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 $A = <a_1, a_2, \ldots, a_m>$, where the elements of the sequence (called tokens) are drawn from the universe $[n] = \{1, 2, \ldots, n\}$
- 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 $\mathcal{A}(\sigma)$ = output of a randomized streaming algorithm $\mathcal{A}$ on input σ
- Let ϕ = function that $\mathcal{A}$ is supposed to compute
- We say the algorithm (ε, δ)-approximates ϕ if

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

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

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

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

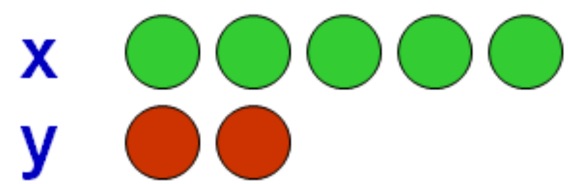
- Example: <x, 3>, <y, 2>, <x, 2> encodes the arrival of

  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

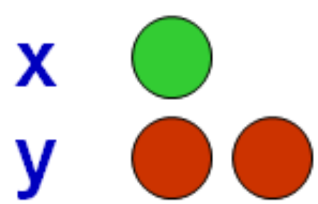
  Most general streaming model

  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

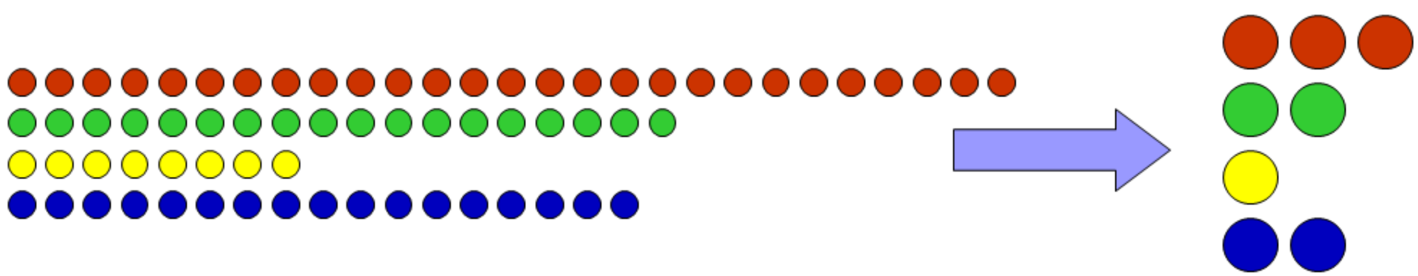
  A small random sample S of the data is often enough to represent all the data

- 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 $k/k + i$

- 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 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
$$(1/N)\ X\ (N/(N+1))\ =\ 1/N+1$$
- Probability that e_N survives $=\ 1-(1/(N+1))=\ N/(N+1)$
- 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
= (N/(N+1)) X ((N+1)/ (N+2))
- The probability of e_N surviving till the end
= (N/(N+1)) X ((N+1)/ (N+2)) X ……. X ((N_T -1)/ N_T) = N/ N_T

Probability of survival of the elements after the initial N

- For the last arriving element to be selected, the probability is N/ N_T
- For the element before the last, the probability of selection
- $= N/ (N_T -1)$
- The probability of the last element replacing the last but one element

  $= (N/ N_T) \times (1/N) = 1/ N_T$
- The probability that the last but one element survives $= 1- 1/ N_T = (N_T -1)/ N_T$
- The probability that the last but one survives till the end

  $= (N/(N_T -1)) \times (N_T -1)/ N_T = N/ N_T$

Similarly we can show that the probability of survival of any element in the sample is 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 $\geq$ 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

  $N + (N/N+1) + \ldots\ldots + (N/ N_T)$
- Overestimating it with an integral the reservoir size can be estimated as

$$\int_{x=N}^{x=NT} \frac{N\, dx}{x} = N \ln(NT/N)$$

- 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 mist 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 , …… 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:

➢C. R. Argon and R.G. Seidel, Randomised Search Trees, Proc of the 30th IEEE Symp on Foundations of Computer Science, 1989, pp 540-545

➢K. Mulmuley, Computational Geometry: An Introduction through Ramdomised Algorithms, Prentice Hall

References

- Crash course - http://people.cs.umass.edu/~mcgregor/slides/10-jhu1.pdf
- Notes
  - http://www.cs.mcgill.ca/~denis/notes09.pdf
  - http://www.cs.dartmouth.edu/~ac/Teach/CS49-Fall11/Notes/lecnotes.pdf
- http://en.wikipedia.org/wiki/Streaming_algorithm

- **Reservoir Sampling**
- Original Paper - http://www.mathcs.emory.edu/~cheung/papers/StreamDB/RandomSampling/1985-Vitter-Random-sampling-with-reservior.pdf
- 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
- Zhang Longbo, Li Zhanhuai, Zhao Yiqiang, Yu Min, Zhang Yang , A priority random sampling algorithm for time-based sliding windows over weighted streaming data , **SAC '07:** Proceedings of the 2007 ACM symposium on Applied computing, May 2007

Overview

- Introduction to Streaming Algorithms
- Sampling Techniques
- Sketching Techniques

