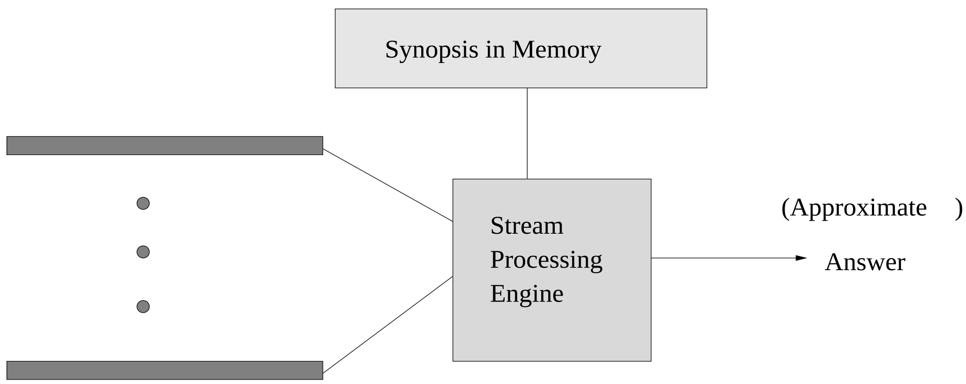
Query and Mining Data Streams

Data Streams: You Only Get One Look

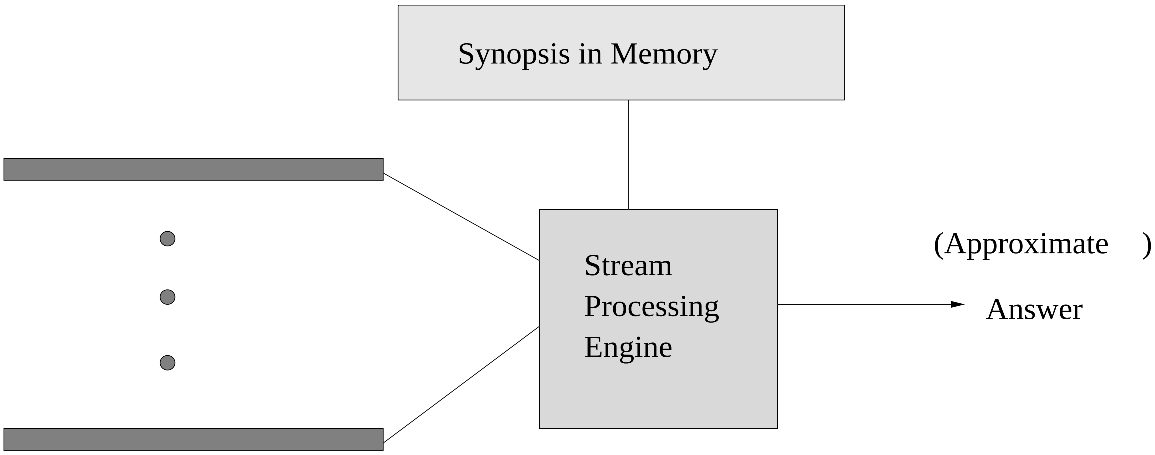
(adopted from tutorials, VLDB 2002, PODS 2002)



Motivation:

A growing number of applications:

- network monitoring and traffic management
- Call detail records in telecommunications
- Transactions in retail chains
- ATM operations in banks
- Log records generated by Web Servers
- Sensor network data



Characteristics

Massive volumes of data
e.g. several terabytes(10^{12} bytes)

Records arrive at a rapid rate

Goal: Mine patterns, process queries, compute statistics on data streams in real time.

Stream Projects

- Amazon/Cougar (Cornell) - sensors
- Aurora (Brown/MIT) - sensor monitoring, dataflow
- Hancock (AT&T) - telecom streams
- Niagara (OGI/Wisconsin) - internet XML databases
- OpenCQ (Georgia) - triggers, view maintenance
- Stream (Stanford) - general purpose
- Tapestry (Xerox) - content-based filtering
- Telegraph (Berkeley) - adaptive engine for sensors
- Tribeca (Bellcore) - network monitoring

Data Stream Processing Algorithms

- Generally, algorithms compute approximate answers

  - Difficult to compute answers accurately with limited memory

- Bounds on error

  - Although answer is approximate, there are bounds on the error.

  - The bound may be probabilistic:

  e.g. With probability of at least $1-\delta$, the computed answer is within a fraction ϵ of the actual answer.

