How Philippe Flipped Coins to Count Data

Jérémie Lumbroso
LIP6 / INRIA Rocquencourt

December 16th, 2011

0. DATA STREAMING ALGORITHMS

Stream: a (very large) sequence S over (also very large) domain $\mathcal{D}$

$$S = s_1 \; s_2 \; s_3 \; \cdots \; s_\ell, \qquad s_j \in \mathcal{D}$$

consider S as a multiset

$$\mathcal{M} = {m_1}^{f_1} \; {m_2}^{f_2} \; \cdots \; {m_n}^{f_n}$$

Interested in **estimating** the following *quantitive* statistics:

— **A.** **Length** $:= \ell$

— **B.** **Cardinality** $:= \mathrm{card}(m_i) \equiv n$ (distinct values) $\leftarrow$ this talk

— **C.** **Frequency moments** $:= \sum_{v \in \mathcal{D}} {f_v}^p \quad p \in \mathbb{R}_{\geqslant}$

Constraints:

- very little processing memory
- on the fly (single pass + simple main loop)
- no statistical hypothesis
- accuracy within a few percentiles

Historical context

- **1970**: average-case $\rightarrow$ deterministic algorithms on random input
- **1976-78**: first randomized algorithms (primality testing, matrix multiplication verification, find nearest neighbors)
- **1979**: Munro and Paterson, find median in one pass with $\Theta(\sqrt{n})$ space with high probability
  $\Rightarrow$ (almost) first streaming algorithm

In **1983**, Probabilistic Counting by Flajolet and Martin is (more or less) the first streaming algorithm (one pass + constant/logarithmic memory).

Google scholar

Probabilistic counting algorithms for data base applications
P **Flajolet**... - Journal of computer and system sciences, 1985 - Elsevier
Abstract This paper introduces a class of probabilistic counting algorithms with which one can estimate the number of distinct elements in a large collection of data (typically a large file stored on disk) in a single pass using only a small additional storage (typically less ...
Cited by 628 - Related articles - All 36 versions

Probabilistic counting
P **Flajolet**... - Foundations of Computer Science, ..., 1983 - ieeexplore.ieee.org
Abstract We present here a class of probabilistic algorithms with which one can estimate the number of distinct elements in a collection of data (typically a large file stored on disk) in a single pass, using only 0 (1) auxiliary storage and 0 (1) operations per element. We ...
Cited by 111 - Related articles - All 7 versions

Combining both versions: cited about 750 times = **second most cited** element of Philippe's bibliography, after only *Analytic Combinatorics*.

Databases, IBM, California...

In the 70s, IBM researches <u>relational databases</u> (first PRTV in UK, then System R in US) with high-level query language: user should not have to know about the structure of the data.

⇒ query optimization; requires cardinality (estimates)

```
SELECT name FROM participants
WHERE
   sex = "M" AND
   nationality = "France"
```

Min. comparisons: compare first `sex` or `nationality`?

G. Nigel N. Martin (IBM UK) invents first version of "probabilistic counting", and goes to IBM San Jose, in **1979**, to share with System R researchers. Philippe discovers the algorithm in **1981** at IBM San Jose.

As I said over the phone, I started working on your
algorithm when Kyu-Young Whang considered implementing it
and wanted explanations/estimations. I find it simple, eleg
and ~~[illegible]~~ amazingly powerful.

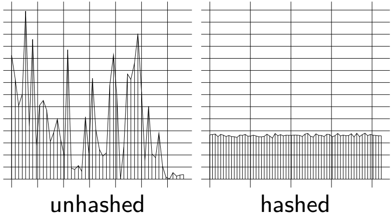
1. HASHING: reproducible randomness

- **1950s:** hash functions as tools for hash tables
- **1969:** Bloom filters $\rightarrow$ first time in an approximate context
- **1977/79:** Carter & Wegman, *Universal Hashing*, first time considered as probabilistic objects + proved uniformity is possible in practice

hash functions **transform data into i.i.d. uniform** random variables or in infinite strings of random bits:

$$h : \mathcal{D} \rightarrow \{0, 1\}^{\infty}$$

that is, if $h(x) = b_1 b_2 \cdots$,
then $\mathbb{P}[b_1 = 1] = \mathbb{P}[b_2 = 1] = \ldots = 1/2$



- Philippe's approach was experimental
- later theoretically validated in **2010:** Mitzenmacher & Vadhan proved hash functions "work" because they exploit the entropy of the hashed data

2. $\boxed{\text{PROBABILISTIC COUNTING (1983)}}$

(with G. Nigel N. Martin)

For each element in the string, we hash it, and look at it

$$S = s_1\ s_2\ s_3\ \cdots \qquad \Rightarrow \quad h(s_1)\ h(s_2)\ h(s_3)\ \cdots$$

