IR: Information Retrieval
FIB, Master in Innovation and Research in Informatics

Slides by Marta Arias, José Luis Balcázar, Ramon Ferrer-i-Cancho, Ricard Gavaldá
Department of Computer Science, UPC

Fall 2018
http://www.cs.upc.edu/~ir-miri
10. Streaming Algorithms
1. Streaming Data and Stream Algorithms
2. Counting Items
3. Counting Distinct Items
4. Finding Frequent Items
5. Sliding Windows
Data streams everywhere

- Telcos - phone calls
- Satellite, radar, sensor data
- Computer systems and network monitoring
- Search logs, access logs
- RSS feeds, social network activity
- Websites, clickstreams, query streams
- E-commerce, credit card sales
- ...
We will have even more streaming data in the future

- We generate far more data than we can store
- Social networks: Planet-scale streams
- Smart cities
- Connected cars
- Sensors on our bodies
- Internet of Things
- Open data; governmental and scientific
In algorithmic words... 

The Data Stream axioms:

1. One pass over infinite sequence of items
   - item $t$ available at time $t$ only
2. Low time per item - read, process, discard
3. Sublinear memory - only summaries or sketches
4. Anytime, real-time answers
5. The stream evolves over time
Approximate answers are often OK

For most problems on data streams,

- it is difficult to give always the exact answer,
- but one can give an approximate answer often
Approximate answers are often OK

- Algorithms are randomized: random bits or numbers
- Different runs give different outputs
- But most runs give approximately correct answers
The Item Counting Problem
How many items have we read so far in the data stream?

To count up to \( t \) elements exactly, \( \log t \) bits are necessary

But \( \log \log t \) bits suffice for approximate solutions
Approximate counting, v1
Init: $c \leftarrow 0$

Update:
- draw a random number $x \in [0, 1]$
- if $(x \leq 1/2)$ $c \leftarrow c + 1$

Query: return $2c$

$\mathbb{E}[2c] = t, \quad \sigma \simeq \sqrt{t/2}$

Space $\log(t/2) = \log t - 1 \rightarrow \text{we saved 1 bit!}$
Approximate counting: Saving $k$ bits

Approximate counting, v2
Init: $c \leftarrow 0$

Update:
\[
\text{draw a random number } x \in [0, 1] \\
\text{if } (x \leq 2^{-k}) c \leftarrow c + 1
\]

Query: return $2^k c$

$E[c] = \frac{t}{2^k}$, \hspace{1cm} $\sigma \simeq \sqrt{\frac{t}{2^k}}$

Memory $\log t - k \rightarrow$ we saved $k$ bits!

$x \leq 2^{-k}$: AND of $k$ random bits, $\log k$ memory
Approximate counting: Morris’ counter

Morris’ counter [Morris77]

Init: \( c \leftarrow 0 \)

Update:
\[
\text{draw a random number } x \in [0, 1] \\
\text{if } (x \leq 2^{-c}) \quad c \leftarrow c + 1
\]

Query: return \( 2^c - 2 \)

\[
E[c] \simeq \log t, \quad E[2^c - 2] = t, \quad \sigma \simeq t/\sqrt{2}
\]

Memory = bits used to hold \( c = \log c = \log \log t \) bits
Morris’ approximate counter

- Can count up to 1 billion with $\log \log 10^9 = 5$ bits
- Can count up to $2^{64}$ with 6 bits
- Problem? large variance, $\sigma \simeq 0.7 t$
Use basis $b < 2$ instead of basis 2:

- Places $t$ in the series $1, b, b^2, \ldots, b^i, \ldots$ ("resolution" $b$)
- $E[b^c] \approx t$, $\sigma \approx \sqrt{(b - 1)/2} \cdot t$
- Space $\log \log t - \log \log b$ (>$\log \log t$, because $b < 2$)
- For $b = 1.08$, 3 extra bits, $\sigma \approx 0.2 t$
Reducing the variance, method II

- Run $r$ parallel, independent copies of the algorithm
- On Query, average their estimates
- $E[\text{Query}] \approx t$, $\sigma \approx t/\sqrt{2r}$ (why?)
- Space $r \log \log t$
- Time per item multiplied by $r$

Worse performance, but more generic technique
Morris’ counter: A non-streaming application

In [VanDurme+09]

- Counting $k$-grams in a large text corpus
- Number of $k$-grams grows exponentially with $k$
- Highly diverse frequencies
- Should fit in RAM
- Use Morris’ counters (5 bits) instead of standard counters
3. Counting distinct elements

The Distinct Element Counting Problem

How many *distinct* elements have we seen so far in the data stream?
Motivation

Item spaces and # distinct elements can be large

- I’m a web searcher. How many different queries did I get?
- I’m a router. How many pairs (sourceIP,destinationIP) have I seen?
  - itemspace: potentially $2^{128}$ in IPv6
- I’m a text message service. How many distinct messages have I seen?
  - itemspace: essentially infinite
- I’m an streaming classifier builder. How many distinct values have I seen for this attribute $x$?
Counting distinct elements