Break

- Counting Distinct Numbers
- Q&A

Sketching

- Sketching is another general technique for processing stream

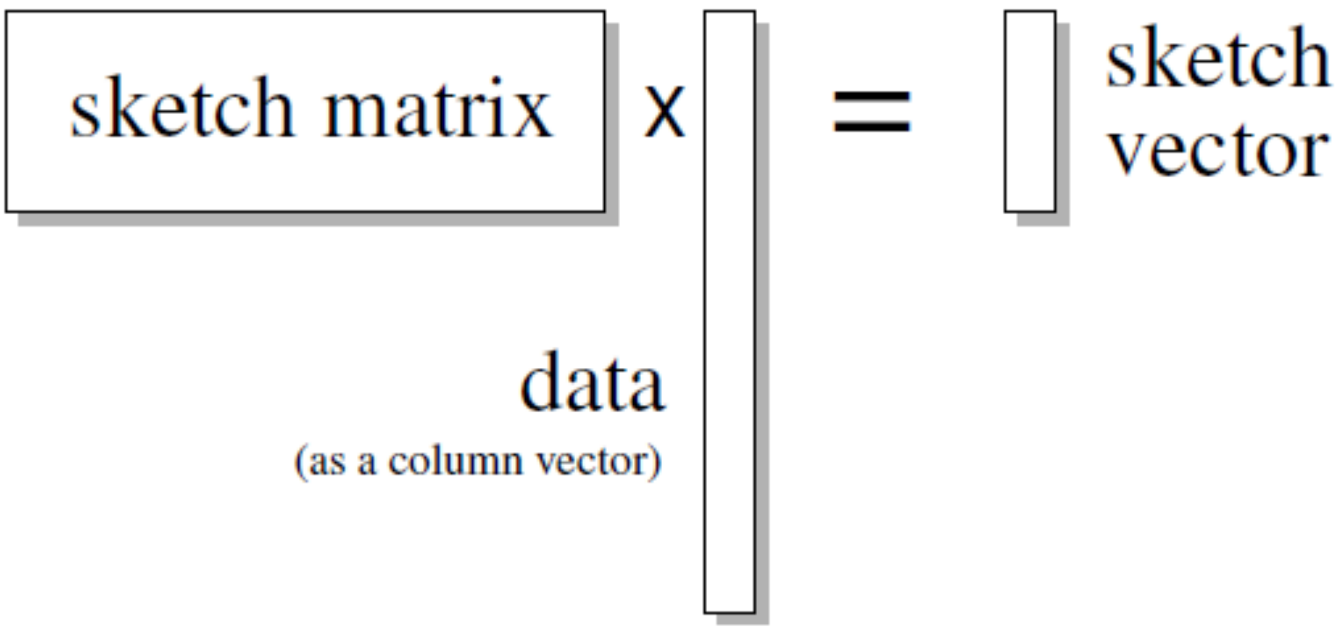


Fig: Schematic view of linear sketching

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

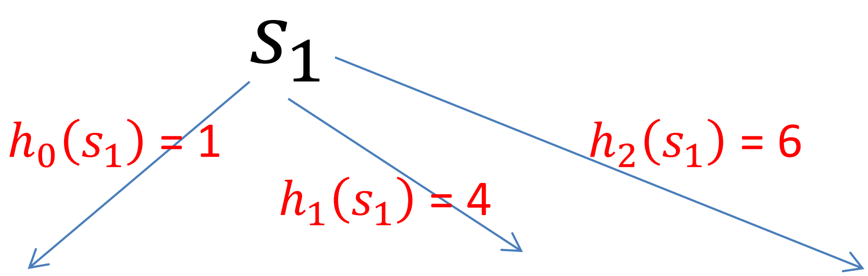
- Consists of
  - vector of n Boolean values, initially all set false
  - k independent hash functions, $h_0, h_1, \ldots, h_{\mathrm{k}-1}$, each with range {0, 1, … , n-1}

F	F	F	F	F	F	F	F	F	F
0	1	2	3	4	5	6	7	8	9

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.

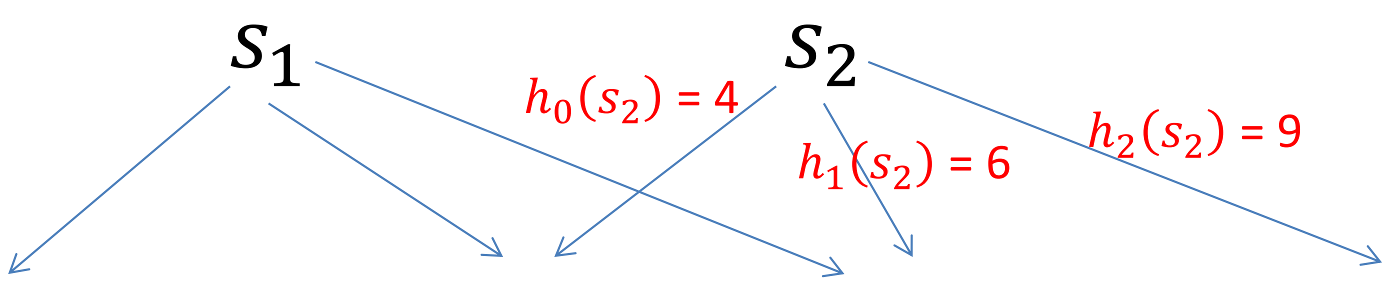


F	T	F	F	T	F	T	F	F	F
0	1	2	3	4	5	6	7	8	9

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.



