

How to Count with Randomness?
Comment compter avec le hasard?

Philippe Flajolet, INRIA, Rocquencourt

<http://algo.inria.fr/flajolet>

Joint with (2003–2007):

Marianne Durand, Éric Fusy, Olivier Gandouet, Frédéric Meunier

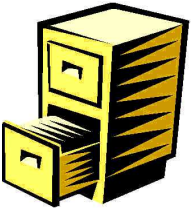
— PMA, Paris, September 20, 2007 in “*Probabiliste par hasard*”.

A single problem:

Given a long sequence of items, how many different ones are there?



How many different words in Shakespeare?



How many different cities in a file `<persons;cities>`?



How many different connections through a router in 1H?

stream: $s = s_1 s_2 \cdots s_\ell, \quad s_j \in \mathcal{D}.$

Length can be $\ell \gg 10^9$. Cardinality can be $n \propto 10^7$.



Routers in the range of Terabits/sec (10^{12} b/s).



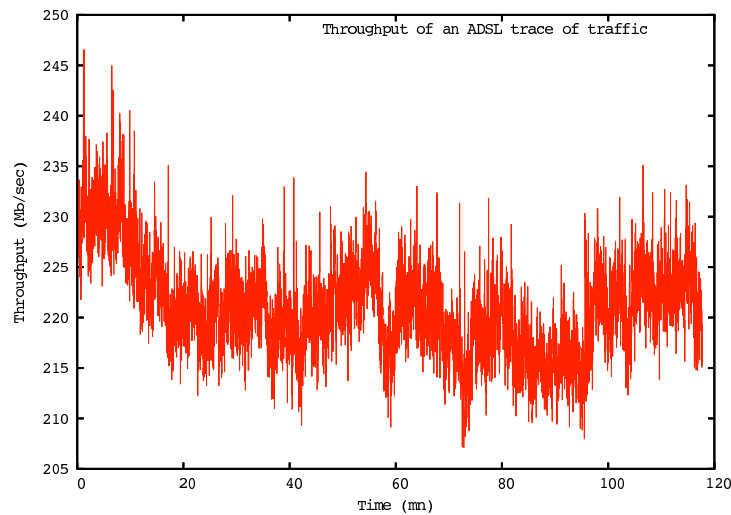
Google indexes 6 billion pages & prepares to index 100 Petabytes of data (10^{17} B).

Can estimate cardinalities, QUICK and EASY

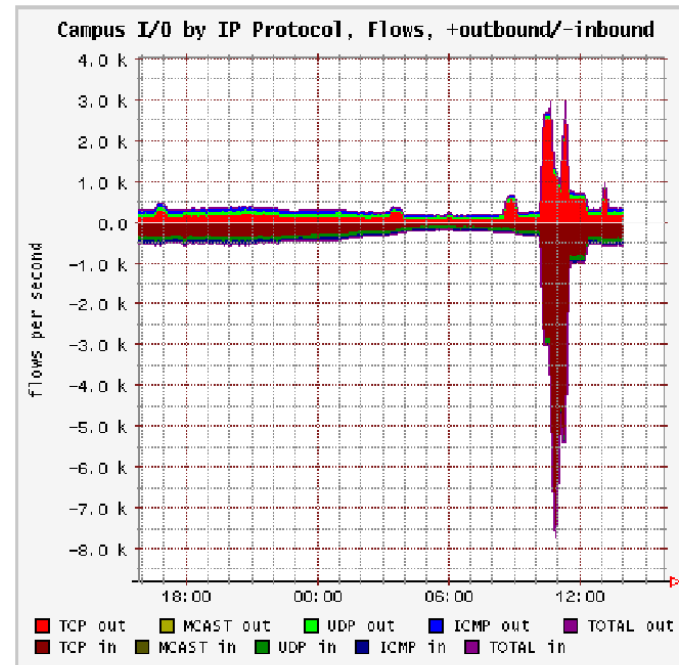
What for?