Sampling: A Simple Case

A small random sample S of the data often well represents all the data

For a fast approximate answer to a query, apply a "modified" query to S.

Example:

Data stream R :

$\boxed{9,\ 3,\ 5,\ 2,\ 7,\ 1,\ 6,\ 5,\ 8,\ 4,\ 9,\ 1}$ (n=12)

Sample S : $\boxed{9,\ 5,\ 1,\ 8}$

- *Select average value from R where $R.e$ is odd*

  Answer = 5

- *Select count from R where $R.e$ is odd*

  answer = $12 \times 3/4 = 9$

$\boxed{\text{Frequent Elements in Streams and Bags}}$

(Richard M. Karp, S. Shenker, C. H. Papadimitriou
ACM TODS 2003)

Given a sequence $x = (x_1, ..., x_N) \in \Sigma^*$,

where Σ is an alphabet with n symbols, and

a real number θ $0 \leq \theta \leq 1$.

Let $f_x(a)$ denote the number of occurrences of a in x.

Find $I(x, \theta) \subset \Sigma$

$$I(x, \theta) = \{a \in \Sigma : f_x(a) > \theta N\}.$$

Example:

$\Sigma = \{a, b, c, d, e, f\}$

$n = 5$

x = (a b a c d a a c e c)

$N = 10$; let $\theta = 0.3$;

$\theta N = 0.3 \times 10 = 3$

$f_x(a) = 4$

$f_x(b) = 1$

$f_x(c) = 3$

$f_x(d) = 1$

$f_x(e) = 1$

Hence $I(x, \theta) = \{a, c\}$.

Performance of an algorithm:

–Time to process an occurrence of a symbol

–Number of passes

–Space

Fact 1: The output set $I(x, \theta)$ cannot contain more than $1/\theta$ symbols.

Proof: otherwise there are more than $\frac{1}{\theta}\theta N = N$ occurrences of symbols from $I(x, \theta)$ in x. $\square$

For example, if $\theta = 0.3$,

Then there can be at most 1/0.3 or 3 symbols in $I(x, \theta)$.

Fact 2:

Any on-line algorithm for computing $I(x, \theta)$ needs in the worst case $\Omega(nlog(N/n))$ bits.

Proof:

Suppose at the middle of x,

no symbol so far has exceeded θN occurrences.

The algorithm must at that point have enough memory to remember the precise tally of each symbol.

For example

x1 = (**a b a c d** a a c e c)

x2 = (**f b f c d** a a c e c)

Suppose the algorithm cannot distinguish the two subsequences in bold.

After half of sequence $x1$ is processed, the memory record leads to a = 2

for $x2$ the memory record also leads to a = 2

If $\theta = 0.3$,

in the end $I(x1, \theta)$ and $I(x2, \theta)$ both contain a.

If for $x1$ the memory record leads to a = 0

for $x2$ the memory record also leads to a = 0

In the end a is not in $I(x1, \theta)$ nor $I(x2, \theta)$.

In the correct answer a is only in $I(x1, \theta)$.

otherwise, if the algorithm reaches the same memory configurations in two computations on two half-sequences with distinct combinations of tallies,

say differing in the tally of a,

there is a way to complete the two sequences by the same second half so that

a is in $I(x, \theta)$ for one sequence and not in the other.

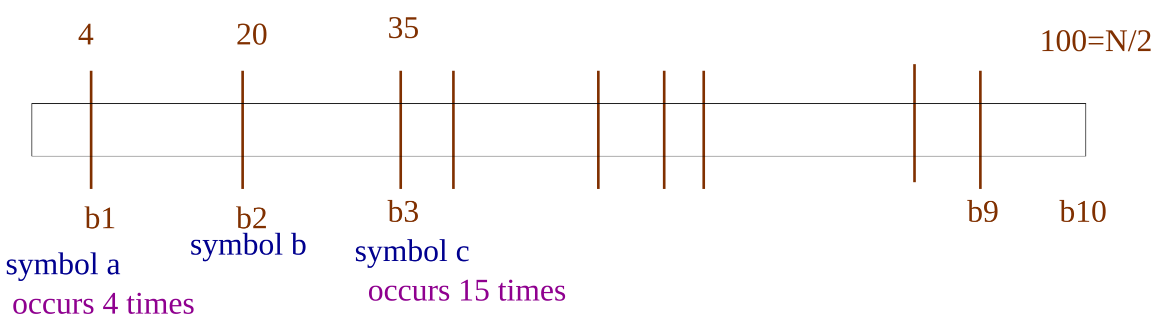
One of these two results must be incorrect.

