algorithmes probabilistes pour la fouille QUANTITATIVE de données MASSIVES

[et quelques applications]

Jérémie Lumbroso
GREYC, Université de Caen

séminaire du groupe “Combinatoire Énumerative et Algébrique”
équipe Combinatoire et Algorithmes, LaBRI

17 mai 2013

grands flux de données

Flux : une (très grande) séquence S sur un domaine $\mathcal{D}$ (aussi très grand)

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

S peut être vue comme un multi-ensemble

$$\mathcal{M} = {e_1}^{f_1}\ {e_2}^{f_2}\ \cdots\ {e_n}^{f_n}$$

Ex.: $\mathcal{S} =$ run sally run see sally run $\Rightarrow$ $\mathcal{M} =$ run^3 sally^2 see^1

On veut **estimer** les statistiques *quantitatives* :

— **A.** **Taille** $:= \ell$

— **B.** **Cardinalité** $:= \mathrm{card}\{s_j\} \equiv n$ (éléments distincts)

— **C.** **Moments de fréquence** $:= \left(\sum_{v\in\mathcal{D}} {f_v}^p\right)^{1/p}$ $p \in \mathbb{R}_{\geqslant}$

— **D.** **Souris** := élém. apparaissant 1, 2, 3, ... 10 fois

— **E.** **Icebergs** := # élém. avec fréquence **relative** $f_v/\ell > \theta$

[où θ est un seuil fixé, e.g. 50%]

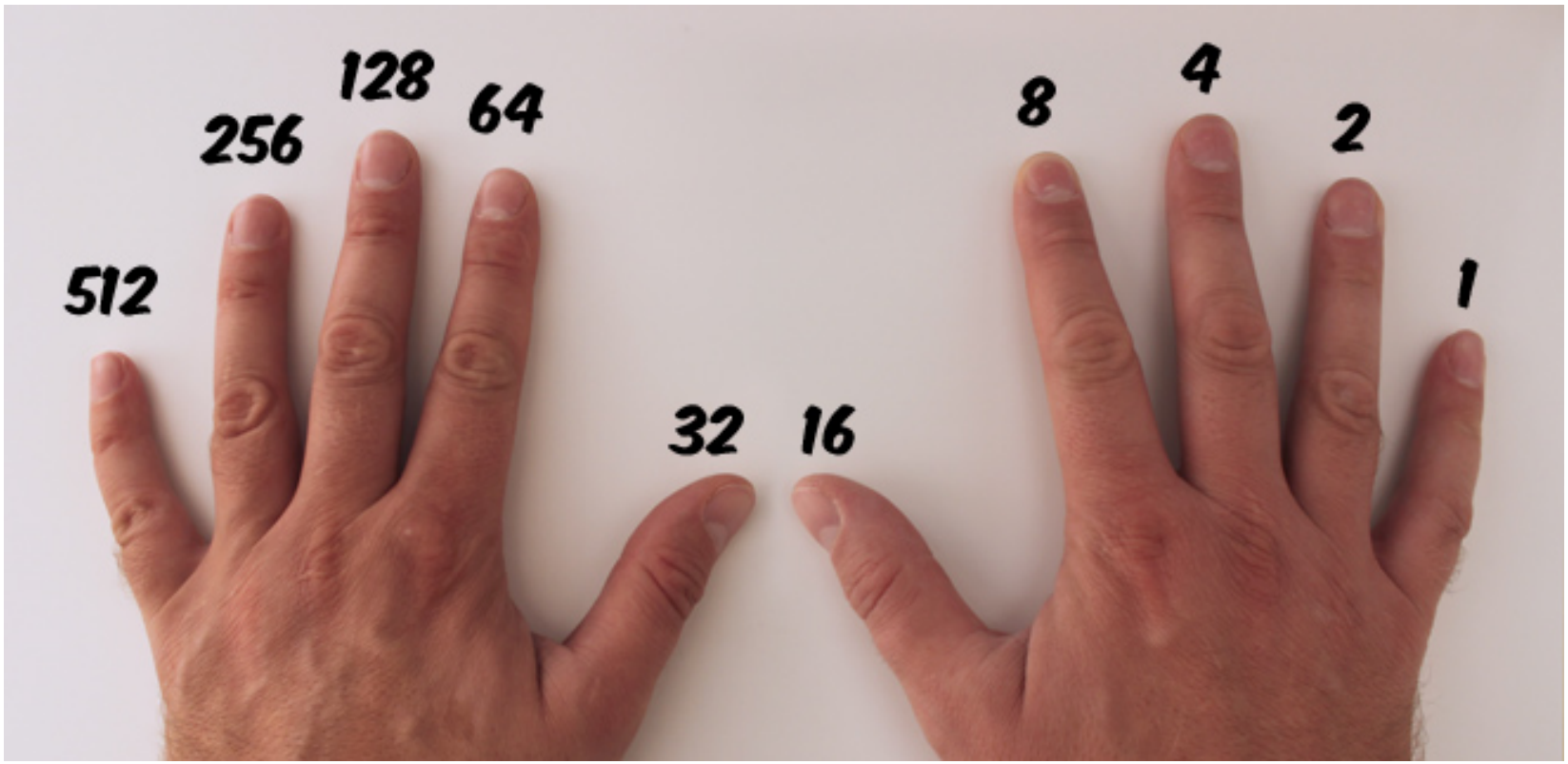
Contraintes :

- très peu de mémoire auxiliaire
- une seule passe (et un faible nombre d'instructions)
- aucune hypothèse statistique
- précision à quelques pourcents près

Prologue : il faut $\log_2 N$ bits pour compter jusqu'à N

Prologue : il faut $\log_2 N$ bits pour compter jusqu'à N

bit : plus petite unité d'information, soit 0 soit 1



avec **10** doigts/bits on peut compter jusqu'à
$2^{10} - 1 = 512 + 256 + \ldots + 1$

1. Approximate Counting (compter taille ℓ)

Donc avec 8 bits, on peut compter jusqu'à $2^8 - 1 = 255$ éléments.

Question : est-il it possible d'en compter **plus** ??
$\Rightarrow$ **OUI**, avec tirages à pile/face !

Première idée : incrémenter une fois sur deux
(= avec prob. 1/2)

- **Initialisation :** $C := 0$
- **Incrément :** avec probabilité 1/2, $C := C + 1$
- **Sortie :** $2 \cdot C$

$\mathbb{E}[2 \cdot C] = n$ et seulement 3% d'erreur

Limites : économie de **1 bit** (avec 8 bits, compter jusqu'à 2^{8+1} - 1 = 511)
[pas super intéressant !]

Deuxième idée : généraliser, et incrémenter 1 toutes les 2^k fois

[prob $1/2^k$ = tirer k pièces, toutes égales à 1]

- **Initialisation :** $C := 0$
- **Incrément :** avec probabilité $1/2^k$, $C := C + 1$
- **Sortie :** $2^k \cdot C$

mieux : économise ***k*** bits, i.e., compter jusqu'à 2^{8+k} avec 8 bits

Limites :

- n'économise toujours qu'un nombre linéaire de bits
- avec $k = 8$, erreur de 55%
- pire : **systématiquement** imprécis pour les petites valeurs $< 2^k$

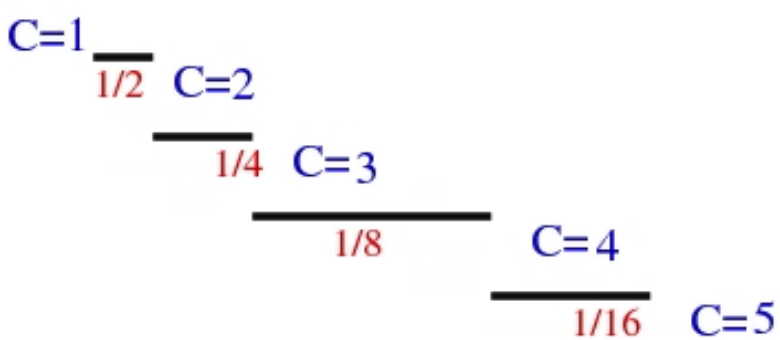
[car la plus petite valeur renvoyée est $2^k \cdot C$]

la BONNE idée

Troisième idée : faire dépendre probabilité d'incrément de valeur du compteur C

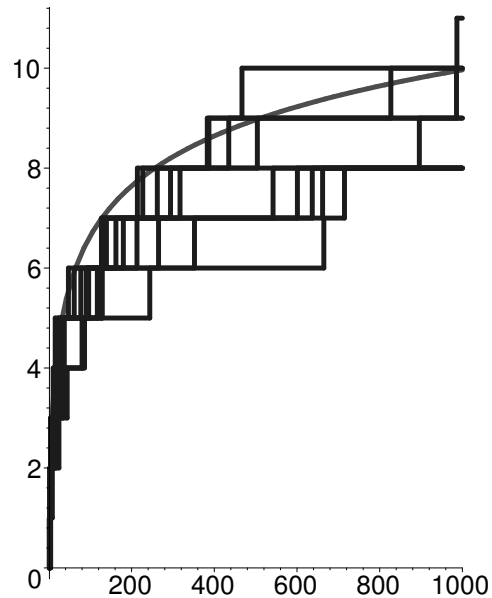
- **Initialisation :** $C := 0$
- **Incrément :** avec probabilité $1/2^C$, $C := C + 1$
- **Sortie :** $2^C - 2$

Cela devient de plus en plus "dur" d'incrémenter :



Enfin :

- précis pour les **petites** valeurs
- avec 8 bits, compter jusqu'à 2^{16} avec **15% d'err.**
- $\log\log \ell$ pour compter (approx.) jusqu'à ℓ