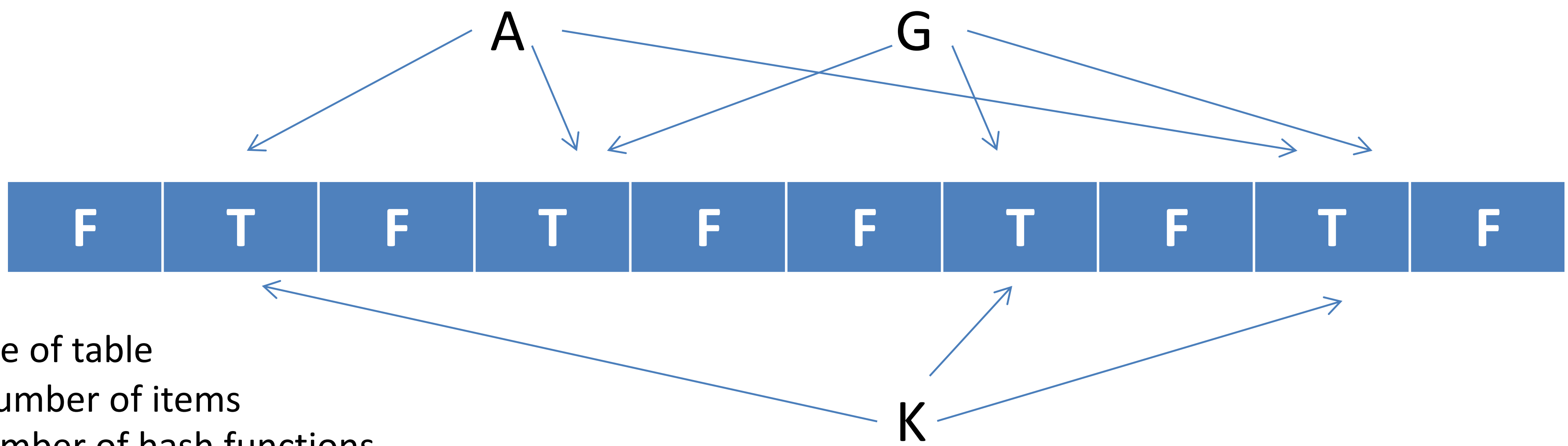
F	T	F	F	T	F	T	T	F	T
0	1	2	3	4	5	6	7	8	9

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



n = size of table
m = number of items
k = number of hash functions

Consider a particular bit 0 <= j <= n-1

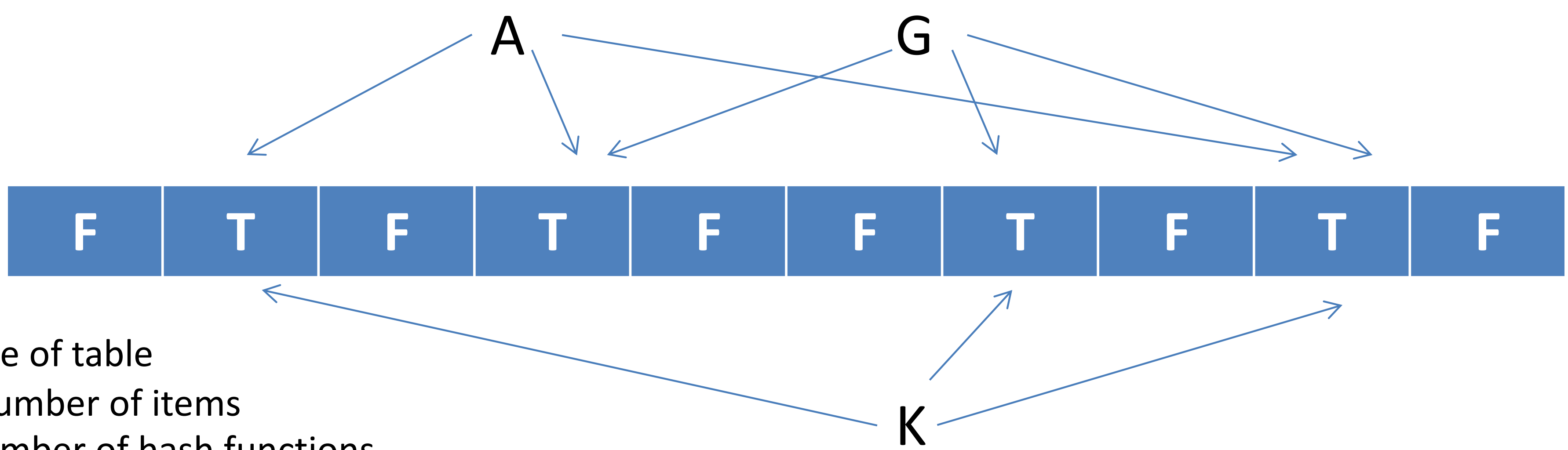
Probability that $h_i(x)$ does not set bit j: $P_{h_i \sim H}(h_i(x) \neq j) = \left(1 - \frac{1}{n}\right)$