Hence the algorithm must use at least $log|K|$ bits, where K is the set of all tally combinations not exceeding θN.

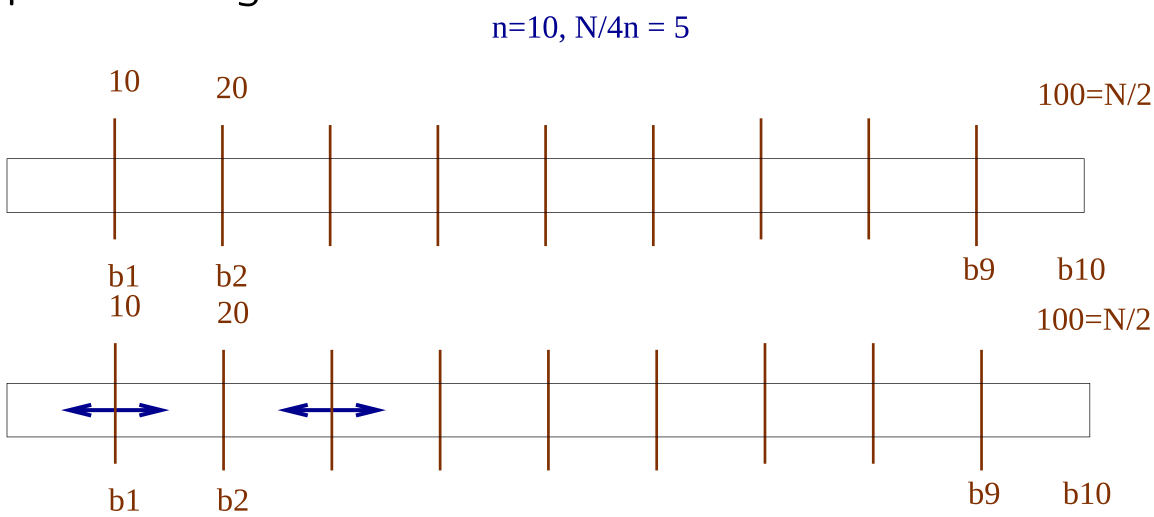
We shall derive a lower bound for $|K|$.

Each sequence of tallies in K can be seen as an $n+1$-tuple $\{b_0 = 0, b_1, b_2, ..., b_{n-1}, b_n = N/2\}$

The b_i's are the "boundaries where the tally of symbol a_i is $b_i - b_{i-1} - 1$.



Consider the sequence X with the most symmetrical partitioning.



Alter X by increasing or decreasing the values of b_i by $\lfloor N/4n \rfloor$.

In the above example, b_1 to b_9 can vary from $b_i - 5$ to $b_i + 5$,

i.e. 11 different values.

There are in total $(2\lfloor N/4n \rfloor + 1)^{n-1}$ different sequences.

Taking logarithms,
the algorithm uses $\Omega(nlog(N/n))$ bits.

An algorithm

Let the input sequence be $x[1], ...x[N]$
K is a set of symbols,
count is an array of integers indexed by K.

$K \leftarrow \phi$

FOR $i := 1, ...N$ DO

```
    IF x[i] is in K
    THEN count[x[i]] := count[x[i]] + 1
    ELSE insert x[i] in K, set count[x[i]] := 1

    IF |K| > 1/θ THEN
      FOR all a in K DO
          count[a] := count[a] − 1
          IF count[a] = 0
          THEN delete a from K
```

output K

Theorem: *A superset K of $I(x, \theta)$ with $|K| \leq 1/\theta$ can be computed with $O(1/\theta)$ memory.*

Proof:

Consider a symbol a not in K.

Each occurrence of a was eliminated with

$1/\theta - 1$ occurrences of other symbols.

Hence $f_x(a)/\theta < N$, or $f_x(a) < \theta N$.

Once the set K is found, a second pass can find the tallies of all symbols in K.

Deleting those with tally less than θN, the result is $I(x, \theta)$.

Approximate Frequency Counts over Data Streams

(G.S. Manku, R. Motwani, VLDB 2002)

Examples of data streams:

stock tickers, network traffic measurements, web-server logs, sensor networks, telecom call records.

A data stream is a stream of tuples, where each tuple is either a single item or a set of items.

Problem: To find the frequency counts exceeding a user-specified threshold s

Problem definition

Let N be the current length of the stream.

