ element is chosen is $N/(N + 1)$
- Probability that if $(N + 1)^{st}$ element is chosen by evicting $e_N$ is $1/N$
- Hence probability of $e_N$ being evicted in this case is
  
  \[
  \frac{1}{N} \times \frac{N}{(N + 1)} = \frac{1}{N + 1}
  \]
- Probability that $e_N$ survives = $1 - \left( \frac{1}{(N + 1)} \right) = \frac{N}{(N + 1)}$
- Similarly the case $e_N$ survives when $(N+2)^{nd}$ element arrives = $(N+1)/(N+2)$
- The probability of $e_N$ surviving two new records
  
  \[
  = \frac{N}{(N+1)} \times \frac{(N+1)}{(N+2)}
  \]
- The probability of $e_N$ surviving till the end
  
  \[
  = \frac{N}{(N+1)} \times \frac{(N+1)}{(N+2)} \times \ldots \times \frac{(N_T - 1)}{N_T} = \frac{N}{N_T}
  \]
Probability of survival of the elements after the initial N

- For the last arriving element to be selected, the probability is \( \frac{N}{N_T} \)
- For the element before the last, the probability of selection
  - \( = \frac{N}{(N_T - 1)} \)
- The probability of the last element replacing the last but one element
  - \( = \left(\frac{N}{N_T}\right) \times \left(\frac{1}{N}\right) = \frac{1}{N_T} \)
- The probability that the last but one element survives = 1 - \( \frac{1}{N_T} = \frac{(N_T - 1)}{N_T} \)
- The probability that the last but one survives till the end
  - \( = \left(\frac{N}{(N_T - 1)}\right) \times \left(\frac{N_T - 1}{N_T}\right) = \frac{N}{N_T} \)

Similarly we can show that the probability of survival of any element in the sample is \( \frac{N}{N_T} \)
Calculating the Maximum Reservoir Size
Some Observations

• Initially the reservoir contains N elements
• Hence the size of the reservoir space is also N
• New records are added to the reservoir only when it will replace any element present previously in the reservoir.
• If it is not replacing any element, then it is not added to the reservoir space and we move on to the next element.
• However we find that when an element is evicted from the reservoir, it still exists in the reservoir storage space.
• The position in the array that held its pointer, now holds some other element’s pointer. But the element is still present in the reservoir space.
• Hence the total number of elements in the reservoir space at any particular time ≥ N.
Maximum Size of the Reservoir

- The new elements are added to the reservoir with initial probability $N/N+1$
- This probability steadily drops to $N/ N_T$
- The statistical expectation of the size $S$ of the reservoir space can thus be calculated as
  $$N + (N/N+1) + \ldots + (N/ N_T)$$
- Overestimating it with an integral the reservoir size can be estimated as
  $$\int_{x=N}^{x=NT} N \frac{dx}{x} = N \ln(NT/N)$$
- Thus, reservoir estimate is:
  $$S = N[1 + \ln (N_T/N)]$$
- Hence we find that the space needed is $O(N \log(N_T))$
Priority Sample for Sliding Window
Reservoir Sampling Vs Sliding Window

Reservoir Sampling

• Works well when we have only inserts into a sample
• The first element in the data stream can be retained in the final sample
• It does not consider the expiry of any record

Sliding Window

• Works well when we need to consider “timeliness” of the data
• Data is considered to be expired after a certain time interval
• “Sliding window” in essence is such a random sample of fixed size (say k) “moving” over the most recent elements in the data stream
Types of Sliding Window

• **Sequence-based**
  -- they are windows of size k moving over the k most recently arrived data. Example being chain-sample algorithm

• **Time-stamp based**
  -- windows of duration t consist of elements whose arrival timestamp is within a time interval t of the current time. Example being Priority Sample for Sliding Window
Principles of the Priority Sampling algorithm

• As each element arrives, it is assigned a randomly-chosen priority between 0 and 1
• An element is *ineligible* if there is another element with a later timestamp and higher priority
• The element selected for inclusion in the sample is thus the most *active* element with the *highest* priority
• If we have a sample size of k, we generate k priorities \( p_1, p_2, \ldots, p_k \) for each element. The element with the highest \( p_i \) is chosen for each i
Memory Usage for Priority Sampling

• We will be storing only the eligible elements in the memory
• These elements can be made to form right spine of the datastructure “treap”
• Therefore expected memory usage is $O(\log n)$, or $O(k \log n)$ for samples of size $k$

