Fully analysing the adaptive-sampling algorithm applied to frequency counts in streams

Jérémie Lumbroso
(joint work with Philippe Flajolet)

Projet ALGORITHMS, INRIA Rocquencourt &
Laboratoire d'Informatique de Paris 6 (UPMC).

AofA'11
June 13th, 2011—Będlewo

1. PROBLEM STATEMENT

Definition: a stream $\mathcal{S}$,

$$\mathcal{S} = s_1 s_2 \cdots s_N.$$

size of the stream: $|\mathcal{S}| = N$
cardinality (= number of distinct elements): $\|\mathcal{S}\| = n$

For instance,

$$\mathcal{S} = \textit{run}, \textit{sally}, \textit{run}, \textit{see}, \textit{sally}, \textit{run} \qquad |\mathcal{S}| = 6 \qquad \|\mathcal{S}\| = 3.$$

Problem: How many elements appear in $\mathcal{S}$ with frequency $k = 1, 2, \ldots$

Constraints

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

MOTIVATION

- **Network security**: detect attacks (distributed denial of service)
- **Customer satisfaction** on websites (e.g., e-commerce)
- **Statistical analysis**
- **Graph analysis**: counting triangles, etc.
- + other applications related to sampling

Algorithmic tool: sampling

To find the number of k-frequency elements, i.e., for $k = 1$: mice,

- **straight sample** of size m of stream of size N:

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

  could infer[1] 'a' repeated m/N in S, but can't say anything about small frequency elements (needle in haystack) $\longrightarrow$ not useful

- sampling of **distinct values** (with their counts)

  `(a, 1) (x, 134) (b, 1) (c, 4) (d, 3) (g, 1) (h, 1)`

  more useful $\longrightarrow$ Adaptive Sampling

[1] minus hidden element, see Good-Turing, etc.

Bibliographic context of Adaptive Sampling

1. Original idea: Wegman, 1983 (unpublished).
2. Used as a cardinality estimation algorithm:
   - 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. "Rediscovery" of the core idea: Gibbons, 2001, *Distinct sampling for highly-accurate answers to distinct values queries and event reports*

2. ADAPTIVE SAMPLING

- use good hash functions

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

  to **transform data into i.i.d. uniform** random variables

- (uniformly) shrink the universe of items until it fits the sample size
- tradeoff: resulting sample size is approximate, and oscillates

Parameter: m cache capacity (targeted size)
Input: a stream $\mathcal{S} = (s_1, \ldots, s_N)$

---

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)

some known properties

Let p = sampling depth and $|C|$ = cache size.

Theorem [Flajolet 90]:

$$X := |C| \cdot 2^p \tag{1}$$

estimates cardinality of $\mathcal{S}$

- it is unbiased: $\mathbb{E}_n[X] = n$
- standard error is

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

Thus with $m = 1000$ we get 4 % accuracy.

Sample Execution

ITEM	**HASH**	**CACHE after insertion**	**depth**	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
POZNA	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

3. ESTIMATING k-frequent elements

Let:

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

Idea:

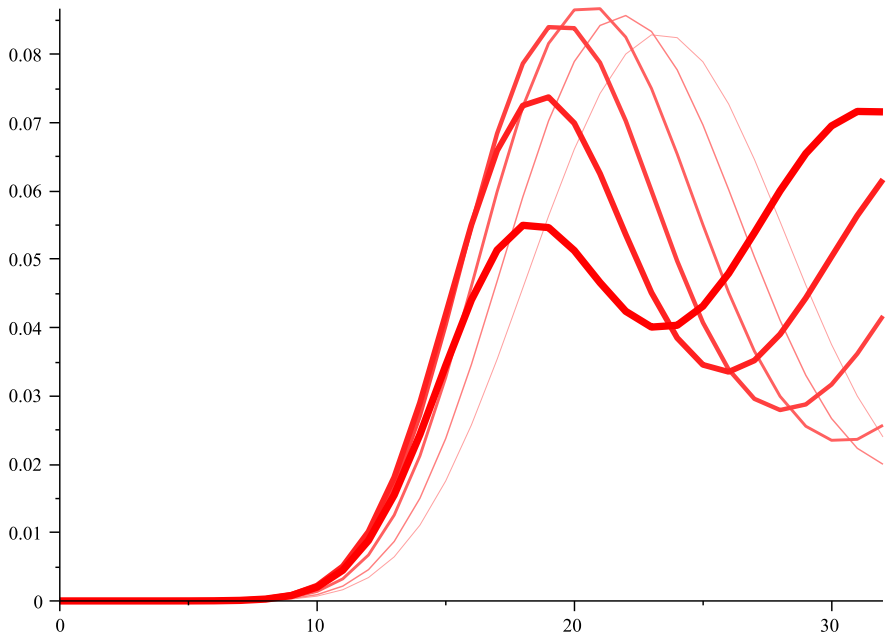
- estimate number of k-frequent elements in $\mathcal{S}$ by $\#(C, k) \cdot 2^p$
- estimate percentage of k-frequent elements in $\mathcal{S}$

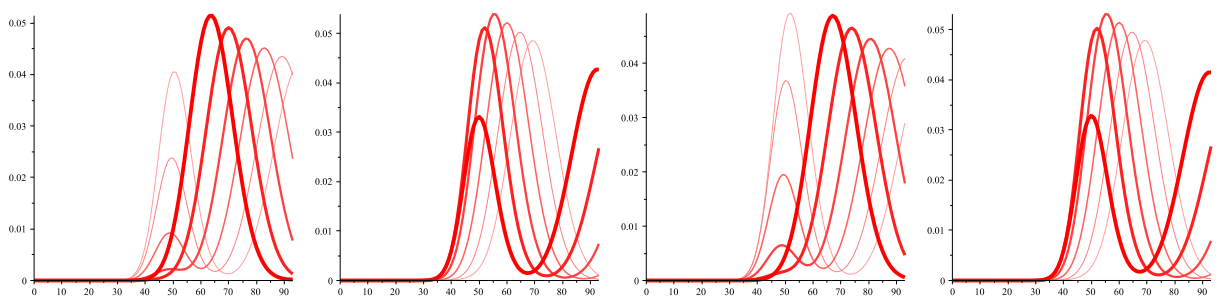
$$\frac{\#(C, k) \cdot 2^p}{|C| \cdot 2^p} = \frac{\#(C, k)}{|C|}$$

Distribution of the cache's size

(size = number of elements contained in cache)

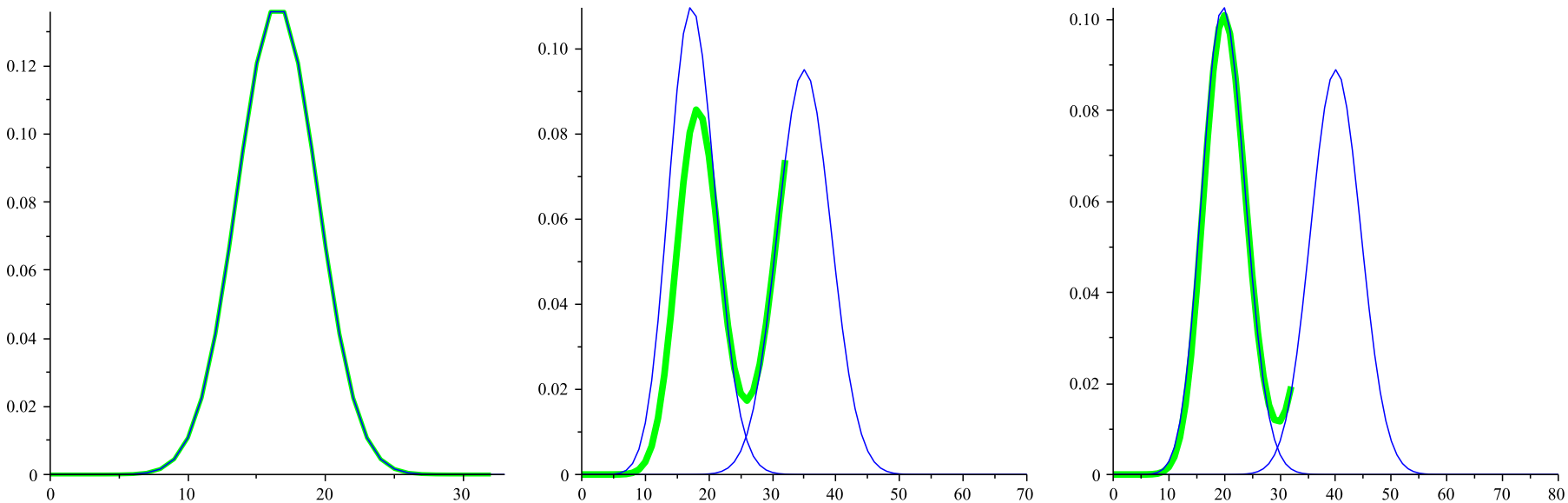
$$\mathbb{P}_n[X = k] = \delta_{k0} + \sum_{j=0}^{\infty} \left[\binom{n}{k} \frac{(1 - 1/2^j)^{n-k}}{2^{jk}} - \sum_{i=k}^{m} \binom{i}{k} \binom{n}{i} \frac{(1 - 1/2^j)^{n-i}}{2^{(j+1)i}} \right].$$