— Network management, worms and viruses, traffic monitoring

Traces of attacks: Number of active connections in time slices.



(Raw ADSL traffic)



(Attack)

Incoming/Outgoing flows at 40Gbits/second. Code Red Worm: 0.5GBytes of compressed data per hour (2001). CISCO: in 11 minutes, a worm infected 500,000,000 machines.

Left: ADSL FT@Lyon 1.5×10^8 packets (21h–23h). Right: (Estan-Varghese-Fisk) different incoming/outgoing connections

Rules of the game

- **Limited storage**: cannot store elements; use \approx **one page** of print \equiv 4kB..
- **Limited time**: proceed online = single pass, read once data.
- Allow to **estimate** rather than compute exactly.



HASHING

A **hash function** $h : \mathcal{D} \mapsto [0, 1]$ scrambles data *uniformly*:

“chevaleret” $\mapsto 0x7BF937A2$



Angel-daemon scenario: n values, replicated and permuted at will, then made into **random uniform** $[0, 1]$.



Solutions are known: $s \mapsto [s \text{ in base } 29] \bmod 1,000,003 \ \& \ \exists \text{ theorems!}$



OBSERVABLES

= *depend on cardinality of underlying set only!*

— **Patterns**: `00001` occurs with probability 2^{-5} ;

Thus (!) I have ≥ 32 elements

(Fl-Martin'85) (Durand-Fl'03) (Fl-Fusy-Gandouet-Meunier'07)

— **Order**: If $\min = 10^{-5}$ then I have $\approx 10^5$ elems!?

(Bar-Youssef'02) (Giroire2006)

These can be turned into (rough) “**indicators**” of cardinality.

Task 1: find “good” observables

Task 2: transform an observable into a *decent* algorithm

— **2a:** find a good *evaluation function*

— **2b:** *correct bias*, either systematic or non-asymptotic

— **2c:** *get guarantees*: estimate accuracy aka standard error. . .

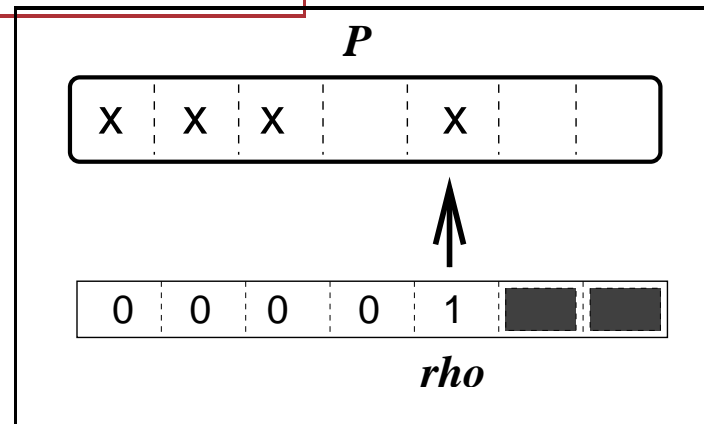
++take computational costs into account.

The Probabilistic Counting Observable

First Counting Algorithm

for cardinalities
(Fl-Martin'85)

:



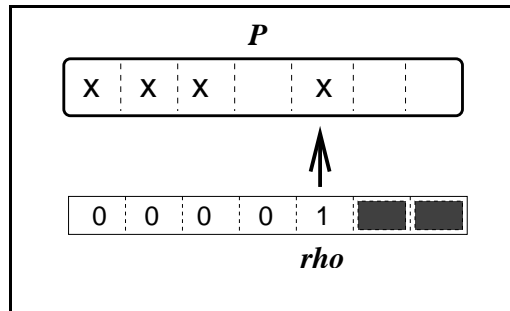
Algorithm: Probabilistic Counting

Input: a stream S ; Output: cardinality $|S|$

For each $x \in S$ do /* $\rho \equiv$ position of leftmost 1-bit */

Set $\text{BITMAP}[\rho(\text{hash}(w))] := 1$; od;

Return P where P is position of *first* 0.



