Cardinality Estimation

Robert Sedgewick
Princeton University

with special thanks to Jérémie Lumbroso

Philippe Flajolet, mathematician and computer scientist extraordinaire

Philippe Flajolet 1948–2011

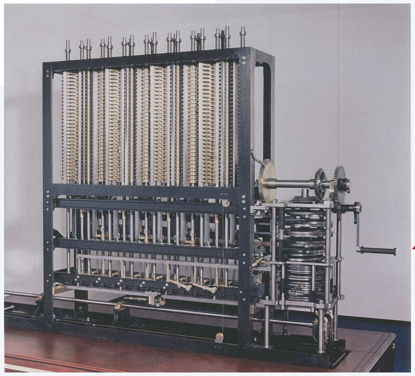
Analysis of Algorithms

" PEOPLE WHO ANALYZE ALGORITHMS have double happiness. First of all they experience the sheer beauty of elegant mathematical patterns that surround elegant computational procedures. Then they receive a practical payoff when their theories make it possible to get other jobs done more quickly and more economically."

– Don Knuth

Understood since Babbage: AofA is a *scientific* endeavor.

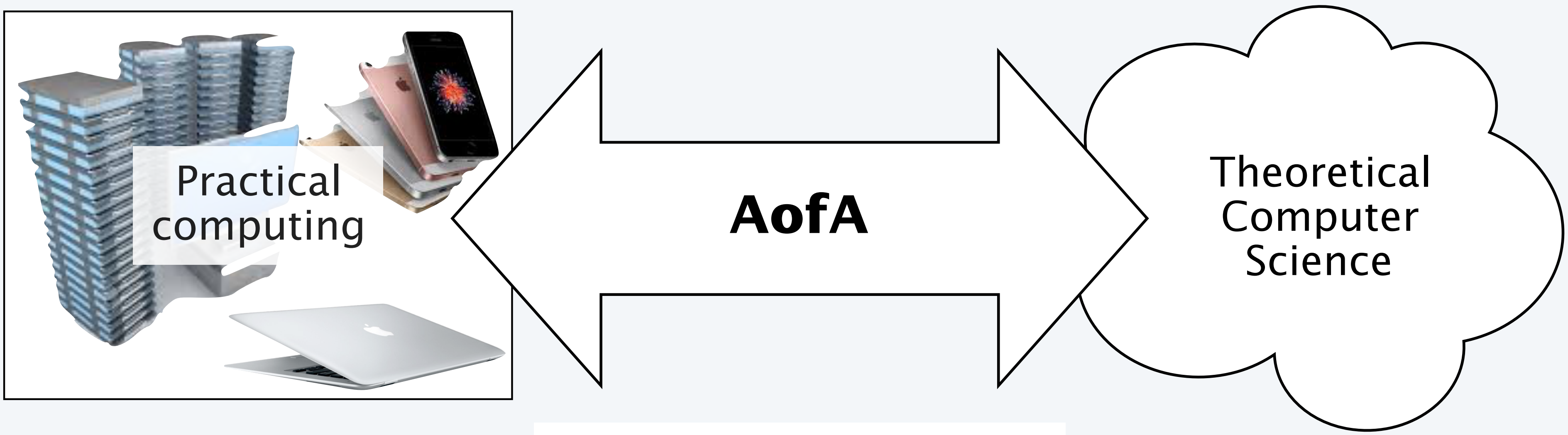
- Start with a working program (algorithm implementation).
- Develop mathematical model of its behavior.
- Use the *model* to formulate hypotheses on resource usage.
- Use the *program* to validate hypotheses.
- Iterate on basis of insights gained.



Analytic Engine

how many times do we have to turn the crank?

Analysis of Algorithms (present-day context)



Practical computing
- Real code on real machines
- Thorough validation
- Limited math models

AofA
- Theorems *and* code
- Precise math models
- Experiment, validate, iterate

Theoretical computer science
- Theorems
- Abstract math models
- Limited experimentation

Cardinality Estimation

- Warmup: exact cardinality count
- Probabilistic counting
- Stochastic averaging
- Refinements
- Final frontier

Cardinality counting

Q. In a given stream of data values, how many different values are present?

Example. How many unique visitors in a web log?

log.07.f3.txt

```
109.108.229.102
pool-71-104-94-246.lsanca.dsl-w.verizon.net
117.222.48.163
pool-71-104-94-246.lsanca.dsl-w.verizon.net
1.23.193.58
188.134.45.71
1.23.193.58
gsearch.CS.Princeton.EDU
pool-71-104-94-246.lsanca.dsl-w.verizon.net
81.95.186.98.freenet.com.ua
81.95.186.98.freenet.com.ua
81.95.186.98.freenet.com.ua
CPE-121-218-151-176.lnse3.cht.bigpond.net.au
```

6 million strings

UNIX (1970s-present)

```
% sort -u log.07.f3.txt | wc -l
1112365
```

"unique"

SQL (1970s-present)

```
SELECT
DATE_TRUNC('day',event_time),
COUNT(DISTINCT user_id),
COUNT(DISTINCT url)
FROM weblog
```

State of the art in the wild for decades. Sort, then count.

Warmup: exact cardinality count using linear probing

Hashing with linear probing

- Compute a *hash function* that transforms data value into a table index.
- Scan to find value or an empty slot.
- Keep table less than half full by resizing.

cnt $\boxed{6}$

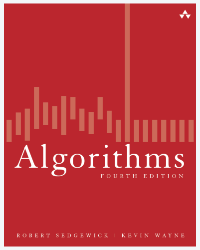
4	109.108.229.102
14	81.95.186.98.freenet.com.ua
12	117.222.48.163
5	1.23.193.58
5	188.134.45.71
5	1.23.193.58
14	81.95.186.98.freenet.com.ua
14	81.95.186.98.freenet.com.ua
1	117.211.88.36
	msnbot-131-253-46-251.msn.com
	msnbot-131-253-46-251.msn.com
	gsearch.CS.Princeton.EDU
	118-171-27-8.dynamic.hinet.net
	cpe-76-170-182-222.socal.rr.com
	203-88-22-144.live.vodafone.in
	117.211.88.36

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Warmup: exact cardinality count using a hash table

Hashing with linear probing

- Compute a *hash function* that transforms data value into a table index.
- Scan to find value or an empty slot.
- Keep table less than half full by resizing.



http://algs4.cs.princeton.edu/34hash/LinearProbingHashST.java

Textbook method, implemented in most modern programming environments.

Exact cardinality count in Java

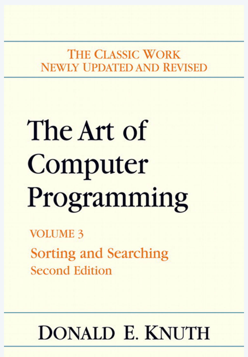
- Input is an “iterable”
- HashSet implements linear probing
- add() adds new value (noop if already there)
- size() gives number of distinct values added

```
public static long count(Iterable<String> stream)
{
   HashSet<String> hset = new HashSet<String>();
   for (String x : stream)
       hset.add(x);
   return hset.size();
}
```

Mathematical analysis of exact cardinality count with linear probing

Theorem. Expected time and space cost is linear.

Proof. Immediate from classic Knuth Theorem 6.4.K.



Theorem K. *The average number of probes needed by Algorithm L, assuming that all M^N hash sequences (35) are equally likely, is*

$$C_N = \tfrac{1}{2}(1 + Q_0(M, N-1)) \quad \text{(successful search)}, \tag{40}$$

$$C'_N = \tfrac{1}{2}(1 + Q_1(M, N)) \quad \text{(unsuccessful search)}, \tag{41}$$

where

$$\begin{aligned} Q_r(M,N) &= \binom{r}{0} + \binom{r+1}{1}\frac{N}{M} + \binom{r+2}{2}\frac{N(N-1)}{M^2} + \cdots \\ &= \sum_{k\ge 0}\binom{r+k}{k}\frac{N}{M}\frac{N-1}{M}\cdots\frac{N-k+1}{M}. \end{aligned} \tag{42}$$

Proof. Details of the calculation are worked out in exercise 27. (For the variance, see exercises 28, 67, and 68.) ∎

Q. Do the hash functions that we use *uniformly* and *independently* distribute keys in the table?

A. Not likely.

Scientific validation of exact cardinality count with linear probing

Hypothesis. Time and space cost is *linear for the hash functions we use and the data we have.*

Quick experiment. Doubling the problem size should double the running time.

