Estimating Entropy (and its Friends) on Data Streams

Amit Chakrabarti

Dartmouth College
Hanover, NH, USA

Largely based on joint work with
Graham Cormode and Andrew McGregor

What is a Data Stream?

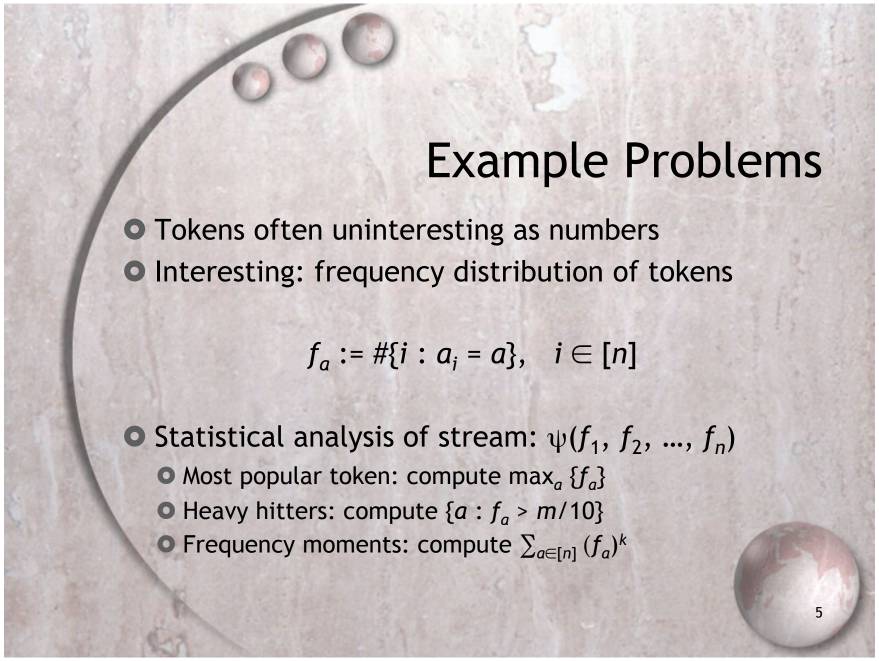
- A **huge** amount of data whizzing by
- Relevance: explosion of data in our heavily networked world
  - 1 billion credit card transactions/month, worldwide
  - 3 billion telephone calls/day, in U.S.
  - 1 billion IP packets/hour, at an average router
  - 2.5 billion emails/hour, worldwide (2006 est.)
- Want to mine such a huge data stream, but can't store it all

What is Entropy?

- A measure of randomness or information content
- Thermodynamics, anyone?
- For probability distribution $\mathbf{p} = (p_1, p_2, \ldots, p_n)$, entropy $\mathrm{H}(\mathbf{p}) := \sum_{i \in [n]} p_i \log(1/p_i)$
- Rich mathematical theory (information theory), initiated by Claude Shannon

Data Stream Model

- Input stream = sequence $\langle a_1, a_2, \ldots, a_m \rangle$
- Each token $a_i \in [n] := \{1, 2, \ldots, n\}$
- m, n huge
- Compute function $\phi(a_1, a_2, \ldots, a_m)$ using
  - sublinear space << m, n; ideally, polylog(m, n)
  - small number of passes; ideally, one pass