Theorem: P estimates $\log_2(\varphi n)$, with $\varphi \doteq 0.77351$.

$$\varphi = \frac{e^\gamma}{\sqrt{2}} \prod_{n \geq 2}^* n^{\epsilon(n)}, \quad \epsilon(n) := (-1)^{\sum \text{bits}(n)}.$$

- Straight averaging over m trials: $\text{Ave} = \frac{1}{m} [P_1 + \dots + P_m]$; return $\frac{1}{\varphi} 2^{\text{Ave}}$.

The **standard error** is about $\frac{0.78}{\sqrt{m}}$ = the best known to-day

Of course, this is not quite right, since $\mathbb{E}[2^X] \neq 2^{\mathbb{E}[X]}$; also there are tiny fluctuations; but all this can be patched with **analysis** \leadsto

Mellin transform $f^*(s) = \int_0^\infty f(x) dx$, from real to *complex*.

♥ **Maps** asymptotics of f at 0 and $+\infty$ **to singularities of f^*** in \mathbb{C} :

$$C \cdot x^\alpha \xleftrightarrow{\mathcal{M}} \pm \frac{C}{s + \alpha}.$$

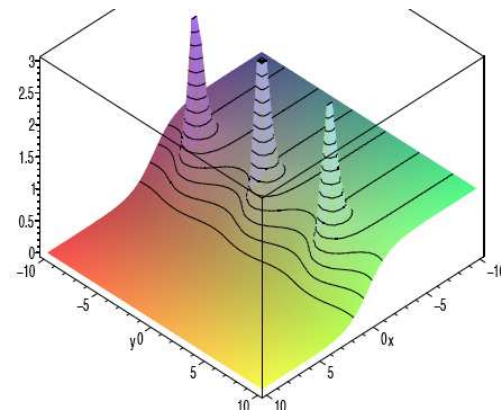
Reason: Inversion theorem $\frac{1}{2i\pi} \int_{c-i\infty}^{c+i\infty} f^*(s)x^{-s} ds$ + Residues.

♥ **Factorizes** harmonic sums:

$$\sum \lambda \cdot f(\mu x) \xrightarrow{\mathcal{M}} f^*(s) \cdot \sum \frac{\lambda}{\mu^s}.$$

For dyadic sums: $\sum f(x2^{-k}) \xrightarrow{\mathcal{M}} \frac{f^*(s)}{1 - 2^s}$

$$\alpha = 2ik\pi / \log 2 \implies x^{-\alpha} = e^{-2ik\pi \log_2 x}$$



We seem to be home, but not quite!

FORGOT computation cost: want m in the order of 10^2 – 10^3 !!

— Stochastic averaging = *one* hash function and $O(1)$ per record:

Split (mentally) stream: e.g., $S \mapsto (S_{000}, \dots, S_{111})$, for $m = 8$;

Work out each $P_j := P(S_j)$ separately; /* cost $O(1)$ per element */

Let $Ave := \frac{1}{m} [P_1 + \dots + P_m]$; /* used to estimate $\frac{n}{m}$ */

Return $\frac{m}{\varphi} 2^{Ave}$.

Because of (ir)regularities of distributions, we're Okay; only get into asymptotic regime later.