Morris 1978, Flajolet 1985

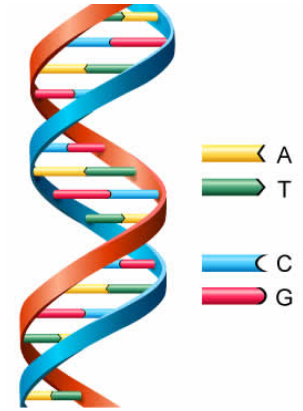
\+ même idée que recherche dichotomique non bornée

application : recherche de motifs dans les génômes

Génôme : longue séquences de lettres $\{A, C, G, T\}$

compter occurrences des sous-séquences de taille k

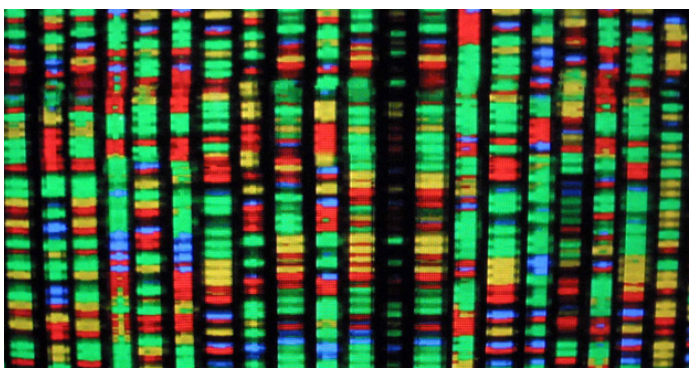
[intéressée par **mots absents** + **mots très fréquents**]



Exemple : A A C T A A C G T A A A pour $k = 2$

AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
4	3	-	-	-	-	1	1	-	-	-	1	2	-	-	-

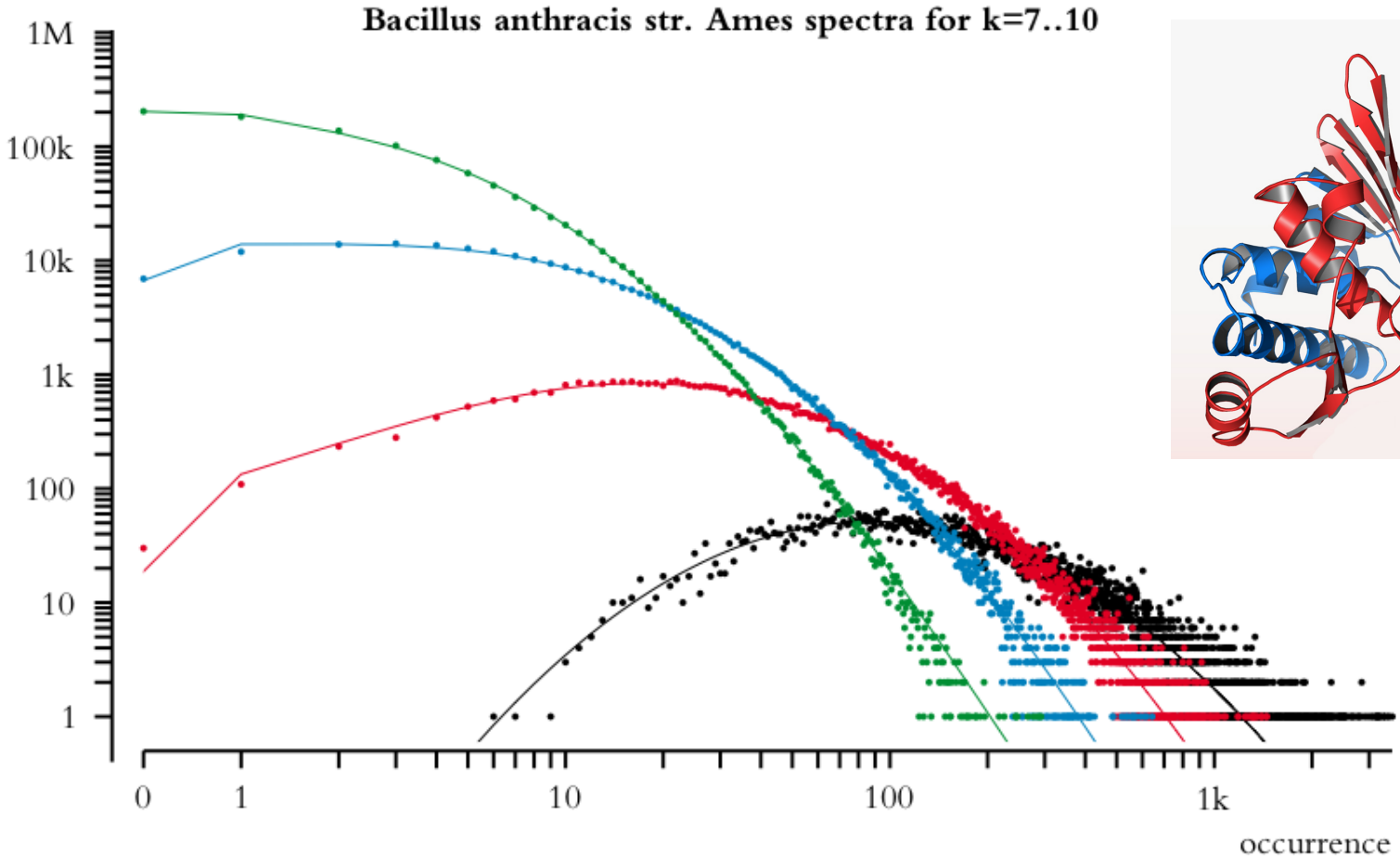
- AA apparait 4 fois
- ...
- GA est absent, appelé un *nullomer* (pareil pour AG, AT, etc.)
  [significatif : dans aléa uniforme, toutes sous-séquences apparaissent]

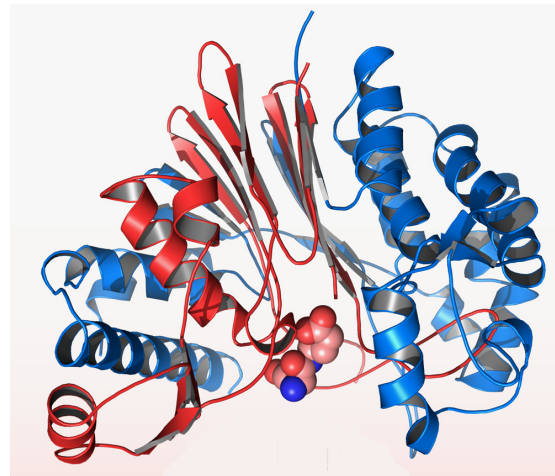


Limites d'un comptage exact :

- pour $k = 13$, avec compteurs de 4 Ko, nécessite **256 Go** de mémoire
- $k > 14$ nécessite Approx. Counting!

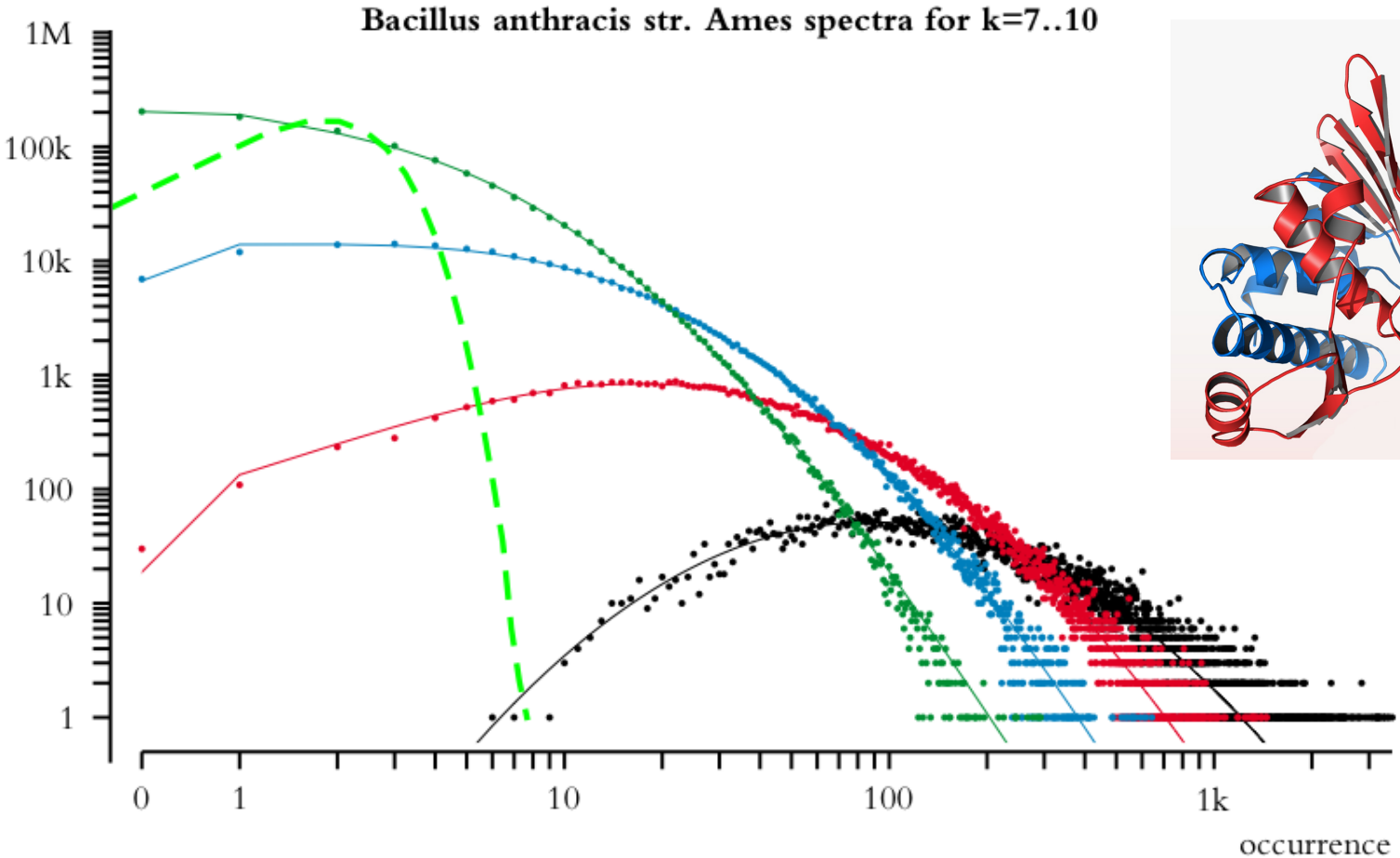
motifs dans le génôme du **bacille du charbon** (5.23 M)