Driver to read N strings and count distinct values

```
public static void main(String[] args)
{
   int N = Integer.parseInt(args[0]);
   StringStream stream = new StringStream(N);
   long start = System.currentTimeMillis();

   StdOut.println(count(stream));

   long now = System.currentTimeMillis();
   double time = (now - start) / 1000.0;
   StdOut.println(time + " seconds");
}
```

```
% java Hash 2000000 < log.07.f3.txt
483477
3.322 seconds

% java Hash 4000000 < log.07.f3.txt
883071
6.55 seconds

% java Hash 6000000 < log.07.f3.txt
1097944
9.49 seconds
```

✓

```
% sort -u log.07.f3 | wc -l
1097944
```

sort-based method takes about 3 minutes

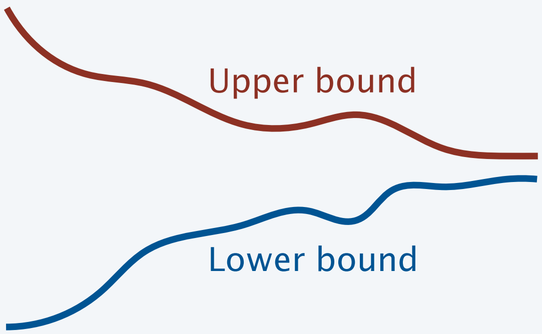
Q. Is hashing with linear probing effective?

A. Yes. Validated in countless applications for *over half a century*.

Complexity of exact cardinality count

Q. Does there exist an *optimal* algorithm for this problem?

A. Depends on definition of "optimal".



Guaranteed linear-time? NO. Linearithmic lower bound.

Guaranteed linearithmic? YES. Balanced BSTs or mergesort.

Linear-time with high probability assuming the existence of random bits?
YES. Perfect hashing.

Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert, and Tarjan
Dynamic Perfect Hashing: Upper and Lower Bounds
SICOMP 1994.

Within a small constant factor of the cost of touching the data in practice?
YES. Linear probing.

M. Mitzenmacher and S. Vadhan
Why Simple Hash Functions Work: Exploiting the Entropy in a Data Stream.
SODA 2008.

Hypothesis. Standard linear probing is "optimal".

Note: uniformity of hash function affects *only* the running time (not the value computed).

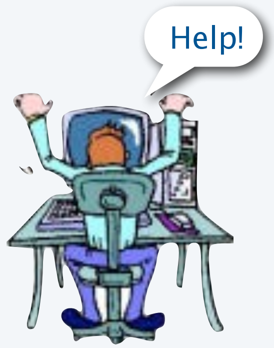
Exact cardinality count requires linear space

Q. I can't use a hash table. The stream is much too big to fit all values in memory. Now what?

A. Bad news: You cannot get an exact count.

A. Good news: You *can* get an accurate estimate (stay tuned).

```
109.108.229.102
pool-71-104-94-246.lsanca.dsl-w.verizon.net
117.222.48.163
pool-71-104-94-246.lsanca.dsl-w.verizon.net
1.23.193.58
188.134.45.71
1.23.193.58
gsearch.CS.Princeton.EDU
pool-71-104-94-246.lsanca.dsl-w.verizon.net
81.95.186.98.freenet.com.ua
81.95.186.98.freenet.com.ua
81.95.186.98.freenet.com.ua
CPE-121-218-151-176.lnse3.cht.bigpond.net.au
```



Typical modern applications
- Can only afford to process each value *once*. ← and cannot spend much time on any value
- *Estimate* of count still very useful.

Cardinality Estimation

- Warmup: exact cardinality count
- **Probabilistic counting**
- Stochastic averaging
- Refinements
- Final frontier

Cardinality *estimation*

is a fundamental problem with many applications *where memory is limited*.

Q. *About* how many different values appear in a given stream?

Constraints

- Make *one pass* through the stream.
- Use *as few operations per value* as possible
- Use *as little memory* as possible.
- Produce *as accurate an estimate* as possible.

typical applications

How many unique visitors to my website?

Which sites are the most/least popular?

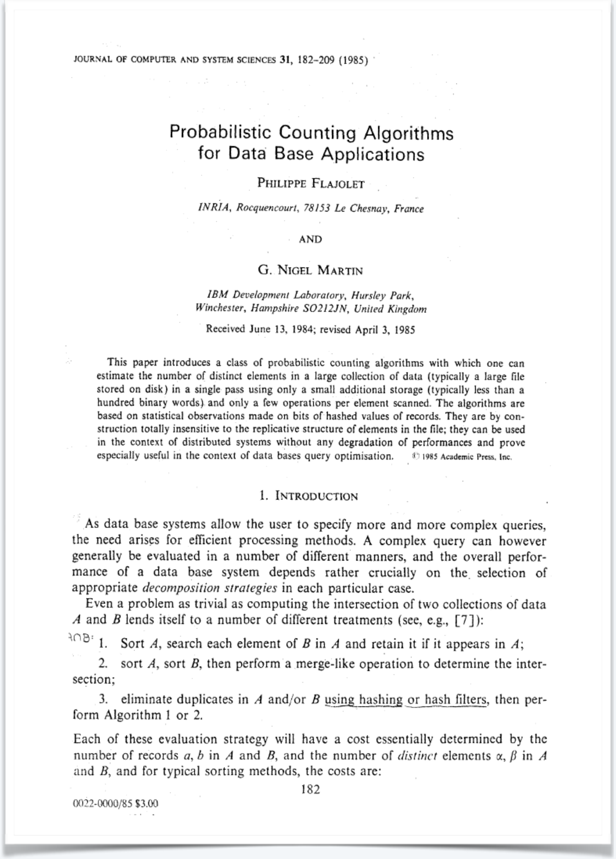
How many different websites visited by each customer?

How many different values for a database join?

To fix ideas on scope: Think of *millions* of streams each having *trillions* of values.

Probabilistic counting with stochastic averaging (PCSA)

Flajolet and Martin
Probabilistic Counting Algorithms for Data Base Applications
FOCS 1983, *JCSS* 1985.



JOURNAL OF COMPUTER AND SYSTEM SCIENCES **31**, 182–209 (1985)

Probabilistic Counting Algorithms for Data Base Applications

PHILIPPE FLAJOLET

INRIA, Rocquencourt, 78153 Le Chesnay, France

AND

G. NIGEL MARTIN

IBM Development Laboratory, Hursley Park, Winchester, Hampshire SO212JN, United Kingdom

Received June 13, 1984; revised April 3, 1985

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 than a hundred binary words) and only a few operations per element scanned. The algorithms are based on statistical observations made on bits of hashed values of records. They are by construction totally insensitive to the replicative structure of elements in the file; they can be used in the context of distributed systems without any degradation of performances and prove especially useful in the context of data bases query optimisation.

1. INTRODUCTION

As data base systems allow the user to specify more and more complex queries, the need arises for efficient processing methods. A complex query can however generally be evaluated in a number of different manners, and the overall performance of a data base system depends rather crucially on the selection of appropriate *decomposition strategies* in each particular case.

Even a problem as trivial as computing the intersection of two collections of data A and B lends itself to a number of different treatments (see, e.g., [7]):

$A \cap B$: 1. Sort A, search each element of B in A and retain it if it appears in A;

2. sort A, sort B, then perform a merge-like operation to determine the intersection;

3. eliminate duplicates in A and/or B using hashing or hash filters, then perform Algorithm 1 or 2.

