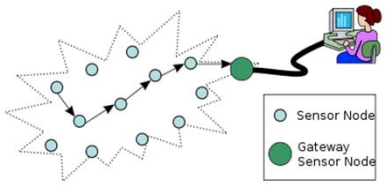
Streaming Algorithms

Data Streams

- A data stream is a sequence of data that is too large to be stored in available memory
- Examples:
  - Network traffic
  - Sensor networks
  - Approximate query optimization and answering in large databases
  - Scientific data streams
  - ..And more



Basic Data Stream Model

- Single pass over the data: $i_1, i_2,\ldots,i_n$
  - Typically, we assume n is known (but not always)
- Bounded storage (typically n^{α} or $\log^c n$)
  - Units of storage: bits, words, coordinates or „elements" (e.g., points, nodes/edges)
- Fast processing time per element
  - Randomness OK (in fact, almost always necessary)

8 2 1 9 1 9 2 4 6 3 9 4 2 3 4 2 3 8 5 2 5 6 …

Counting Distinct Elements

- Stream elements: numbers from $\{1...m\}$
- Goal: estimate the number of distinct elements DE in the stream
  - Up to $1\pm\varepsilon$
  - With probability 1-P
- Simpler goal: for a given $T>0$, provide an algorithm which, with probability 1-P:
  - Answers YES, if $DE> (1+\varepsilon)T$
  - Answers NO, if $DE< (1-\varepsilon)T$
- Run, in parallel, the algorithm with
  $T=1, 1+\varepsilon, (1+\varepsilon)^2,\ldots, n$
  - Total space multiplied by $\log_{1+\varepsilon} n \approx \log(n)/\varepsilon$

Vector Interpretation

Stream: 8 2 1 9 1 9 2 4 4 9 4 2 5 4 2 5 8 5 2 5

Vector X:

- Initially, x=0
- Insertion of i is interpreted as

$$x_i = x_i + 1$$

- Want to estimate DE(x) = the number of non-zero elements of x

Counting Complexity?

- One path over the data $\Rightarrow$ O(*n*) time
- Space complexity: O(*m*)
  - *m* is the number of DE
- We want better space complexity

An Approximation

- 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.

[Flajolet & Martin, J. of Comp. and Sys. Sc., 85]
[Alon, Matias, & Szegedy, STOC, 1996]

Intuition

- The more different elements you see, the more likely you are to see something unusual.
  - Here, "unusual" means "hash value ends in a lot of 0's."
- With a good random hash function
  - ½ of the elements end with 0
  - ¼ of the elements end with 00
  - 1/8 of the elements end with 000
  - 2^{-i} of the element end with i 0s

Analysis

- 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.

Analysis (2)

- Since 2^{-r} is small, $1 - (1-2^{-r})^m \approx 1 - e^{-m2}$.
- 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 $2^r << m$, $1 - (1 - 2^{-r})^m \approx 1 - e^{-m2} \approx 1$.
- Thus, 2^R will almost always be around m.

Why it doesn't work?

- E(2^R) is not bounded.
  - Probability halves when R -> R +1, but value doubles, up to maximum possible R.
- Workaround involves using many hash functions and getting many samples.
- How are samples combined?
  - Average? What if one very large value?
  - Median? All values are a power of 2.

Solution

- Partition your samples into small groups.
  - About log of the number of samples.
- Run in parallel the algorithm with different hash function
  - Take the average of groups.
- Then take the median of the averages.

Counting Distinct Elements

- Stream elements: numbers from {1...m}
- Goal: estimate the number of distinct elements DE in the stream
  - Up to $1\pm\varepsilon$
  - With probability 1-P
- Simpler goal: for a given T>0, provide an algorithm which, with probability 1-P:
  - Answers YES, if DE> $(1+\varepsilon)T$
  - Answers NO, if DE< $(1-\varepsilon)T$
- Run, in parallel, the algorithm with
  $T=1, 1+\varepsilon, (1+\varepsilon)^2,..., n$
  - Total space multiplied by $\log_{1+\varepsilon} n \approx \log(n)/\varepsilon$

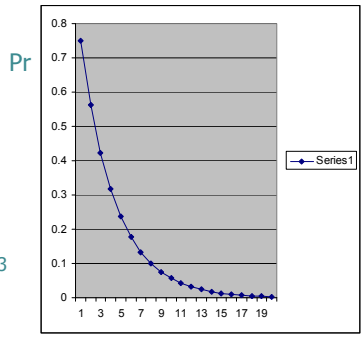
Estimating DE(x)

Vector X:

1 2 3 4 5 6 7 8 9

Set S: + + + (T=4)