Some guarantees:

1. All items whose true frequency exceeds sN are output.

   There are no **false negative**.

2. No item whose true frequency is less than $(s - \epsilon)N$ is output

3. Estimated frequencies are less than the true frequencies by at most ϵN.

An algorithm maintains an $\epsilon-$deficient synopsis if its output satisfies the above properties.

E.g. s = 0.5, $\epsilon = 0.1$,

No item with frequency less than (0.5-0.1)N = 0.4N is output

Estimated frequency are less than the true frequency by at most 0.1 N

Sticky Sampling Algorithm

Data structure $\mathcal{S}$:

$\mathcal{S}$ is a set of entries of the form (c, f)

c is an element

f estimates the frequency of c.

Initially $\mathcal{S}$ is empty.

The initial sampling rate r is set to 1.

Sampling an element with rate r means to select the element with probability $\frac{1}{r}$.

For each incoming element c,

- If c already exists in $\mathcal{S}$,

  increment the corresponding frequency f.

- Sample c with rate r.

  If the element is selected by sampling, add an entry $(c, 1)$ to $\mathcal{S}$.

Sticky Sampling Algorithm (Cont)

The sampling rate r varies over the lifetime of a stream:

Let δ be a small value.

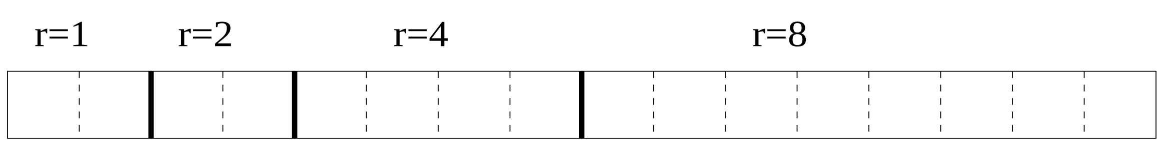
Let $t = \frac{1}{\epsilon}(s^{-1}\delta^{-1})$

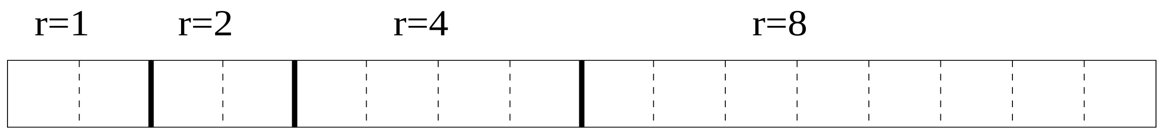
The first $2t$ elements are sampled at rate $r = 1$.

The next $2t$ elements are sampled at rate $r = 2$.

The next $4t$ elements are sampled at rate $r = 4$.

$\ldots$





Whenever sampling rate changes,

scan entries in $\mathcal{S}$.

For each entry (c, f),

repeatedly toss an coin with probability of $\frac{1}{r}$ for heads.

If coin toss is successful, stop.

If coin toss is not successful the frequency f is decremented by 1.

If f becomes 0, the entry is deleted from $\mathcal{S}$.

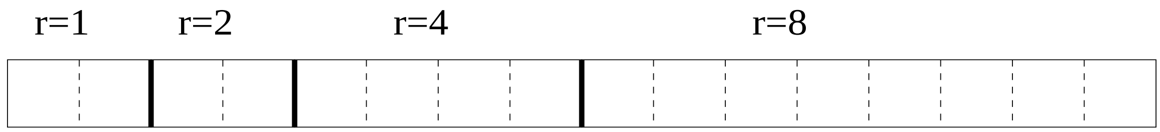
Effectively, we have transformed $\mathcal{S}$ to exactly the state when it would have been, had we been sampling with the new rate from the very beginning.

Theorem: Sticky Sampling computes an ϵ-deficient synopsis with probability at least $1 - \delta$ using at most $\frac{2}{\epsilon} log(s^{-1}\delta^{-1})$ expected number of entries.

Proof: The first $2t$ elements find their way into S.

N is the current total number of entries.

When the sampling rate is $r \geq 2$, we have $N = rt + rt'$, for some $t' \in [0, t)$.



It follows that

$$N \geq rt \;;\; \frac{1}{r} \geq \frac{t}{N}$$

Any error in the frequency count of an element c corresponds to a sequence of unsuccessful coin tosses during the first occurrences of c.

The probability that this sequence length exceeds ϵN is at most

$$(1 - \frac{1}{r})^{\epsilon N}$$

which is less than

$$(1 - \frac{t}{N})^{\epsilon N}$$

which in turn is less than $e^{-\epsilon t}$.