Each of these evaluation strategy will have a cost essentially determined by the number of records a, b in A and B, and the number of *distinct* elements α, β in A and B, and for typical sorting methods, the costs are:

0022-0000/85 $3.00

Contributions
- Introduced problem
- Idea of *streaming algorithm*
- Idea of "small" *sketch* of "big" data
- Detailed analysis that yields tight bounds on accuracy
- Full validation of mathematical results with experimentation
- Practical algorithm that has remained effective for decades

Bottom line: Quintessential example of the effectiveness of scientific approach to AofA.

PCSA first step: Use hashing

Transform value to a "random" computer word.

- Compute a *hash function* that transforms data value into a 32- or 64-bit value.
- Cardinality count is unaffected (with high probability).
- Built-in capability in modern systems.
- *Allows use of fast machine-code operations.*

20th century: use 32 bits (millions of values)
21st century: use 64 bits (quadrillions of values)

Example: Java

- All data types implement a hashCode() method (though we often override the default).
- String data type stores value (computed once).

```
String value = "gsearch.CS.Princeton.EDU"
int x = value.hashCode();
```

current Java default is 32-bit int value

Bottom line: Do cardinality estimation on streams of (binary) integers.

```
01111000100111110111000111001000
01111000100111110111000111001000
01110101010110110000000011011010
00110100010001111100010100111010
00010000111001101000111010010011
00001001011011100000010010010111
00001001011011100000010010010111
```

"Random" *except* for the fact that some values are equal.

Initial hypothesis

Hypothesis. Knuth's assumption about hash functions is reasonable in this context.

Implication. Need to run experiments to validate any hypotheses about performance.

No problem!

- AofA is a scientific endeavor (we always validate hypotheses).
- End goal is development of algorithms that are useful in practice.
- It is the responsibility of the *designer* to validate utility before claiming it.
- After decades of experience, discovering a performance problem due to a bad hash function would be a significant research result.

Unspoken bedrock principle of AofA.
Experimenting to validate hypotheses is **WHAT WE DO!**

Probabilistic counting starting point: three integer functions

Definition. $p(x)$ is the **number of 1s** in the binary representation of x.

Definition. $r(x)$ is the **number of *trailing* 1s** in the binary representation of x. ← *position of rightmost 0*

Definition. $R(x) = 2^{r(x)}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	$p(x)$	$r(x)$	$R(x)$	
1	0	1	1	1	1	0	1	1	1	1	1	0	1	0	1	**12**	**1**	**2**	**= 1 0**
1	0	1	0	1	0	1	0	1	0	0	0	1	1	1	0	**8**	**0**	**1**	**= 1**
0	1	1	0	1	0	0	1	0	1	0	**1**	**1**	**1**	**1**	**1**	**10**	**5**	**32**	**= 1 0 0 0 0**

Bit-whacking magic:
$R(x)$ is easy to compute.

0	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	*x*
1	0	0	1	0	1	1	0	1	0	1	0	0	0	0	0	*~x*
0	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	*x + 1*
0	0	0	0	0	0	0	0	0	0	**1**	**0**	**0**	**0**	**0**	**0**	*~x & (x + 1)*

3 instructions on a typical computer

Exercise: Compute $p(x)$ as easily.

Beeler, Gosper, and Schroeppel
HAKMEM item 169, MIT AI Laboratory AIM 239, 1972
`http://www.inwap.com/pdp10/hbaker/hakmem/hakmem.html`

Note: $r(x) = p(R(x) - 1)$.

see also Knuth volume 4A

Bottom line: $p(x)$, $r(x)$, and $R(x)$ all can be computed with *just a few machine instructions*.

Probabilistic counting (Flajolet and Martin, 1983)

Maintain a single-word *sketch* that summarizes a data stream $x_0, x_1, \ldots, x_N, \ldots$

- For each x_N in the stream, update sketch by *bitwise or* with $R(x_N)$.
- Use position of rightmost 0 (with slight correction factor) to estimate $\lg N$.

estimate of lg N

typical sketch
***N* = 10⁶**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
sketch	0	0	0	0	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$\boldsymbol{x_N}$	0	0	1	1	0	1	0	1	0	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	1	1	1	1
$\boldsymbol{R(x_N)}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
***sketch* | $\boldsymbol{R(x_N)}$**	0	0	0	0	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

$R(x) = 2^k$ with probability $1/2^k$

leading bits almost surely 0

trailing bits almost surely 1

Rough estimate of $\lg N$ is $r(sketch)$.

Rough estimate of N is $R(sketch)$. ← *correction factor needed (stay tuned)*

Probabilistic counting trace

x	*r(x)*	*R(x)*	*sketch*
011000100110001110100111101110**11**	2	100	00000000000000000000000000000**1**00
0110011100100011000111110000010**1**	1	10	000000000000000000000000000001**1**0
000100010001110001101101101100**11**	2	100	00000000000000000000000000000**1**10
010001000111011100000001110**11111**	5	100000	00000000000000000000000000**1**00110
01101000001011000101110001000100	0	1	0000000000000000000000000010011**1**
0011011110110000000010100101010**1**	1	10	000000000000000000000000001001**1**1
00110100011000111010101111111100	0	1	0000000000000000000000000010011**1**
00011000010000100001011100110**111**	3	1000	00000000000000000000000000 10**1**111
00011001100110011110010000**111111**	6	1000000	0000000000000000000000000**1**101111
01000101110001001010110011111100	0	1	0000000000000000000000000110**1111**

R(sketch) $= 10000_2$
$= 16$

Probabilistic counting (first try)

```
public long R(long x)
{ return ~x & (x+1); }

public long estimate(Iterable<String> stream)
{
   long sketch;
   for (s : stream)
      sketch = sketch | R(s.hashCode());
   return R(sketch);
}
```

Maintain a *sketch* of the data

- A single word
- OR of all values of R(x) in the stream

Estimate is smallest value not seen.

Early example of "a simple algorithm whose analysis isn't"

Q. (Martin) Seems a bit low. How much?

A. (unsatisfying) Obtain empirically.

Probabilistic counting (Flajolet and Martin)

```
public long R(long x)
{ return ~x & (x+1); }

public long estimate(Iterable<String> stream)
{
   long sketch;
   for (s : stream)
      sketch = sketch | R(s.hashCode());
   return R(sketch)/.77351;
}
```

Maintain a *sketch* of the data
- A single word
- OR of all values of R(x) in the stream

Estimate is smallest value not seen.

with correction for bias

Early example of "a simple algorithm whose analysis isn't"

Q. Value of correction factor? How accurate?

A. (Flajolet) Analytic combinatorics!

Mathematical analysis of probabilistic counting

Theorem. *The expected number of trailing 1s in the PC sketch is*

$$\lg(\phi N) + P(\lg N) + o(1) \quad \textit{where} \;\; \phi \doteq .77351$$

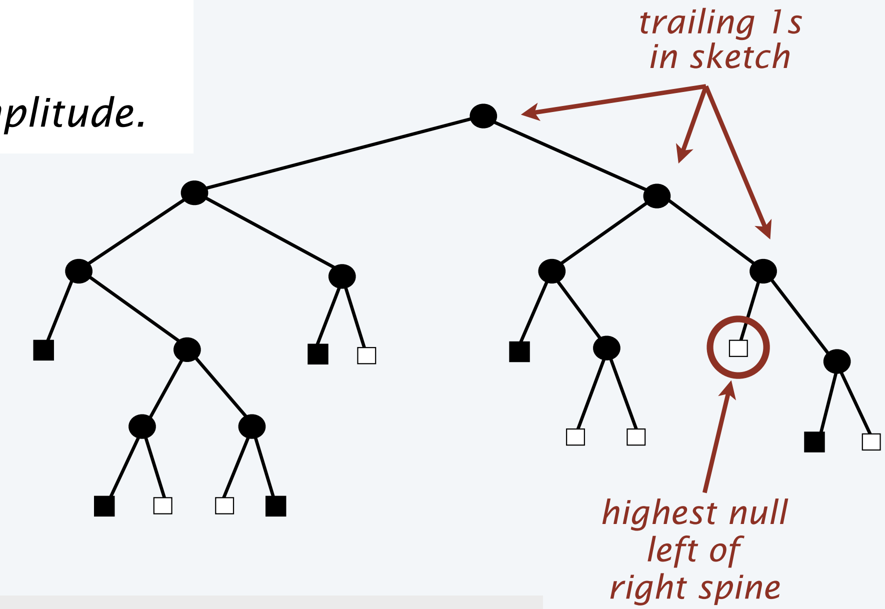
and P is an oscillating function of lg *N of very small amplitude.*

Proof (omitted).