Ref:
References

• Crash course - http://people.cs.umass.edu/~mcgregor/slides/10-jhu1.pdf
• Notes
  ▪ http://www.cs.dartmouth.edu/~ac/Teach/CS49-Fall11/Notes/lecnotes.pdf
• http://en.wikipedia.org/wiki/Streaming_algorithm

• Reservoir Sampling
• Notes and explanations
  ▪ http://en.wikipedia.org/wiki/Reservoir_sampling
  ▪ http://blogs.msdn.com/b/spt/archive/2008/02/05/reservoir-sampling.aspx
• Paul F Hultquist, William R Mahoney and R.G. Seidel, Reservoir Sampling, Dr Dobb’s Journal, Jan 2001, pp 189-190
• B Babcock, M Datar, R Motwani, SODA ’02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, January 2002
Overview

• Introduction to Streaming Algorithms
• Sampling Techniques
• **Sketching Techniques**

Break

• Counting Distinct Numbers
• Q&A
Sketching

• Sketching is another general technique for processing stream

Fig: Schematic view of linear sketching
How Sketching is different from Sampling

• Sample “sees” only those items which were selected to be in the sample whereas the sketch “sees” the entire input, but is restricted to retain only a small summary of it.

• There are queries that can be approximated well by sketches that are provably impossible to compute from a sample.
Bloom Filter
Set Membership Task

• x: Element
• S: Set of elements
• Input: x, S
• Output:
  – True (if x in S)
  – False (if x not in S)
Bloom Filter

• Consists of
  – vector of \( n \) Boolean values, initially all set false
  – \( k \) independent hash functions, \( h_0, h_1, \ldots, h_{k-1} \), each with range \( \{0, 1, \ldots, n-1\} \)

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

\( n = 10 \)
Bloom Filter

• For each element $s$ in $S$, the Boolean value with positions $h_0(s), h_1(s), \ldots, h_{k-1}(s)$ are set true.

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>F</th>
<th>T</th>
<th>F</th>
<th>F</th>
<th>T</th>
<th>F</th>
<th>T</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$k = 3$
Bloom Filter

• For each element \( s \) in \( S \), the Boolean value with positions \( h_0(s), h_1(s), \ldots, h_{k-1}(s) \) are set true.

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>T</th>
<th>F</th>
<th>F</th>
<th>T</th>
<th>F</th>
<th>T</th>
<th>T</th>
<th>F</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

\( k = 3 \)
Error Types

• False Negative
  – Never happens for Bloom Filter

• False Positive
  – Answering “is there” on an element that is not in the set
Probability of false positives

Consider a particular bit $0 \leq j \leq n-1$

Probability that $h_i(x)$ does not set bit $j$: $P_{h_i \sim H}(h_i(x) \neq j) = \left(1 - \frac{1}{n}\right)^{km}$

Probability that bit $j$ is not set $P_{h_1 \ldots h_k \sim H}(Bit(j) = F) \leq \left(1 - \frac{1}{n}\right)^{km}$

We know that, $\left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e} = e^{-1}$

$\Rightarrow \left(1 - \frac{1}{n}\right)^{km} = \left(\left(1 - \frac{1}{n}\right)^n\right)^{km/n} \approx (e^{-1})^{km/n} = e^{-km/n}$
Probability of false positives

\[ \text{Probability of false positive} = \left(1 - e^{-\frac{km}{n}}\right)^k \]

Note: All k bits of new element are already set

False positive probability can be minimized by choosing
\[ k = \log_2 \left(\frac{n}{m}\right) \]

Upper Bound Probability would be
\[ \left(1 - e^{-\log_2 \left(\frac{n}{m}\right) \cdot \left(\frac{m}{n}\right)}\right)^{\log_2 \left(\frac{n}{m}\right)} \]
\[ \Rightarrow (0.5)^{\log_2 \left(\frac{n}{m}\right)} \]
Bloom Filters: cons

- Small false positive probability
- No deletions
- Can not store associated objects
References

• Graham Cormode, *Sketch Techniques for Approximate Query Processing*, AT&T Research

• Michael Mitzenmacher, *Compressed Bloom Filters*, Harvard University, Cambridge
Count Min Sketch