***h*(*v*) transforms *v* into string of random bits** (0 or 1 with prob. 1/2).
So you expect to see:

$$0xxxx... \to \mathbb{P} = 1/2 \qquad 10xxx... \to \mathbb{P} = 1/4 \qquad 110xx... \to \mathbb{P} = 1/8$$

Indeed

$$\mathbb{P}\left[\ \begin{array}{|c|c|c|c|c|c}\hline 1 & 1 & 0 & x & x & \cdots \\ \hline\end{array}\ \right] = \mathbb{P}[b_1 = 1] \cdot \mathbb{P}[b_2 = 1] \cdot \mathbb{P}[b_3 = 0] = \frac{1}{8}$$

2. $\boxed{\text{PROBABILISTIC COUNTING (1983)}}$

(with G. Nigel N. Martin)

For each element in the string, we hash it, and look at it

$$S = s_1 \ s_2 \ s_3 \ \cdots \qquad \Rightarrow \quad h(s_1)\ h(s_2)\ h(s_3)\ \cdots$$

$h(v)$ transforms v into string of random bits (0 or 1 with prob. 1/2).
So you expect to see:

$$0xxxx\ldots \to \mathbb{P} = 1/2 \qquad 10xxx\ldots \to \mathbb{P} = 1/4 \qquad 110xx\ldots \to \mathbb{P} = 1/8$$

Indeed

$$\mathbb{P}\left[\ \boxed{1 \mid 1 \mid 0 \mid x \mid x \mid \cdots}\ \right] = \mathbb{P}[b_1 = 1] \cdot \mathbb{P}[b_2 = 1] \cdot \mathbb{P}[b_3 = 0] = \frac{1}{8}$$

Intuition: because strings are uniform, prefix pattern $1^k0\cdots$ appears with probability $1/2^{k+1}$
$\Rightarrow$ seeing prefix $1^k0\cdots$ means it's likely there is $n \geqslant 2^{k+1}$ different strings

Idea:

- keep track of prefixes $1^k0\cdots$ that have appeared
- estimate cardinality with 2^p, where $p =$ size of largest prefix

Bias correction: how analysis is FULLY INVOLVED in design

Described idea works, but presents **small bias** (i.e. $\mathbb{E}[2^p] \neq n$).

Without analysis (original algorithm)

```
After all the values have been processed, then:

if M(MAP)=000,  then RESULT=LO(MAP)-1

if M(MAP)=111,  then RESULT=LO(MAP)+1

otherwise RESULT=LO(MAP).

For example,
   if MAP was 000000000000000000000001111111
   LO(MAP) is 8 and M(MAP) is 000: RESULT=7
   if MAP was 000000000000000000000111011111111
   LO(MAP) is 8 and M(MAP) is 111: RESULT=9
   if MAP was 000000000000000000000100111111111
   LO(MAP) is 8 and M(MAP) is 010: RESULT=8
```

the three bits immediately after the first 0 are sampled, and depending on whether they are 000, 111, etc. a small ± 1 correction is applied to $p = \rho(\mathrm{bitmap})$

With analysis (Philippe)

Philippe determines that

$$\mathbb{E}[2^p] \approx \phi n$$

where $\phi \approx 0.77351\ldots$ is defined by

$$\phi = \frac{e^{\gamma}\sqrt{2}}{3} \prod_{p=1}^{\infty} \left[\frac{(4p+1)(4p+2)}{(4p)(4p+3)} \right]^{(-1)^{\nu(p)}}$$

such that we can apply a simple correction and have unbiased estimator,

$$Z := \frac{1}{\phi} 2^p \qquad \mathbb{E}[Z] = n$$

The basic algorithm

- $h(x)$ = hash function, transform data x into uniform $\{0,1\}^\infty$ string
- $\rho(s)$ = position of first bit equal to 0, i.e. $\rho(1^k 0 \cdots) = k + 1$

procedure ProbabilisticCounting(S : stream)
    $\mathrm{bitmap} := [0, 0, \ldots, 0]$
    for all $x \in S$ **do**
        $\mathrm{bitmap}[\rho(h(x))] := 1$
    end for
    $P := \rho(\mathrm{bitmap})$
    return $\frac{1}{\phi} \cdot 2^P$
end procedure

Ex.: if $\mathrm{bitmap} = 1111000100 \cdots$ then $P = 5$, and $n \approx 2^5/\phi = 20.68\ldots$

Typically estimates are one binary order of magnitude off the exact result: **too inaccurate for practical applications**.

Stochastic Averaging

To improve accuracy of algorithm by $1/\sqrt{m}$, elementary idea is to use m different hash functions (and a different bitmap table for each function) and **take average**.

$\Rightarrow$ very costly (hash m time more values)!