1980s: Flajolet *tour de force*

1990s: trie parameter

21st century: standard AC



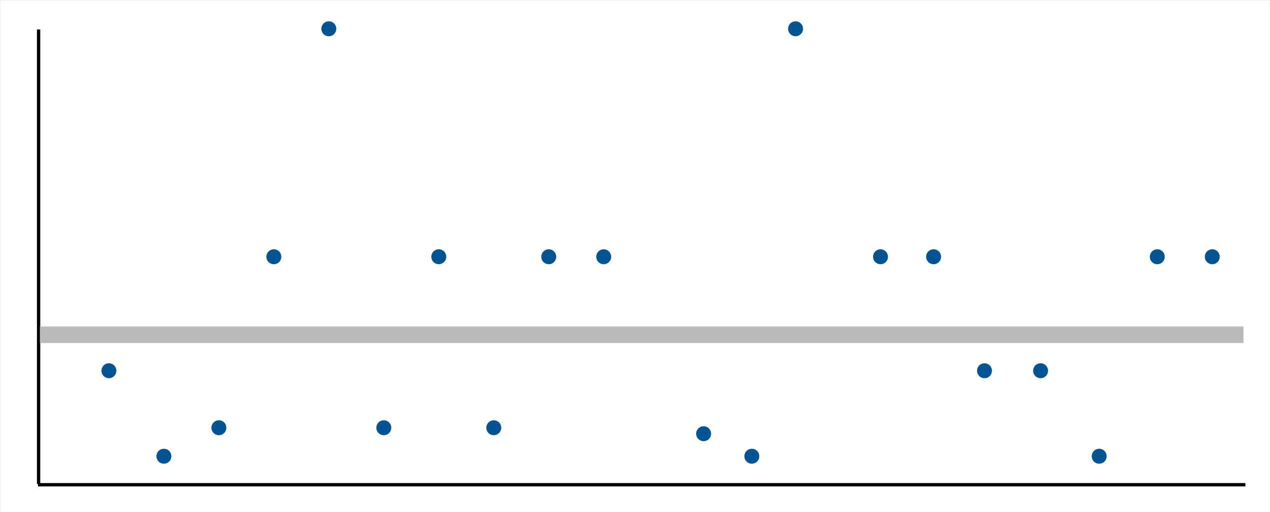
Kirschenhofer, Prodinger, and Szpankowski
Analysis of a splitting process arising in probabilistic counting and other related algorithms
ICALP 1992.

In other words. In PC code, `R(sketch)/.77351` is an *unbiased statistical estimator* of *N*.

Validation of probabilistic counting

Hypothesis. Expected value returned is N *for random values from a large range.*

Quick experiment. 100,000 31-bit random values (20 trials)



Flajolet and Martin: Result is “typically one binary order of magnitude off.”

Of course! (Always returns a power of 2 divided by .77351.)

16384/.77351 = 21181
32768/.77351 = 42362
65536/.77351 = 84725
...

Need to incorporate more experiments for more accuracy.

Cardinality Estimation

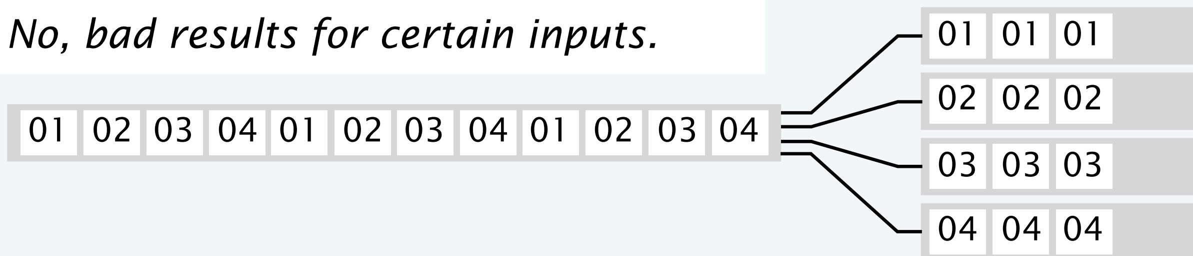
- Rules of the game
- Probabilistic counting
- Stochastic averaging
- Refinements
- Final frontier

Stochastic averaging

Goal: Perform M independent PC experiments and average results.

Alternative 1: M independent hash functions? *No, too expensive.*

Alternative 2: M-way alternation? *No, bad results for certain inputs.*



Alternative 3: *Stochastic averaging*

- Use second hash to divide stream into 2^m independent streams
- Use PC on each stream, yielding 2^m sketches .
- Compute *mean* = average number of trailing bits in the sketches.
- Return $2^{mean}/.77531$.

key point: equal values all go to the same stream

PCSA trace

use initial m bits for second hash

M = 4

x	*R(x)*	*sketch[0]*	*sketch[1]*	*sketch[2]*	*sketch[3]*
101001111011101**1**	100	0000000000000000	0000000000000000	0000000000000100	0000000000000000
000111110000010**1**	10	0000000000000010	0000000000000000	0000000000000100	0000000000000000
011011011011001**1**	100	0000000000000010	0000000000000100	0000000000000100	0000000000000000
0000000111011111	100000	0000000000100010	0000000000000100	0000000000000100	0000000000000000
0101110001000100	1	0000000000100010	0000000000000101	0000000000000100	0000000000000000
0000101001010101	10	0000000000100010	0000000000000101	0000000000000100	0000000000000000
1010101111111100	1	0000000000100010	0000000000000101	0000000000000101	0000000000000000
0001011100110111	1000	0000000000101010	0000000000000101	0000000000000101	0000000000000000
1110010000111111	1000000	0000000000101010	0000000000000101	0000000000000101	0000000001000000
1010110011111101	10	0000000000101010	0000000000000101	0000000000000111	0000000001000000
0001110100110100	1	0000000000101011	0000000000000101	0000000000000111	
		0000000000101011	0000000000000101	0000000000000111	0000000001000000
r (sketch[ ])		*2*	*1*	*3*	*0*

Probabilistic counting with stochastic averaging in Java

```
public static long estimate(Iterable<Long> stream, int M)
{
   long[] sketch = new long[M];
   for (long x : stream)
   {
      int k = hash2(x, M);
      sketch[k] = sketch[k] | R(x);
   }
   int sum = 0;
   for (int k = 0; k < M; k++)
      sum += r(sketch[k]);
   double mean = 1.0 * sum / M;
   double phi = .77351;
   return (int) (M * Math.pow(2, mean)/phi);
}
```

Flajolet-Martin 1983

Idea. *Stochastic averaging*

- Use second hash to split into $M = 2^m$ independent streams
- Use PC on each stream, yielding 2^m sketches .
- Compute *mean* = average # trailing 1 bits in the sketches.
- Return $2^{mean}/.77351$.

Theoretical analysis of PCSA

Definition. The *relative accuracy* is the standard deviation of the estimate divided by the actual value.

Theorem (paraphrased to fit context of this talk).

Under the uniform hashing assumption, PCSA

- *Uses 64M bits.*
- *Produces estimate with a relative accuracy close to* $0.78/\sqrt{M}$.