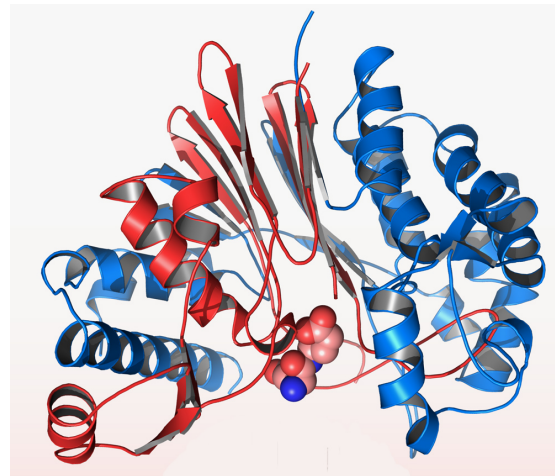




distribution des séquences de nucléotides de taille $k = 7, 8, 9, 10$ source :
Csűrös 2007, `http://www.iro.umontreal.ca/~csuros/spectrum/`

motifs dans le génôme du **bacille du charbon** (5.23 M)





distribution de 5.23 M séquences **aléatoires**

distribution des séquences de nucléotides de taille $k = 7, 8, 9, 10$ source :

Csűrös 2007, `http://www.iro.umontreal.ca/~csuros/spectrum/`

2. HACHAGE: un aléa reproductible

Hachage : des structures de données aux algos probabilistes

- **1950s:** fonctions de hachage = composants des tables de hachage
- **1969:** avec filtres de Bloom, utilisées pour 1ère fois dans contexte approximatif
- **1977/79:** Carter & Wegman, *Hachage Universel*, considérées pour 1ère fois comme objets probabilistes + preuve que l'uniformité est possible en pratique

fonctions de hachage **transforment les données** en séquences infinies de bits aléatoires indép. ou en var. aléatoires i.i.d. **uniformes** dans **[0, 1]** :

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

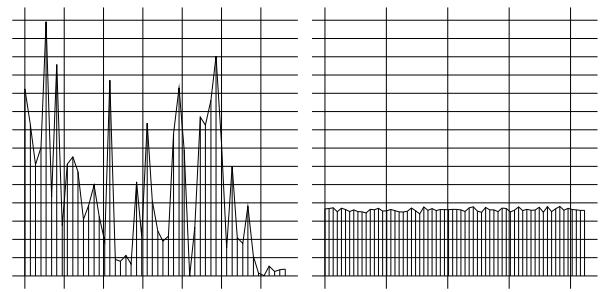
e.g., si $h(x) = b_1 b_2 \cdots$ alors
$\mathbb{P}[b_1 = 1] = \mathbb{P}[b_2 = 1] = \ldots = 1/2$

$$g : \mathcal{D} \to [0,1]$$

$\mathbb{P}[g(x) \in [y, y + \mathrm{d}y]] = \mathrm{d}y$
si $y \in [0,1]$, et 0 sinon

- modèles **équivalents** : passe de l'un à l'autre par un développement dyadique

$$g(x) = \sum_{k=1}^{\infty} \frac{b_i}{2^k}$$



non-hachées hachées

- comportement empirique, validé théoriquement en **2010**: Mitzenmacher & Vadhan montre que les fonctions de hachage exploitent l'entropie des données

3. estimation nb. éléments DISTINCTS

3. estimation nb. éléments DISTINCTS

- **échantillonnage** : intrinsèquement limité pour raisons statistiques, mais + de 100 réf., notamment en *biologie* [Bunge & Fitzpatrick 1993]
- **collectionneur de coupons** [Whang *et al.* 1990, Estan *et al.* 2003...]
- **statistiques d'ordre**
  - Probabilistic Counting [FlMa85]: $\min\left(\mathbb{N} \setminus \{G_1, \ldots, G_n\}\right)$
  - variables uniformes [Bar-Yossef *et al.* 2002, Giroire 2005...]: $X_{(k)}, k > 2$
  - LogLog, etc. [FlDu03, FlFuGaMe07]: $\max\{G_1, \ldots, G_n\}$
  - estimation par le minimum [L. 2010]: $\min\{X_1, \ldots, X_2\}$

où

$$G_i \in \mathrm{Geo}(1/2) \qquad X_i \in \mathcal{U}(0,1) \qquad X_{(k)} \in \mathrm{Beta}(k, n-k+1)$$

Démarche générique (dans l'ex. $\ell = 8, n = 6$)

- un flux — Allons mon pauvre coeur allons mon vieux complice
- hachage (v.a. en Post-it®) — Allons mon pauvre coeur allons mon vieux complice
  $\underset{0.26}{\text{Allons}}\ \underset{0.58}{\text{mon}}\ \underset{0.22}{\text{pauvre}}\ \underset{0.68}{\text{coeur}}\ \underset{0.26}{\text{allons}}\ \underset{0.58}{\text{mon}}\ \underset{0.52}{\text{vieux}}\ \underset{0.14}{\text{complice}}$
- fonction insensible aux répétions — $\min = \mathbf{0.26}, 0.26, \mathbf{0.22}, 0.22, 0.22, 0.22, 0.22, \mathbf{0.14}$
- ... dont l'espérance dépend de n

$(\mathbb{E}_n[F] = f(n) \quad \Rightarrow \quad \mathbb{E}_n\left[f^{-1}(F)\right] \approx n)$ $\qquad \mathbb{E}_n[\min] = 1/(n+1), \qquad 1/0.14 - 1 \approx 6$

---

ces techniques utilisent **l'ultime valeur calculée** = dépend uniquement de la composition (+ fonction hachage)

première statistique d'ordre (le minimum)

M := minimum de n variables aléatoire uniformes i.i.d. (= loi beta)

$$\mathbb{P}_n[M \in [x, x + \mathrm{d}x]] = n(1-x)^{n-1}\mathrm{d}x \tag{1}$$

ainsi

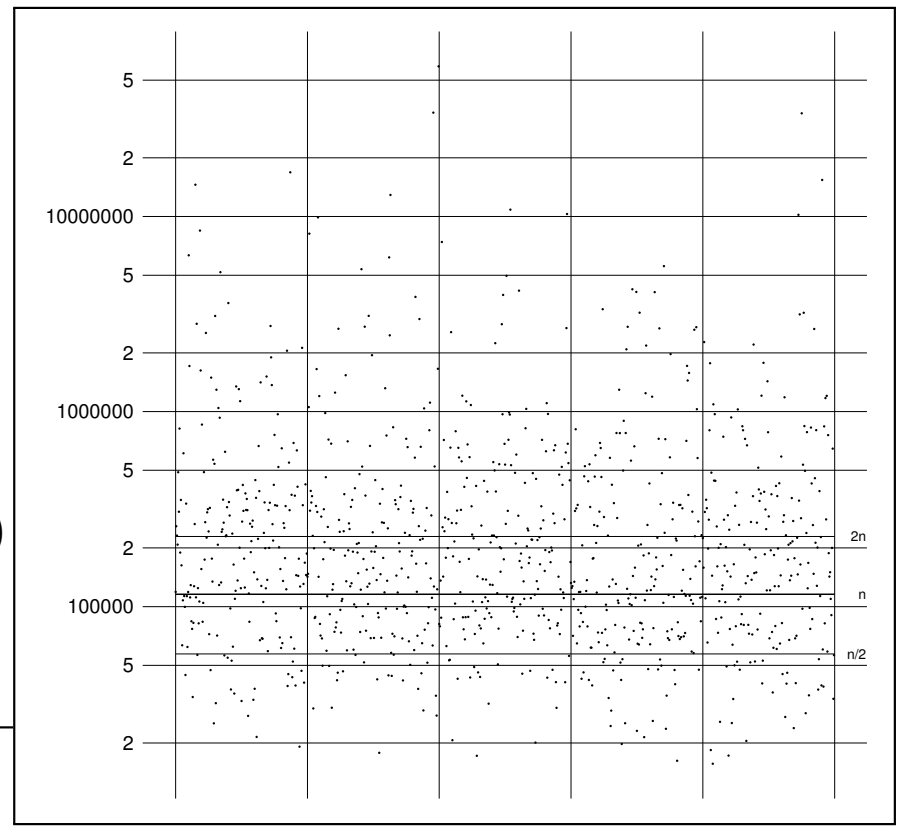
$$\mathbb{E}_n[M] = \int_0^1 x \cdot n(1-x)^{n-1}\mathrm{d}x = \boxed{\frac{1}{n+1}}. \tag{2}$$

Avantages :

- calculable en une passe
- calculable avec un unique registre
- $\mathbb{E}_n[M] = \frac{1}{n+1}$.

Désavantages :

- le minimum oscille beaucoup, en effet $\sigma_n[M] = \frac{1}{n+1}$ (meme ordre)
- fonction $x \mapsto \frac{1}{x}$ diverge en 0.



$$\mathbb{P}_n\left[M \leqslant \tfrac{t}{n}\right] \sim 1 - \exp(-t)$$

moyenne stochastique (Flajolet & Martin 85)

Pour améliorer précision d'algorithme par $1/\sqrt{m}$, utiliser m fonctions de hachage différentes (et un registre/minimum diff. pour chaque fonct.) et **prendre la moyenne**.

$\Rightarrow$ très couteux (hacher m fois plus) !