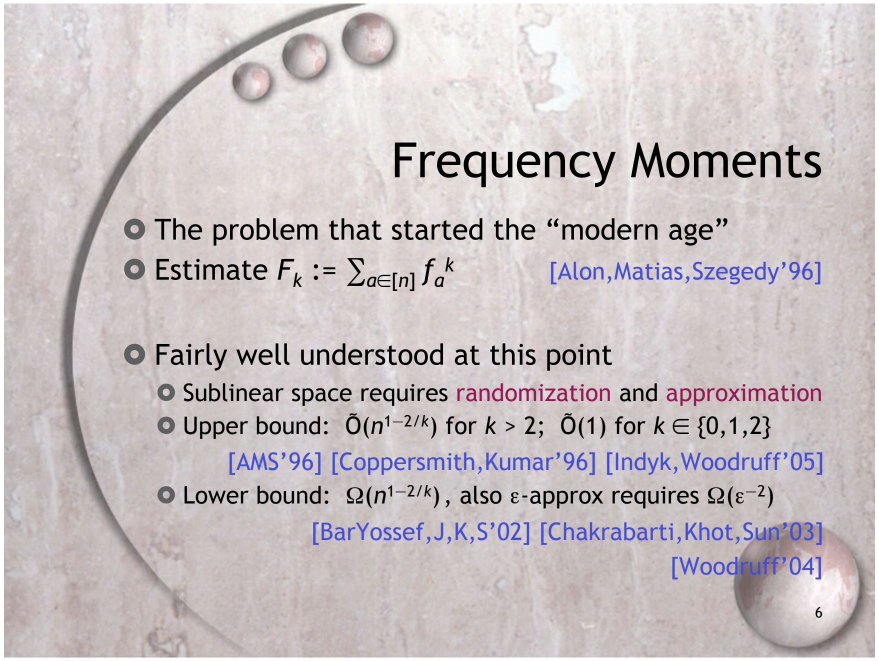


Example Problems

- Tokens often uninteresting as numbers
- Interesting: frequency distribution of tokens

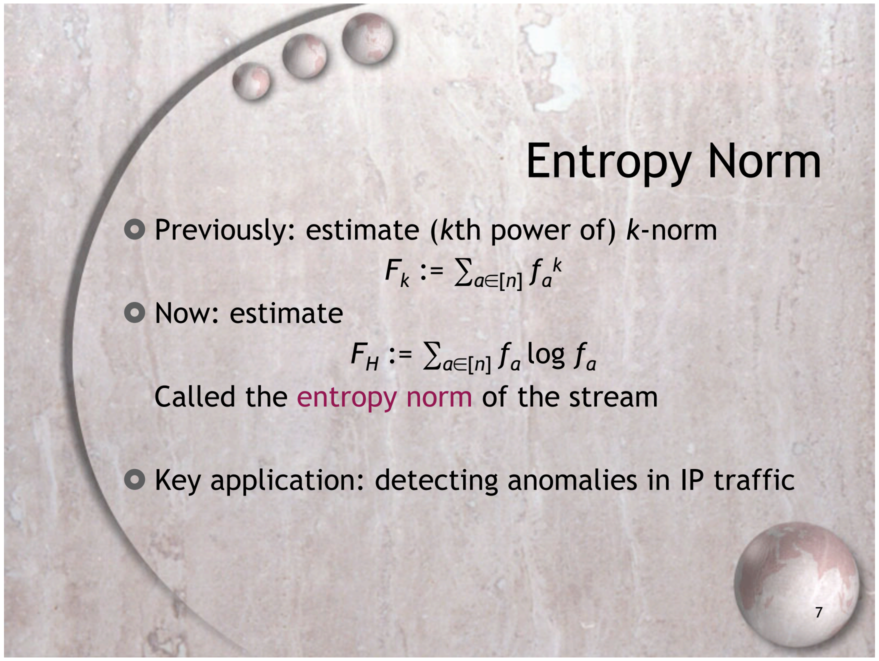
$$f_a := \#\{i : a_i = a\}, \quad i \in [n]$$

- Statistical analysis of stream: $\psi(f_1, f_2, \ldots, f_n)$
  - Most popular token: compute $\max_a \{f_a\}$
  - Heavy hitters: compute $\{a : f_a > m/10\}$
  - Frequency moments: compute $\sum_{a\in[n]} (f_a)^k$