Application: Data mining of the Internet graph

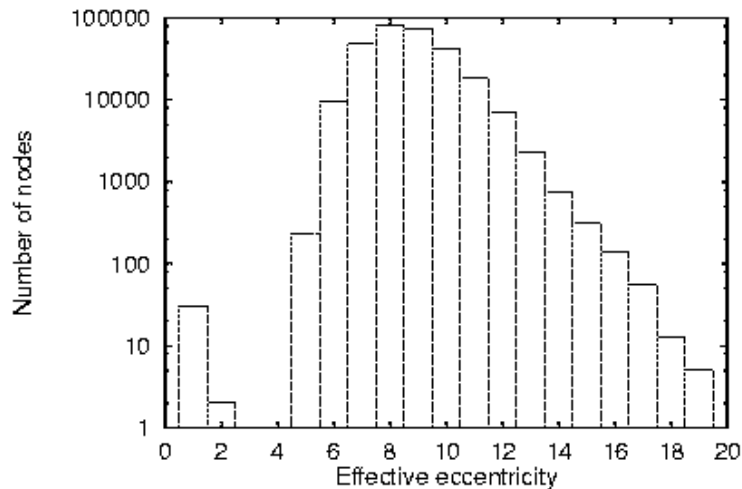
(Palmer, Gibbons, Faloutsos², Siganos 2001)

Internet graph: 285k nodes, 430k edges.

For each vertex v , define ball $B(v; R)$ of radius R .

Want: histograms of $|B(v, R)|$ $R = 1 \dots 20$

Get it in minutes of CPU rather than a day ($400\times$ speedup)

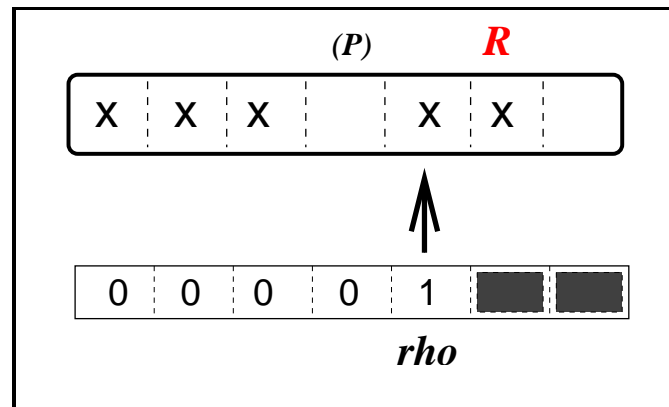


b) Histogram of diameters

LogLog Counting

Claim: IS the best on the market! (Durand-F, 2003/DFFGM, 2007)

- Hash and get $\rho(h(x)) :=$ position of rightmost 1-bit
- To set S associate $R(S) := \max_{v \in S} \rho(h(v))$.



This observable is mathematically worse, but much less costly:

$$\log_2 N \rightsquigarrow \log_2 \log_2 N;$$

that is, replace WORDS (32bits) by BYTES (8bits).

Algorithm LogLog Counting:

- Use rightmost **1**-bit as “observable”.
- Do **stochastic averaging** with $m = 2^\ell$. E.g., $S \cong \langle S_{00}, S_{01}, S_{10}, S_{11} \rangle$.
- Return: $\frac{m}{\hat{\varphi}} 2^{\text{Average}}$, where $\hat{\varphi} = e^{-\gamma} \sqrt{2}$.

+ Switch to “hit Counting” for small cardinalities.

++ Optimize by **harmonic means**:

~> HyperLogLog by (FFGM'07) (\ll Chassaing-Gerini'06)

Theorem. LogLog and HyperLogLog are *asymptotically unbiased*.

— They need m “bytes”, each of length $\log_2 \log N_{\max}$.

— Standard error (accuracy) is:

LogLog: $\frac{1.30}{\sqrt{m}}$, where $1.30 \doteq \sqrt{\frac{1}{12} \log^2 2 + \frac{1}{6} \pi^2}$; HyperLogLog: $\frac{1.04}{\sqrt{m}}$

— Distribution is approximately Gaussian.

— They can be effectively corrected in the non-asymptotic regime.

Proof: need $\text{coeff}[z^n] \left(e^z \sum_k 2^{k/m} \left[e^{-z/2^{k-1}} - e^{-z/2^k} \right] \right)^m$.