Aiguiller elements aléatoirement en m sous-flux avec les premiers bits de la valeur hachée

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

qui sont ensuite tronqués (seul $b_4 b_5 b_6 \cdots$ = valeur utilisée).

Par exemple pour $m = 4$,

$$h(x) = \begin{cases} 00 b_3 b_4 \cdots & \rightarrow & M_{00} = \min\{M_{00}, 0.b_3 b_4 \cdots\} \\ 01 b_3 b_4 \cdots & \rightarrow & M_{01} = \min\{M_{01}, 0.b_3 b_4 \cdots\} \\ 10 b_3 b_4 \cdots & \rightarrow & M_{10} = \min\{M_{10}, 0.b_3 b_4 \cdots\} \\ 11 b_3 b_4 \cdots & \rightarrow & M_{11} = \min\{M_{11}, 0.b_3 b_4 \cdots\} \end{cases}$$

Algorithme principal (L. 2010)

Paramètre : m paramètre de contrôle (= précision requise)
Entrée : un flux $S = (s_1, \ldots, s_\ell)$

---

initialiser m registres, M_1 jusqu'à M_m, à 1

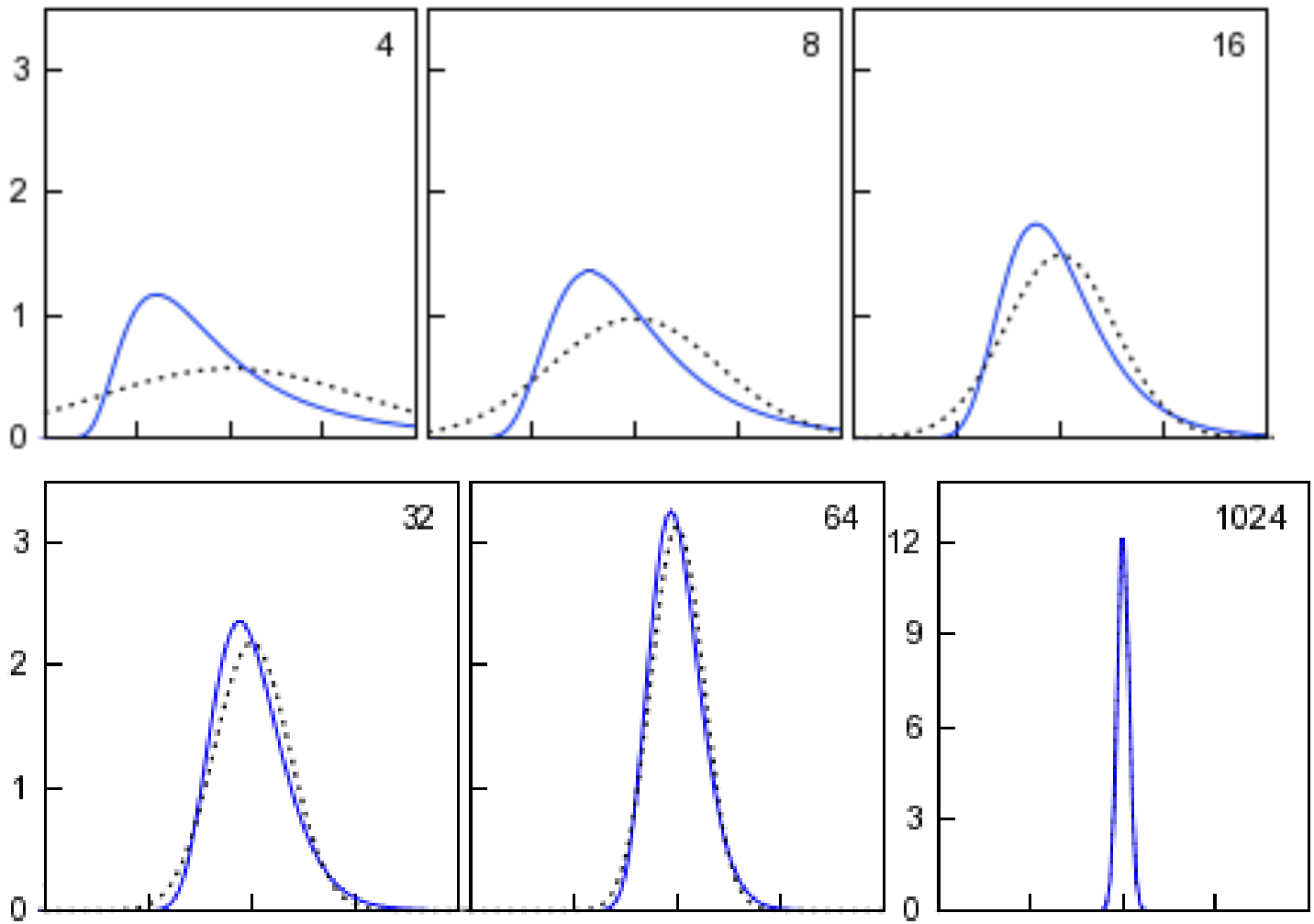
pour tout $x \in S$ **faire**

$A := h(x)$	{hacher x, avec $h(x) \in (0,1)$}
$j := \lfloor mA \rfloor + 1$	{calcul de l'index du sous-flux attribué à x}
$M_j := \min\left(M_j, mA - \lfloor mA \rfloor\right)$	{mise à jour du minimum du je sous-flux}

renvoyer $\mathcal{Z}^* = m \cdot \dfrac{(m-1)}{M_1 + \ldots + M_m}$

pour la suite : je note $\mathcal{Z}^*$ mon estimateur

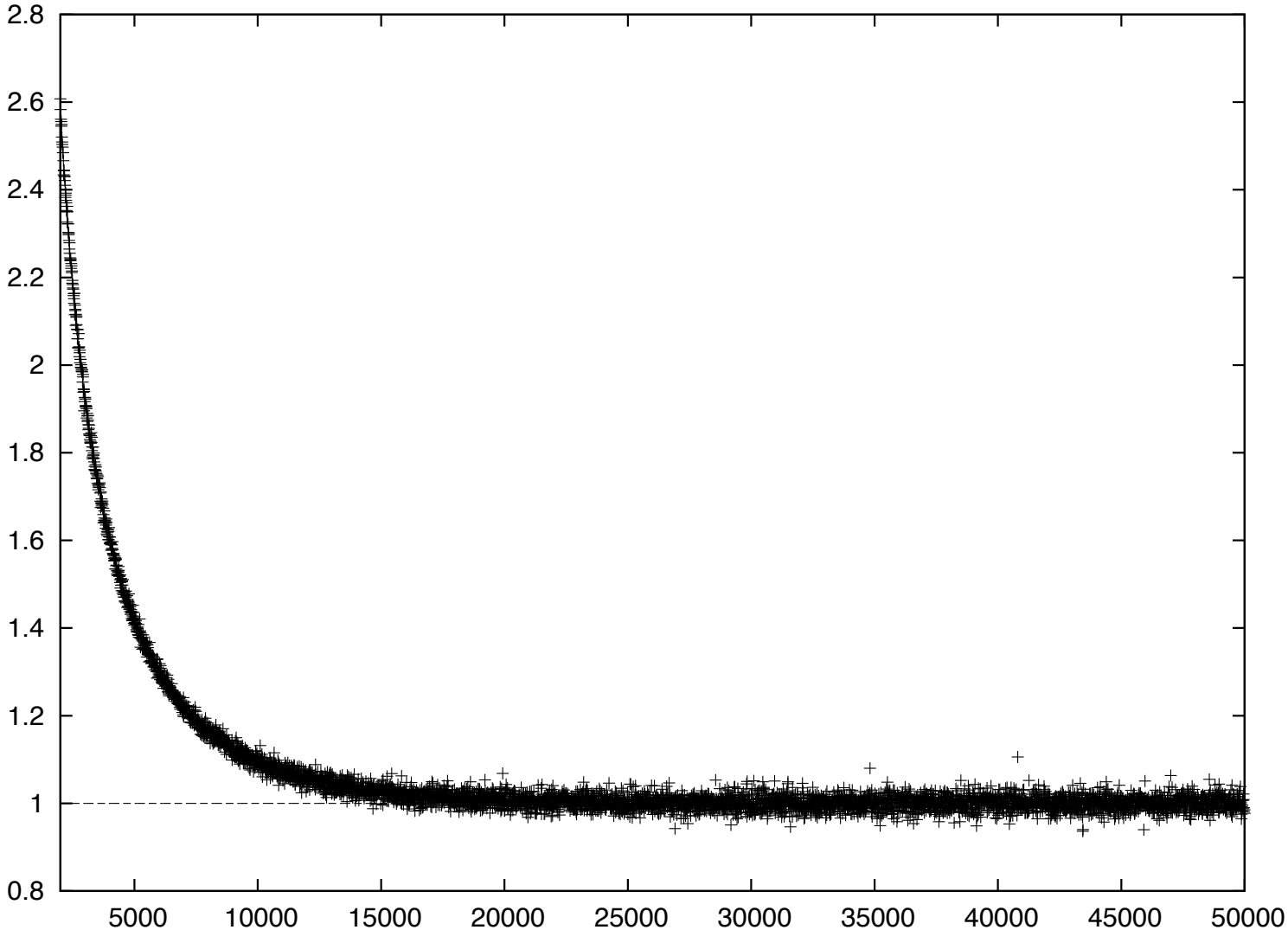
DISTRIBUTION LIMITE vs. une GAUSSIENNE, $m = 4..1024$



l'estimateur est distribué selon une loi gamma (converge en une gaussienne car loi gamma = somme de loi exponentielles + CLT)

corrections **non-asymptotiques**

Graphe de $\mathcal{Z}^\star/n$ en fonction de cardinalité $n = 2000..50000$ $(m = 4096)$



A) modèle POISSON (**CERTAINS** registres vides)

Calculs pré-asymptotiques: n relativement petit comparé à m

registres vides **biaisent** moyenne $\longrightarrow$ **détecter + ignorer registres vides**.

approx. de Poisson de l'allocation des registres (i.e.: $N_j \in \mathrm{Poi}(\lambda)$).

$$\mathbb{P}[M_j \in [x, x+\mathrm{d}x]] = \lambda \mathrm{e}^{-\lambda x}\mathrm{d}x + \mathrm{e}^{-\lambda}\mathbf{1}_{\{x=1\}} \tag{3}$$

soit $k = \#\{\text{registres non-vide}\}$:

$$\mathbb{E}\left[\frac{1}{M_1 + \ldots + M_m}\right] = \sum_{k=0}^{m} \int_0^\infty \binom{n}{k} \left(\lambda \cdot \frac{1-\mathrm{e}^{-a-\lambda}}{a+\lambda}\right)^k \left(\mathrm{e}^{-a-\lambda}\right)^{m-k} \mathrm{d}a$$

après calculs + méthode de Laplace, on trouve

Théorème: soit $n/m = \lambda$ tel que $0 < \lambda < C < \infty$, alors

$$\mathbb{E}_n[\mathcal{Z}^\star] \sim \frac{n}{1-\mathrm{e}^{-\lambda}}. \tag{4}$$

B) "Comptage Linéaire[1] (**TROP** de registres vides)

Non-asymptotique : changement de point-de-vue

n boules lancées dans m urnes.