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

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

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

Probability of false positives



n = size of table
m = number of items
k = number of hash functions

Probability of false positive = $\left(1 - e^{-km/n}\right)^{k}$

Note: All k bits of new element are already set

False positive probability can be minimized by choosing k = $\log_e(2) \cdot n/m$

Upper Bound Probability would be $\left(1 - e^{-\log_e(2)\cdot(n/m)\cdot(m/n)}\right)^{\log_e(2)\cdot n/m}$

$\Rightarrow (0 \cdot 5)^{\log_e(2)\cdot n/m}$

Bloom Filters: cons

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

References

- **Graham Cormode**, *Sketch Techniques for Approximate Query Processing*, ATT 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_∞ : 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 $\hat{a}_i = \min_j count[j, h_j(i)]$

- Guarantees error less than εF_1 in size $O(\lceil \frac{e}{\varepsilon} \rceil * \lceil \ln \frac{1}{\delta} \rceil)$
  – 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 (ε, δ) is represented by a two-dimensional array (a small summary of input) counts with width w and depth d : $count[1,1] \ldots count[d, w]$

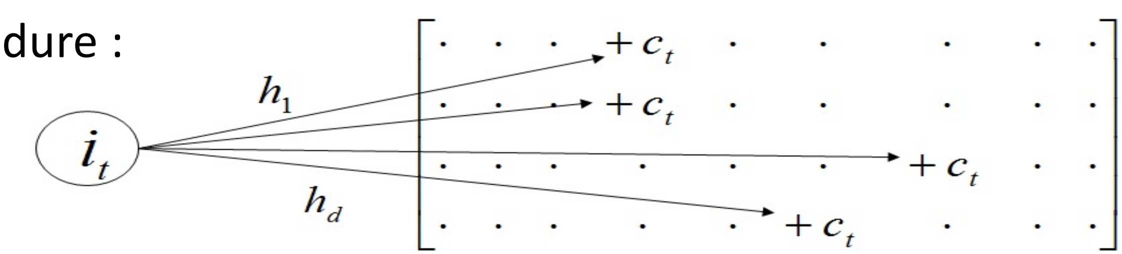
Given parameters (ε, δ), set $w = \left\lceil \frac{e}{\varepsilon} \right\rceil$ and $d = \left\lceil \ln \frac{1}{\delta} \right\rceil$ Each entry of the array is initially zero.

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

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 :



Count Min Sketch Algorithm

Initialize :

$t \leftarrow \log(1/\delta)$;

$k \leftarrow 2/\varepsilon$;

$C[1 \ldots t][1 \ldots k] \leftarrow \vec{0}$;

Pick t independent hash functions $h_1, h_2, \ldots, h_t : [n] \to [k]$, each from a 2-universal family ;

Process (j, c):

for $i = 1$ **to** t **do**

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

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

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

```
x: next element in data stream

for each hash function (h_k)
     v = h_k(x)
     update table_k[v] +1
```

Data Stream

3
145
99
84
12
.
.
.

Initialize :

$t \leftarrow \log(1/\delta)$;

$k \leftarrow 2/\varepsilon$;

$C[1\ldots t][1\ldots k] \leftarrow \vec{0}$;

Pick t independent hash functions $h_1, h_2, \ldots, h_t : [n] \rightarrow [k]$, each from a 2-universal family ;

Process (j, c):

for $i = 1$ **to** t **do**

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

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

Count-Min sketch

h_1	0	0	0	0	0	0	0	0	0	0
h_2	0	0	0	0	0	0	0	0	0	0
h_3	0	0	0	0	0	0	0	0	0	0
h_4	0	0	0	0	0	0	0	0	0	0

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 ($a_{i_t}(t) > 0$)

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

Theorem 1 $a_i \le \hat{a}_i$ $P[\hat{a}_i > a_i + \varepsilon \|\vec{a}\|_1] \le \delta$

PROOF : Introduce indicator variables

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

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

Define the variable $X_{i,j} = \sum_{k=1}^{n} I_{i,j,k} a_k$

By construction, $count[j, h_j(i)] = a_i + X_{i,j} \Longrightarrow \min count[j, h_j(i)] \ge a_i$

For the other direction, observe that

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

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

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

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

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

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

Remark : The constant e 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 $\xrightarrow{\text{(at most)}}$ $2\log_2 n$ <u>dyadic range queries</u> $\Rightarrow$ single point query
- For each set of dyadic ranges of length $2^y, \quad y = 0 \ldots \log_2 n - 1$ a sketch is kept

$\Rightarrow$ $\log_2 n$ CM Sketches

SELECT COUNT (*) FROM D
WHERE D.val >=l AND D.val <=h

$Q(l, r)$

Compute the dyadic ranges (at most $2\log_2 n$) which canonically cover the range $\Rightarrow$ Pose that many point queries to the sketches $\Rightarrow$ Sum of queries