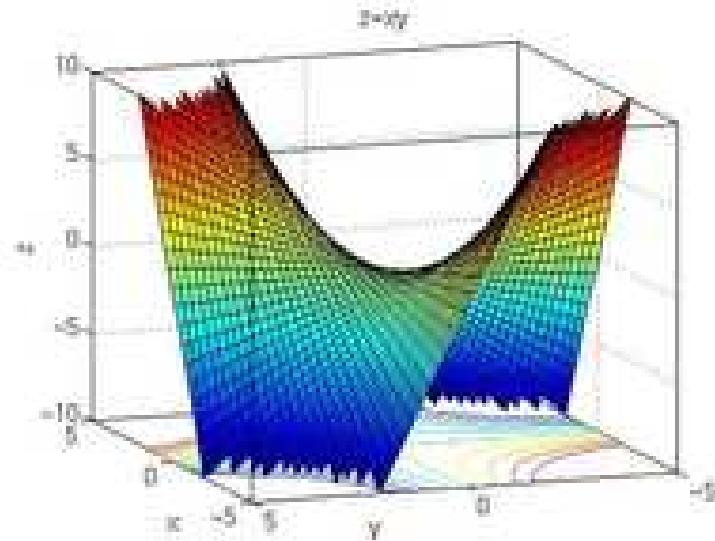
Analytic depoissonization [JaSz95⁺]

- Recover asymptotics of f_n from $\phi(z) := \sum_n f_n e^{-z} \frac{z^n}{n!} \equiv \text{Poisson GF?}$
- Intuition: with luck $f_n \sim \phi(n)$

Here: “Luck” means good lifting of $\phi(z)$ to $\mathbb{C} \equiv \text{Poisson flow of complex rate!}$

E.g.: \exists *cone*. Inside: $\phi(z) \sim z^\alpha$. Outside: $\phi(z)$ is exponentially smaller.

$$f_n = \frac{n!}{2i\pi} \oint e^z \phi(z) \frac{dz}{z^{n+1}} \approx \phi(n)$$



Whole of Shakespeare:

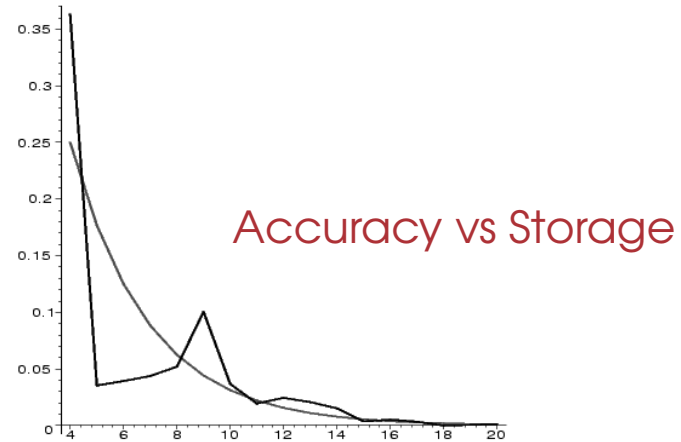


$m = 256$ small "bytes" of 4 bits each \equiv 128bytes

```
ghfffghfghggghggggghghheehfhfhhgghghghhfghfffhhiigfhhffgfiihfhhh  
igigighfghihfffghigihghigfhhgeegeghggghhgghhfhidiigihighihehhhfgg  
hfgighigffghdieghhhggghhfhghhfiieffghghihifgggffihgihfggighgiiif  
fjgfgjhhjiihfjhgehghfhhfhjhiggghghihigghihihgiighgfhlgjfgjjmfl
```

Estimate $n^\circ \approx 30,897$ against $n = 28,239$ distinct words
Error is +9.4% for 128 bytes(!!)

Features: Errors \approx *Gaussian*, seldom more than $3 \times$ standard error.

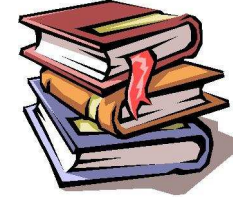


Mahābhārata: 8MB, 1M words, 177601 diff.