Résultat classique : soit $W_k := \#\{\text{urnes avec } k \text{ boules}\}$

$$\mathbb{E}[W_k] \sim m \cdot \left[\frac{\lambda^k}{k!} \exp(-\lambda) \right]$$

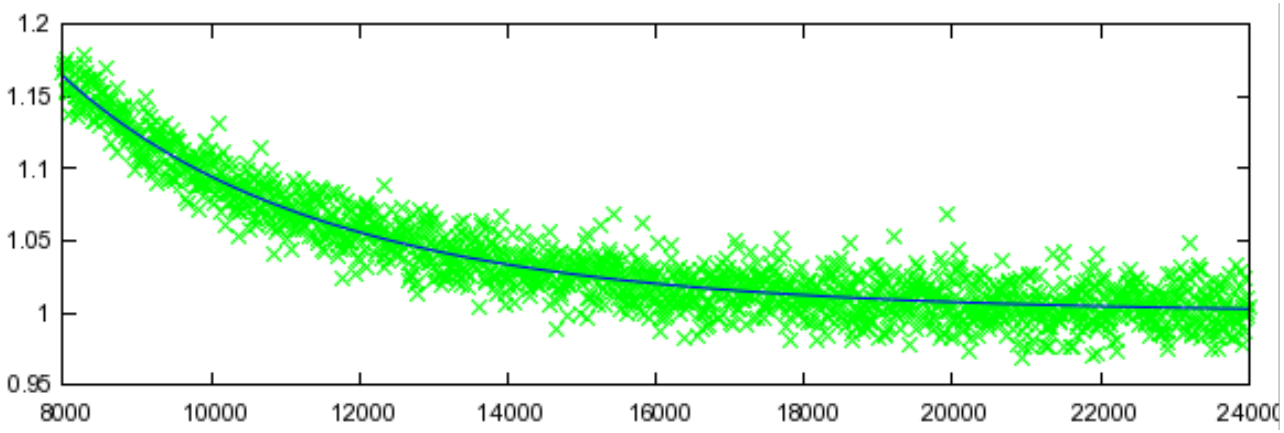
où $\lambda := n/m$ est constant, avec $n \to \infty$ et $m \to \infty$.

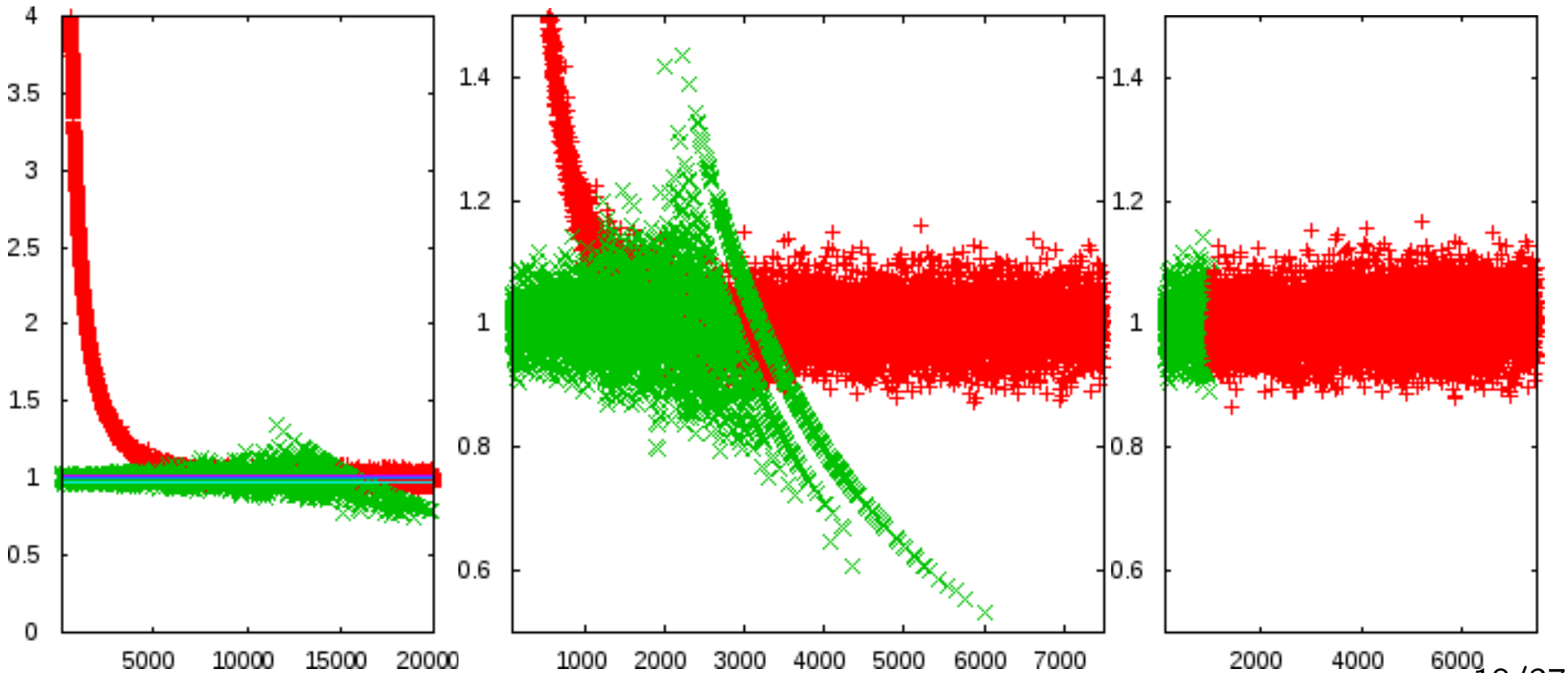
comme $\mathbb{E}[W_0] \sim m \cdot \exp\left(-\frac{n}{m}\right)$ alors, avec $\widehat{w}_0$ registres vides observés,

$$n \approx -m \log\left(\frac{\widehat{w}_0}{m}\right) \tag{5}$$

[1] d'après Whang et al, "*A Linear-Time Probabilistic Counting Algorithm for Database Applications*," ACM Trans. on Database Systems, Vol. 15, No. 2, pp. 208-229, June 1990.