LEMMA 4. *Setting* $\beta = 2^{1/q}$, *with* $q \geqslant 1$, *one has for fixed* q

$$\mathrm{E}[\beta^{R_n}] = n^{1/q}(d_q + P_q(\log_2 n)) + o(n^{1/q}),$$

where

probability

$$1-\left(1-\frac{1}{2^k}\right)^n-\left(1-\frac{1}{2^{k-1}}\right)^n+\left(1-\frac{1}{2^k}-\frac{1}{2^{k-1}}\right)^n$$

a quantity which is

$$1-e^{-n/2^k+O(n/2^{2k})}-e^{-n/2^{k-1}+O(n/2^{2k})}+e^{-3n/2^{k-1}+O(n/2^{2k})}$$

or $O(n/2^{2k})$, which in the given range of values of k is $O(n^{-3/2}4^{-\delta})$. Thus

$$\sum_{k>(5/4)\log_2 n} 2^k p_{n,k} = O\left(n^{5/4-3/2}\sum_{\delta \geqslant 0} 4^{-\delta}2^{\delta}\right) = O(n^{-1/4}), \qquad (28)$$

and the same bound applies if 2 is replaced by β in the above sum.

We now consider the error that comes from the replacement of the $p_{n,k}$ by their asymptotic equivalent for "small" k. From the bounds of Theorem 2, one finds

$$\sum_{k \leqslant (5/4)\log_2 n} \beta^k \left[p_{n,k}-\psi\left(\frac{n}{2^k}\right)+\psi\left(\frac{n}{2^{k+1}}\right)\right] = O\left(\frac{n^{5/4q}}{n^{0.49}}\right) = O(n^{0.76/q}), \qquad (29)$$

LEMMA 5. *If* n *elements are distributed into* m *cells* (m *fixed*), *where the probability that any element goes to a given cell has probability* $1/m$, *then the probability that at least one of the cells has a number of elements* N *satisfying*

$$|N-n/m| > \sqrt{n}\log n$$

nt $h>0$.

$-1/m$; let N_1 be the number of elements that fall into istribution

$$\Pr(N_1=k)=\binom{n}{k}p^k q^{n-k}, \qquad (30)$$

and taking logarithms of (30), for $k = pn+\delta$ and $\delta \ll n$, one finds

$$\Pr(N_1=pn+\delta)=\exp\left(-\frac{\delta^2+O(\delta)}{2npq}+O\left(\frac{\delta^3}{n^2}\right)\right).$$

If $\delta = \sqrt{n}\log n$, the probability (30) is exponentially small. We conclude the proof by observing that the binomial distribution is unimodal and

$$\Pr\left[\bigcup_{1\leqslant j\leqslant m}\left|N_j-\frac{n}{m}\right|>\sqrt{n}\log n\right] < m\Pr\left[\left|N_1-\frac{n}{m}\right|>\sqrt{n}\log n\right]. \quad ▮$$

We can now conclude the proof of the first part of Theorem 4. Let S denote the sum $R^{\langle 1\rangle}+R^{\langle 2\rangle}+\cdots+R^{\langle m\rangle}$. We have

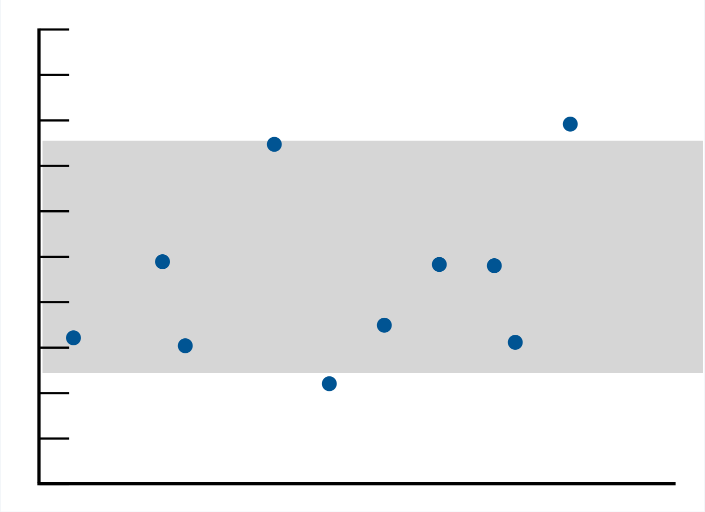
$$\Pr(S=k)=\sum_{\substack{n_1+n_2+\cdots+n_m=n\\k_1+k_2+\cdots+k_m=k}}\frac{1}{m^n}\binom{n}{n_1,n_2,\ldots,n_m}p_{n_1,k_1}p_{n_2,k_2}\cdots p_{n_m,k_m}. \qquad (31)$$

Validation of PCSA analysis

Hypothesis. Value returned is accurate to $0.78/\sqrt{M}$ *for random values from a large range.*

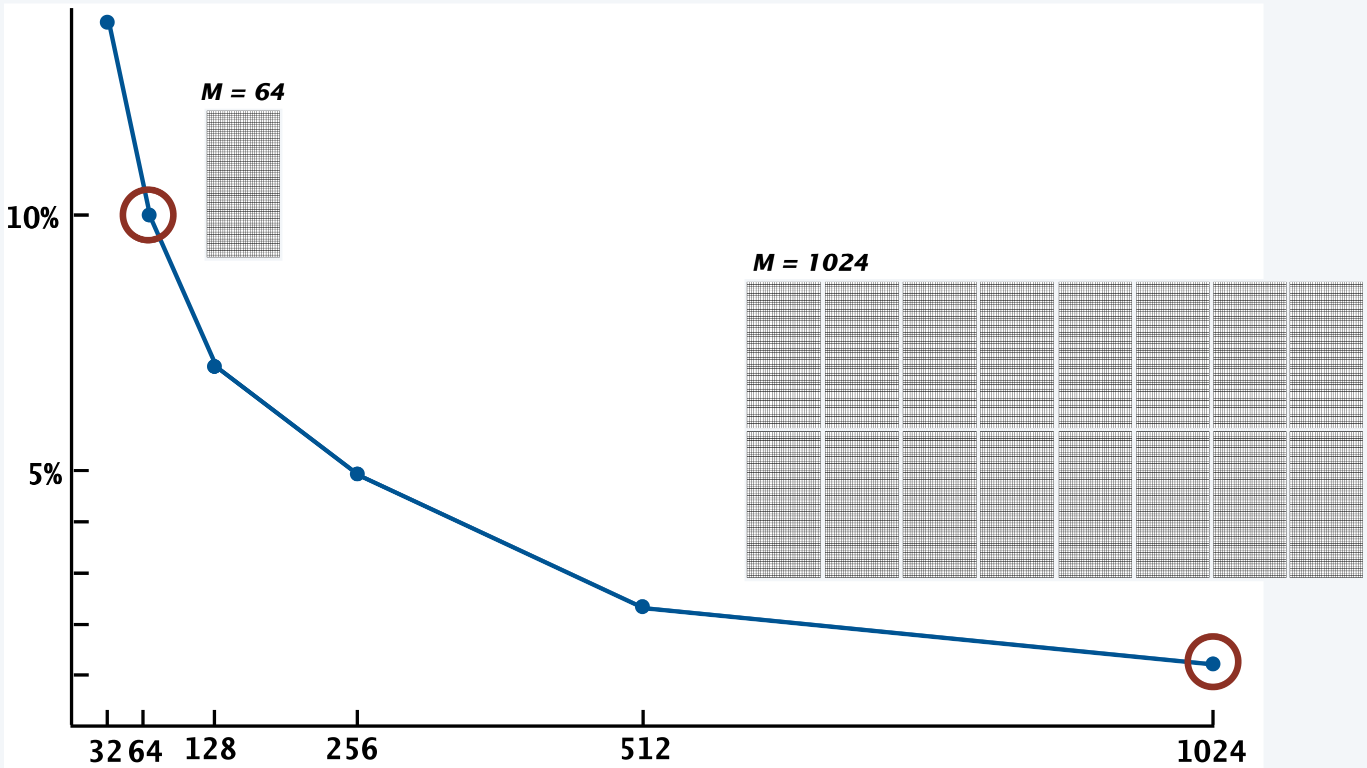
Experiment. 1,000,000 31-bit random values, $M = 1024$ (10 trials)

```
% java PCSA 1000000 31 1024 10
964416
997616
959857
1024303
972940
985534
998291
996266
959208
1015329
```



Space-accuracy tradeoff for probabilistic counting with stochastic averaging

Relative accuracy: $\frac{0.78}{\sqrt{M}}$



Bottom line.