HTTP server: 400Mb log pages 1.8 M distinct req.

| m | 2^6 (50by) | 2^{10} (0.8kby) | 2^{14} (12kb) | 2^{18} (200kb) |
|------------|--------------|-------------------|-----------------|------------------|
| Obs: | 8.9% | 2.6% | 1.2% | 0.32% |
| σ : | 11% | 2.8% | 0.7% | 0.36% |

Document similarity



= An application of cardinality counts. For multisets A and B , define $\text{sim}(A, B) := \frac{|A \cap B|}{|A \cup B|}$ (Broder).

Here: $|A| := \text{card}(A)$. Let $\text{Reg}(A)$ be **signature** of A , i.e., (LogLog) **register dump** of A , so that $|A| = \text{estim}(\text{Reg}(A))$.

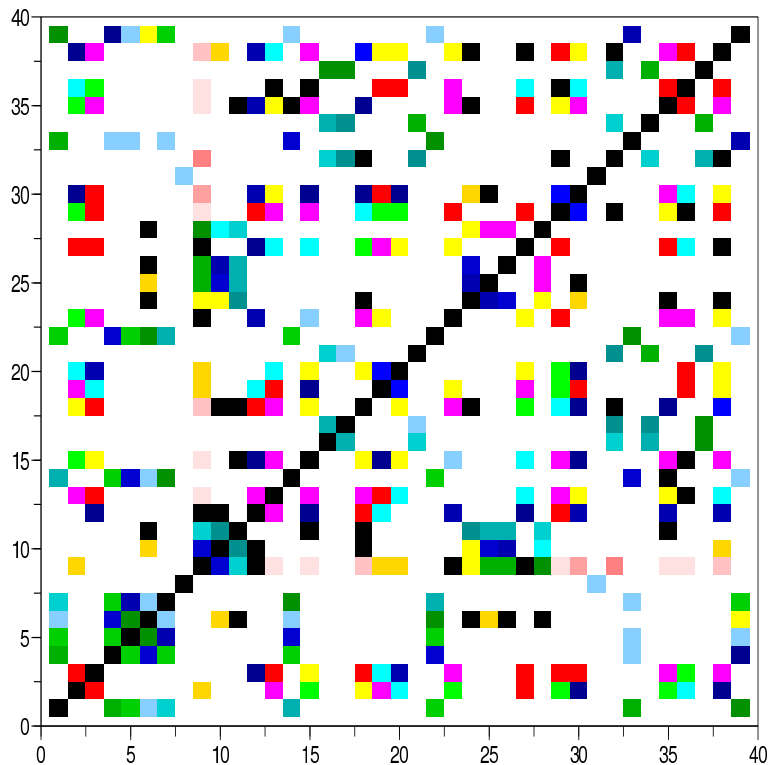
$$\begin{cases} |A| = \text{estim}(\text{Reg}(A)); & |B| = \text{estim}(\text{Reg}(B)) \\ |A \cup B| = \text{estim} \left(\text{Reg}(A) \oplus_{\max} \text{Reg}(B) \right); & |A \cap B| = |A \cup B| - |A| - |B|. \end{cases}$$

- For r files, **pairwise comparisons** have cost $O(\sum |F_j|) + O(r^2)$, as opposed to $O(\sum |F_j|)^2$:
 \implies For 10^5 files of size 10^5 , work in **minutes** instead of **days**!