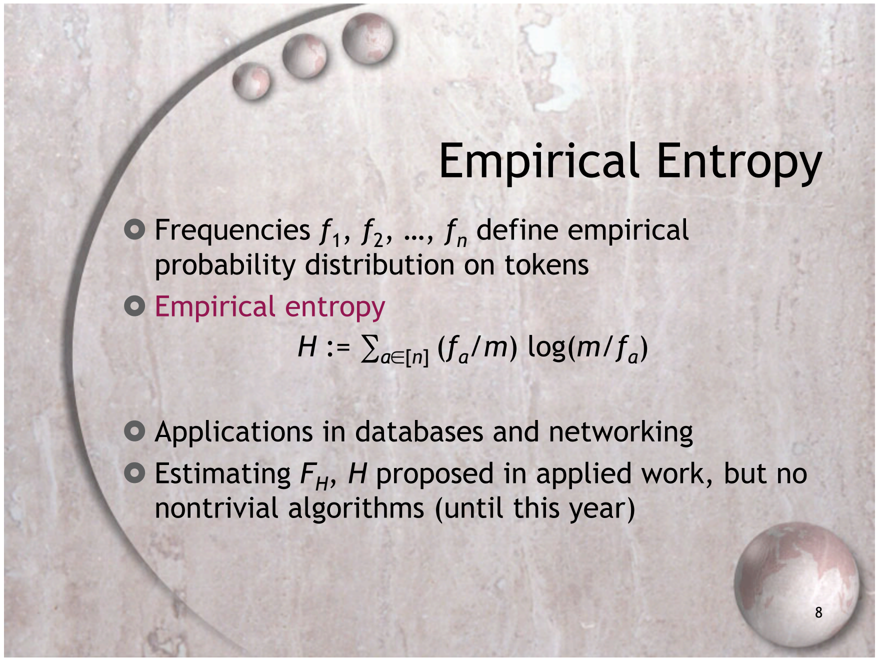
Frequency Moments

- The problem that started the “modern age”
- Estimate $F_k := \sum_{a\in[n]} f_a{}^k$ [Alon,Matias,Szegedy’96]
- Fairly well understood at this point
  - Sublinear space requires randomization and approximation
  - Upper bound: $\tilde{O}(n^{1-2/k})$ for $k > 2$; $\tilde{O}(1)$ for $k \in \{0,1,2\}$ [AMS’96] [Coppersmith,Kumar’96] [Indyk,Woodruff’05]
  - Lower bound: $\Omega(n^{1-2/k})$, also ε-approx requires $\Omega(\varepsilon^{-2})$ [BarYossef,J,K,S’02] [Chakrabarti,Khot,Sun’03] [Woodruff’04]



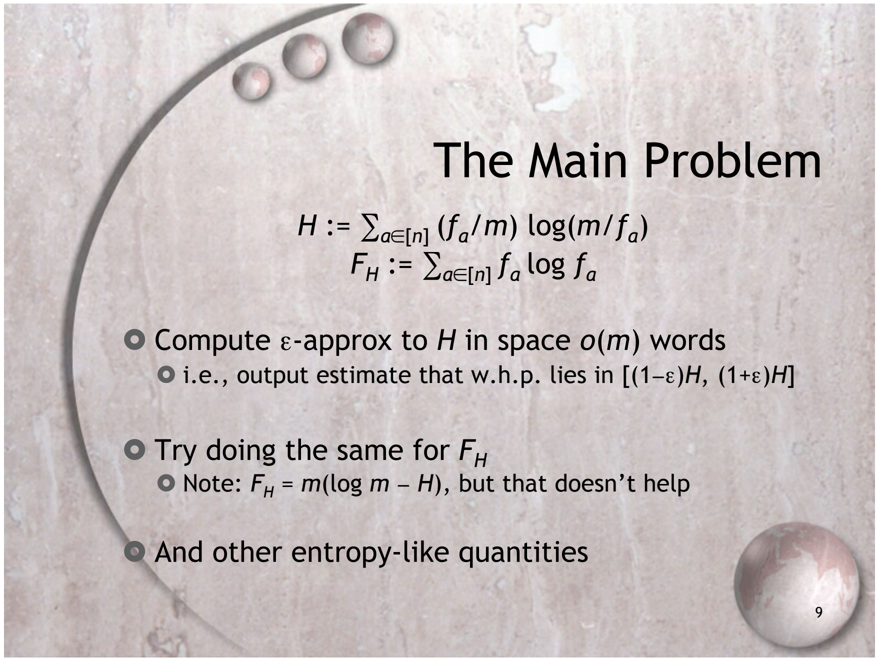
Entropy Norm

- Previously: estimate (kth power of) k-norm
  $$F_k := \sum_{a\in[n]} f_a{}^k$$
- Now: estimate
  $$F_H := \sum_{a\in[n]} f_a \log f_a$$
  Called the entropy norm of the stream
- Key application: detecting anomalies in IP traffic