- The Count-Min sketch is a simple technique to summarize large amounts of frequency data.
- It was introduced in 2003 by G. Cormode and S. Muthukrishnan, and since then has inspired many applications, extensions and variations.
- It can be used for as the basis of many different stream mining tasks — Join aggregates, range queries, frequency moments, etc.
- \( F_k \) of the stream as \( \sum_i (f_i)^k \) — the \( k' \)th Frequency Moment, where \( f_i \) be the frequency of item \( i \) in the stream
  - \( F_0 \) : count 1 if \( f_i \neq 0 \) — number of distinct items
  - \( F_1 \) : length of stream, easy
  - \( F_2 \) : sum the squares of the frequencies — self join size
  - \( F_k \) : related to statistical moments of the distribution
  - \( F_\infty \) : dominated by the largest \( f_k \), finds the largest frequency
    - The space complexity of approximating the frequency moments by Alon, Matias, Szegedy in STOC 1996 studied this problem
    - They presented AMS sketch estimate the value of \( F_2 \)
- Estimate \( a[i] \) by taking \( \hat{a}_i = \min_j \text{count}[j, h_j(i)] \)
- Guarantees error less than \( \mathcal{E}1 \) in size \( O\left(\left\lceil \frac{\epsilon}{\epsilon'} \right\rceil \ln \frac{1}{\delta} \right) \)
  - Probability of more error is less than \( (1 - \delta) \)
- Count Min Sketch gives best known time and space bound for Quantiles and Heavy Hitters problems in the Turnstile Model.
Count Min Sketch

- Model input data stream as vector
  \[ \vec{a}(t) = (a_1(t), \ldots, a_i(t), \ldots a_n(t)) \]
  Where initially \[ a_i(0) = 0 \quad \forall i \]
- The \( t^{th} \) update is \((i_t, c_t)\)
  \[ a'_{i'}(t) = a'_{i'}(t - 1) \quad \forall i' \neq i_t \]
  \[ a_{i_t}(t) = a_{i_t}(t - 1) + c_t \]

- A Count-Min (CM) Sketch with parameters \((\varepsilon, \delta)\) is represented by a two-dimensional array (a small summary of input) counts with width \(w\) and depth \(d : count[1,1] \ldots count[d, w] \)

Given parameters \((\varepsilon, \delta)\), set \(w = \left\lceil \frac{e}{\varepsilon} \right\rceil\) and \(d = \left\lceil \ln \frac{1}{\delta} \right\rceil\) Each entry of the array is initially zero.

\(d\) hash functions are chosen uniformly at random from a pairwise independent family which map vector entry to \([1\ldots w]\). i.e. \(h_1, \ldots, h_d : [1\ldots n] \rightarrow [1\ldots w]\)

When \((i_t, c_t)\) arrives, set \(\forall 1 \leq j \leq d\)

\[ count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t \]

Update procedure:

![Diagram showing the update procedure for Count Min Sketch]](image_url)
Count Min Sketch Algorithm

Initialize:
1. $t \leftarrow \log(1/\delta)$;
2. $k \leftarrow 2/\varepsilon$;
3. $C[1 \ldots t][1 \ldots k] \leftarrow 0$;
4. Pick $t$ independent hash functions $h_1, h_2, \ldots, h_t : [n] \rightarrow [k]$, each from a 2-universal family;

Process $(j, c)$:
5. for $i = 1$ to $t$ do
6. 

\[
C[i][h_i(j)] \leftarrow C[i][h_i(j)] + c;
\]

Output:
On query $a$, report $\hat{f}_a = \min_{1 \leq i \leq t} C[i][h_i(a)]$

Analysis

Time to produce the estimate $O(\ln \frac{1}{\delta})$

Space used $O(\frac{1}{\varepsilon} \ln \frac{1}{\delta})$

Time for updates $O(\ln \frac{1}{\delta})$
Example

Initialize:
1. $t \leftarrow \log(1/\delta)$;
2. $k \leftarrow 2^c$;
3. $C[1..r][1..k] \leftarrow 0$;
4. Pick $r$ independent hash functions $h_1, h_2, \ldots, h_r : [n] \rightarrow [k]$, each from a 2-universal family;

Process $(j, c)$:
5. for $i = 1$ to $r$
6. $C[i][h_i(j)] \leftarrow C[i][h_i(j)] + c$

Output:
- On query $a$, report $\hat{f}_a = \min_{1 \leq i \leq r} C[i][h_i(a)]$.
Approximate Query Answering

- Point query: $Q(i) \xrightarrow{\text{approx.}} a_i$

- Range queries: $Q(l, r) \xrightarrow{\text{approx.}} \sum_{i=l}^{r} a_i$

- Inner product queries: $Q(\vec{a}, \vec{b}) \xrightarrow{\text{approx.}} \vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i$
Point Query

- Non-negative case \( a_i(t) > 0 \)

\[
Q(i) \quad \Rightarrow \quad \hat{a}_i = \min_j \text{count}[j, h_j(i)]
\]