Split elements in m substreams randomly using first few bits of hash

$$h(v) = b_1 b_2 b_3 b_4 b_5 b_6 \cdots$$

which are then discarded (only $b_4 b_5 b_6 \cdots$ is used as hash value).

For instance for $m = 4$,

$$h(x) = \begin{cases} 00 b_3 b_4 \cdots & \rightarrow & \mathrm{bitmap}_{00}[\rho(b_3 b_4 \cdots)] = 1 \\ 01 b_3 b_4 \cdots & \rightarrow & \mathrm{bitmap}_{01}[\rho(b_3 b_4 \cdots)] = 1 \\ 10 b_3 b_4 \cdots & \rightarrow & \mathrm{bitmap}_{10}[\rho(b_3 b_4 \cdots)] = 1 \\ 11 b_3 b_4 \cdots & \rightarrow & \mathrm{bitmap}_{11}[\rho(b_3 b_4 \cdots)] = 1 \end{cases}$$

Theorem [FM85]. The estimator Z of Probabilistic Counting is an **asymptotically unbiased** estimator of cardinality, in the sense that

$$\mathbb{E}_n[Z] \sim n$$

and has accuracy using m bitmaps is

$$\frac{\sigma_n[Z]}{n} = \frac{0.78}{\sqrt{m}}$$

Concretely, need $O(m \log n)$ memory (instead of $O(n)$ for exact).

Example: can count cardinalities up to $n = 10^9$ with error $\pm 6\%$, using only 4096 bytes = 4 kB.

3. from Prob. Count. to LogLog (2003)

(with Marianne Durand)

PC: bitmaps require k bits to count cardinalities up to $n = 2^k$

Reasoning backwards (from observations), it is reasonable, when estimating cardinality $n = 2^3$, to observe a bitmap $11100\cdots$; remember

- $b_1 = 1$ means $n \geqslant 2$
- $b_2 = 1$ means $n \geqslant 4$
- $b_3 = 1$ means $n \geqslant 8$

WHAT IF instead of keeping track of **all** the 1s we set in the bitmap, we only kept track of the **position of the largest**? It only requires $\log\log n$ bits!

In algorithm, replace

$$\mathrm{bitmap}_i[\rho(h(x))] := 1 \qquad \text{by} \qquad \mathrm{bitmap}_i := \max\{\rho(h(x)), \mathrm{bitmap}_i\}$$

For example, compared evolution of "bitmap":

Prob. Count.:	$00000\cdots$	$00100\cdots$	$10100\cdots$	$11100\cdots$	$11110\cdots$
LogLog:	1	4	4	4	5

loss of precision in LogLog?

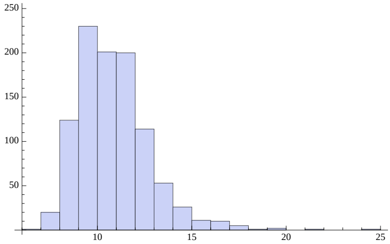
Probabilistic Counting and LogLog **often** find the same estimate:

Probabilistic Counting					**5**				
LogLog					**5**				
bitmap	1	1	1	1	0	0	0	0	⋯

But sometimes differ:

Probabilistic Counting					**5**				
LogLog								**8**	
bitmap	1	1	1	1	0	0	1	0	⋯

Other way of looking at it, the **distribution** of the rank (= max of n geometric variables with $p = 1/2$) used by LogLog has **long tails**:



SuperLogLog (same paper)

The accuracy (want it to be smallest possible):

- **Probabilistic Counting:** $0.78/\sqrt{m}$ for m registers of 32 bits
- **LogLog:** $1.36/\sqrt{m}$ for m small registers of 5 bits

In LogLog, loss of accuracy due to some (rare but real) registers that are too big, too far beyond the expected value.

SuperLogLog is LogLog, in which we remove δ largest registers before estimating, i.e., $\delta = 70\%$.

- involves a two-time estimation
- analysis is much more complicated
- but accuracy much better: $\mathbf{1.05/\sqrt{m}}$

Analysis Duper Loglog Nov 1, 2006

Geometric RV.

$$\mathbb{P}(X=k)=\frac{1}{2^k} \qquad k=1,2,3,\dots$$

$$\mathbb{P}(X\geq k)=\frac{1}{2^{k-1}} \qquad k=1,2,3,\dots$$

$$\mathbb{P}(X<k)=\frac{1}{2^{k-1}}\,; \qquad \mathbb{P}(X\leq k)=1-1/2^k.$$

Max Geom $\quad M_\nu = \max\left(X^{(1)},\dots,X^{(\nu)}\right) \qquad X^{(j)}\in \text{geom}\left(\tfrac{1}{2}\right)$

$$\mathbb{P}(M_n\leq k)=\left(1-\frac{1}{2^k}\right)^\nu \quad \begin{cases} \text{valid for } \nu>0,\ k=0,1,2,3,\dots \\ \text{or } \nu=0 \text{ with convention } 0^0=1 \\ [\max(\{\phi\})=0.] \end{cases}$$