- Attain 10% relative accuracy with a sketch consisting of 64 words.
- Attain 2.4% relative accuracy with a sketch consisting of 1024 words.

Scientific validation of PCSA

Hypothesis. Accuracy is as specified *for the hash functions we use and the data we have.*

Validation (Flajolet and Martin, 1985). Extensive reproducible scientific experiments (!)

Validation (RS, this morning).

log.07.f3.txt

```
109.108.229.102
pool-71-104-94-246.lsanca.dsl-w.verizon.net
117.222.48.163
pool-71-104-94-246.lsanca.dsl-w.verizon.net
1.23.193.58
188.134.45.71
1.23.193.58
gsearch.CS.Princeton.EDU
pool-71-104-94-246.lsanca.dsl-w.verizon.net
81.95.186.98.freenet.com.ua
81.95.186.98.freenet.com.ua
81.95.186.98.freenet.com.ua
CPE-121-218-151-176.lnse3.cht.bigpond.net.au
```

```
% java PCSA 6000000 1024 < log.07.f3.txt
1106474
```

<1% larger than actual value

Q. Is PCSA effective?

A. ABSOLUTELY!

Summary: PCSA (Flajolet-Martin, 1983)

is a *demonstrably* effective approach to cardinality estimation

Q. *About* how many different values are present in a given stream?

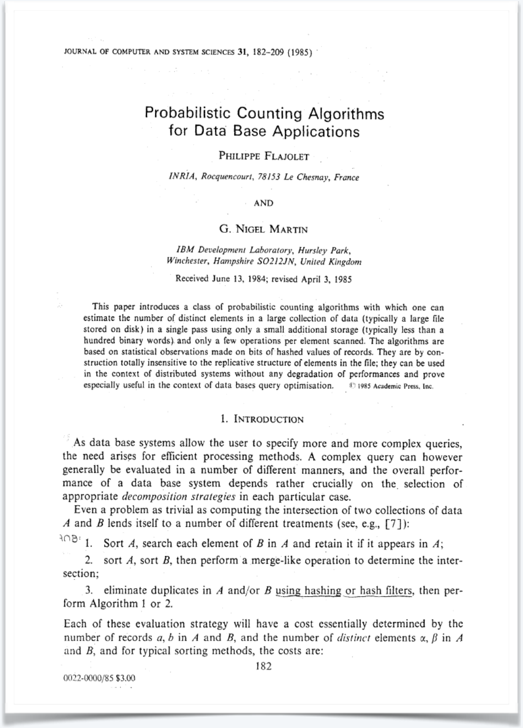
PCSA

- Makes *one pass* through the stream.
- Uses *a few machine instructions per value*
- Uses *M* words to achieve relative accuracy $0.78/\sqrt{M}$ ✓

Results validated through extensive experimentation.

Open questions

- Better space-accuracy tradeoffs?
- Support other operations?



JOURNAL OF COMPUTER AND SYSTEM SCIENCES 31, 182–209 (1985)

Probabilistic Counting Algorithms for Data Base Applications

PHILIPPE FLAJOLET

INRIA, Rocquencourt, 78153 Le Chesnay, France

AND

G. NIGEL MARTIN

IBM Development Laboratory, Hursley Park, Winchester, Hampshire SO212JN, United Kingdom

Received June 13, 1984; revised April 3, 1985

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 than a hundred binary words) and only a few operations per element scanned. The algorithms are based on statistical observations made on bits of hashed values of records. They are by construction totally insensitive to the replicative structure of elements in the file; they can be used in the context of distributed systems without any degradation of performances and prove especially useful in the context of data bases query optimisation. © 1985 Academic Press, Inc.

1. INTRODUCTION

As data base systems allow the user to specify more and more complex queries, the need arises for efficient processing methods. A complex query can however generally be evaluated in a number of different manners, and the overall performance of a data base system depends rather crucially on the selection of appropriate *decomposition strategies* in each particular case.

Even a problem as trivial as computing the intersection of two collections of data *A* and *B* lends itself to a number of different treatments (see, e.g., [7]):

$A \cap B$: 1. Sort *A*, search each element of *B* in *A* and retain it if it appears in *A*;

2. sort *A*, sort *B*, then perform a merge-like operation to determine the intersection;

3. eliminate duplicates in *A* and/or *B* using hashing or hash filters, then perform Algorithm 1 or 2.

Each of these evaluation strategy will have a cost essentially determined by the number of records a, b in *A* and *B*, and the number of *distinct* elements α, β in *A* and *B*, and for typical sorting methods, the costs are:

182

0022-0000/85 $3.00

A poster child for AofA/AC

"IT IS QUITE CLEAR that other observable regularities on hashed values of records could have been used…

– Flajolet and Martin

Cardinality Estimation

- Rules of the game
- Probabilistic counting
- Stochastic averaging
- **Refinements**
- Final frontier

We can do better (in theory)

Alon, Matias, and Szegedy

The Space Complexity of Approximating the Frequency Moments

STOC 1996; JCSS 1999.

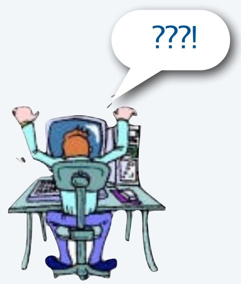
Contributions
- Studied problem of estimating higher moments
- Formalized idea of *randomized* streaming algorithms
- Won Gödel Prize in 2005 for "foundational contribution"

Theorem (paraphrased to fit context of this talk).

With strongly universal hashing, PC, for any $c > 2$,
- *Uses $O(\log N)$ bits.*
- Is accurate to a factor of *c, with probability at least 2/c*.

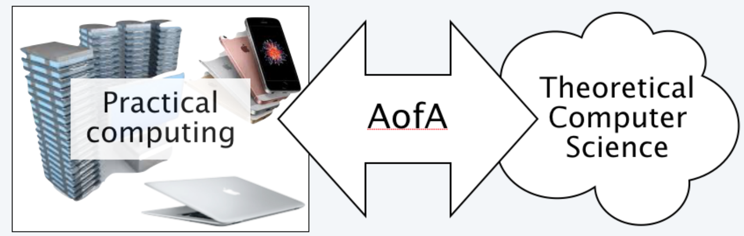
Replaces "uniform hashing" assumption with "random bit existence" assumption

BUT, no impact on cardinality estimation in practice
- "Algorithm" just changes hash function for PC
- Accuracy estimate is too weak to be useful
- No validation



Interesting quote

> *" Flajolet and Martin [assume] that one may use in the algorithm an explicit family of hash functions which exhibits some ideal random properties. Since we are not aware of the existence of such a family of hash functions ..."*
>
> *– Alon, Matias, and Szegedy*



No! They hypothesized that practical hash functions would be as effective as random ones.
They then validated that hypothesis by proving tight bounds that match experimental results.

Points of view re *hashing*

- Theoretical computer science. Uniform hashing assumption is not proved.
- Practical computing. Hashing works for many common data types.
- AofA. *Extensive experiments have validated precise analytic models.*

Points of view re *random bits*

- Theoretical computer science. Random bits exist.
- Practical computing. No, they don't! And randomized algorithms are inconvenient, btw.
- AofA. *More effective path forward is to validate precise analysis even if stronger assumptions are needed.*

We can do better (in theory)

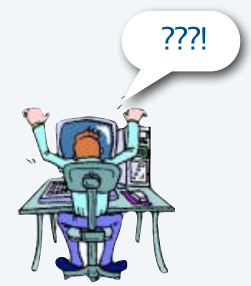
Bar-Yossef, Jayram, Kumar, Sivakumar, and Trevisan
Counting Distinct Elements in a Data Stream
RANDOM 2002.