**Theorem 1**

\[
a_i \leq \hat{a}_i \quad \text{and} \quad P[\hat{a}_i > a_i + \epsilon \|\vec{a}\|_1] \leq \delta
\]

**PROOF:** Introduce indicator variables

\[
I_{i,j,k} = \begin{cases} 
1 & \text{if } (i \neq k) \wedge (h_j(i) = h_j(k)) \\
0 & \text{otherwise}
\end{cases}
\]

\[
E(I_{i,j,k}) = \Pr[h_j(i) = h_j(k)] \leq \frac{1}{w} = \frac{\epsilon}{e}
\]

Define the variable

\[
X_{i,j} = \sum_{k=1}^{n} I_{i,j,k} a_k
\]

By construction,

\[
\text{count}[j, h_j(i)] = a_i + X_{i,j} \quad \Rightarrow \quad \min \text{count}[j, h_j(i)] \geq a_i
\]
For the other direction, observe that

\[ E(X_{i,j}) = E\left( \sum_{k=1}^{n} I_{i,j,k} a_k \right) = \sum_{k=1}^{n} a_k E(I_{i,j,k}) \leq \frac{\varepsilon}{e} \| \bar{a} \|_1 \]

\[ \Pr[\hat{a}_i > a_i + \varepsilon \| \bar{a} \|_1] = \Pr[\forall j. \text{count}[j, h_j(i)] > a_i + \varepsilon \| \bar{a} \|_1] \]

\[ = \Pr[\forall j. a_i + X_{i,j} > a_i + \varepsilon \| \bar{a} \|_1] \]

\[ = \Pr[\forall j. X_{i,j} > eE(X_{i,j})] < e^{-d} \leq \delta \]

Markov inequality

\[ \Pr[X \geq t] \leq \frac{E(X)}{t} \quad \forall t > 0 \]

**Analysis**

- Time to produce the estimate: \( O(\ln \frac{1}{\delta}) \)
- Space used: \( O\left(\frac{1}{\varepsilon} \ln \frac{1}{\delta}\right) \)
- Time for updates: \( O(\ln \frac{1}{\delta}) \)

**Remark:** The constant \( \varepsilon \) is used here to minimize the space used.
Range Query

- Dyadic range: \([x2^y + 1 \ldots (x + 1)2^y]\) for parameters \(x, y\)
- Range query
  - (at most) \(2\log_2 n\) dyadic range queries
  - Single point query
- For each set of dyadic ranges of length \(2^y\), \(y = 0 \ldots \log_2 n - 1\)
  - A sketch is kept
  - \(\log_2 n\) CM Sketches

\[
Q(l, r) \\
\text{Compute the dyadic ranges (at most } 2\log_2 n \text{) which canonically cover the range}
\]

\[
\text{Pose that many point queries to the sketches}
\]

\[
\text{Sum of queries } = \hat{a}[l, r]
\]

\[
\text{SELECT COUNT(*) FROM D WHERE D.val } \geq l \text{ AND D.val } \leq h
\]
Range Sum Example

- AMS approach to this, the error scales proportional to $\sqrt{F_2(f)} \frac{F_2(f')}{F_2(f')}$
  So here the error grows proportional to the square root of the length of the range.

- Using the Count-Min sketch approach, the error is proportional to $F_1(h-l+1)$, i.e. it grows proportional to the length of the range.

- Using the Count-Min sketch to approximate counts, the accuracy of the answer is proportional to $(F_1 \log n)/w$. For large enough ranges, this is an exponential improvement in the error.

**e.g.** To estimate the range sum of [2...8], it is decomposed into the ranges [2...2], [3...4], [5...8], and the sum of the corresponding nodes in the binary tree as the estimate.
Theorem 4  
\[ a[l, r] \leq \hat{a}[l, r] \]
\[ \Pr[\hat{a}[l, r] > a[l, r] + 2\varepsilon \log n\|\bar{a}\|_1] \leq \delta \]

Proof:  
\[ a_i \leq \hat{a}_i \]
\[ a[l, r] \leq \hat{a}[l, r] \]

\[ E(\Sigma \text{error for each estimator}) = 2 \log n \]
\[ E(\text{error for each estimator}) \leq 2 \log n \frac{\varepsilon}{e} \|\bar{a}\|_1 \]
\[ \Pr[\hat{a}[l, r] - a[l, r] > 2 \log n\|\bar{a}\|_1] < e^{-d} \leq \delta \]

Analysis

Time to produce the estimate  \( O\left(\log(n) \log \frac{1}{\delta}\right) \)