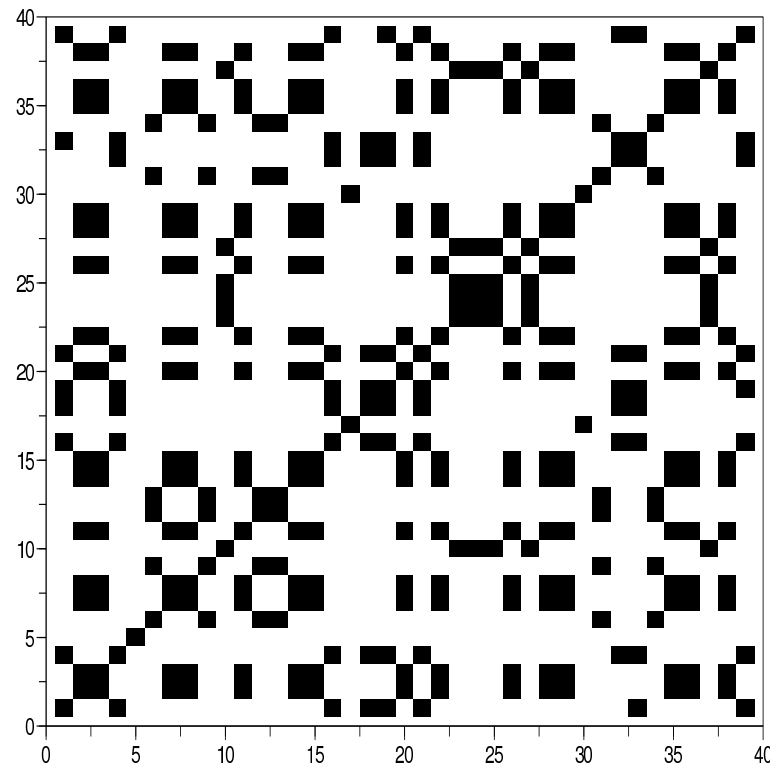
प्रणवः कश्यपः A blind test, by Pranav Kashyap

39 files of 20k words each, encrypted word-by-word.