Empirical Entropy

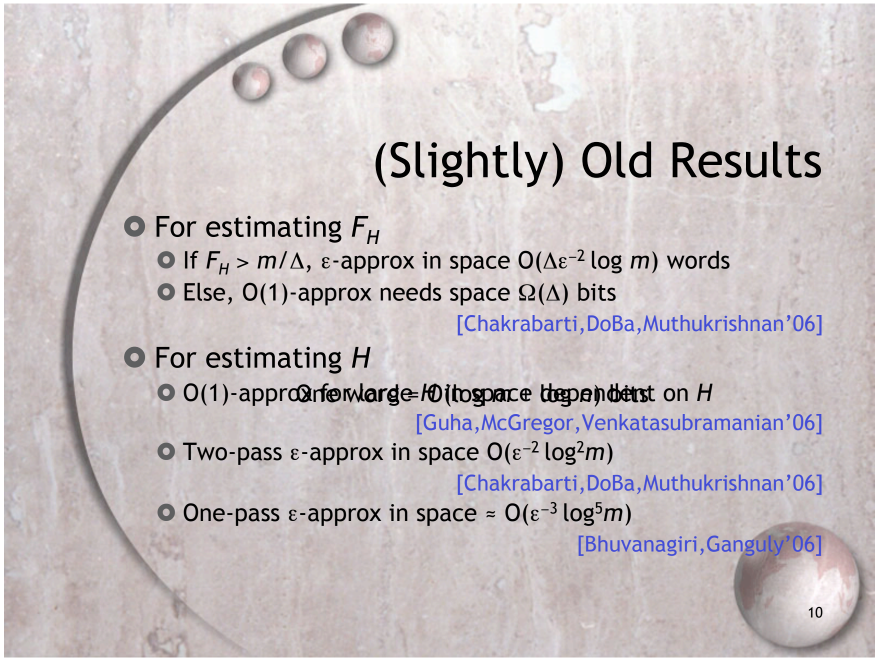
- Frequencies $f_1, f_2, \ldots, f_n$ define empirical probability distribution on tokens
- Empirical entropy
  $$H := \sum_{a\in[n]} (f_a/m) \log(m/f_a)$$
- Applications in databases and networking
- Estimating F_H, H proposed in applied work, but no nontrivial algorithms (until this year)



The Main Problem

$$H := \sum_{a\in[n]} (f_a/m)\log(m/f_a)$$
$$F_H := \sum_{a\in[n]} f_a \log f_a$$

- Compute ε-approx to H in space $o(m)$ words
  - i.e., output estimate that w.h.p. lies in $[(1-\varepsilon)H, (1+\varepsilon)H]$
- Try doing the same for F_H
  - Note: $F_H = m(\log m - H)$, but that doesn't help
- And other entropy-like quantities



(Slightly) Old Results

- For estimating F_H
  - If $F_H > m/\Delta$, ε-approx in space $O(\Delta\varepsilon^{-2}\log m)$ words
  - Else, O(1)-approx needs space $\Omega(\Delta)$ bits

[Chakrabarti,DoBa,Muthukrishnan'06]

- For estimating H
  - O(1)-approx for large H in space dependent on H

One word = $O(\log m + \log n)$ bits

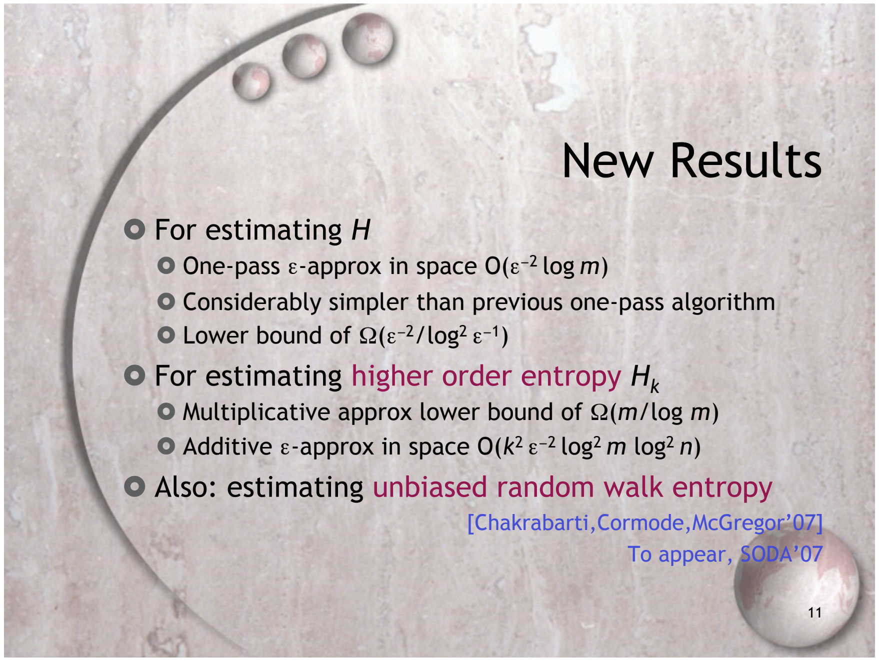
[Guha,McGregor,Venkatasubramanian'06]