Space used  \( O\left(\frac{\log(n)}{\varepsilon} \log \frac{1}{\delta}\right) \)

Time for updates  \( O\left(\log(n) \log \frac{1}{\delta}\right) \)

Remark: the guarantee will be more useful when stated without terms of \( \log n \) in the approximation bound.
Inner Product Query

Set
\[
(\hat{\mathbf{a}} \cdot \hat{\mathbf{b}})_j = \sum_{k=1}^{w} \text{count}_{\mathbf{a}}[j,k] \times \text{count}_{\mathbf{b}}[j,k]
\]

\[
Q(\hat{\mathbf{a}}, \hat{\mathbf{b}}) \quad \rightarrow \quad (\hat{\mathbf{a}} \cdot \hat{\mathbf{b}})_j = \min_j (\hat{\mathbf{a}} \cdot \hat{\mathbf{b}})_j
\]

Theorem 3
\[
(\hat{\mathbf{a}} \cdot \hat{\mathbf{b}}) \leq (\hat{\mathbf{a}} \cdot \hat{\mathbf{b}})
\]
\[
\text{Pr}[(\hat{\mathbf{a}} \cdot \hat{\mathbf{b}}) > \hat{\mathbf{a}} \cdot \hat{\mathbf{b}} + \varepsilon \|\hat{\mathbf{a}}\|_1 \|\hat{\mathbf{b}}\|_1] \leq \delta
\]

Analysis
- Time to produce the estimate: \(O\left(\frac{1}{\varepsilon \log \frac{1}{\delta}}\right)\)
- Space used: \(O\left(\frac{1}{\varepsilon \log \frac{1}{\delta}}\right)\)
- Time for updates: \(O\left(\log \frac{1}{\delta}\right)\)

Application
- The application of inner-product computation to Join size estimation

Corollary
- The Join size of two relations on a particular attribute can be approximated up to \(\varepsilon \|\hat{\mathbf{a}}\|_1 \|\hat{\mathbf{b}}\|_1\) with probability \(1 - \delta\) by keeping space \(O\left(\frac{1}{\varepsilon \log \frac{1}{\delta}}\right)\)
Resources

Applications
  – Compressed Sensing
  – Networking
  – Databases
  – Eclectics (NLP, Security, Machine Learning, ...)

Details
  – Extensions of the Count-Min Sketch
  – Implementations and code

List of open problems in streaming
  – Open problems in streaming
References for Count Min Sketch

• Basics

• Journal

• Surveys

• Coverage in Textbooks

• Tutorials
  – Video explaining sketch data structures with emphasis on CM sketch. Graham Cormode.

• Lectures
Overview

• Introduction to Streaming Algorithms
• Sampling Techniques
• Sketching Techniques

Break

• Counting Distinct Numbers
• Q&A
Overview

• Introduction to Streaming Algorithms
• Sampling Techniques
• Sketching Techniques

Break

• Counting Distinct Numbers
• Q&A
Stream Model of Computation

Memory: poly(1/\(\varepsilon\), log N)

Query/Update Time: poly(1/\(\varepsilon\), log N)

\(N\): # items so far, or window size

\(\varepsilon\): error parameter

Main Memory
(Synopsis Data Structures)
Counting Distinct Elements - Motivation

- Motivation: Various applications
  - Port Scanning
  - DDoS Attacks
  - Traffic Accounting
  - Traffic Engineering
  - Quality of Service

Packet Filtering:
No of Packets – 6 (n)
No of Distinct Packets – 3 (m)
Counting Distinct Elements - Problem

- **Problem:** Given a stream $X = \langle x_1, x_2, \ldots, x_m \rangle \in [n]^m$ of values. Let $F_0$ be the number of distinct elements in $X$. Find $F_0$ under the constraints for algorithms on data streams.

- **Constraints:**
  - Elements in stream are presented sequentially and single pass is allowed.
  - Limited space to operate. Expected space complexity $O(\log(\min(n, m)))$ or smaller.
  - Estimation Guarantees: With Error $\epsilon < 1$ and high probability
Naïve Approach

- Counter $C(i)$ for each domain value $i$ in $[n]$
- Initialize counters $C(i) \leftarrow 0$
- Scan $X$ incrementing appropriate counters
- **Solution:** Distinct Values = Number of $C(i) > 0$
- **Problem**
  - Memory size $M << n$
  - Space $O(n)$ — possibly $n >> m$
    - (e.g., when counting distinct words in web crawl)
  - Time $O(n)$
Algorithm History