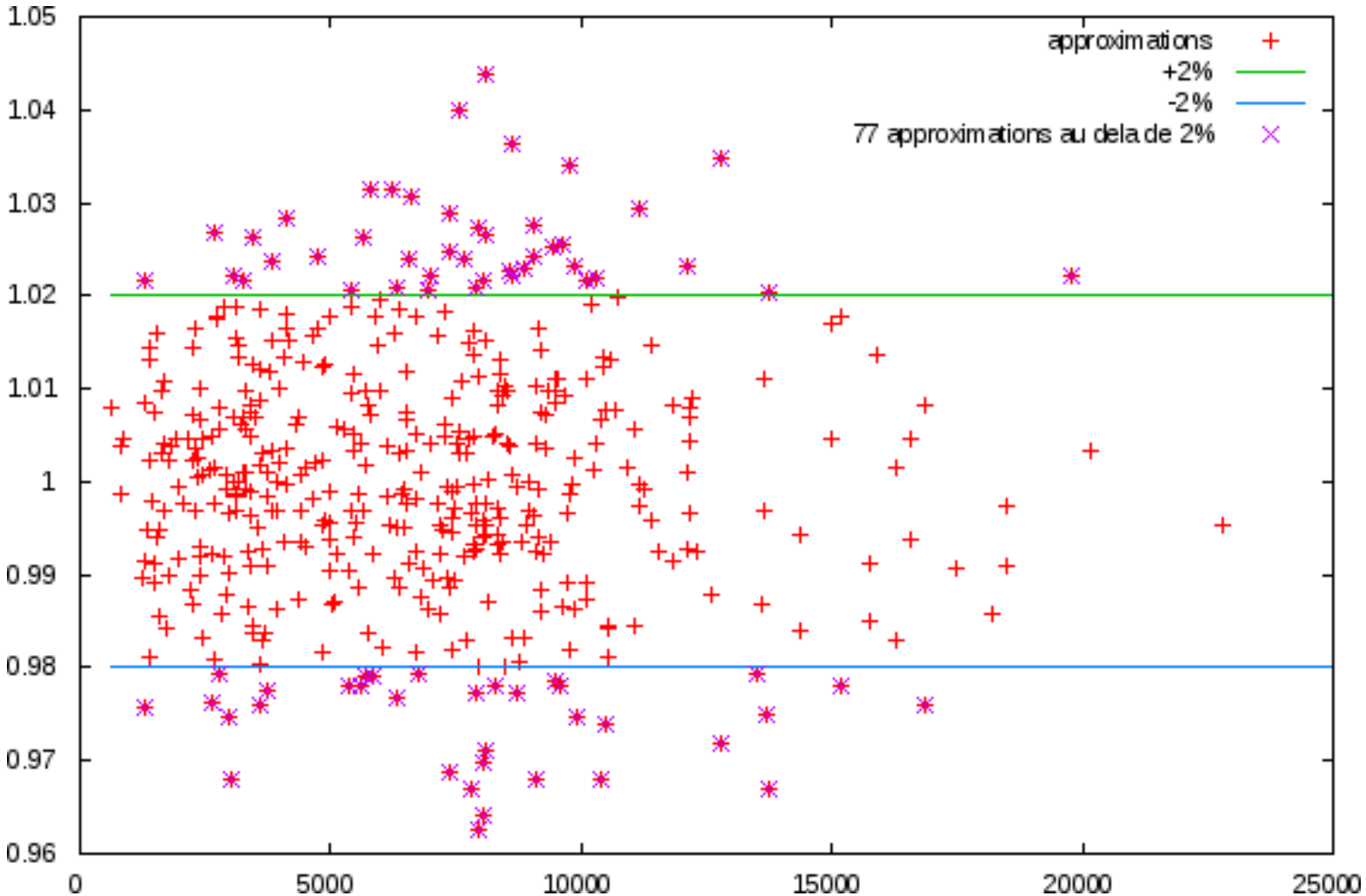
C) algorithme final = COMPOSITE





estimation de 500 textes en **moins de 4 secondes**

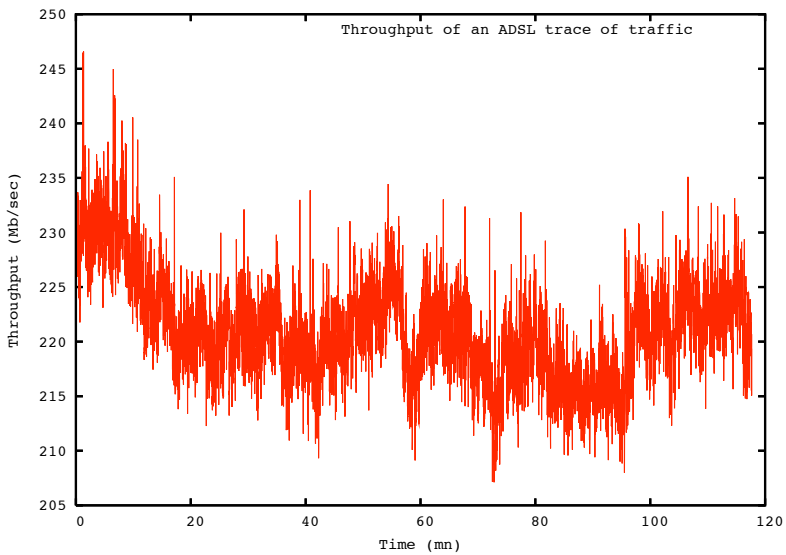
(contre **45** secondes ...)

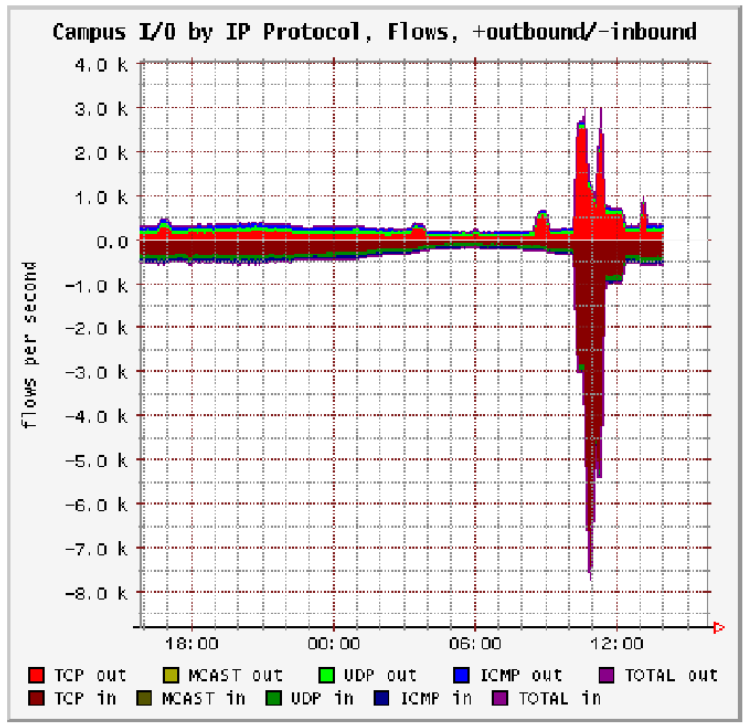


$\ell \approx 3 \cdot 10^7$ (taille) $n = 3 \cdot 10^6$ (cardinalité) $m = 4096$

application : détection attaque réseau DDoS

DDoS : déni de service distribué
⇒ un réseau “zombie” (énorme nombre de machines infectées contrôlées à distance) se connecte simultanément au serveur, le surchargeant (connexion = processus couteux)





le serveur n’a aucun contrôle sur le flux réseau entrant, mais des noeuds en amonts du réseau peuvent mitiger la situation

ADRESSE SOURCE	PORT SRC	ADRESSE DEST	PORT DEST	DONNEES
233.123.8.56	90	101.245.127.4	763	01100101010010101010

chaque connexion est un quadruplet nouveau (adresse source, port source, adresse dest, port dest) ⟶ compter quadruplets distincts

union et indice de similarité

bitmap : le tableau des m registres qui contiennent les minima calculées lors de l'algorithme

par construction, calculer la cardinalité de l'union de deux flux A et B, étant donnés leur bitmap respectif, se fait en temps $O(m)$

$$\forall i \in \{1, ..., m\}, \quad \text{bitmap}_{A\cup B}[i] := \min\{\text{bitmap}_A[i], \text{bitmap}_B[i]\}$$

Pour calculer cardinalité intersection, utiliser

$$|X \cap Y| = |X \cup Y| - |X| - |Y|$$

indice de similarité de deux fichiers,

$$S(A, B) := \frac{\text{card}(A \cap B)}{\text{card}(A \cup B)}$$

établir matrice de r fichiers F_j, coûte $O(\sum F_j + (rm)^2)$ au lieu de $O((\sum F_j)^2)$
$\Rightarrow$ pour 10^5 fichiers de taille 10^5, cela passe de **minutes** au lieu de **jours**

application : AltaVista, supprimer les quasi-doublons

Broder (97) utilise indice de similarité

$$S(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

$S(A,B) = 0$: ensembles disjoints
$S(A,B) = 1$: ensembles conjoints

si $S(A,B) > 0.99$, considérer documents A et B comme étant les mêmes

$\Rightarrow$ **eliminer quasi-doublons !**

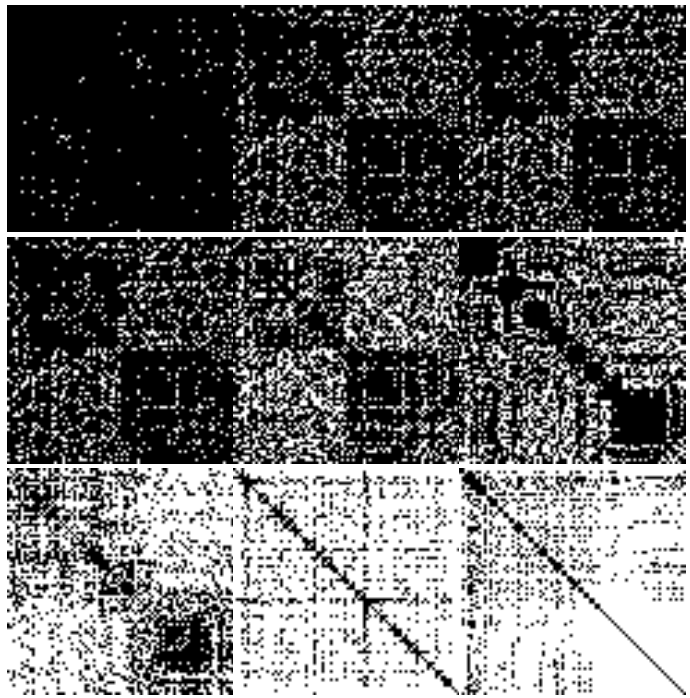
In order to show you the most relevant results, we have omitted some entries very similar to the 500 already displayed.
If you like, you can repeat the search with the omitted results included.

‹ Goooooooooogle

Previous 1 2 3 4 5 6 7 8 9 10

application : classification de corpus

matrices de comparaison de textes



50 textes, moitié en anglais, moitié en français → lesquels ??

- régler seuil de regroupement (en dessous duquel un corpus est divisé)
- régler seuil de visibilité (seuil au dessus duquel un point devient noir)

4. souris et PROFILS de FRÉQUENCES

4. souris et PROFILS de FRÉQUENCES

Soit un flux de taille ℓ (avec n éléments distincts)

$$\mathcal{S} = x_1\ x_2\ x_3\ \cdots\ x_\ell$$

- **échantillon standard** de taille m (chaque x_i pris avec prob. $\approx m/\ell$)

  `a x x x x b b x c d d d b h x x ...`

  permet inférer 'a' répété $\approx \ell/m$ fois dans S, mais impossible de parler des éléments rares, noyés dans la masse = **problème** de l'aiguille dans une botte de foin
- **échantillon distinct** (avec compteurs)

  `(a, 9) (x, 134) (b, 25) (c, 12) (d, 30) (g, 1) (h, 11)`

  prendre chaque élément avec probabilité $1/n$ = **indépendemment** de sa **fréquence** d'apparition

Autre exemple : échantillon de 1 élém. du flux $(1, 1, 1, 1, 2, 1, 1, \ldots, 1)$, $\ell = 1000$; avec échantillonnage standard, prob. $999/1000$ de prendre 1 et $1/1000$ de prendre 2 ; avec échantillonnage distinct, prob. $1/2$ de prendre 1 et prob. $1/2$ de prendre 2.