- Item space $I$, $|I| = n$, identified with range $[n]$
- $d = \text{number of distinct elements in stream}$
- Often omit subindex $t$
- Solving \textit{exactly} requires $O(d)$ memory
- Approximate solutions using $O(d)$, $O(\log d)$ and $O(\log \log d)$ bits
Linear counting [Whang+90] ≈ Bloom filters

- build a bit vector $B$ of size $s$
- choose a hash function $f : [n] \rightarrow s$

**Update**($x$): $B[f(x)] \leftarrow 1$

**Query:**
- $w =$ fraction of 0s in $B$
- return $s \cdot \ln(1/w)$

(Details omitted) Good performance if $s = O(d)$
Cohen’s algorithm [Cohen97]

\[ E[\text{gap between two 1’s in } B] = \frac{s - d}{d + 1} \approx \frac{s}{d} \]

Query: return \( s / (\text{size of first gap in } B) \)
Cohen’s algorithm [Cohen97]

**Trick:** Don’t store $B$, remember smallest key inserted in $B$

**Init:** $\text{posmin} = s$; choose hash function $h : [n] \rightarrow s$

**Update**($x$): if $(h(x) < \text{posmin})$ $\text{posmin} \leftarrow h(x)$

**Query:** return $s / \text{posmin}$;

---

Storing $\text{posmin}$ uses memory $O(\log d)$
Probabilistic Counting [Flajolet-Martin 85]

Bloom filter. But: Observe values of hash function $f(i)$, in binary

Idea: To see $f(i) = 0^{k-1}1 \ldots$, about $2^k$ distinct values inserted
And we don’t need to store $B$, just the smallest $k$
Flajolet-Martin probabilistic counter

Init: \( p \leftarrow 0 \)

Update(\( x \)):
  
  - let \( b \) be the position of the leftmost 1 bit of \( f(x) \)
  
  - if \( (b > p) \) \( p \leftarrow b \)

Query: return \( 2^p \)

\[
E[2^p] = \frac{d}{\varphi}, \text{ for a constant } \varphi = 0.77 \ldots
\]

Memory = (bits to store \( p \)) = \( \log p = \log \log d_{\text{max}} \) bits
Flajolet-Martin: reducing the variance

HyperLogLog [Flajolet+07]

- Maintain $c$ copies of the sketch
- Use first $\log c$ bits of $f(x)$ to decide copy for $x$
- Same $f$ can be used for all copies

- One sketch update per item - fast
- Query: Harmonic average of the $c$ copies

- Memory = $c \log \log d_{\text{max}}$
- Variance $\simeq 1.03 d/\sqrt{c}$

“cardinalities up to $10^9$ can be approximated within say 2% with 1.5 Kbytes of memory”
Application: Computing Distance Distributions in Graphs

For a directed graph $G = (V, E)$ and $u \in V$, neighborhood & distance functions

- $B(u, t) = \text{set of vertices at distance } \leq t \text{ from } u$
- $N(u, t) = |B(u, t)| - |B(u, t - 1)|$
- $N(t) = \sum_v N(u, t) = \text{number of pairs } (u, v) \text{ at distance } t$

Useful:

- $N(1)$ gives average degree
- $N(2)$ gives e.g. clustering coefficients
- max nonzero $N(t)$ gives diameter, etc.
Computing distance distributions

**Traditional algorithm:**

1. For each $v$, $B(v, 0) = \{v\}$

2. For $t = 1, 2, \ldots$
   
   for each $v \in V$,
   
   $B(v, t) = B(v, t - 1)$
   
   for each $(v, u) \in E$, $B(v, t) = B(v, t) \cup B(u, t - 1)$

3. For $t = 1, 2, \ldots$
   
   output $N(t) = \sum_v |B(v, t)| - N(t - 1)$

**Problems:**

- Random access to edges. Disk faults
- Memory $|V|^2$ in connected graphs, even if $|E| \ll |V|^2$
Computing distance distributions

Required:

- Access edges sequentially, as they are stored in disk
- Work well on small-world, sparse graphs
- Memory linear (or little more) in number of vertices
- Limited number of passes
HyperANF

ANF [Palmer, Gibbons, Faloutsos02]: Memory $O(n \log n)$
  - Graph with 2 billion links $\rightarrow$ 30 minutes on 90 machines

HyperANF [Boldi, Rosa, Vigna11]: Memory $O(n \log \log n)$
  - 15 minutes on a laptop
HyperANF

Key observation 1:
We eventually need only $|B(v, t)|$, not $B(v, t)$ itself.

Key observation 2:
$|B(v, t)|$ is the number of distinct elements connected by one edge to nodes in $B(v, t - 1)$.