  - Two-pass ε-approx in space $O(\varepsilon^{-2}\log^2 m)$

[Chakrabarti,DoBa,Muthukrishnan'06]

  - One-pass ε-approx in space ≈ $O(\varepsilon^{-3}\log^5 m)$

[Bhuvanagiri,Ganguly'06]

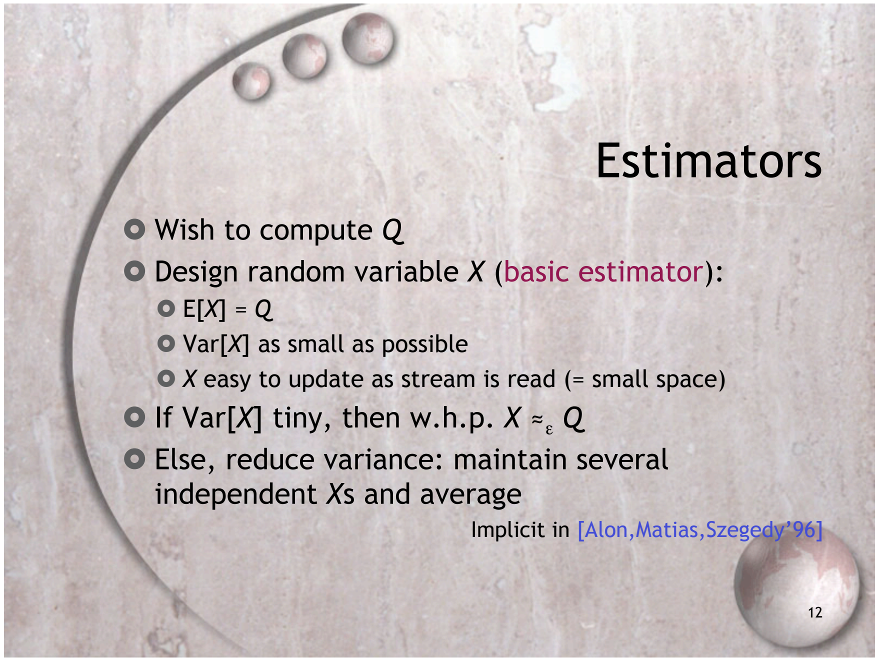


New Results

- For estimating H
  - One-pass ε-approx in space $O(\varepsilon^{-2}\log m)$
  - Considerably simpler than previous one-pass algorithm
  - Lower bound of $\Omega(\varepsilon^{-2}/\log^2 \varepsilon^{-1})$
- For estimating higher order entropy H_k
  - Multiplicative approx lower bound of $\Omega(m/\log m)$
  - Additive ε-approx in space $O(k^2\varepsilon^{-2}\log^2 m \log^2 n)$
- Also: estimating unbiased random walk entropy

[Chakrabarti,Cormode,McGregor'07]

To appear, SODA'07



Estimators

- Wish to compute Q
- Design random variable X (basic estimator):
  - $E[X] = Q$
  - $\mathrm{Var}[X]$ as small as possible
  - X easy to update as stream is read (= small space)
- If $\mathrm{Var}[X]$ tiny, then w.h.p. $X \approx_\varepsilon Q$
- Else, reduce variance: maintain several independent Xs and average

Implicit in [Alon,Matias,Szegedy'96]

Estimators: Brief Analysis

- Basic estimator X, E[X] = Q

- Let Y = average of $3\varepsilon^{-2}\mathrm{Var}[X]/Q^2$ copies of X
  Then, $\Pr[|Y-Q| > \varepsilon Q] \le 1/3$ (Chebyshev)
- Let Z = average of $5\varepsilon^{-2}\mathrm{Max}[X]/Q$ copies of X
  Then, $\Pr[|Z-Q| > \varepsilon Q] \le 1/3$ (Chernoff)

