CS60021: Scalable Data Mining

Stream Mining

Sourangshu Bhattacharya

Filtering Data Streams

Filtering Data Streams

- **Each element of data stream is a tuple**
- Given a list of keys **S**
- **Determine which tuples of stream are in *S***

- **Obvious solution: Hash table**
  – But suppose we **do not have enough memory** to store all of ***S*** in a hash table
    - E.g., we might be processing millions of filters on the same stream

Applications

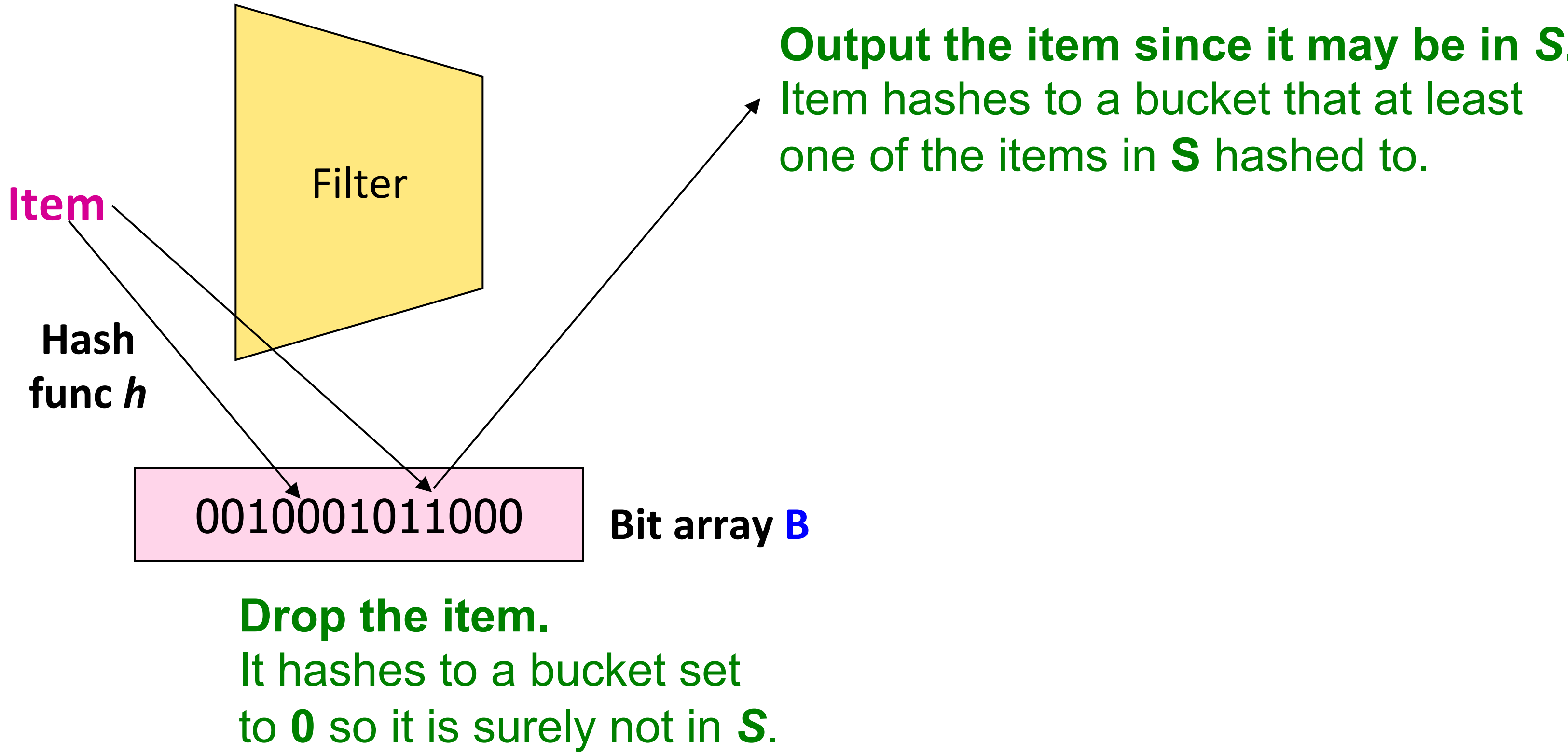
- **Example: Email spam filtering**
  – We know 1 billion "good" email addresses
  – If an email comes from one of these, it is **NOT** spam