- Choose a random set S of coordinates
  - For each i, we have $Pr[i\in S]=1/T$
- Maintain $Sum_S(x) = \Sigma_{i\in S} x_i$
- Estimation algorithm I:
  - YES, if $Sum_S(x)>0$
  - NO, if $Sum_S(x)=0$
- Analysis:
  - $Pr=Pr[Sum_S(x)=0] = (1-1/T)^{DE}$
  - For T large enough: $(1-1/T)^{DE} \approx e^{-DE/T}$
  - Using calculus, for ε small enough:
    - If DE> $(1+\varepsilon)T$, then $Pr \approx e^{-(1+\varepsilon)} < 1/e - \varepsilon/3$
    - if DE< $(1-\varepsilon)T$, then $Pr \approx e^{-(1-\varepsilon)} > 1/e + \varepsilon/3$



[Flajolet & Martin, J. of Comp. and Sys. Sc., 85]
[Alon, Matias, &Szegedy, STOC, 1996]

Generalization: Moments

- Suppose a stream has elements chosen from a set of n values.
- Let m_i be the number of times value i occurs.
- The k^{th} *moment* is the sum of $(m_i)^k$ over all i.

21

Special Cases

- 0^{th} moment = number of different elements in the stream.
  - The problem just considered.
- 1^{st} moment = sum of the numbers of elements = length of the stream.
  - Easy to compute.
- 2^{nd} moment = a measure of how uneven the distribution is.

22

Estimating L_2 norm
can also work for other moments

Alon-Matias-Szegedy'96

- Choose $r_1 \ldots r_m$ to be i.i.d. r.v., with
  $$Pr[r_i=1]=Pr[r_i=-1]=1/2$$
- Maintain
  $$Z=\sum_i r_i x_i$$
  under increments/decrements to x_i
- Algorithm I:
  $$Y=Z^2$$
- "Claim": Y "approximates" $||x||_2^2$ with "good" probability

Analysis

- We will use Chebyshev inequality
  - Need expectation, variance
- The expectation of $Z^2 = (\sum_i r_i x_i)^2$ is equal to

$$E[Z^2] = E[\sum_{i,j} r_i x_i r_j x_j] = \sum_{i,j} x_i x_j E[r_i r_j]$$

- We have
  - For $i \neq j$, $E[r_i r_j] = E[r_i] E[r_j] = 0$ – term disappears
  - For $i=j$, $E[r_i r_j] = 1$
- Therefore

$$E[Z^2] = \sum_i x_i^2 = ||x||_2^2$$

(unbiased estimator)

Analysis, ctd.

- The second moment of $Z^2 = (\sum_i r_i x_i)^2$ is equal to the expectation of

$$Z^4 = (\sum_i r_i x_i)(\sum_i r_i x_i)(\sum_i r_i x_i)(\sum_i r_i x_i)$$

- This can be decomposed into a sum of
  - $\sum_i (r_i x_i)^4$ →expectation= $\sum_i x_i^4$
  - $6 \sum_{i<j} (r_i r_j x_i x_j)^2$ →expectation= $6\sum_{i<j} x_i^2 x_j^2$
  - Terms involving single multiplier $r_i x_i$ (e.g., $r_1 x_1 r_2 x_2 r_3 x_3 r_4 x_4$) →expectation=0

  Total: $\sum_i x_i^4 + 6\sum_{i<j} x_i^2 x_j^2$

- The variance of Z^2 is equal to

$$E[Z^4] - E^2[Z^2] = \sum_i x_i^4 + 6\sum_{i<j} x_i^2 x_j^2 - (\sum_i x_i^2)^2$$
$$= \sum_i x_i^4 + 6\sum_{i<j} x_i^2 x_j^2 - \sum_i x_i^4 - 2\sum_{i<j} x_i^2 x_j^2$$
$$= 4\sum_{i<j} x_i^2 x_j^2$$
$$\leq 2 (\sum_i x_i^2)^2$$

Analysis, ctd.

- We have an estimator $Y=Z^2$
  - $E[Y] = \sum_i x_i^2$
  - $\sigma^2 = Var[Y] \le 2 (\sum_i x_i^2)^2$
- Chebyshev inequality:

$$Pr[\ |E[Y]-Y| \ge c\sigma\ ] \le 1/c^2$$

- Algorithm II:
  - Maintain $Z_1 \ldots Z_k$ (and thus $Y_1 \ldots Y_k$), define $Y' = \sum_i Y_i /k$
  - $E[Y'] = k \sum_i x_i^2 /k = \sum_i x_i^2$
  - $\sigma'^2 = Var[Y'] \le 2k(\sum_i x_i^2)^2 /k^2 = 2 (\sum_i x_i^2)^2 /k$
- Guarantee:

$$Pr[\ |Y' - \sum_i x_i^2 | \ge c\ (2/k)^{1/2} \sum_i x_i^2\ ] \le 1/c^2$$

- Setting c to a constant and $k=O(1/\varepsilon^2)$ gives $(1\pm\varepsilon)$-approximation with const. probability