$= \hat{a}[l, r]$

Range Sum Example

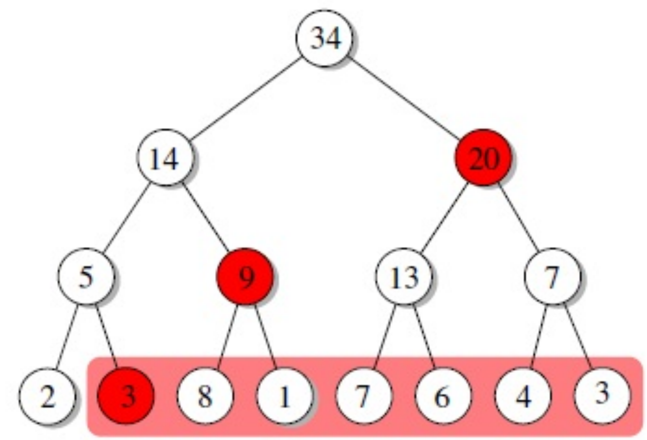
- AMS approach to this, the error scales proportional to $\sqrt{F_2(f)\, F_2(f')}$

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.



Using dyadic ranges to answer a range query

Theorem 4 $a[l,r] \le \hat{a}[l,r]$

$$\Pr[\hat{a}[l,r] > a[l,r] + 2\varepsilon \log n \|\vec{a}\|_1] \le \delta$$

Proof : **Theorem 1** $\xrightarrow{a_i \le \hat{a}_i}$ $a[l,r] \le \hat{a}[l,r]$

E(Σ error for each estimator) $= 2\log n$ E(error for each estimator)

$$\le 2\log n \frac{\varepsilon}{e} \|\vec{a}\|_1$$

$$\Pr[\hat{a}[l,r] - a[l,r] > 2\log n \|\vec{a}\|_1] < e^{-d} \le \delta$$

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 $\qquad (\vec{a} \hat{\cdot} \vec{b})_j = \sum_{k=1}^{w} count_{\vec{a}}[j,k] * count_{\vec{b}}[j,k]$

$Q(\vec{a}, \vec{b}) \quad \Rightarrow \quad (\vec{a} \hat{\cdot} \vec{b}) = \min_j (\vec{a} \hat{\cdot} \vec{b})_j$

Theorem 3 $\quad (\vec{a} \cdot \vec{b}) \le (\vec{a} \hat{\cdot} \vec{b}) \qquad \Pr[(\vec{a} \hat{\cdot} \vec{b}) > \vec{a} \cdot \vec{b} + \varepsilon \|\vec{a}\|_1 \|\vec{b}\|_1] \le \delta$

Analysis

Time to produce the estimate $\quad O(\frac{1}{\varepsilon} \log \frac{1}{\delta})$

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

Time for updates $\quad O(\log \frac{1}{\delta})$

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 $\varepsilon \|\vec{a}\|_1 \|\vec{b}\|_1$ 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**
  - G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. LATIN 2004, J. Algorithm 58-75 (2005) .
  - G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. SDM 2005.
  - G. Cormode and S. Muthukrishnan. Approximating data with the count-min data structure. IEEE Software, (2012).

- **Journal**
  - Alon, Noga; Matias, Yossi; Szegedy, Mario (1999), "The space complexity of approximating the frequency moments", *Journal of Computer and System Sciences* **58** (1): 137–147.

- **Surveys**
  - Network Applications of Bloom Filters: A Survey. Andrei Broder and Michael Mitzenmacher. Internet Mathematics Volume 1, Number 4 (2003), 485-509.
  - Article from "Encyclopedia of Database Systems" on Count-Min Sketch Graham Cormode 09. 5 page summary of the sketch and its applications.
  - A survey of synopsis construction in data streams. Charu Aggarwal.

- **Coverage in Textbooks**
  - Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Michael Mitzenmacher, Eli Upfal. Cambridge University Press, 2005. Describes Count-Min sketch over pages 329--332
  - Internet Measurement: Infrastructure, Traffic and Applications. Mark Crovella, Bala Krishnamurthy. Wiley 2006.

- **Tutorials**
  - Advanced statistical approaches for network anomaly detection. Christian Callegari. ICIMP 10 Tutorial.
  - Video explaining sketch data structures with emphasis on CM sketch Graham Cormode.

- **Lectures**
  - Data Stream Algorithms. Notes from a series of lectures by S. Muthu Muthukrishnan.
  - Data Stream Algorithms. Lecture notes, Chapter 3. Amit Chakrabarti. Fall 09.
  - Probabilistic inequalities and CM sketch. John Byers. Fall 2007.