- **Publish-subscribe systems**
  – You are collecting lots of messages (news articles)
  – People express interest in certain sets of keywords
  – Determine whether each message matches user's interest

First Cut Solution (1)

- **Given a set of keys *S* that we want to filter**
- Create a **bit array *B*** of ***n*** bits, initially all ***0*s**
- Choose a **hash function *h*** with range **[*0,n*)**
- Hash each member of ***s*∈*S*** to one of ***n*** buckets, and set that bit to **1**, i.e., ***B[h(s)]=1***
- Hash each element ***a*** of the stream and output only those that hash to bit that was set to **1**
  - **Output *a* if B[h(a)] == 1**

First Cut Solution (2)



- **Creates false positives but no false negatives**
  - If the item is in ***S*** we surely output it, if not we may still output it

First Cut Solution (3)

- **|S| = 1 billion email addresses**
  |B|= 1GB = 8 billion bits

- If the email address is in ***S***, then it surely hashes to a bucket that has the big set to **1**,
  so it always gets through (***no false negatives***)

- Approximately **1/8** of the bits are set to **1**, so about **1/8th** of the addresses not in ***S*** get through to the output (***false positives***)
  - Actually, less than **1/8th**, because more than one address might hash to the same bit

Analysis: Throwing Darts (1)

- **More accurate analysis for the number of false positives**

- **Consider:** If we throw ***m*** darts into ***n*** equally likely targets, **what is the probability that a target gets at least one dart?**

- **In our case:**
  - **Targets** = bits or buckets
  - **Darts** = hash values of items

<u>Analysis:</u> Throwing Darts (2)

- We have ***m*** darts, ***n*** targets
- **What is the probability that a target gets at least one dart?**

Equals **1/e** as **n** $\rightarrow \infty$

Equivalent

$$\boxed{1 - \underline{\boxed{(1 - 1/n)}^{\,n(m/n)}}}$$

$$\mathbf{1 - e^{-m/n}}$$

Probability some target **X** not hit by a dart

Probability at least one dart hits target **X**

Analysis: Throwing Darts (3)

- **Fraction of 1s in the array B =**
  = probability of false positive = $1 - e^{-m/n}$

- **Example: 10^9** darts, **$8 \cdot 10^9$** targets
  - Fraction of **1s** in **B = $1 - e^{-1/8}$ = 0.1175**
    - Compare with our earlier estimate: **1/8 = 0.125**

Bloom Filter

- Consider: $|S| = m$, $|B| = n$
- Use k independent hash functions $h_1, \ldots, h_k$
- **Initialization:**
  - Set **B** to all **0s**
  - Hash each element $s \in S$ using each hash function h_i, set $B[h_i(s)] = 1$ (for each $i = 1,.., k$)

(**note:** we have a single array B!)

- **Run-time:**
  - When a stream element with key x arrives
    - If $B[h_i(x)] = 1$ for all $i = 1, \ldots, k$ then declare that x is in S
      - That is, x hashes to a bucket set to **1** for every hash function $h_i(x)$
    - Otherwise discard the element x