Key observation 3:
Hyperloglog keeps number of distinct elements, and can implement unions (is mergeable).
Keep a hyperloglog counter $H(v)$ for each $v \in V$. Then

$$B(v, t) = B(v, t - 1)$$
for each $(v, u) \in E$, $B(v, t) = B(v, t) \cup B(u, t - 1)$

$\rightarrow$

$$H'(v) = H(v)$$
for each $(v, u) \in E$, $H'(v) = \text{merge}(H'(v), H(u))$

Big Win: this can be done while reading edges sequentially!
add a few other optimizations and clever programming
HyperANF: applications

Diameter of the Facebook graph [Backstrom+11]

- 720M active users, 69B friendship links
- Average distance is 4.74 (= 3.74 degrees of separation)
- 92% of users are at distance ≤ 5
- 10 hours on 256Gb RAM machine
Finding Frequent Elements

Heavy Hitters, Elephants, Hotlist analysis, Iceberg queries
Finding frequent elements

$\theta$-heavy hitters:

Given a sequence $S$ of $t$ elements, threshold $\theta$, find all elements with frequency $> \theta t$

Many different algorithms [Berinde+09], [Cormode+08]
The Space Saving sketch [Metwally+05]

Init($k$): Create
set of keys $K \leftarrow \emptyset$
vector $count$, indexed by $K$

Update($x$):
if ($x$ is in $K$) increment $count[x]$;
else, if ($|K| < k$) add $x$ to $K$ and set $count[x] = 1$;
else, replace an item with lowest count with $x$
and increase its count by 1

Query:
return the set $K$;
Why Does This Work?

Let $min_t$ be the minimum value of a counter at time $t > 0$. Then

1. $\sum_x count_t[x] = t$
2. $min_t \leq t/k$
3. If $f_t(x) > min_t$, then $x \in K$ at time $t$
4. For every $x \in K$, $f_t(x) \leq count_t[x] \leq f_t(x) + min_t$

Proof of (2), (3) by induction on $t$

No false negatives: all heavy hitters are in $K$

May have false positives: non-heavy-hitters in $K$
The Count-Min Sketch [Cormode-Muthukrishnan 04]

Like Space Saving:

- Provides an approximation $f'_x$ to $f_x$, for every $x$
- Can be used (less directly) to find $\theta$-heavy hitters
- Uses memory $O(1/\theta)$

Unlike Space Saving:

- It is randomized - hash functions instead of counters
- Supports additions and deletions
- Can be used as basis for several other queries
- For example, RangeSum queries and Heavy Hitters
  
  Given items $a$ and $b$, return frequency([a..b])
The Sliding Window Model

- Only last $n$ items matter
- Clear way to bound memory
- Natural in applications: emphasizes most recent data
- Data that is too old does not affect our decisions

Examples:
- Study network packets in the last day
- Detect top-10 queries in search engine in last month
- Analyze phone calls in last hours
Statistics on Sliding Windows

- Want to maintain mean, variance, histograms, frequency moments, hash tables, ... 
- SQL on streams. Extension of relational algebra 
- Want quick answers to queries at all times
Basic Problem: Counting 1's

Obvious algorithm, memory $n$:

- Keep window explicitly
- At each time $t$, add new bit $b$ to head, remove oldest bit $b'$ from tail,
- Add $b$ and subtract $b'$ from count

Fact:
$\Omega(n)$ memory bits are necessary to solve this problem exactly
Exponential Histograms
[Datar, Gionis, Indyk, Motwani, 2002]

**Deterministic** algorithm to estimate the number of 1’s in a window of length $n$ with multiplicative error $1/k$ using $O(k \log n)$ counters up to $n$ which means $O(k(\log n)^2)$ bits of memory

Example:

- $n = 10^6; k = 10$, error=10 %, 200 counters, 4,000 bits
Exponential Histograms

- Each bit has a timestamp - time at which it arrived
- Bits with timestamp $\leq$ now $- n$ are expired
- Bucket of capacity $s$ records $s$ 1's
- We have up to $k$ buckets of capacities 1, 2, 4, 8 ...
- Errors: expired bits in the last bucket
- Errors $\leq$ capacity of last bucket
  $\leq$ (capacity of all buckets) $/ k \simeq n / k$
- → Relative error $\leq 1 / k$
Exponential Histograms

- Error $\leq n/k$ for a window of size $n$
- Relative error $\leq 1/k$
- Choose max bucket size $s = \log(n/k)$
- Total capacity $k(1 + 2 + \cdots + 2^s) \geq n$, enough!
- Required counters $= k \log(n/k)$
Exponential Histograms

Init: Create empty set of buckets

Query: Return total number of bits in buckets — (last bucket / 2)
Insert rule(bit):

- If bit is a 0, ignore it. Otherwise, if it’s a 1:
- Add a bucket with 1 bit and current timestamp $t$ to the front
- for $i = 0, 1, \ldots$
  - If more than $k$ buckets of capacity $2^i$, merge two oldest as newest bucket of capacity $2^{i+1}$, with timestamp of the older one
- if oldest bucket timestamp $< t - n$, drop it (all expired)
Generalizations

Technique can be applied to maintain many natural aggregates:

- Max, min, variance
- Distinct elements
- Histograms
- Hash tables
Conclusions

- More and more data will be streaming data
- Streaming model is becoming a fundamental algorithmic paradigm
- Many problems admit surprisingly efficient approximation solutions
- Randomization and approximation useful tools