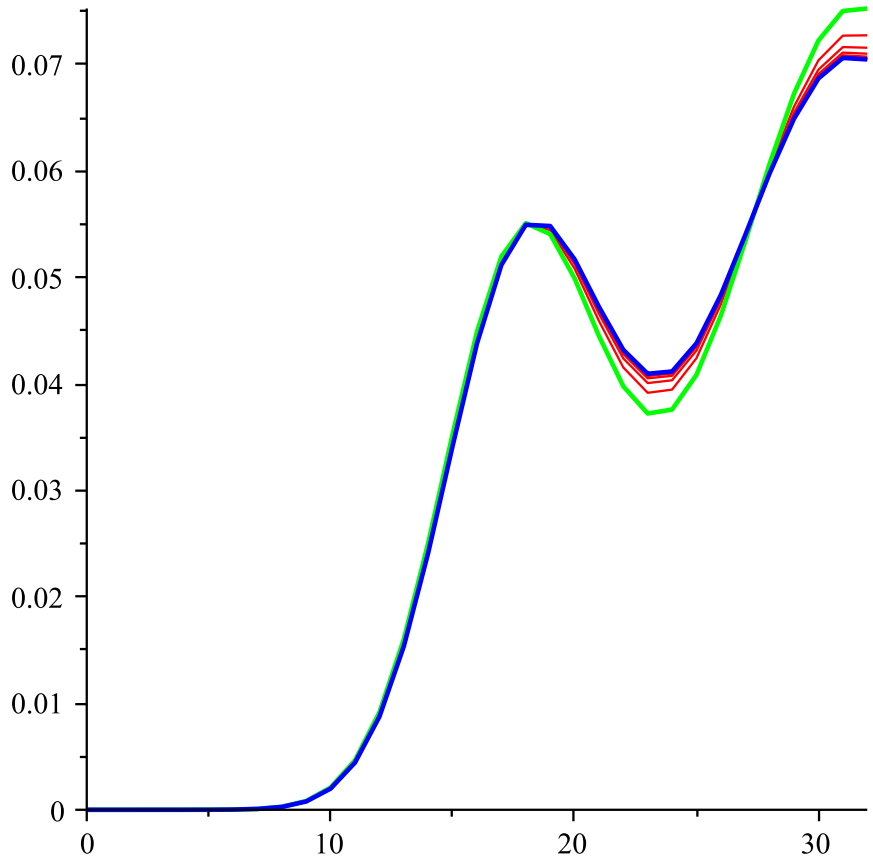
Distribution of the cache's size (2)

Cache capacity $m = 32$. Cardinality $n = 33, 70, 80$.