Bloom Filter -- Analysis

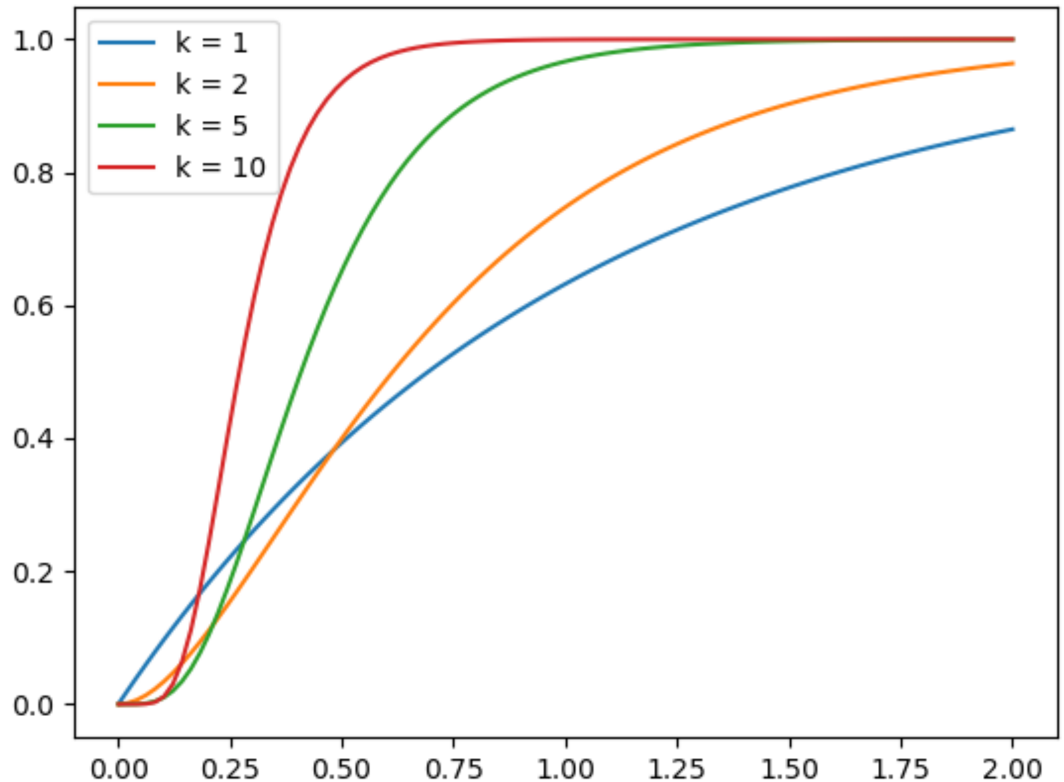
- **What fraction of the bit vector B are 1s?**
  - Throwing $k \cdot m$ darts at n targets
  - So fraction of **1**s is $(1 - e^{-km/n})$

- But we have k independent hash functions and we only let the element x through **if all** k hash element x to a bucket of value **1**