Overview

- Introduction to Streaming Algorithms
- Sampling Techniques
- Sketching Techniques

Break

- Counting Distinct Numbers
- Q&A

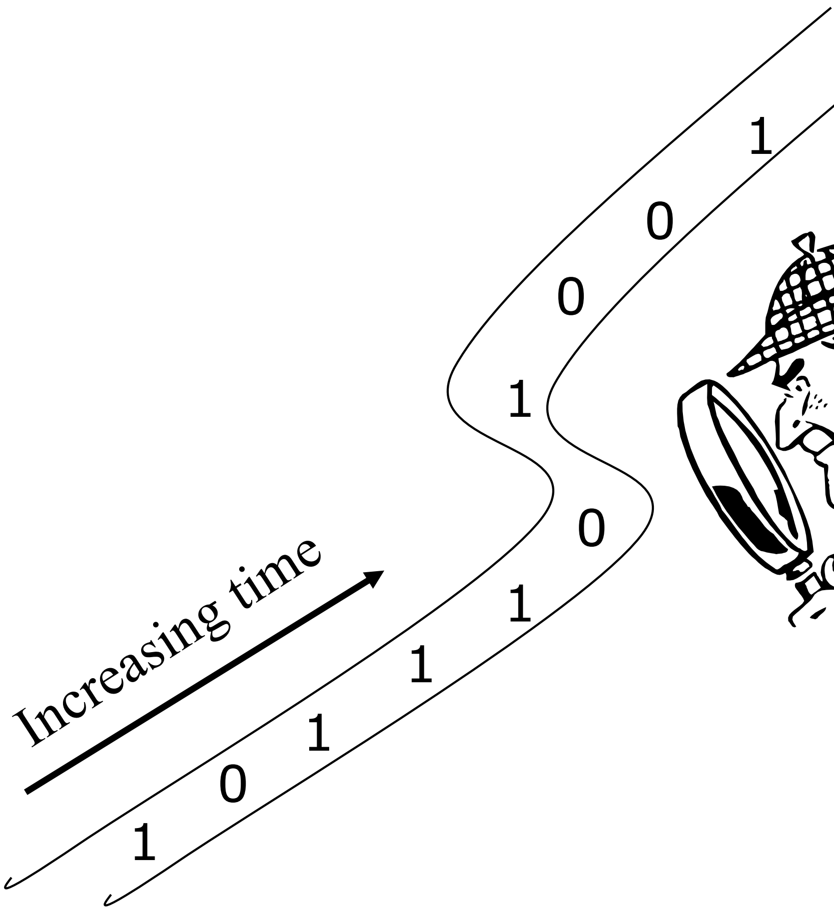
Overview

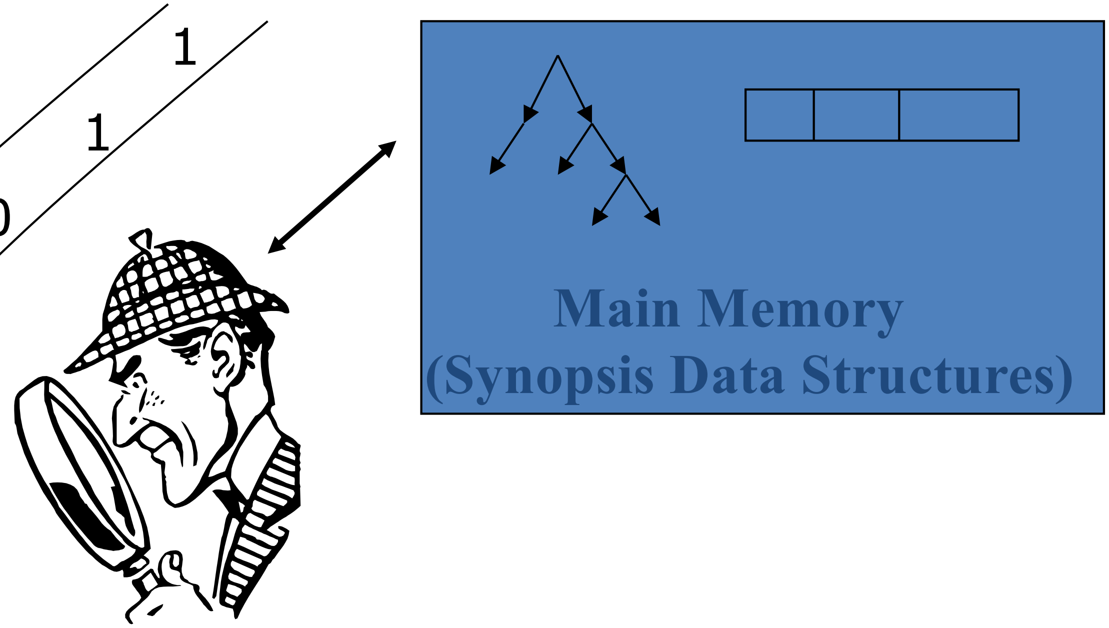
- Introduction to Streaming Algorithms
- Sampling Techniques
- Sketching Techniques

Break

- Counting Distinct Numbers
- Q&A

Stream Model of Computation





Data Stream

Memory: poly(1/ε, log N)