Distribution of the cache's size (3)

Cache capacity $m = 32$. Cardinalities $n = 128 \cdot 2^i$ (smallest, largest).



On Austen's complete works

Execution on Austen's works, with $m = 20$, can[2] give the following cache:

(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)

During the execution, the depth was $p = 10$.

Estimated cardinality is $|C| \cdot 2^p = 18 \cdot 2^{10} = 18432$.
Estimated count of elements with frequency 1: $8 \cdot 2^{10} = 8192$.
Estimated percentage of elements with frequency 1: $8/|C| = 44.4$ %.

% of items with frequency	1	2	3	4	Cardinality
Estimated	44.4 %	16.7 %	5.6 %	5.6 %	18432
Exact	47.0 %	11.3 %	4.2 %	3.0 %	19967

[2] an actual - but particularly auspicious - execution

4. REAL-VALUE HASH VERSION

Parameter: m cache capacity, q filtering parameter
Input: a stream $\mathcal{S} = (s_1, \ldots, s_N)$

---

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

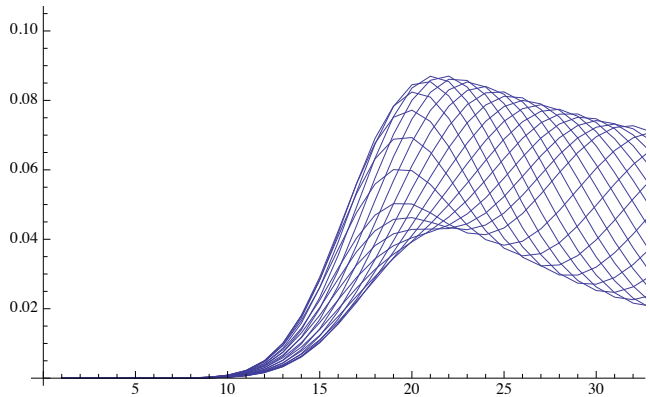
forall $x \in \mathcal{S}$ **do**

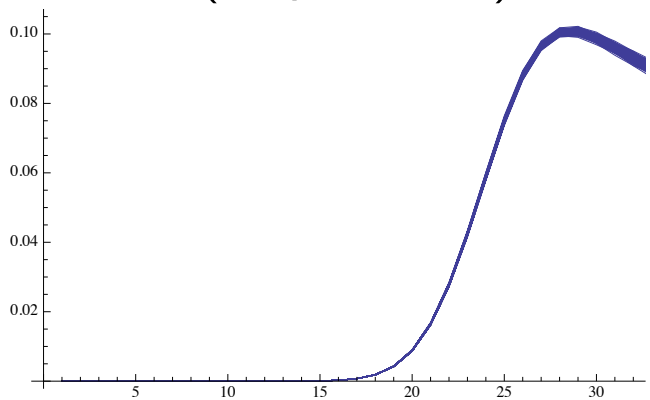
if $h(x) \leqslant q^p \cdots$ **then**
if $x \in C$ **then** increase the frequency count of x in C
else $C := C \cup \{(x, 1)\}$

if $|C| > m$ **then** {overflow of cache}
$p := p + 1$
filter(C) {remove items less than $q^p \cdots$}

return (C, p)

- classical variation found in survey [Mehta and Shrivastava, 2008]
- allows for better control of cache size (empties less)





- increases concentration of numerical estimates (here, for $m = 256$)

	$q = 1/2$	$q = \sqrt{2}/2$	$q = 9/10$
Mean	6.11	5.35	5.23
Std. Dev.	4.94	4.03	3.86

5. NEXT STEPS

Analysis:

- distribution of $\#(C,k)\cdot 2^p$ almost done
- next, risk analysis of $\#(C,k)\cdot 2^p$
- next, distribution and risk analysis of percentage

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

  is harder as ratio of two RVs.

Algorithmic:

- fine-tuning of the q filtering parameter
- adapt algorithm to large-frequency problems (iceberg, k-top)
- seek other applications, unrelated to frequency