- Y (or Z) serves as a final estimator
  Space $\propto \mathrm{Var}[X]/Q^2$ or $\mathrm{Max}[X]/Q$

Designing an Estimator

- Input $\langle a_1, a_2, \ldots, a_m\rangle$; f_i = frequency of $a \in [n]$
- Want to compute $\sum_{a\in[n]} \phi(f_a)/m$, for some ϕ
  - To compute H, use $\phi(x) = x \log(m/x)$
  - To compute F_H, use $\phi(x) = mx \log x$

- Pick $J \in_{\mathrm{unif}} [m]$
- Let $R = \#\{k : a_k = a_J, J \le k \le m\}$
- Basic estimator $X = \phi(R) - \phi(R-1)$

Key Algorithmic Steps
Why This Works

- Want: $\sum_{a\in[n]} \phi(f_a)/m$
  - Pick $J \in_{\mathrm{unif}} [m]$
  - Let $R = \#\{k : a_k = a_J, J \le k \le m\}$
  - Basic estimator $X = \phi(R) - \phi(R-1)$

i.e., sample one token from the input stream

$$\mathrm{E}[X] = \sum_{a=1}^{n} \frac{f_a}{m} \sum_{r=1}^{f_a} \frac{1}{f_a}\big(\phi(r) - \phi(r-1)\big) = \frac{1}{m} \sum_{a\in[n]} \phi(f_a)$$

- Using some calculus, we can show
  - For F_H, $\mathrm{Var}[X]/F_H^2$ is "small"
  - For H, $\mathrm{Max}[X]/H$ is "small"…… well…… $\le (\log m)/H$

Dealing with $H = o(1)$

- If $H << 1$, space usage $(\log m)/H$ could be high
- When is $H < 1$?
  - Only when some $f_a > m/2$
  - i.e., when the input stream A has a dominator, a^*

- If we knew about a^* in advance…
  - Let $A' = A$ – (all occurrences of a^*)
  - Design estimator X' for A', similar to X for A
  - Compute H from X' and $|A'|$
- Easy two-pass algorithm, but how about one-pass?

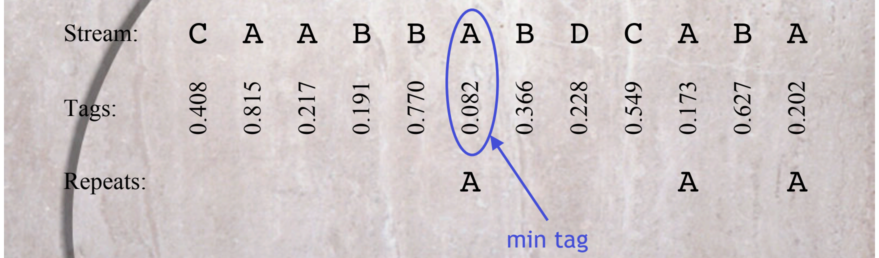
Dealing with $H = o(1)$

- In one pass, we need to
  - Sample one token from A
  - Sample one token from A', if a^* exists
  - Identify a^*
  - Estimate $|A'|$ within 1±ε

- Last two tasks: nice undergrad exercise today

Once a research problem: [Misra,Gries'82]

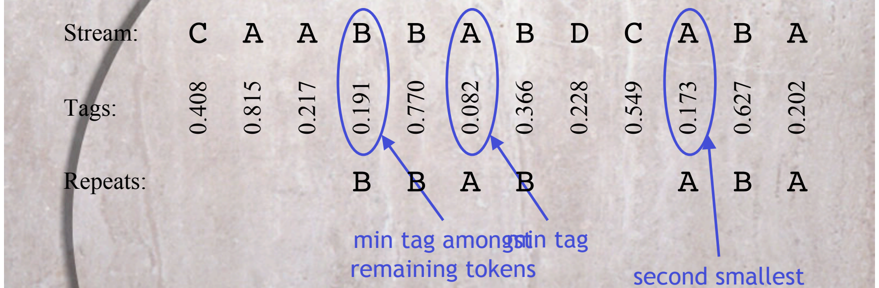
Sampling One Token



Stream:	C	A	A	B	B	A	B	D	C	A	B	A
Tags:	0.408	0.815	0.217	0.191	0.770	0.082	0.366	0.228	0.549	0.173	0.627	0.202
Repeats:						A				A		A

- Assign random tag ∈ [0,1] for each token
- Choose token with min tag (= uniform random choice)
- Implementation: keep track of (min tag, corresponding token, number of repeats)

Sampling Two Tokens



Stream:	C	A	A	B	B	A	B	D	C	A	B	A
Tags:	0.408	0.815	0.217	0.191	0.770	0.082	0.366	0.228	0.549	0.173	0.627	0.202
Repeats:				B	B	A	B			A	B	A

min tag amongst remaining tokens
min tag
second smallest tag, but we don't want this; same token as min tag!