- So, false **positive probability** = $(1 - e^{-km/n})^k$

Bloom filter analysis

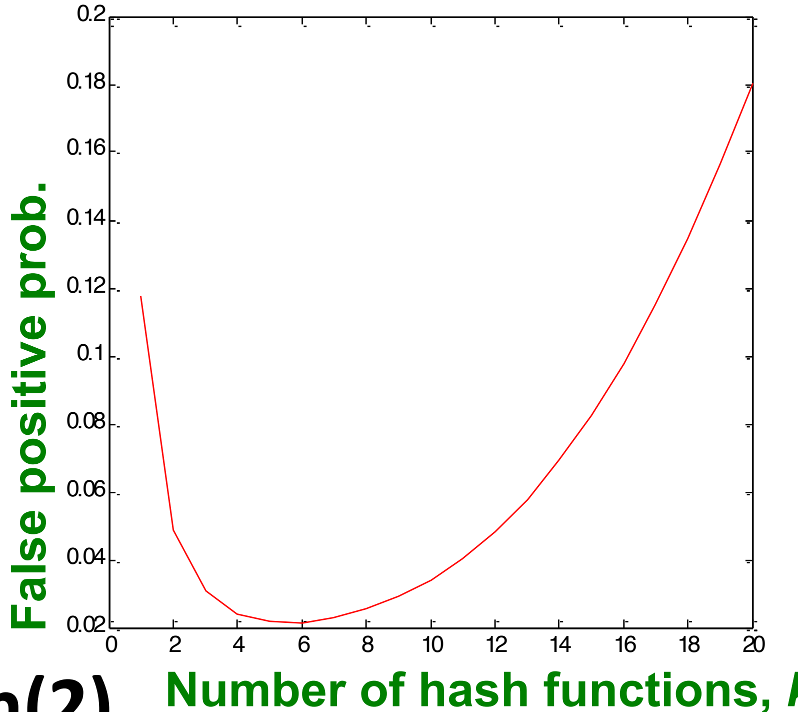
Prob. Of
false positive



m/n

Bloom Filter – Analysis (2)

- ***m* = 1 billion, *n* = 8 billion**
  - **k = 1**: $(1 - e^{-1/8})$ = **0.1175**
  - **k = 2**: $(1 - e^{-1/4})^2$ = **0.0493**



- **What happens as we keep increasing *k*?**

- "Optimal" value of ***k***: ***n/m* ln(2)**
  - **In our case:** Optimal **k = 8 ln(2) = 5.54 ≈ 6**
    - **Error at k = 6**: $(1 - e^{-1/6})^2$ = **0.0235**

Bloom Filter: Wrap-up

- **Bloom filters guarantee no false negatives, and use limited memory**
  - Great for pre-processing before more expensive checks
- **Suitable for hardware implementation**
  - Hash function computations can be parallelized

- Is it better to have **1** big **B** or ***k*** small **B**s?
  - **It is the same:** $(1 - e^{-km/n})^k$ vs. $(1 - e^{-m/(n/k)})^k$
  - But keeping **1 big B** is simpler

(2) Counting Distinct Elements

Counting Distinct Elements

- **Problem:**
  - Data stream consists of a universe of elements chosen from a set of size ***N***
  - Maintain a count of the number of distinct elements seen so far

- **Obvious approach:**
  Maintain the set of elements seen so far
  - That is, keep a hash table of all the distinct elements seen so far

Applications

- **How many different words are found among the Web pages being crawled at a site?**
  - Unusually low or high numbers could indicate artificial pages (spam?)

- **How many different Web pages does each customer request in a week?**

- **How many distinct products have we sold in the last week?**

Using Small Storage

- **Real problem: What if we do not have space to maintain the set of elements seen so far?**
- **Estimate the count in an unbiased way**
- **Accept that the count may have a little error, but limit the probability that the error is large**

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 **0s** in ***h*(*a*)**
  - **r(a)** = position of first 1 counting from the right
    - E.g., say ***h(a) = 12***, then ***12*** is ***1100*** in binary, so ***r(a) = 2***
- Record ***R* = the maximum *r*(*a*) seen**
  - $R = \max_a r(a)$, over all the items ***a*** seen so far
- **Estimated number of distinct elements =** 2^R

Why It Works: Intuition

- **Very very rough and heuristic intuition why Flajolet-Martin works:**
  - ***h(a)*** hashes ***a*** with **equal prob.** to any of ***N*** values
  - Then ***h(a)*** is a sequence of **$\log_2$ N** bits, where ***2^{-r}*** fraction of all ***a***s have a tail of ***r*** zeros
    - About 50% of ***a***s hash to *****0**
    - About 25% of ***a***s hash to ****00**
    - So, if we saw the longest tail of ***r=2*** (i.e., item hash ending ***100**) then we have probably seen **about *4*** distinct items so far
  - **So, it takes to hash about *2^r* items before we see one with zero-suffix of length *r***