Theoretical Computer Science

Contribution
Improves space-accuracy tradeoff at extra stream-processing expense.

Theorem (paraphrased to fit context of this talk).
With strongly universal hashing, there exists an algorithm that
- *Uses O(M log log N) bits.* ← *PCSA uses M lg N bits*
- Achieves relative accuracy $O(1/\sqrt{M})$.

STILL no impact on cardinality estimation in practice
- Infeasible because of high stream-processing expense.
- Big constants hidden in O-notation
- No validation

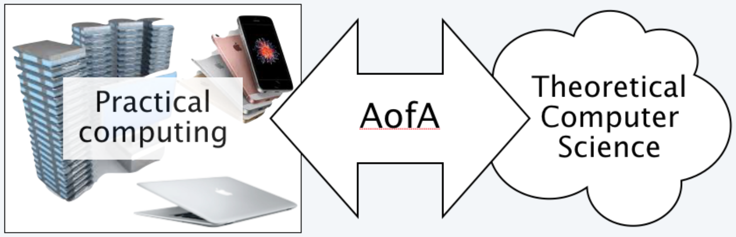


We can do better (in theory *and* in practice)

Durand and Flajolet

LogLog Counting of Large Cardinalities

ESA 2003; LNCS volume 2832.



Contributions (independent of BYJKST)

- Presents **LogLog** algorithm, an easy variant of PCSA
- Improves space-accuracy tradeoff *without* extra expense per value
- Full analysis, fully validated with experimentation

Theorem (paraphrased to fit context of this talk).

Under the uniform hashing assumption, ***LogLog***

- *Uses M lg lg N bits.*
- *Achieves relative accuracy close to* $1.30/\sqrt{M}$.

lg N bits can save a value (PCSA)

lg lg N bits can save a bit index in a value

Not much impact on cardinality estimation in practice *only because*

- PCSA was effectively deployed in practical systems
- Idea led to a better algorithm a few years later (stay tuned)

We can do better (in theory and in practice): LogLog algorithm (2003)

```
public static long estimate(Iterable<Long> stream, int M)
{
   int[] bytes = new int[M];
   for (long x : stream)
   {
      int k = hash2(x, M);
      if (bytes[k] < Bits.r(x)) bytes[k] = Bits.r(x);
   }
   int sum = 0;
   for (int k = 0; k < M; k++)
      sum += Bits.r(sketch[k]);
   double mean = 1.0 * sum / M;
   double alpha = .77351;
   return (int) (alpha * M * Math.pow(2, mean + 1.0));
}
```

Code to pack into
M loglogN bits omitted

Durand-Flajolet 2003

Idea. Keep track of min($r(x)$)
(with stochastic averaging).

Durand and Flajolet
LogLog Counting of Large Cardinalities
ESA 2003; LNCS volume 2832.

We can do better (in theory and in practice): HyperLogLog algorithm (2007)

```
public static long estimate(Iterable<Long> stream, int M)
{
   int[] bytes = new int[M];
   for (long x : stream)
   {
      int k = hash2(x, M);
      if (bytes[k] < Bits.r(x)) bytes[k] = Bits.r(x);
   }
   double sum = 0.0;
   for (int k = 0; k < M; k++)
      sum += Math.pow(2, -1.0 - bytes[k]);
 return (int) (alpha * M * M / sum);
}
```

Flajolet-Fusy-Gandouet-Meunier 2007

Idea. Use Harmonic mean.

Flajolet, Fusy, Gandouet, and Meunier
HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm
AofA 2007; DMTCS 2007.

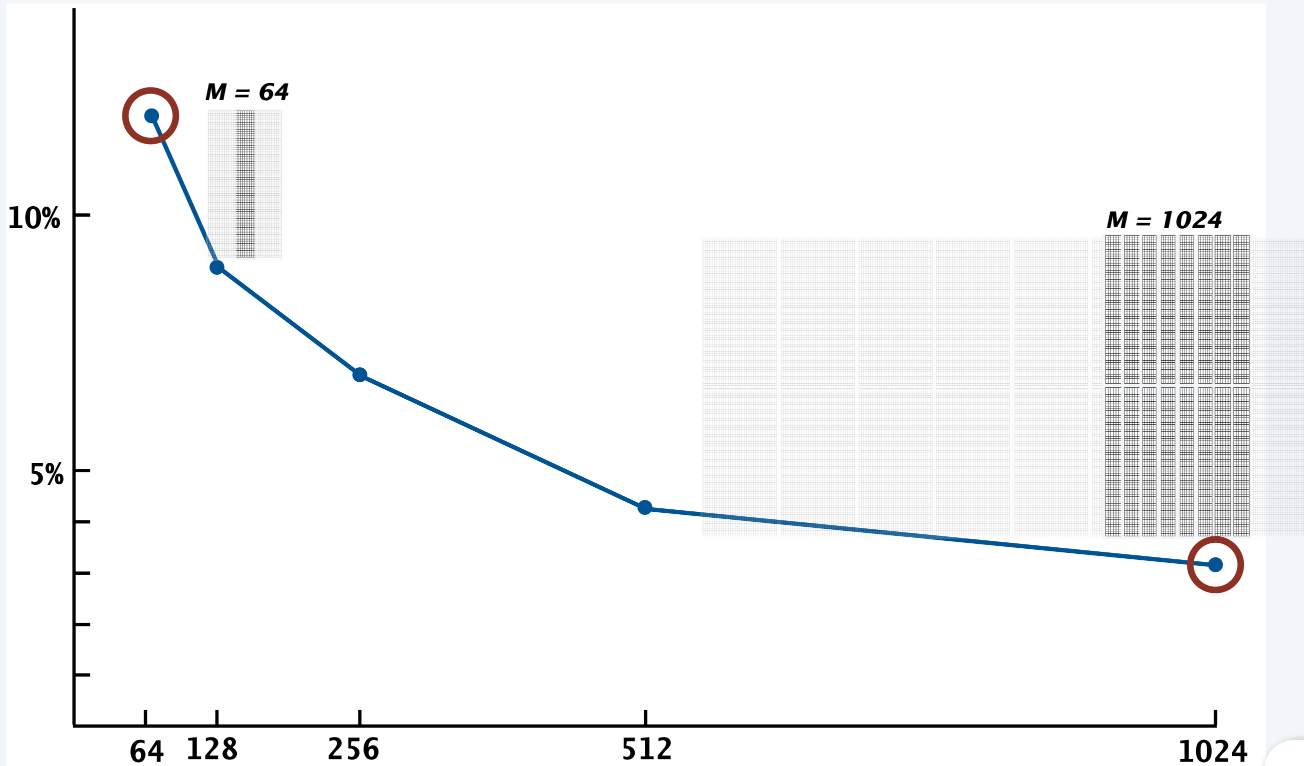
Theorem (paraphrased to fit context of this talk).

Under the uniform hashing assumption, ***HyperLogLog***

- *Uses M log log N bits.*
- *Achieves relative accuracy close to* $1.02/\sqrt{M}$.

Space-accuracy tradeoff for HyperLogLog

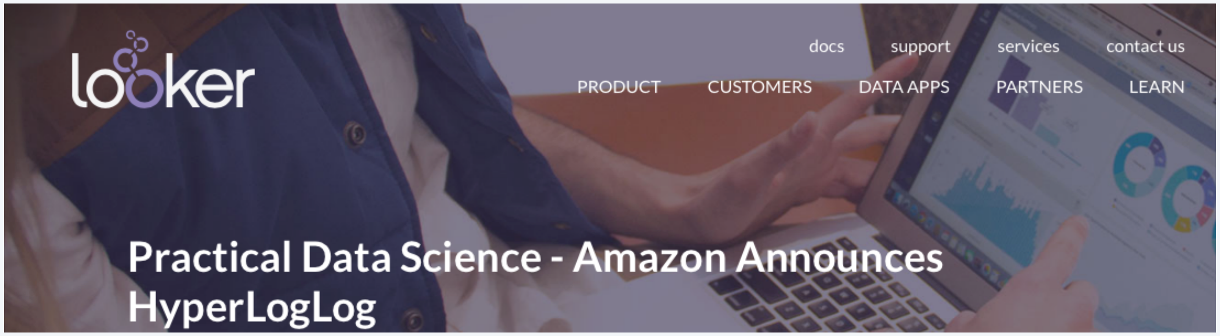
Relative accuracy: $\dfrac{1.02}{\sqrt{M}}$



Bottom line (for $N < 2^{64}$).