Normalizing = $\boxed{\text{Correct bias by considering } \alpha = 2\log 2}$

Let $S=M_\nu^{(1)}+\dots+M_\nu^{(m)}$, the sum of $\underline{m}$ independent copies

$$\mathbb{E}\left(\frac{S}{m}\times 2\log 2\right)\approx x^{-1}. \qquad \mathbb{V}\text{ar}\left(\frac{S}{m}\times 2\log 2\right)\approx\frac{x^{-2}}{m}(3\log 2-1).$$

Set $\beta=\sqrt{3\log 2-1}$, $\beta=1.03896.$ (5)

4. "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm" (2007)

(with Eric Fusy, Frédéric Meunier & Olivier Gandouet)

- **2005:** Giroire (PhD student of Philippe's) publishes thesis with cardinality estimator based on order statistics
- **2006:** Chassaing and Gerin, using statistical tools find best estimator based on order statistics in an information theoretic sense

The note suggests using a harmonic mean: initially dismissed as a theoretical improvement, it turns out simulations are *very* good. Why?

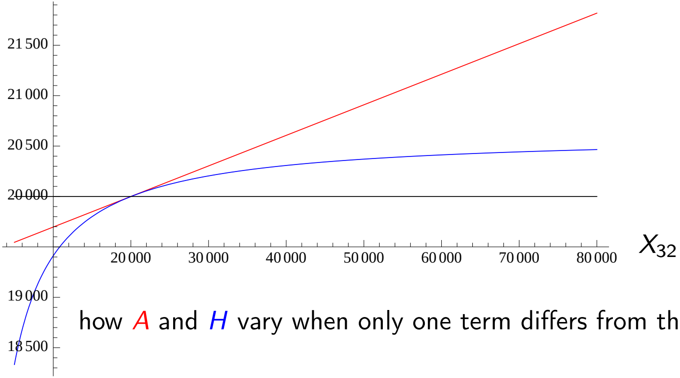
Harmonic means ignore too large values

X_1, X_2, ..., X_m are estimates of a stream's cardinality

Arithmetic mean **Harmonic mean**

$$A := \frac{X_1 + X_2 + \ldots + X_m}{m} \qquad H := \frac{m}{\frac{1}{X_1} + \frac{1}{X_2} + \ldots + \frac{1}{X_m}}$$

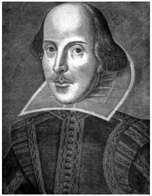
Plot of A and H for $X_1 = \ldots = X_{31} = 20\,000$ and X_{32} varying between and 5 000 and 80 000 (two binary orders of magnitude)



how A and H vary when only one term differs from the rest

The end of an adventure. HyperLogLog = sensibly same precision as SuperLogLog, but **substitutes** algorithmic cleverness with **mathematical elegance**.

Accuracy is $\mathbf{1.03/\sqrt{m}}$ with m small loglog bytes (≈ 4 bits).

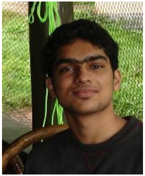


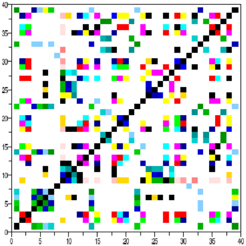
Whole of Shakespeare summarized:

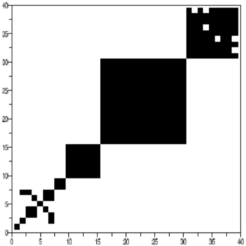
```
ghfffghfghgghggggghghheehfhfhhgghghghhfgffffhhhiigfhhffgfiihfhhh
igigighfgihfffghigihghigfhhgeegeghgghhhgghhfhidiigihighihehhhfgg
hfgighigffghdieghhhggghhfghhfiiheffghghihifgggffihgihfggighgiiif
fjgfgjhhjiifhjgehgghfhhfhjhigggh ghihigghhihihgiighgfhlgjfgjjjmfl
```

Estimate $\tilde{n} \approx 30\,897$ against $n = 28\,239$. Error is $\pm 9.4\%$ for 128 bytes.

Pranav Kashyap: word-level encrypted texts, classification by language.







Left out of discussion:

- Philippe's discovery and analysis of Approximate Counting, 1982 (handle a counter up to n with $\log\log n$ memory)
- a beautiful algorithm, Adaptive Sampling, 1989, which was ahead of its time, and was grossly unappreciated... until it was rediscovered in 2000

PDF + all algorithms implemented in Mathematica + video of Philippe giving a wide-audience talk on data streaming algorithms:

`http://lip6.fr/Jeremie.Lumbroso/pfac.php`