- Assign tags, choose first token as before
- Delete all occurrences of first token
- Choose token with min remaining tag; count repeats
- Implementation: keep track of two triples (min tag, corresponding token, number of repeats)

Implementation: Some Details

Maintain (tag1, tok1, rep1), (tag2, tok2, rep2); tag1 < tag2
tok1 will be sample from A, tok2 will be sample from A'

```
On reading next token, a:
    x = random tag ∈ [m³]
    if a == tok1:
        if x < tag1 then (tag1,tok1,rep1) = (x,a,1) else rep1++
    else:
        if a == tok2 then rep2++
        if x < tag1:
            (tag2,tok2,rep2) = (tag1,tok1,rep1)
            (tag1,tok1,rep1) = (x,a,1)
        else:
            if x < tag2 then (tag2,tok2,rep2) = (x,a,1)
```

Example Run

Stream:	C	A	A	C
Tags:	0.408	0.815	0.217	0.391

a = C A

x = 0.408 0.815 0.217 0.391

(tag1,tok1,rep1) = (0.408, C, 1) (0.217, A, 1) (1.000, ⊥, 1)

(tag2,tok2,rep2) = (0.815, A, 1) (0.408, C, 1) (0.391, C, 2) (1.000, ⊥, 1)

Lower Bound

GAP-HAMM communication problem:

- Alice holds $x \in \{0,1\}^N$, Bob holds $y \in \{0,1\}^N$
- Promise: $\Delta(x,y)$ is either $\leq N/2$ or $\geq N/2 + \sqrt{N}$
- Which is the case?
- Model: one message from Alice to Bob

Requires $\Omega(N)$ bits of communication

[Indyk,Woodruff'03]

Lower Bound, Reduction

Alice: $x \in \{0,1\}^N$, Bob: $y \in \{0,1\}^N$

Entropy estimation algorithm $\mathscr{A}$

- Alice runs $\mathscr{A}$ on $\langle(1,x_1), (2,x_2), \ldots, (N,x_N)\rangle$
- Alice sends over memory contents to Bob
- Bob continues $\mathscr{A}$ on $\langle(1,y_1), (2,y_2), \ldots, (N,y_N)\rangle$

Alice	0	1	0	0	1	1
	(1,0)	(2,1)	(3,0)	(4,0)	(5,1)	(6,1)
	(1,1)	(2,1)	(3,0)	(4,0)	(5,1)	(6,0)
Bob	1	1	0	0	1	0

Observe: there are

- $2\Delta(x,y)$ tokens with frequency 1 each
- $N - \Delta(x,y)$ tokens with frequency 2 each

So, $H = \log N + \Delta(x,y)/N$

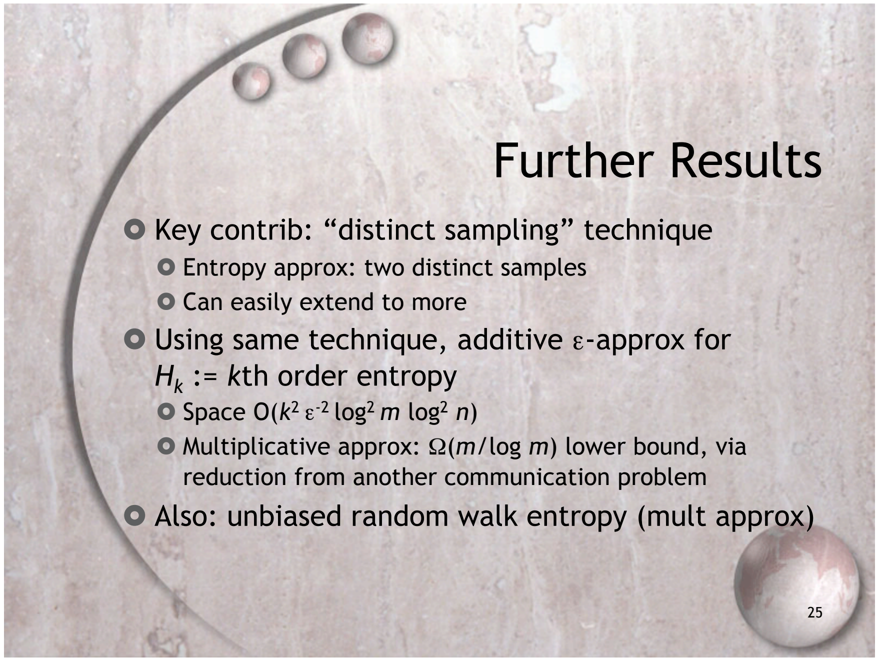
Either $H \leq \log N + 1/2$ or $H \geq \log N + 1/2 + 1/\sqrt{N}$