- *How many languages? Which are which?*

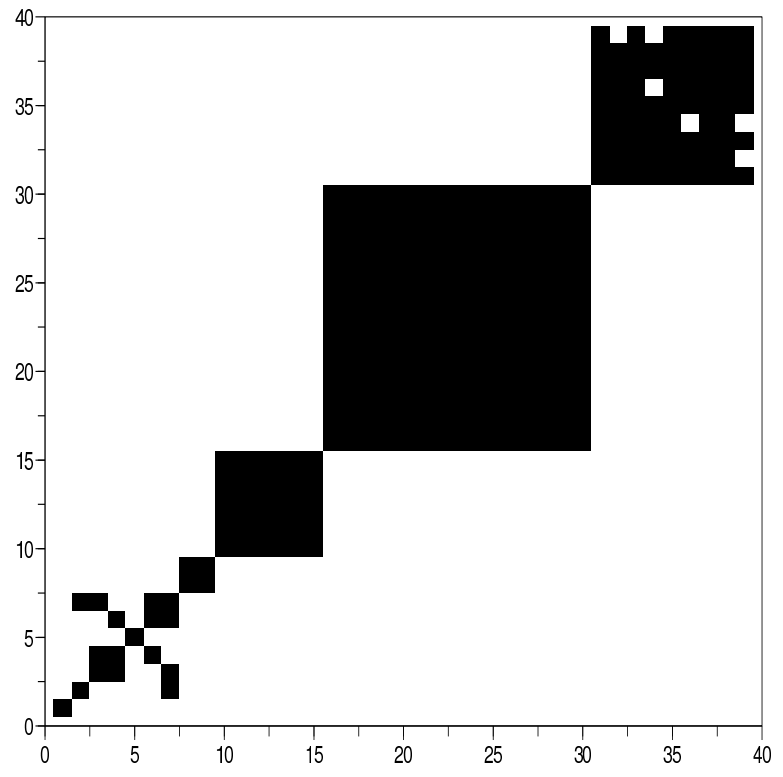


Raw comparison data

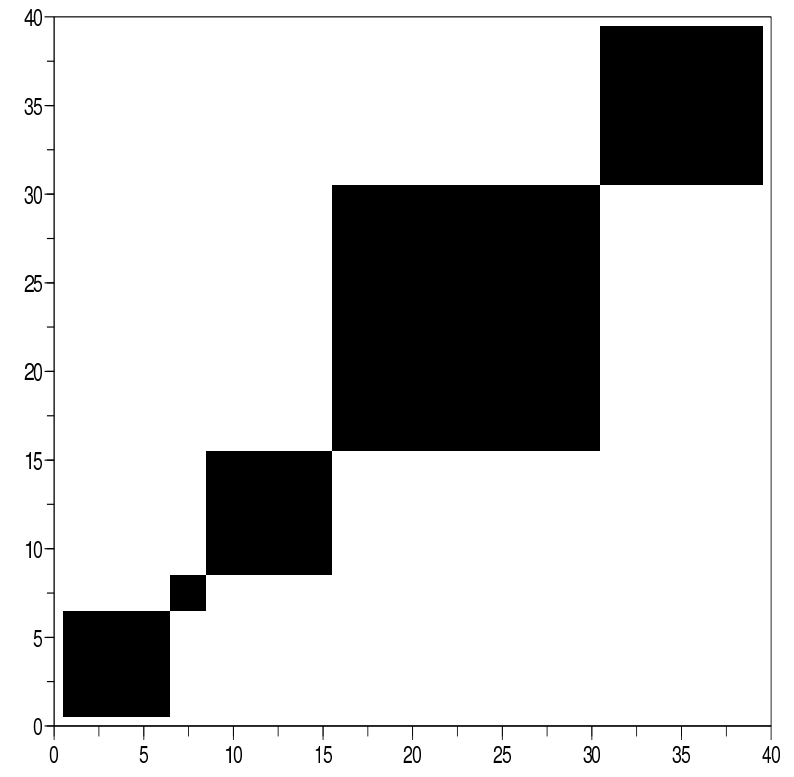


With thresholding ($\theta = 0.25$)

प्रणवः **कश्यपः**



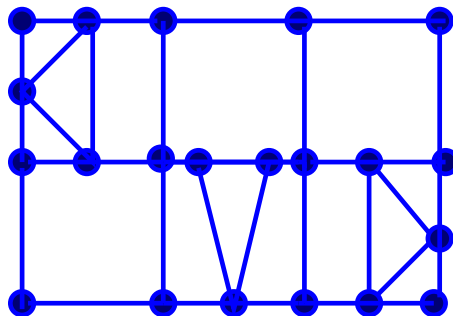
Blind classification ($\theta = 0.25$)



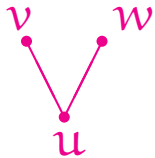
Actual ($\theta = 0.20$)

Counting triangles in graphs

Suggested by (Bar-Yossef, Kumar, Sivakumar, 2002) who propose to use F_2 (!)



Consider graph Γ of max-degree D given by adjacency list.

- Define a *vee* (V) as any triple $\{u, v, w\}$ such that 
- Make a stream of all vee's with cost $O(nD^2)$.
- Isolated vee's are 1-mice; triangles are 3-elephants.
- Use cardinality on top of p -sampling: e.g., $p = \frac{1}{2}$ gives

$$N = N_m + N_e, \quad N_0 = \frac{1}{2}N_m + \frac{7}{8}N_e.$$

Conclusions

For streams, using **practically $O(1)$ storage**, one can:

- **Sample** distinct values;
- **Estimate** F_0, F_1, F_2, F_p ($0 < p \leq 2$) even for *huge* data sets;
- **Estimate icebergs, # of mice and elephants.**

♡ Need **virtually no assumption on nature of data.**



The algorithms are based on **randomization** \mapsto **Analysis fully applies**

- They **work exactly** as predicted on real-life data;
- They often have **a wonderfully elegant structure**;
- Their analysis involves **beautiful methods** for AofA: “Symbolic modelling by generating functions, Singularity analysis, Saddle Point and analytic depoissonization, Mellin transforms, stable laws and Mittag-Leffler functions, etc.”

That's All, Folks!