"Adaptive Sampling" : concept et contexte

- avec fonction de hachage transformer données en i.i.d. uniformes
- à chaque fois que nécessaire réduire (uniformément) l'univers des éléments retenus jusqu'à celui-ci soit plus petit que la taille max.
- **désavantage :** taille de l'échantillon est approximative, et oscille

Bibliographie

1. Idée originale : Wegman, 1983 (non publié).
2. Utilisé comme algorithme d'estimation de cardinalité :
   - Flajolet, 1984, 1990, *On Adaptive Sampling*
   - Astrahan et al., 1987, *Approximating the number of unique values of an attribute without sorting*
   - Louchard, 2000, *Probabilistic analysis of adaptive sampling*
3. "Redécouverte" des idées principales : Gibbons, 2001, *Distinct sampling for highly-accurate answers to distinct values queries and event reports*
4. Comme algorithme pour le profilage d'un flux :
   - Flajolet, 1997, *Adaptive Sampling* (in Encyclopaedia of Mathematics)
   - [Helmi, L., Martinez, Viola, 2012, *Data Streams as Random Permutations: the Distinct Element Problem*]
   - Louchard, L., Swan, 2013

Paramètre : m capacité du cache (taille visée)
Entrée : un flux $\mathcal{S} = (s_1, \ldots, s_\ell)$

---

initialize $C := \emptyset$ (cache) and $p := 0$ (depth)

forall $x \in \mathcal{S}$ **do**

    if $h(x) = 0^p \cdots$ **then**
        if $x \in C$ **then** increase the frequency count of x in C
        else $C := C \cup \{(x, 1)\}$

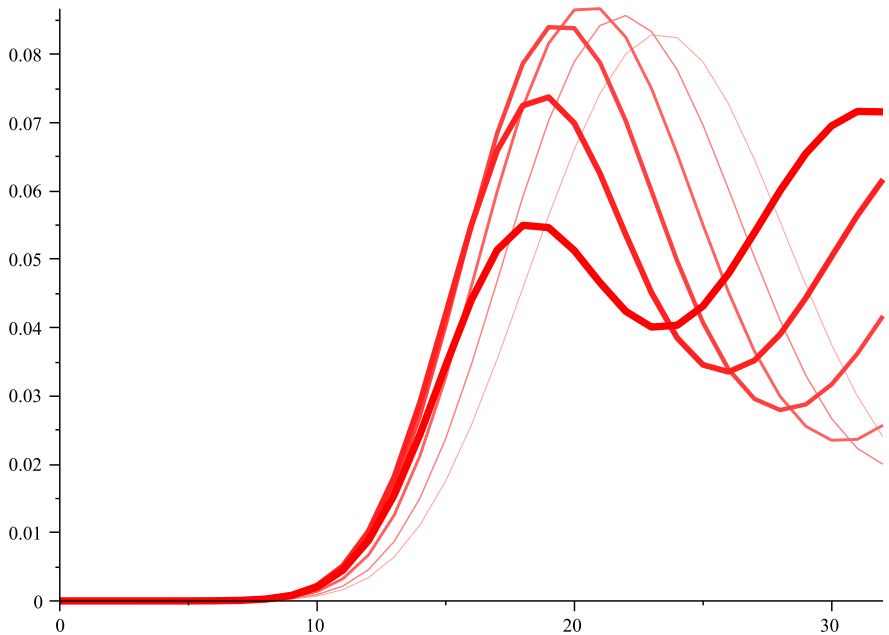
    if $|C| > m$ **then** {overflow of cache}
        $p := p + 1$
        filter(C) {remove items that don't match $0^p \cdots$}

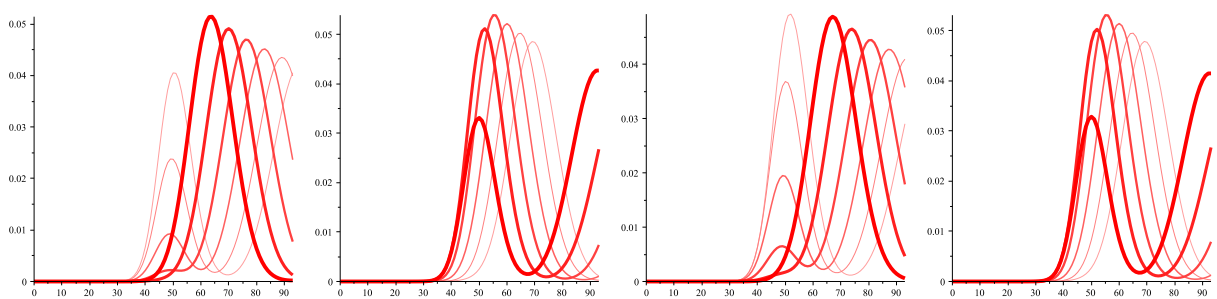
return (C, p)

distribution de la taille du cache

(taille = nombre d'éléments contenus dans le cache)

$$\mathbb{P}_n[X = k] = \delta_{k0} + \sum_{j=0}^{\infty} \left[\binom{n}{k} \frac{(1 - 1/2^j)^{n-k}}{2^{jk}} - \sum_{i=k}^{m} \binom{i}{k} \binom{n}{i} \frac{(1 - 1/2^j)^{n-i}}{2^{(j+1)i}} \right].$$





cet algorithme estime la cardinalité

Soit $p =$ profondeur d'échantillonnage et $|C| =$ taille du cache.

Théorème [Flajolet 90]:

$$X := |C| \cdot 2^p \tag{6}$$

estime la cardinalité de $\mathcal{S}$

- c'est un estimateur non-biaisé: $\mathbb{E}_n[X] = n$
- l'erreur standard est

$$\sim \frac{1}{\sqrt{(m-1)\log 2}} \doteq \frac{1.20}{\sqrt{m}}$$

Ainsi avec $m = 1000$ on a une précision 4 %.

exemple d'exécution

ELEM	**HASH**	**CACHE après insertion**	**prof.**	Est.#	Exact #
UDINE	10101	$(10101, 1)$	0	1	1
NICE	00101	$(10101, 1), (00101, 1)$	0	2	2
PARIS	11011	$(10101, 1), (00101, 1), (11011, 1)$	0	3	3
BORDE	01001	$(00101, 1), (01001, 1)$	1	4	4
ORSAY	11101	$(00101, 1), (01001, 1)$	1	4	5
PARIS	11011	$(00101, 1), (01001, 1)$	1	4	5
BORDE	01001	$(00101, 1), (01001, 2)$	1	4	5
MARSE	01010	$(00101, 1), (01001, 2), (01010, 1)$	1	6	6
RENNE	10100	$(00101, 1), (01001, 2), (01010, 1)$	1	6	7
LEIPZ	00010	$(00101, 1), (00010, 1)$	2	8	8
CAEN	10001	$(00101, 1), (00010, 1)$	2	8	9
QUEBE	00111	$(00101, 1), (00010, 1), (00111, 1)$	2	12	10
MARSE	01010	$(00101, 1), (00010, 1), (00111, 1)$	2	12	10
CAEN	10001	$(00101, 1), (00010, 1), (00111, 1)$	2	12	10
NICE	00101	$(00101, 2), (00010, 1), (00111, 1)$	2	12	10
WIEN	00100	$(00010, 1)$	3	8	11

... estimer les éléments k-fréquents

Let:

- p = sampling depth;
- $|C|$ = cache size;
- $\#(C, k)$ be the number of items of cache C that have frequency k.

Idea:

- estimate number of k-frequent elements in $\mathcal{S}$ by $\#(C, k) \cdot 2^p$
- estimate percentage of k-frequent elements in $\mathcal{S}$

$$\frac{\#(C, k) \cdot 2^p}{|C| \cdot 2^p} = \frac{\#(C, k)}{|C|}$$

l'oeuvre complète de Jane Austen

Exécution sur l'oeuvre de Austen, avec $m = 20$, peut[2] donner le cache suivant:

(fix, 37) (become, 73) (agree, 51) (abominate, 3) (clever, 69) (saunter, 2) (creations, 2) (alas, 41) (papered, 1) (housekeepers, 4) (grazier, 1) (berth, 1) (loins, 1) (bragging, 1) (fullgrown, 2) (soured, 1) (hardily, 1) (monasteries, 1)

La profondeur finale atteinte est $p = 10$.

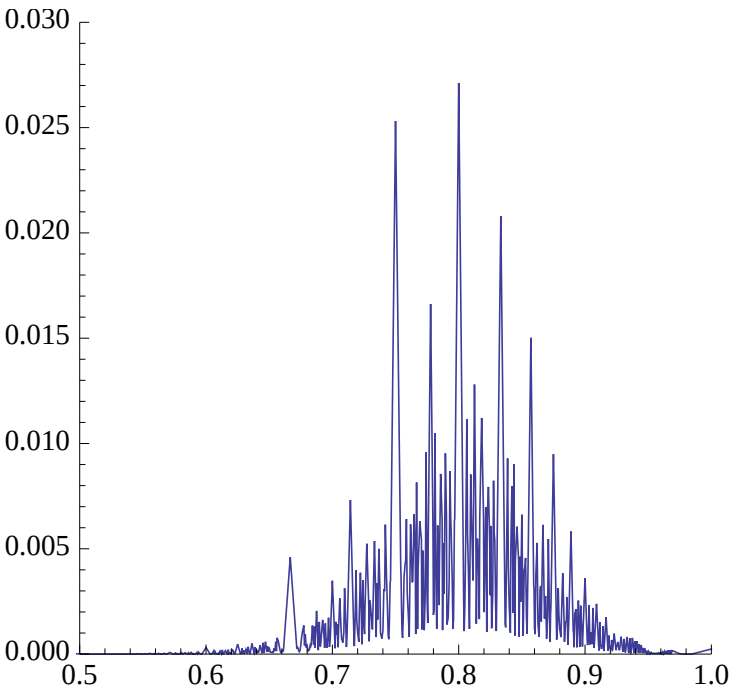
La cardinalité estimée est $|C| \cdot 2^p = 18 \cdot 2^{10} = 18432$.
Le nombre estimé d'éléments de fréquence 1: $8 \cdot 2^{10} = 8192$.
La proportion estimée d'éléments de fréquence 1: $8/|C| = 44.4$ %.