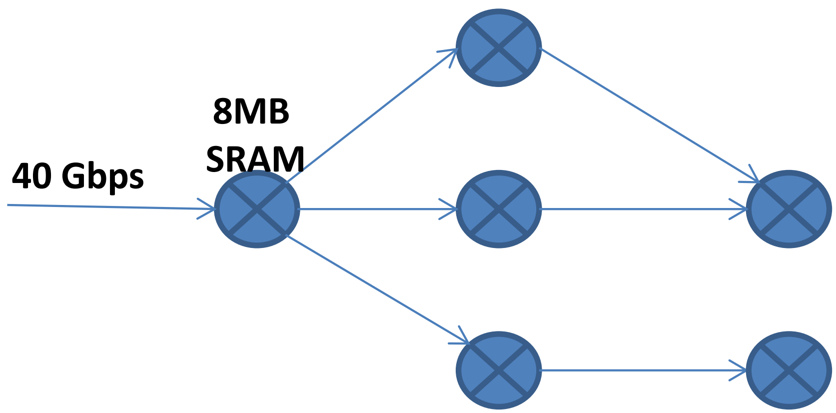
Query/Update Time: poly(1/ε, log N)

N: # items so far, or window size

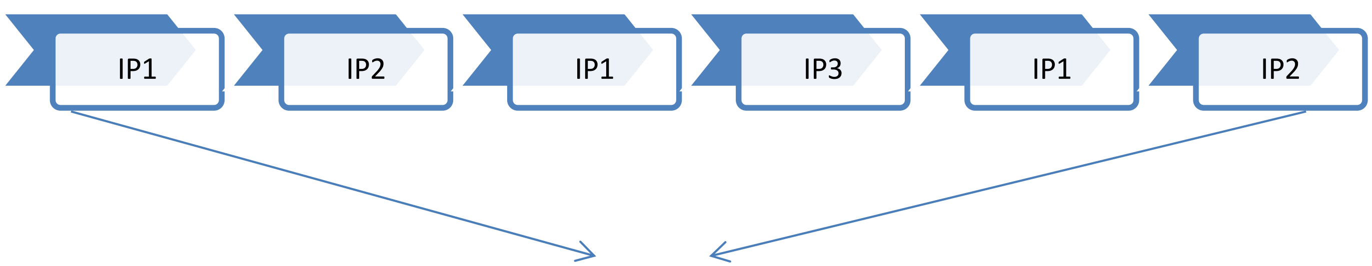
ε: error parameter

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 $X = < x_1, x_2, \ldots \ldots, x_m > \in [n]^m$ of values. Let $F0$ be the number of distinct elements in X. Find $F0$ 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 $\boldsymbol{\varepsilon} < \mathbf{1}$ and high proability

Naïve Approach

- Counter C(i) for each domain value i in [n]
- Initialize counters C(i)← 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 ε in random oracle model
- Alon, Matias and Szegedy
  - $O(\log n)$ space/update time for fixed ε 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 bit(y, k) = k^{th} bit in the binary representation of y

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

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

$$\rho(y) = L \qquad \textit{if } y = 0$$

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

Flajolet-Martin Approach

```
for (i:=0 to L-1) do BITMAP[i]:=0;
for (all x in M) do
    begin
        index:=ρ(h(x));
        if BITMAP[index]=0 then
                BITMAP[index]:=1;
    end
R := the largest index in BITMAP whose value equals to 1
Estimate := 2^R
```

Examples of bit(*y*, *k*) & ρ(*y*)

- $y=10=(1010)_2$
  - $bit(y,0)=0$ $bit(y,1)=1$ $bit(y,2)=0$ $bit(y,3)=1$
  - $$y=\sum_{k\geq 0} bit(y,k)\cdot 2^k$$

int *y*	binary format	ρ(*y*)
0	0000	4 (=*L*)
1	0001	0
2	0010	1
3	0011	0
4	0100	2
5	0101	0
6	0110	1
7	0111	0
8	1000	3

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 $F0 \approx \log_2 \varphi n$ 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 $1 - (1 - \boxed{2^{-r}})^m$.

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 $2^r >> m$, $1 - (1 - 2^{-r})^m \approx 1 - (1 - m2^{-r})$ $\approx m/2^r \approx 0$.

  First 2 terms of the Taylor expansion of e^x
- If $2^r << m$, $1 - (1 - 2^{-r})^m \approx 1 - e^{-m2^{-r}} \approx 1$.
- Thus, 2^R will almost always be around m.

Algorithm History

- Flajolet and Martin introduced problem
  - $O(\log n)$ space for fixed ε in random oracle model
- Alon, Matias and Szegedy
  - $O(\log n)$ space/update time for fixed ε 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=Θ(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 → 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 $O\left(\frac{log\frac{K}{\varepsilon}}{loglog\frac{K}{\varepsilon}}\right)$ - 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 ± ε)

Rough Estimator (RE)

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

- 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 ∈ [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 $A, b, \mathrm{est} = 0$.
5. Run an instantiation RE of RoughEstimator.
6. **Update(i):** Set $x \leftarrow \max\{C_{h_3(h_2(i))}, \mathrm{lsb}(h_1(i)) - b\}$.
   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 $R > 2^{\mathrm{est}}$:

   (a) $\mathrm{est} \leftarrow \log(R)$, $b_{\mathrm{new}} \leftarrow \max\{0, \mathrm{est} - \log(K/32)\}$.