- Attain 12.5% relative accuracy with a sketch consisting of 64x6 = $\boxed{\text{396 } bits}$.
- Attain 3.1% relative accuracy with a sketch consisting of 1024x6 = 6144 *bits*.

Validation of Hyperloglog



cradlepoint

neustar

Periscope DATA

AMAZON REDSHIFT

BigQuery

redis

Research at Google

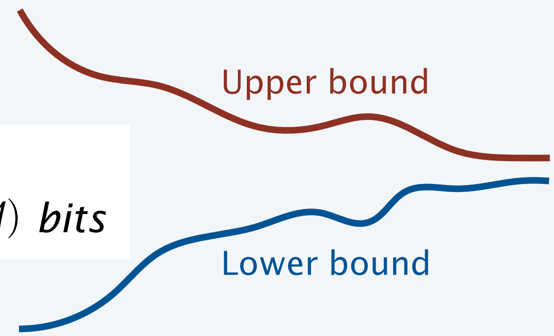
S. Heule, M. Nunkesser and A. Hall
HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm. Extending Database Technology/International Conference on Database Theory 2013.

Cardinality Estimation

- Rules of the game
- Probabilistic counting
- Stochastic averaging
- Refinements
- Final frontier

We can do a bit better (in theory) *but not much better*

Indyk and Woodruff
Tight Lower Bounds for the Distinct Elements Problem, FOCS 2003.



Theorem (paraphrased to fit context of this talk).
Any algorithm that achieves relative accuracy $O(1/\sqrt{M})$ *must use* $\Omega(M)$ *bits*

loglogN improvement possible

Kane, Nelson, and Woodruff
Optimal Algorithm for the Distinct Elements Problem, PODS 2010.

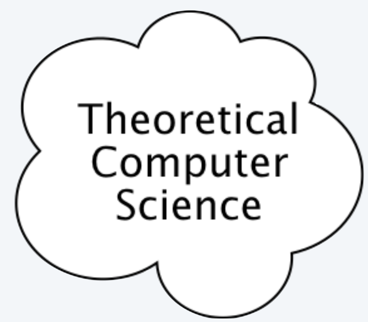
Theorem (paraphrased to fit context of this talk).
With strongly universal hashing there exists an algorithm that

- *Uses* O(*M*) *bits.*
- *Achieves relative accuracy* $O(1/\sqrt{M})$.

optimal

Unlikely to have impact on cardinality estimation in practice

- Tough to beat HyperLogLog's low stream-processing expense.
- Constants hidden in O-notation not likely to be < 6
- *No validation*



Can we beat HyperLogLog in practice?

Necessary characteristics of a better algorithm
- Makes *one pass* through the stream.
- Uses *a few dozen machine instructions per value*
- Uses *a few hundred bits*
- Achieves 10% relative accuracy or better

> *" I've long thought that there should be a simple algorithm that uses a small constant times M bits..."*
>
> *– Jérémie Lumbroso*

	machine instructions per stream element	*memory bound*	*memory bound when $N < 2^{64}$*	*# bits for 10% accuracy when $N < 2^{64}$*
HyperLogLog	20–30	$M \log\log N$	$6M$	768
BetterAlgorithm	*a few dozen*			*a few hundred*

Also, results need to be validated through extensive experimentation.

A proposal: HyperBitBit (Sedgewick, 2016)

```
public static long estimate(Iterable<String> stream, int M)
{
   int lgN =  5;
   long sketch = 0;
   long sketch2 = 0;
   for (String x : stream)
   {
      long x = hash(s);
      int k = hash2(x, 64);
      if (r(x) > lgN)     sketch  = sketch  | (1L << k);
      if (r(x) > lgN + 1) sketch2 = sketch2 | (1L << k);
      if (p(sketch) > 31)
      { sketch = sketch2; lgN++; sketch2 = 0; }
   }
   return (int) (Math.pow(2, lgN + 5.4 + p(sketch)/32.0));
}
```

bias (determined empirically)

Idea.

- lgN is estimate of $\lg N$
- sketch is 64 indicators whether to increment lgN
- sketch2 is is 64 indicators whether to increment lgN *by 2*
- Update when half the bits in sketch are 1

Initial experiments

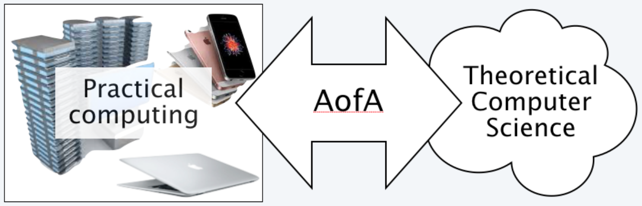
Exact values for web log example

```
% java Hash 1000000 < log.07.f3.txt
242601
% java Hash 2000000 < log.07.f3.txt
483477
% java Hash 4000000 < log.07.f3.txt
883071
% java Hash 6000000 < log.07.f3.txt
1097944
```

HyperBitBit estimates

```
% java HyperBitBit 1000000 < log.07.f3.txt
234219
% java HyperBitBit 2000000 < log.07.f3.txt
499889
% java HyperBitBit 4000000 < log.07.f3.txt
916801
% java HyperBitBit 6000000 < log.07.f3.txt
1044043
```

	1,000,000	2,000,000	4,000,000	6,000,000
Exact	242,601	483,477	883,071	1,097,944
HyperBitBit	234,219	499,889	916,801	1,044,043
ratio	1.05	1.03	0.96	1.03



Conjecture. *On practical data,* ***HyperBitBit****, for* $N < 2^{64}$,

- *Uses* 128 + 6 *bits.*
- *Estimates cardinality within 10% of the actual.*

Next steps.

- *Analyze.*
- *Experiment.*
- *Iterate*

Small sample of work on related problems

1970	Bloom	set membership
1984	Wegman	unbiased sampling estimate
2000	Indyk	L1 norm
2004	Cormode–Muthukrishnan	frequency estimation deletion and other operations
2005	Giroire	fast stream processing
2012	Lumbroso	full range, asymptotically unbiased
2014	Helmi–Lumbroso–Martinez–Viola	uses neither sampling nor hashing

Summary/timeline for cardinality estimation

			hashing assumption	feasible and validated?	memory bound	relative accuracy constant	# bits for 10% accuracy when $N < 2^{64}$
1985	Flajolet-Martin	**PCSA**	*uniform*	✓	$M \log N$	0.78	4096
1996	Alon-Matias-Szegedy	[*theorem*]	*strong universal*	✗	$O(M \log N)$	O(1)	?
2002	Bar-Yossef-Jayram-Kumar-Sivakumar-Trevisan	[*theorem*]	*strong universal*	✗	$O(M \log \log N)$	O(1)	?
2003	Durand-Flajolet	**LogLog**	*uniform*	✓	$M \operatorname{lglg} N$	1.30	1536
2007	Flajolet-Fusy-Gandouet-Meunier	**HyperLogLog**	*uniform*	✓	$M \operatorname{lglg} N$	1.02	768
2010	Kane-Nelson-Woodruff	[*theorem*]	*strong universal*	✗	$O(M) + \operatorname{lglg} N$	O(1)	?
2016?		**HyperBitBit**			$2M + \operatorname{lglg} N$	?	134 (?)

Philippe Flajolet, mathematician, data scientist, and computer scientist extraordinaire

Philippe Flajolet 1948–2011

Cardinality Estimation

Robert Sedgewick
Princeton University

with special thanks to Jérémie Lumbroso