Why It Works: More formally

- **Now we show why Flajolet-Martin works**

- **Formally, we will show that probability of finding a tail of *r* zeros:**
  - **Goes to 1 if $m \gg 2^r$**
  - **Goes to 0 if $m \ll 2^r$**

  where m is the number of distinct elements seen so far in the stream

- **Thus, 2^R will almost always be around *m*!**

Why It Works: More formally

- **What is the probability that a given *h*(*a*) ends in at least *r* zeros is 2^{-r}**
  - **h(a)** hashes elements uniformly at random
  - Probability that a random number ends in at least ***r*** zeros is $\mathbf{2^{-r}}$
- Then, the probability of **NOT** seeing a tail of length ***r*** among ***m*** elements:

$$(1-2^{-r})^{m}$$

Prob. all end in fewer than ***r*** zeros.

Prob. that given **h(a)** ends in fewer than ***r*** zeros

Why It Works: More formally

- **Note:** $(1-2^{-r})^m = (1-2^{-r})^{2^r(m2^{-r})} \approx e^{-m2^{-r}}$
- **Prob. of NOT finding a tail of length *r* is:**
  - If ***m << 2^r***, then prob. tends to **1**
    - $(1-2^{-r})^m \approx e^{-m2^{-r}} = 1$ as **m/2^r → 0**
    - So, the probability of finding a tail of length ***r*** tends to **0**
  - If ***m >> 2^r***, then prob. tends to **0**
    - $(1-2^{-r})^m \approx e^{-m2^{-r}} = 0$ as **m/2^r → ∞**
    - So, the probability of finding a tail of length ***r*** tends to **1**
- **Thus, 2^R will almost always be around *m*!**

Why It Doesn't Work

- **E[2^R] is actually infinite**
  - Probability halves when $R \rightarrow R+1$, but value doubles
- **Workaround involves using many hash functions h_i and getting many samples of R_i**
- **How are samples R_i combined?**
  - **Average?** What if one very large value 2^{R_i} ?
  - **Median?** All estimates are a power of **2**
  - **Solution:**
    - Partition your samples into small groups
    - Take the median of groups
    - Then take the average of the medians

(3) Computing Moments

Generalization: Moments

- **Suppose a stream has elements chosen from a set *A* of *N* values**

- **Let m_i be the number of times value *i* occurs in the stream**

- **The k^{th} *moment* is**

$$\sum_{i \in A} (m_i)^k$$

Special Cases

$$\sum_{i \in A} (m_i)^k$$

- **0th moment =** number of distinct elements
  - The problem just considered
- **1st moment =** count of the numbers of elements = length of the stream
  - Easy to compute
- **2nd moment =** ***surprise number S*** **=** a measure of how uneven the distribution is

Example: Surprise Number

- **Stream of length 100**
- **11 distinct values**

- Item counts: **10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9**
  Surprise *S* = 910

- Item counts: **90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1**
  Surprise *S* = 8,110

AMS method

- AMS method works for all moments
- Gives an unbiased estimate.
- We will just concentrate on the 2^{nd} moment S.
- We pick and keep track of many variables X:
  - For each variable X, store X.el and X.val
    - X.el corresponds to the item I
    - X.val corresponds to the count of item I
  - Note this requires a count in main memory, so number of Xs is limited
- Our goal is to compute $S = \Sigma_i m_i^2$

One random variable (X)

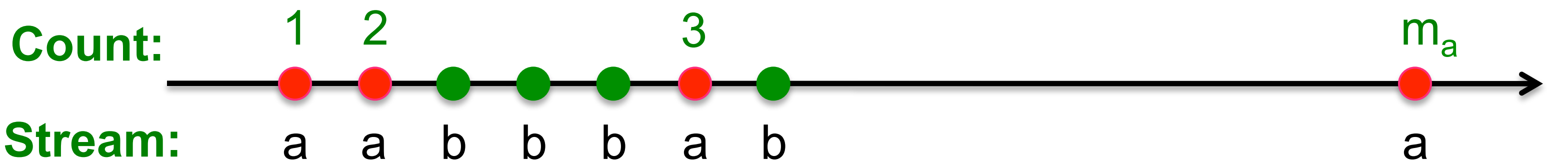
- How to set X.val and X.el ?
  - Assume stream has length n (we relax this later)
  - Pick some random time **t (t<n)** to start, so that any time is equally likely
  - Let at time t the stream have item **i**. We set ***X.el = i***
  - Then we maintain count **c** (***X.val = c***) of the number of is in the stream starting from the chosen time **t**
- Then the estimate of the 2nd moment ($\Sigma_i m_i^2$) is:

$$S = f(X) = n\,(2\,c - 1)$$

  - Note, we will keep track of multiple **Xs, (X_1, X_2,… X_k)** and our final estimate will be:

$$S = 1/k\ \Sigma_j f(X_j)$$

Expectation Analysis



- 2nd moment is $\mathbf{S = \Sigma_i m_i^2}$
- $\mathbf{C_t}$ - number of times item at time **t** appears from time **t** onwards $\mathbf{(c_1=m_a\ ,\ c_2=m_a-1,\ c_3=m_b)}$
- $E[f(X)] = 1/n\ \Sigma_{t=1}^{n}\ n\ (2c_t - 1)$
  $= 1/n\ \Sigma_i\ n\ (1 + 3 + 5 + \cdots + 2\ m_i - 1)$

m_i … total count of item i in the stream (we are assuming stream has length **n**)

Group times by the value seen

Time t when the last i is seen (c_t=1)

Time t when the penultimate i is seen (c_t=2)

Time t when the first i is seen (c_t=m_i)

Higher-Order Moments

- For estimating kth moment we essentially use the same algorithm but change the estimate:
  - For **k=2** we used $\mathbf{n\ (2 \cdot c - 1)}$
    For **k=3** we use: $\mathbf{n\ (3 \cdot c^2 - 3c + 1)}$ (where c=X.val)
- Why?
  - For **k=2**: Remember we had $\mathbf{(1+3+5+\cdots+(2m_i-1))}$ and we showed terms **2c-1 (for c=1,...,m)** sum to $\mathbf{m^2}$
  - $2c - 1 = c^2 - (c-1)^2$
  - For **k=3**: $c^3 - (c-1)^3 = 3c^2 - 3c + 1$
- Generally: Estimate = $\mathbf{n\ (c^k - (c-1)^k)}$

Combining Samples

- **In practice:**
  - Compute $f(X) = n(2c - 1)$ for as many variables X as you can fit in memory
  - Average them in groups
  - Take median of averages
- **Problem: Streams never end**
  - We assumed there was a number n, the number of positions in the stream
  - But real streams go on forever, so n is a variable – the number of inputs seen so far

Streams Never End: Fixups

- **(1)** The variables ***X*** have ***n*** as a factor – keep ***n*** separately; just hold the count in ***X***
- **(2)** Suppose we can only store ***k*** counts. We must throw some ***X***s out as time goes on:
  - **Objective:** Each starting time ***t*** is selected with probability ***k/n***
  - **Solution: (fixed-size sampling!)**
    - Choose the first ***k*** times for ***k*** variables
    - When the ***n***th element arrives (***n*** > ***k***), choose it with probability ***k/n***
    - If you choose it, throw one of the previously stored variables **X** out, with equal probability

AMS ALGORITHM USING SKETCHES

Generalization of AMS Algorithm

- Stream of pair (i,c), i € {1,⋯,U} and c is positive integer.
- $\boldsymbol{x[i] = x[i] + c}$ for each update
- Join size: $\boldsymbol{x.y = \Sigma_{i=1}^{U} (x[i]\, y[i])}$
- Pth Moment: $\boldsymbol{F_P(x) = \Sigma_{i=1}^{U} x[i]^2}$

$$\|\boldsymbol{x} - \boldsymbol{y}\|_2 = \sqrt{F_2(\boldsymbol{x} - \boldsymbol{y})}.$$