- Flajolet and Martin introduced problem
  - $O(\log n)$ space for fixed $\varepsilon$ in random oracle model
- Alon, Matias and Szegedy
  - $O(\log n)$ space/update time for fixed $\varepsilon$ with no oracle
- Gibbons and Tirthapura
  - $O(\varepsilon^{-2} \log n)$ space and $O(\varepsilon^{-2})$ update time
- Bar-Yossef et al
  - $O(\varepsilon^{-2} \log n)$ space and $O(\log 1/\varepsilon)$ update time
  - $O(\varepsilon^{-2} \log \log n + \log n)$ space and $O(\varepsilon^{-2})$ update time, essentially
  - Similar space bound also obtained by Flajolet et al in the random oracle model
- Kane, Nelson and Woodruff
  - $O(\varepsilon^{-2} + \log n)$ space and $O(1)$ update and reporting time
  - All time complexities are in unit-cost RAM model
Flajolet-Martin Approach

• Hash function $h$: map $n$ elements to $L = \log_2 n$ bits (uniformly distributed over the set of binary strings of length $L$)

• For $y$ any non-negative integer, define $\text{bit}(y, k) = k^{\text{th}}$ bit in the binary representation of $y$

\[ y = \sum_{k \geq 0} \text{bit}(y, k) \cdot 2^k \]

\[ \rho(y) = \min_{k \geq 0} \{ \text{bit}(y, k) \neq 0 \text{ if } y > 0 \} \]

\[ \rho(y) = L \quad \text{if } y = 0 \]

$\rho(y)$ represents the position of the least significant – bit in the binary representation of $y$
Flajolet-Martin Approach

\[
\text{for } (i:=0 \text{ to } L-1) \text{ do } BITMAP[i]:=0; \\
\text{for (all } x \text{ in } M) \text{ do} \\
\quad \text{begin} \\
\quad \quad \text{index:=} \rho(h(x)); \\
\quad \quad \text{if } BITMAP[\text{index}]=0 \text{ then} \\
\quad \quad \quad \text{BITMAP[\text{index}]}:=1; \\
\quad \text{end} \\
R := \text{the largest } \text{index in } BITMAP \text{ whose value equals to } 1 \\
Estimate := 2^R
Examples of $\text{bit}(y, k)$ & $\rho(y)$

- $y=10=(1010)_2$
  - $\text{bit}(y,0)=0$ $\text{bit}(y,1)=1$
  - $\text{bit}(y,2)=0$ $\text{bit}(y,3)=1$

$$y = \sum_{k \geq 0} \text{bit}(y,k) \cdot 2^k$$

<table>
<thead>
<tr>
<th>int $y$</th>
<th>binary format</th>
<th>$\rho(y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>4 (=L)</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>3</td>
</tr>
</tbody>
</table>
Flajolet-Martin Approach – Estimate Example

• Part of a Unix manual file M of size 26692 lines is loaded of which 16405 are distinct.

• If the final BITMAP looks like this:
  
  0000,0000,1100,1111,1111,1111

• The left most 1 appears at position 15

• We say there are around $2^{15}$ distinct elements in the stream. But $2^{14} = 16384$.

• Estimate $F_0 \approx \log_2 \varphi n$ where $\varphi = 0.77351$ is the correction factor.
Flajolet-Martin* Approach

• Pick a hash function $h$ that maps each of the $n$ elements to at least $\log_2 n$ bits.
• For each stream element $a$, let $r(a)$ be the number of trailing 0’s in $h(a)$.
• Record $R =$ the maximum $r(a)$ seen.
• Estimate $= 2^R$.

* Really based on a variant due to AMS (Alon, Matias, and Szegedy)
Why It Works

- The probability that a given $h(a)$ ends in at least $r$ 0’s is $2^{-r}$.
- If there are $m$ different elements, the probability that $R \geq r$ is $1 - (1 - 2^{-r})^m$. 

Prob. all h(a)’s end in fewer than $r$ 0’s. 

Probability any given h(a) ends in fewer than $r$ 0’s.
Why It Works (2)

• Since \(2^{-r}\) is small, 
  \[1 - (1-2^{-r})^m \approx 1 - e^{-m2^{-r}}.\]

• If \(2^r \gg m\), 
  \[1 - (1 - 2^{-r})^m \approx 1 - (1 - m2^{-r}) \approx m/2^r \approx 0.\]

• If \(2^r \ll m\), 
  \[1 - (1 - 2^{-r})^m \approx 1 - e^{-m2^{-r}} \approx 1.\]