   (b) For each $j \in [K]$, set $C_j \leftarrow \max\{-1, C_j + b - b_{\mathrm{new}}\}$

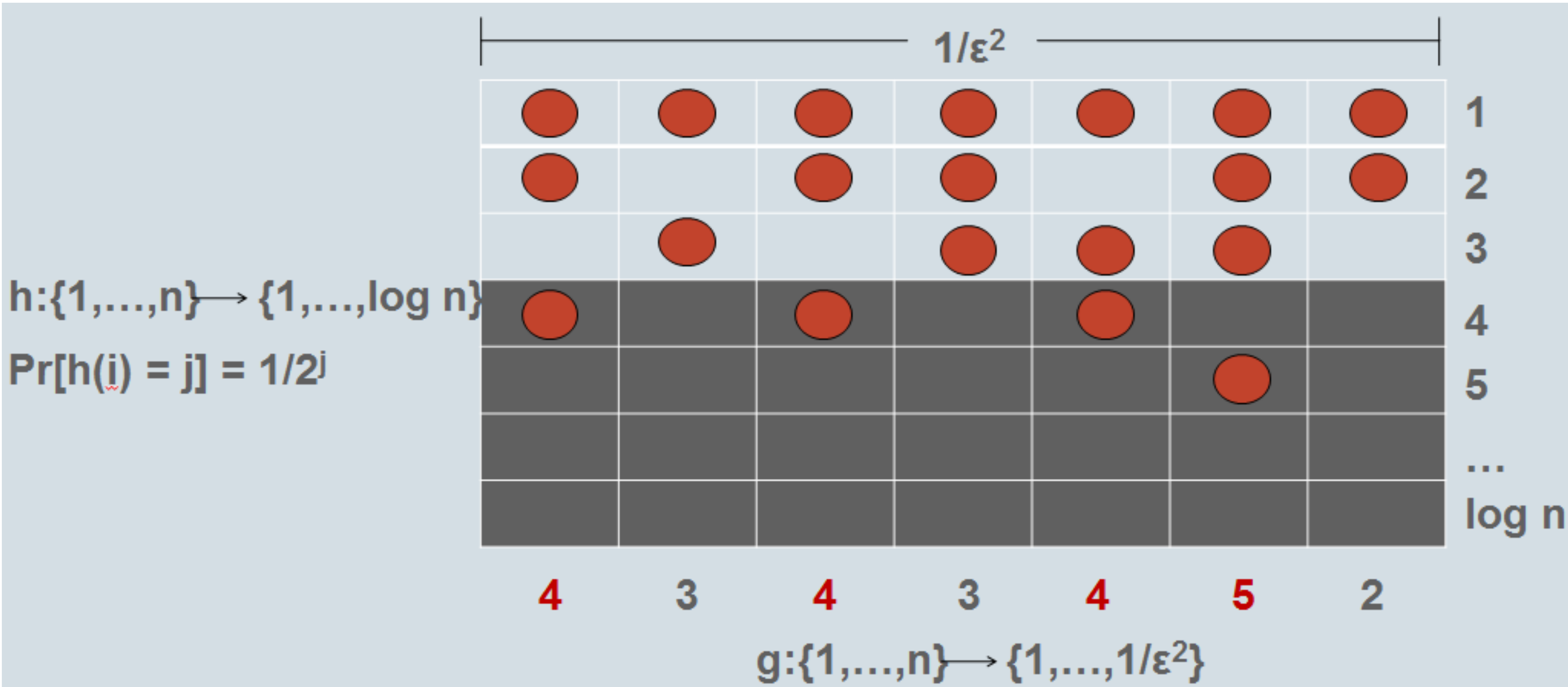
   (c) $b \leftarrow b_{\mathrm{new}}$, $A \leftarrow \sum_{j=1}^{K} \lceil \log(C_j + 2) \rceil$.
7. **Estimator:** Define $T = |\{j : C_j \geq 0\}|$. Output $\widetilde{F}_0 = 2^b \cdot \frac{\ln(1-\frac{T}{K})}{\ln(1-\frac{1}{K})}$.

- 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

Prove Space Complexity

- The hash functions h1, h2 each require $\mathrm{O}(\log n)$ bits to store
- The hash function h3 takes $\mathrm{O}(klogK)$= $\mathrm{O}(log^2(\frac{1}{\varepsilon}))$ bits to store
- The value b takes $\mathrm{O}(loglogn)$ bits
- The value A never exceeds the total number of bits to store all counters, which is $\mathrm{O}(\varepsilon^{-2}logn)$, and thus A can be represented in $\mathrm{O}(\log\left(\frac{1}{\varepsilon}\right) + loglogn)$ bits
- The counters C_j never in total consume more than $\mathrm{O}(\frac{1}{\varepsilon^2})$ bits by construction, since we output FAIL if they ever would
- The Rough Estimator and *est* use O(log(n)) bits

→ Total space complexity: $\mathrm{O}(\varepsilon^{-2} + logn)$

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.
- **D. M. Kane, J. Nelson, and D. P. Woodruff**. *An optimal algorithm for the distinct elements problem*. In Proc. 29th ACM Symposium on Principles of Database Systems, pages 41-52. 2010.
- **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