- h : {1,⋯U} ➔ {+1,-1}

Generalization of AMS Algorithm

UPDATE(i, c, z)
Input: item i, count c, sketch z

for $j = 1$ to w **do**
  for $k = 1$ to d **do**
    $z[j][k] += h_{j,k}(i) * c$

ESTIMATE$F_2(z)$
Input: sketch z

Return ESTIMATEJS(z, z)

ESTIMATEJS(x, y)
Input: sketch x, sketch y
Output: estimate of $\boldsymbol{x} \cdot \boldsymbol{y}$

for $j = 1$ to w **do**
  $avg[j] = 0$;
  for $k = 1$ to d **do**
    $avg[j] += x[j][k] * y[j][k]/w$;
Return(median(avg))

Fig. 1 AMS algorithm for estimating join and self-join size

Generalization of AMS Algorithm

Lemma 1 $\mathsf{E}(Z^2) = F_2(\boldsymbol{x})$

Proof

$$\begin{aligned}
\mathsf{E}(Z^2) &= \mathsf{E}\left(\left(\sum_{i=1}^{U} h(i)\boldsymbol{x}[i]\right)^2\right) \\
&= \mathsf{E}\left(\sum_{i=1}^{U} h(i)^2\boldsymbol{x}[i]^2\right) + \mathsf{E}\sum_{1\le i<j\le U} 2h(i)h(j)\boldsymbol{x}[i]\boldsymbol{x}[j] \\
&= \sum_{i=1}^{U}\boldsymbol{x}[i]^2 + 0 = F_2(\boldsymbol{x}).
\end{aligned}$$

Generalization of AMS Algorithm

- ***$Var(Z^2) \leq 2F_2(x)^2$***

$$\begin{aligned}\mathsf{Var}(Z^2) &= \mathsf{E}(Z^4) - \mathsf{E}(Z^2)^2 \\ &= \mathsf{E}\left(\left(\sum_{i=1}^{U} h(i)\boldsymbol{x}[i]\right)^4\right) - \left(\sum_{i=1}^{U} \boldsymbol{x}[i]^2\right)^2\end{aligned}$$

Generalization of AMS Algorithm

$$= \mathsf{E}\Bigg(\Bigg(\sum_{i=1}^{U} h(i)^4 \boldsymbol{x}[i]^4 + \sum_{1\le i<j\le U} 6h(i)^2 h(j)^2 \boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2$$

$$+ \sum_{i,i\neq j\neq k} 12h(i)^2 h(j)h(k)\boldsymbol{x}[i]^2 \boldsymbol{x}[j]\boldsymbol{x}[k]$$

$$+ \sum_{1\le i\neq j\le U} 4h^3(i)h(j)\boldsymbol{x}[i]^3 \boldsymbol{x}[j]$$

$$+ \sum_{1\le i<j<k<l\le U} 12h(i)h(j)h(k)h(l)\boldsymbol{x}[i]\boldsymbol{x}[j]\boldsymbol{x}[k]\boldsymbol{x}[l]\Bigg)$$

$$-\Bigg(\sum_{i=1}^{U} \boldsymbol{x}[i]^4 + \sum_{1\le i<j\le U} 2\boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2\Bigg)$$

Generalization of AMS Algorithm

$$= \sum_{i=1}^{U} \boldsymbol{x}[i]^4 + \sum_{1 \le i < j \le U} 6\boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2$$

$$- \left(\sum_{i=1}^{U} \boldsymbol{x}[i]^4 + \sum_{1 \le i < j \le U} 2\boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2 \right)$$

$$= 4 \sum_{1 \le i < j \le U} \boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2 \Bigg) \le 2F_2^2.$$

Generalization of AMS Algorithm

Fact 1 (Variance Reduction) *Let X_i be independent and identically distributed random variables. Then*

$$\mathsf{Var}\left(\sum_{i=1}^{w} \frac{X_i}{w}\right) = \frac{1}{w}\mathsf{Var}(X_1).$$