• Thus, \(2^R\) will almost always be around \(m\).
Algorithm History

- Flajolet and Martin introduced problem
  - $O(\log n)$ space for fixed $\varepsilon$ in random oracle model
- Alon, Matias and Szegedy
  - $O(\log n)$ space/update time for fixed $\varepsilon$ with no oracle
- Gibbons and Tirthapura
  - $O(\varepsilon^{-2} \log n)$ space and $O(\varepsilon^{-2})$ update time
- Bar-Yossef et al
  - $O(\varepsilon^{-2} \log n)$ space and $O(\log 1/\varepsilon)$ update time
  - $O(\varepsilon^{-2} \log \log n + \log n)$ space and $O(\varepsilon^{-2})$ update time, essentially
  - Similar space bound also obtained by Flajolet et al in the random oracle model
- Kane, Nelson and Woodruff
  - $O(\varepsilon^{-2} + \log n)$ space and $O(1)$ update and reporting time
  - All time complexities are in unit-cost RAM model
An Optimal Algorithm for the Distinct Elements Problem

Daniel M. Kane, Jelani Nelson, David P. Woodruff
Overview

• Computes a $(1 \pm \varepsilon)$ approximation using an optimal $\Theta(\varepsilon^{-2} + \log n)$ bits of space with 2/3 success probability, where $0 < \varepsilon < 1$ is given.

• Process each stream update in $\Theta(1)$ worst-case time.
Foundation technique 1

• If it is known that $R=\Theta(F_0)$ then $(1 \pm \varepsilon)$ estimation becomes easier

• Run a constant-factor estimation at the end of the stream to achieve $R$ before the main estimation algorithm $\rightarrow$ ROUGH ESTIMATOR
Foundation technique 2

- **Balls and Bins Approach:** use truly random function $f$ to map $A$ balls into $K$ bins and count the number of non-empty bins $X$

  $$E[X] = K(1 - \left(1 - \frac{1}{K}\right)^A)$$

- Instead of using $f$, use $O\left(\frac{\log\frac{K}{\varepsilon}}{\log\log\frac{K}{\varepsilon}}\right)$ - wise independent mapping $g$ then the expected number of non-empty bins under $g$ is the same as under $f$, up to a factor of $(1 \pm \varepsilon)$
Rough Estimator (RE)

1. Set $K_{RE} = \max\{8, \log(n)/\log\log(n)\}$.
2. Initialize $3K_{RE}$ counters $C_1^j, \ldots, C_{K_{RE}}^j$ to $-1$ for $j \in [3]$.
3. Pick random $h_1^j \in \mathcal{H}_2([n],[0,n-1]), h_2^j \in \mathcal{H}_2([n],[K_{RE}^3]), h_3^j \in \mathcal{H}_2(K_{RE}([K_{RE}^3],[K_{RE}])$ for $j \in [3]$.
4. Update(i): For each $j \in [3]$, set $C_{h_3^j(h_2^j(i))}^j \leftarrow \max\{C_{h_3^j(h_2^j(i))}^j, \text{lsb}(h_1^j(i))\}$.
5. Estimator: For integer $r \geq 0$, define $T_r^j = |\{i : C_i^j \geq r\}|$.
   For the largest $r = r^*$ with $T_r^j \geq \rho K_{RE}$, set $\widetilde{F}_0^j = 2r^* K_{RE}$. If no such $r$ exists, $\widetilde{F}_0^j = -1$.
   Output $\widetilde{F}_0 = \text{median}\{\widetilde{F}_0^1, \widetilde{F}_0^2, \widetilde{F}_0^3\}$.

- With probability $1 - o(1)$, the output $\widetilde{F}_0$ of RE satisfies
  
  $$F_0(t) \leq \widetilde{F}_0(t) \leq 8F_0(t)$$

  for every $t \in [m]$ with $F_0(t) \geq K_{RE}$ simultaneously

- The space used is $O(\log(n))$
- Can be implemented with $O(1)$ worst-case update and reporting times
Main Algorithm(1)

1. Set $K = 1/\varepsilon^2$.
2. Initialize $K$ counters $C_1, \ldots, C_K$ to $-1$.
3. Pick random $h_1 \in \mathcal{H}_2([n], [0, n-1])$, $h_2 \in \mathcal{H}_2([n], [K^3])$, $h_3 \in \mathcal{H}_k([K^3], [K])$ for $k = \Omega(\log(1/\varepsilon)/\log\log(1/\varepsilon))$.
4. Initialize $A, b, \text{est} = 0$.
5. Run an instantiation RE of ROUGH_ESTIMATOR.
6. Update(i): Set $x \leftarrow \max\{C_{h_3}(h_2(i)), \text{lsb}(h_1(i)) - b\}$.
   Set $A \leftarrow A - \lceil \log(2 + C_{h_3}(h_2(i))) \rceil + \lceil \log(2 + x) \rceil$.
   If $A > 3K$, Output FAIL.
   Set $C_{h_3}(h_2(i)) \leftarrow x$. Also feed $i$ to RE.
   Let $R$ be the output of RE.
   