To distinguish, approximate H within $(1 \pm (\sqrt{N} \log N)^{-1})$

- For this, Alice's memory contents = $\Omega(N)$ bits
- Translation: $(1\pm\varepsilon)$-approx requires $\Omega(\varepsilon^{-2}/\log^2 \varepsilon^{-1})$ bits

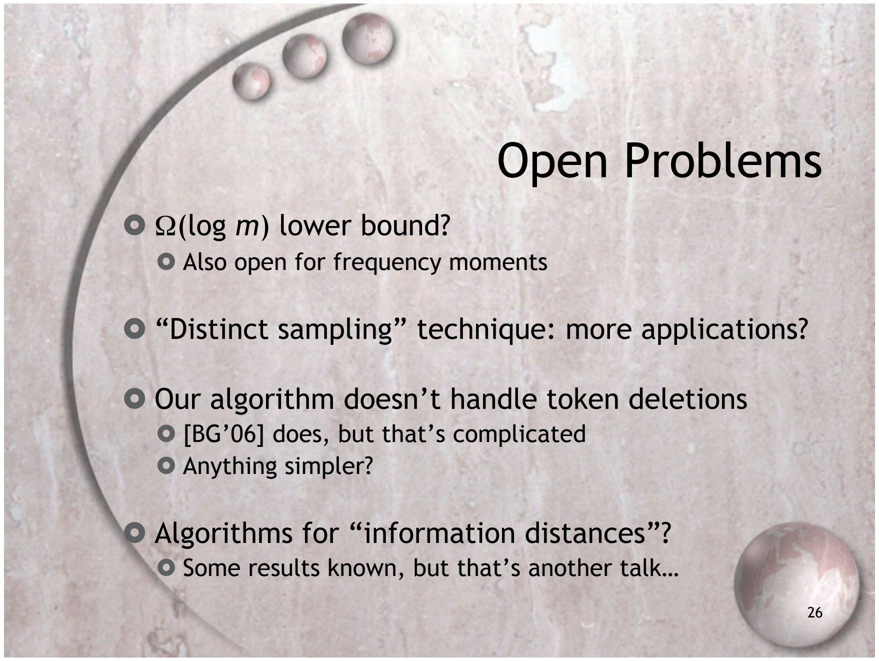
Extensions

- Approximating H within a sliding window
  - Width-W window: $O(\log W)$ space blowup
- Algorithm needs to know m in advance
  - After a tweak it no longer does
- Algorithm needs $O(m \log m)$ random bits
  - Can be reduced to $O(\text{polylog } m)$



Further Results

- Key contrib: “distinct sampling” technique
  - Entropy approx: two distinct samples
  - Can easily extend to more
- Using same technique, additive ε-approx for H_k := kth order entropy
  - Space $O(k^2\,\varepsilon^{-2}\log^2 m\log^2 n)$
  - Multiplicative approx: $\Omega(m/\log m)$ lower bound, via reduction from another communication problem
- Also: unbiased random walk entropy (mult approx)



Open Problems

- $\Omega(\log m)$ lower bound?
  - Also open for frequency moments
- “Distinct sampling” technique: more applications?
- Our algorithm doesn’t handle token deletions
  - [BG’06] does, but that’s complicated
  - Anything simpler?
- Algorithms for “information distances”?
  - Some results known, but that’s another talk…