Fact 2 (The Chebyshev Inequality) *Given a random variable X,*

$$\mathsf{Pr}\left[\left|X - \mathsf{E}(X)\right| \geq k\right] \leq \frac{\mathsf{Var}(X)}{k^2}.$$

Generalization of AMS Algorithm

Theorem 1 *An (ϵ, δ)-approximation of F_2, the self-join size, can be computed in space $O(\frac{1}{\epsilon^2} \log 1/\delta)$ machine words in the streaming model. Each update takes time $O(\frac{1}{\epsilon^2} \log 1/\delta)$.*

Proof Applying the Chebyshev inequality to the average of $w = \frac{16}{\epsilon^2}$ copies of the estimate Z generates a new estimate Y such that

$$\Pr\left[|Y - F_2| \leq \epsilon F_2\right] \leq \frac{\mathsf{Var}(Y)}{\epsilon^2 F_2^2} = \frac{\mathsf{Var}(Z)}{c\epsilon^2 F_2^2} = \frac{2F_2^2}{(16/\epsilon^2)\epsilon^2 F_2^2} = \frac{1}{8}.$$

Generalization of AMS Algorithm

Fact 3 (Application of Chernoff Bounds) *Let R be a range of values $R = [R_{\min}..R_{\max}]$, and let Y_i be $d = 4\log 1/\delta$ independent and identically distributed random variable such that* $\Pr[Y_i \notin R] \leq \frac{1}{8}$. *Then*

$$\Pr\left[\left(\text{median}_{i=1}^{d} Y_i\right) \notin R\right] \leq \delta,$$

that is, if there is constant probability that each Y_i falls within the desired range R, then taking the median of $O(\log 1/\delta)$ copies of Y_i reduces the failure probability to δ.

Hence, applying the Chernoff bound result from Fact 3 to the median of $4\log 1/\delta$ copies of the average Y gives the probability of the results being outside the range of ϵF_2 from F_2 as δ. The space required is that to maintain $O(\frac{1}{\epsilon^2}\log 1/\delta)$ copies of the original estimate. Each of these requires a counter and a 4-wise independent hash function, both of which can be represented with a constant number of machine words under the standard RAM model. □

Join size estimation

Lemma 3 *Let* Z_x *be an entry of a sketch computed for the vector* $\boldsymbol{x}$*, and let* Z_y *be an entry of a sketch computer for* $\boldsymbol{y}$ *using the same hash function. The estimate is correct in expectation, i.e.,* $\mathsf{E}(Z_x * Z_y) = \boldsymbol{x} \cdot \boldsymbol{y}$.

Proof

$$\mathsf{E}(Z_x * Z_y) = \mathsf{E}\left(\sum_{i=1}^{U} h(i)^2 \boldsymbol{x}[i]\boldsymbol{y}[i] + \sum_{1 \le i \ne j \le U} h(i)h(j)\boldsymbol{x}[i]\boldsymbol{y}[j]\right)$$

$$= \sum_{i=1}^{U} \boldsymbol{x}[i]\boldsymbol{y}[i] + 0 = \boldsymbol{x} \cdot \boldsymbol{y}. \qquad \square$$

Join size estimation

$$\mathrm{Var}(Z_x * Z_y) \leq F_2(\boldsymbol{x}) F_2(\boldsymbol{y}).$$

Theorem 2 *Using space* $O(\frac{1}{\epsilon^2} \log 1/\delta)$ *space we can output an estimate of* $\mathbf{x} \cdot \mathbf{y}$ *so that*

$$\Pr\left[\left|(\boldsymbol{x} \cdot \boldsymbol{y}) - est\right| \leq \epsilon \sqrt{F_2(\boldsymbol{x}) F_2(\boldsymbol{y})}\right] \geq 1 - \delta.$$