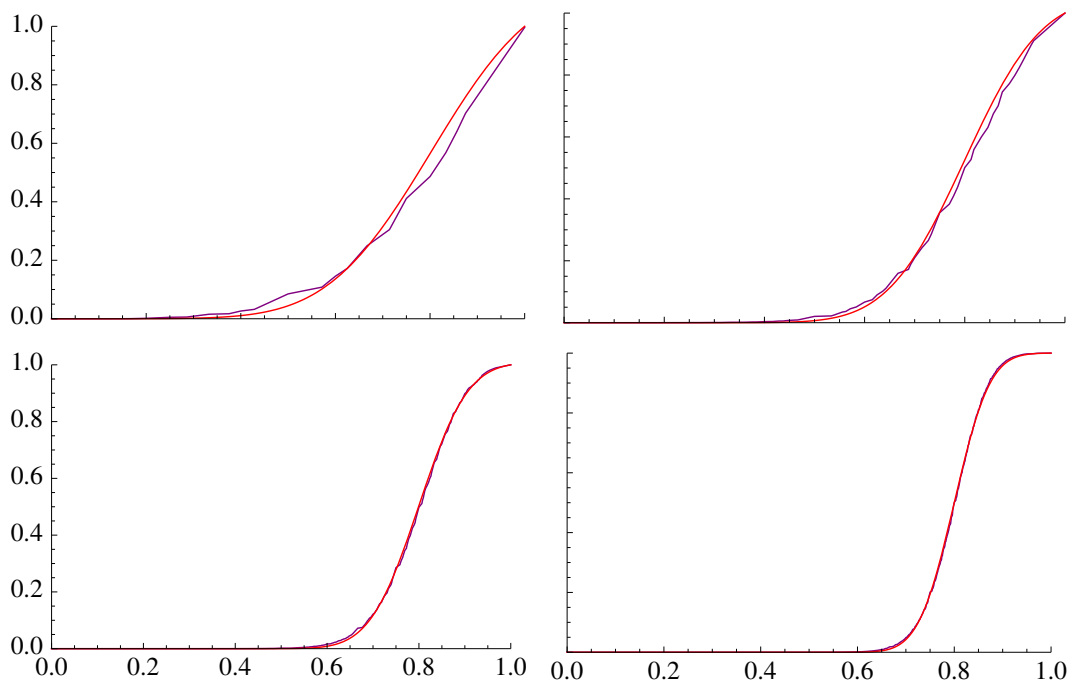
% d'éléments avec fréquence	1	2	3	4	Cardinalité
Estimation	44.4 %	16.7 %	5.6 %	5.6 %	18432
Exact	47.0 %	11.3 %	4.2 %	3.0 %	19967

[2] une véritable – mais particulièrement bonne – exécution!

analyse en distribution

[Louchard, L., Swan, 2013]





applications et extension

- en réseaux, détecter le régime du réseau, et les paquets à faire passer en priorité (éléphants = gros transferts ; souris = connexions)

- dans le marketing (“analytics”), répondre à la volée en permanence à des questions telles : combien d'utilisateurs ont vu 1, 2, 3, publicités...

- dans une base de données répondre rapidement à une requête complexe sur un nombre distinct d'éléments, de manière approchée, à partir d'un cache pré-calculé

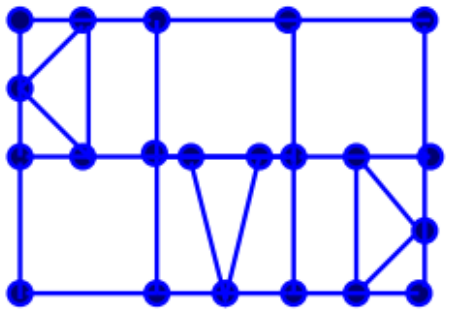
```
select count(distinct elevation)
from terrain.data
where elevation 2500 and type-foret = sapin
```

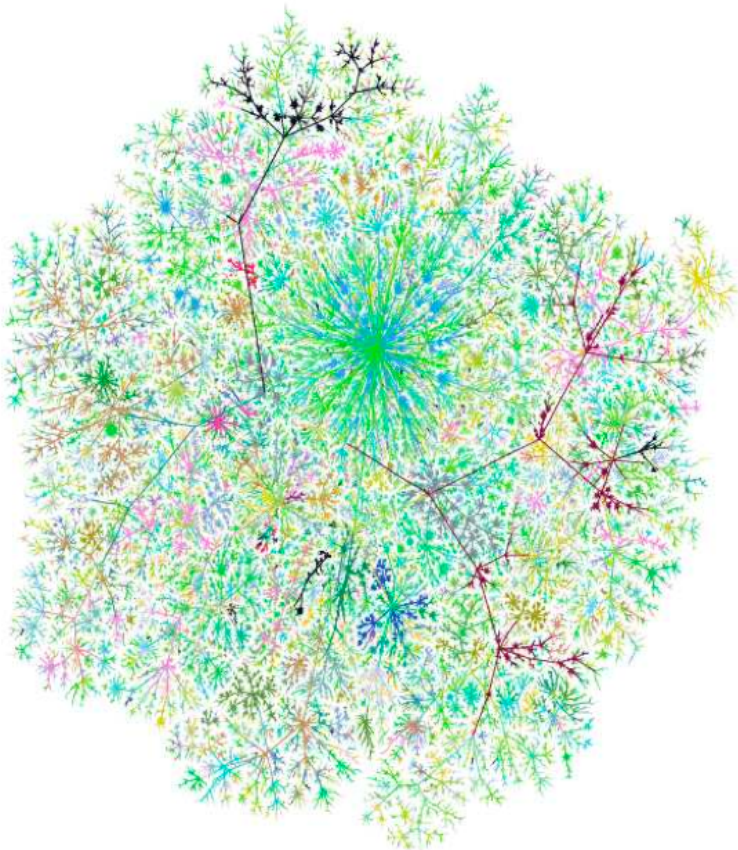
ou

```
select count(distinct pays-de-naissance)
from recensement.data
where emploi = ‘fonctionnaire-region’ or
      emploi = ‘fonctionnaire-departement’ or
      emploi = ‘fonctionnaire-commune’
```

noter que l'algorithme est (comme pour l'estimateur de cardinalité) **parfaitement parallélisable** (union de deux échantillons facile à calculer — condition : avoir même fonction hachage)

application : comptage de triangles





compter triangles dans grands graphes permet :

a) mesures quantifiant la densité de triangles : *clustering coefficient* et *transitivity ratio* [Newman 03, ...]

b) détection de spam [Becchetti *et al.* 08, ...]

c) autres applications en analyse

[Alon *et al.* 97, Bar Yossef *et al.* 02, Tsourakakis 08]

soit un graphe $G = (V, E)$

- pour tout nœud u, pour tout $(v, w) \in V(G, u)$, ajouter le triplet $\{u, v, w\}$ au flux
- **traffic bimodal :** 1 fois $\rightarrow$ V ; 3 fois $\rightarrow \nabla$

avec les algos de cet exposé, deux manières

1. échantillonnage distinct (proportion cache)
2. cardinalité :

$$\ell = n_V + 3n_\nabla \qquad \text{[taille]}$$
$$n = n_V + n_\nabla \qquad \text{[cardinalité]}$$

d'où $(N - n)/2 \approx n_\nabla$.

5. ICEBERGS

***k*-iceberg** : éléments avec fréquence relative $> \ell/k$

(ex.: 2-iceberg = élément majoritaire)

guerre des gangs (pour $k = 2$)

- double registres (valeur, compteur)
- pour tout x
  - si valeur $= x$ alors incrémenter compteur
  - sinon si compteur > 1 alors décrémenter compteur
  - sinon { valeur $\leftarrow x$; compteur $\leftarrow 1$

+ généralisation pour k avec $k - 1$ registres

[Karp, Schenker, Papadimitriou 03]

6. MOMENTS de fréquence

Rappel : soit le grand flux vu comme un multi-ensemble sur domaine $\mathcal{D}$

$$\mathcal{M} = {e_1}^{f_1}\ {e_2}^{f_2}\ \cdots\ {e_n}^{f_n} \qquad e_j \in \mathcal{D}$$

alors **moments de fréquence** :

$$F_p := \left(\sum_{v \in \mathcal{D}} {f_v}^p \right)^{1/r} \qquad p \in \mathbb{R}_{\geqslant}$$

cas particuliers :

- cardinalité = F_0 ;
- taille = F_1 ;
- index de répétition (Simpson, Gini, etc.) = F_2
  donne "mesure" de distribution,
  ex.: (si normalisé) $F_2 = 1$ éléments tous mêmes, $F_2 \to 0$ tous différents

un loi ***p*-stable** si X_1, X_2, et X distribués selon cette loi vérifient

$$c_1 X_1 + c_2 X_2 \stackrel{d}{=} \qquad \text{avec} \qquad \mu := (c_1^p + c_2^p)^{1/p}$$

$p = 1$: Cauchy ; $p = 2$ loi normale

Algorithme (dû à Indyk, 2000)

- $Z := 0$
- pour tout $x \in S$, faire $Z := Z + \text{Stable}_p(x)$
- renvoyer Z

où on associe à chaque élément x un tirage unique d'une loi p-stable (par construction hachage)