   if $R > 2^\text{est}$:
   
   (a) $\text{est} \leftarrow \log(R)$, $b_{\text{new}} \leftarrow \max\{0, \text{est} - \log(K/32)\}$.
   (b) For each $j \in [K]$, set $C_j \leftarrow \max\{-1, C_j + b - b_{\text{new}}\}$
   (c) $b \leftarrow b_{\text{new}}$, $A \leftarrow \sum_{j=1}^{K} \lceil \log(C_j + 2) \rceil$.
7. Estimator: Define $T = |\{j : C_j \geq 0\}|$. Output $\tilde{F}_0 = 2^b \cdot \frac{\ln(1 - \frac{T}{K})}{\ln(1 - \frac{1}{K})}$.

- The algorithm outputs a value which is $(1 \pm \varepsilon)F_0$ with probability at least $11/20$ as long as $F_0 \geq \frac{K}{32}$.
Main Algorithm (2)

• **A**: keeps track of the amount of storage required to store all the $C_i$

• **est**: is such that $2^{est}$ is a $\Theta(1)$-approximation to $F_0$, and is obtained via Rough Estimator

• **b**: is such that we expect $F_0(t)/2^b$ to be $\Theta(K)$ at all points $t$ in the stream.
Main Algorithm (3)

- Subsample the stream at geometrically decreasing rates
- Perform balls and bins at each level
- When $i$ appears in stream, put a ball in cell $[g(i), h(i)]$
- For each column, store the largest row containing a ball
- Estimate based on these numbers

<table>
<thead>
<tr>
<th>H: {1, ..., n} → {1, ..., log n}</th>
<th>Pr[h(i) = j] = 1/2^j</th>
</tr>
</thead>
<tbody>
<tr>
<td>g: {1, ..., n} → {1, ..., 1/ε^2}</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram showing the main algorithm](image)
Prove Space Complexity

- The hash functions $h_1$, $h_2$ each require $O(\log n)$ bits to store
- The hash function $h_3$ takes $O(k \log K) = O(\log^2(\frac{1}{\varepsilon}))$ bits to store
- The value $b$ takes $O(\log \log n)$ bits
- The value $A$ never exceeds the total number of bits to store all counters, which is $O(\varepsilon^{-2} \log n)$, and thus $A$ can be represented in $O(\log \left(\frac{1}{\varepsilon}\right) + \log \log n)$ bits
- The counters $C_j$ never in total consume more than $O(\frac{1}{\varepsilon^2})$ bits by construction, since we output FAIL if they ever would
- The Rough Estimator and $est$ use $O(\log(n))$ bits

→ Total space complexity: $O(\varepsilon^{-2} + \log n)$
Prove Time Complexity

• Use high-performance hash functions (Siegel, Pagh and Pagh) which can be evaluated in $O(1)$ time

• Store column array in Variable-Length Array (Blandford and Blelloch). In column array, store offset from the base row and not absolute index giving $O(1)$ update time for a fixed base level

• Occasionally we need to update the base level and decrement offsets by 1
  – Show base level only increases after $\Theta(\varepsilon^{-2})$ updates, so can spread this work across these updates, so $O(1)$ worst-case update time (Use deamortization)
  – Copy the data structure, use it for performing this additional work so it doesn’t interfere with reporting the correct answer
  – When base level changes, switch to copy

• For reporting time, we can maintain $T$ during updates, and thus the reporting time is the time to compute a natural logarithm, which can be made $O(1)$ via a small lookup table
References

• **Blandford, Blelloch.** *Compact dictionaries for variable-length keys and data with applications.* ACM Transactions on Algorithms. 2008.


• **Pagh, Pagh.** *Uniform Hashing in Constant Time and Optimal Space.* SICOMP 2008.

• **Siegel.** *On Universal Classes of Uniformly Random Constant-Time Hash Functions.* SICOMP 2004.
Summary

• We introduced Streaming Algorithms

• Sampling Algorithms
  – Reservoir Sampling
  – Priority Sampling

• Sketch Algorithms
  – Bloom Filter
  – Count-Min Sketch

• Counting Distinct Elements
  – Flajolet-Martin Algorithm
  – Optimal Algorithm
Q & A