(note that $lim_{n\to\infty}(1 + \frac{t}{n})^n = e^t$)

The number of elements with frequency at least sN is no more than $\frac{1}{s}$.

Therefore, the probability that the estimate for any of them is deficient by ϵN

is at most

$$\frac{e^{-\epsilon t}}{s}$$

The probability of failure should be at most δ.

We can choose t such that

$$\frac{e^{-\epsilon t}}{s} = \delta$$

$$e^{-\epsilon t} = \delta s$$

$$t = \frac{1}{\epsilon} log(s^{-1}\delta^{-1})$$

Space requirement

The expected space requirement is $2t$.

Hence the expected number of entries is at most $\frac{2}{\epsilon} log(s^{-1}\delta^{-1})$

Lossy Counting Algorithm

The incoming stream is divided into buckets of width $w = \lceil \frac{1}{\epsilon} \rceil$ transactions each.

Buckets are labeled with bucket ids, starting from 1.

The current bucket id is $b_{current} = \lceil \frac{N}{w} \rceil$
(N is the current length of the stream.)

bucket ids

1	2	3	4
a b c d	e f g h	i j k l	m n o p

N=16 (current stream length)
w=4 (bucket size)

For an element e,

f_e is its true frequency in the stream

Data structure : $\mathcal{D}$:

$\mathcal{D}$ is a set of entries of the form (e, f, Δ)

- e: an element in the stream
- f: an integer representing its estimated frequency
- Δ: maximum possible error in f

Lossy Counting Algorithm (cont)

Initially $\mathcal{D}$ is empty.

When a new element e arrives,

- If $\mathcal{D}$ has an entry for e,
  increment the frequency f by 1.
- Otherwise, create a new entry $(e, 1, b_{current} - 1)$.

Pruning:

at bucket boundaries, i.e., whenever $N = 0 \bmod w$,
$\mathcal{D}$ is pruned by **deleting** some of its entries

Rule for deletion:

an entry (e, f, Δ) *is deleted if* $f + \Delta \leq b_{current}$

Output: output entries in $\mathcal{D}$ where $f \geq (s - \epsilon)N$.

(s is the threshold)

Some guarantees

- If e does not appear in $\mathcal{D}$, then the true frequency $f_e \leq \epsilon N$.
- If $(e, f, \Delta) \in \mathcal{D}$, then $f \leq f_e \leq f + \epsilon N$.

- - -

Lemma 1: When deletion occurs, $b_{current} \leq \epsilon N$.

$b_{current} = \lceil \frac{N}{w} \rceil$

When deletion occurs, $N = 0 \bmod w$.

$b_{current} = \frac{N}{w}$

$w = \lceil \frac{1}{\epsilon} \rceil$

$b_{current} \leq N\epsilon$

Note: The value of Δ assigned to a new entry (e, f, Δ) is the maximum number of times e could have occurred in the first $b_{current} - 1$ buckets. This is exactly $b_{current} - 1$.

Lemma 2: When (e, f, Δ) gets deleted, $f_e \leq b_{current}$

Proof

Show by induction If an entry (e, f, Δ) is deleted for $b_{current} = 1$. $f = f_e = 1$, thus $f_e \leq b_{current}$

If an entry (e, f, Δ) is deleted for some $b_{current} > 1$.

This entry was inserted at bucket $\Delta + 1$

An entry for e could have been deleted latest at bucket Δ.

By induction, the true frequency of e at that deletion was $\leq \Delta$

f is the true frequency of e ever since it was inserted.

Therefore, $f_e \leq f + \Delta$

The deletion rule says that $f + \Delta \leq b_{current}$

We get $f_e \leq b_{current}$.

- - -

With Lemma 1, we have $f_e \leq \epsilon N$.

Lemma 3: If $(e, f, \Delta) \in \mathcal{D}$, then $f \leq f_e \leq f + \epsilon N$.

Proof: If $\Delta = 0, f = f_e$.

Otherwise, e was possibly deleted sometimes in the first Δ buckets.

From Lemma 2, the true frequency at the deletion was at most Δ.

Therefore, $f_e \leq f + \Delta$.

Since $\Delta \leq b_{current} - 1 \leq \epsilon N$,
we conclude that $f \leq f_e \leq f + \epsilon N$.

Space requirement $|\mathcal{D}|$

$$|\mathcal{D}| \leq \frac{1}{\epsilon} log(\epsilon N)$$