Streaming algorithms

Motivation

Motivation

- Given a long stream of data, process it quickly to compute some interesting statistic

Motivation

- Given a long stream of data, process it quickly to compute some interesting statistic
- **Example:** routers, possible statistics:
  - IP’s that are way more often that a typical one
  - etc.

Model

Model

- Given a stream of elements, estimate a function of the **frequencies** $\mathbf{f_i}$

Model

- Given a stream of elements, estimate a function of the **frequencies** f_i
- **Do not want to store frequencies explicitly**

Model

- Given a stream of elements, estimate a function of the **frequencies f_i**
- **Do not want to store frequencies explicitly**

IP	Count
128.74.251.151	42
77.88.55.55	5
204.79.197.200	1

A puzzle

A puzzle

- Given a stream of **n** elements, figure out if there is an element with $f_i > 0.5\,n$
  - Allow **two passes** and **O(1) words of storage**

A puzzle

- Given a stream of **n** elements, figure out if there is an element with $f_i > 0.5\,n$
  - Allow **two passes** and **O(1) words of storage**
- **Solution:** store a candidate **x** and a count **c**; on seeing **y**:
  - If **c = 0**, then **x = y**, **c = 1**
  - If **x = y**, then **c++**
  - **c--**
  - During the second pass, count f_x

A puzzle

- Given a stream of **n** elements, figure out if there is an element with $f_i > 0.5\,n$
  - Allow **two passes** and **O(1)** **words of storage**
- **Solution:** store a candidate **x** and a count **c**; on seeing **y**:
  - If **c = 0**, then **x = y**, **c = 1**
  - If **x = y**, then **c++**
  - **c--**
  - During the second pass, count f_x
- **Why works:** removing two *distinct* elements preserves majority

A puzzle

- Given a stream of **n** elements, figure out if there is an element with $\mathbf{f_i > 0.5\ n}$
  - Allow **two passes** and **O(1) words of storage**
- **Solution:** store a candidate **x** and a count **c**; on seeing **y**:
  - If **c = 0**, then **x = y**, **c = 1**
  - If **x = y**, then **c++**
  - **c--**
  - During the second pass, count $\mathbf{f_x}$
- **Why works:** removing two *distinct* elements preserves majority

Deterministic heavy hitters

Deterministic heavy hitters

- **Generalization:** find all elements with $f_i > \varepsilon n$ (there are less than $1/\varepsilon$ of them).

Deterministic heavy hitters

- **Generalization:** find all elements with $f_i > \varepsilon n$ (there are less than $1/\varepsilon$ of them).
- **[Misra, Gries 1982]:** maintain a set of at most $1/\varepsilon$ candidates with counters
  - If the current element is in the set, increase the counter
  - Add if can
  - Subtract **1** from all the counters and remove zeros

Deterministic heavy hitters

- **Generalization:** find all elements with $f_i > \varepsilon n$ (there are less than $1/\varepsilon$ of them).
- **[Misra, Gries 1982]:** maintain a set of at most $1/\varepsilon$ candidates with counters
  - If the current element is in the set, increase the counter
  - Add if can
  - Subtract **1** from all the counters and remove zeros
- In the end, get an estimate on f_i within additive εn

Deterministic heavy hitters

- **Generalization:** find all elements with $f_i > \varepsilon n$ (there are less than $1/\varepsilon$ of them).
- **[Misra, Gries 1982]:** maintain a set of at most $1/\varepsilon$ candidates with counters
  - If the current element is in the set, increase the counter
  - Add if can
  - Subtract **1** from all the counters and remove zeros
- In the end, get an estimate on f_i within additive εn
- Second pass to count frequencies for all the candidates

Summary

Summary

- Two pass deterministic exact algorithm for heavy hitters

Summary

- Two pass deterministic exact algorithm for heavy hitters
- Advantages:
  - Simple and practical (almost trivial to implement!)
  - Deterministic
  - Clearly optimal

Summary

- Two pass deterministic exact algorithm for heavy hitters
- Advantages:
  - Simple and practical (almost trivial to implement!)
  - Deterministic
  - Clearly optimal
- Drawbacks:
  - Two passes
  - Does not support deletions
  - Will see later a stronger guarantee (**L_2** vs **L_1**)

Sketches

Sketches

- Function **f(x, y)**, where **x** and **y** are large inputs

Sketches

- Function **$f(x, y)$**, where **x** and **y** are large inputs
- **Sketch:** a random compression function **$sk(x)$** and a decoding algorithm **D** such that with high probability **$D(sk(x), sk(y)) \approx f(x, y)$**

Sketches

- Function **$f(x, y)$**, where **x** and **y** are large inputs
- **Sketch:** a random compression function **$sk(x)$** and a decoding algorithm **D** such that with high probability **$D(sk(x), sk(y)) \approx f(x, y)$**
- **An example:** the **Johnson—Lindenstrauss lemma**! Can compress a vector into **$O(1 / \varepsilon^2)$** dimensions to estimate the Euclidean norm of the difference

Sketches

- Function **f(x, y)**, where **x** and **y** are large inputs
- **Sketch:** a random compression function **sk(x)** and a decoding algorithm **D** such that with high probability **D(sk(x), sk(y)) ≈ f(x, y)**
- **An example:** the **Johnson—Lindenstrauss lemma**! Can compress a vector into **O(1 / ε^2)** dimensions to estimate the Euclidean norm of the difference
- **Linear sketches: sk(x) = Ax** for random **A**

Sketches

- Function **$f(x, y)$**, where **x** and **y** are large inputs
- **Sketch:** a random compression function **$sk(x)$** and a decoding algorithm **D** such that with high probability **$D(sk(x), sk(y)) \approx f(x, y)$**
- **An example:** the **Johnson—Lindenstrauss lemma**! Can compress a vector into **$O(1 / \varepsilon^2)$** dimensions to estimate the Euclidean norm of the difference
- **Linear sketches:** **$sk(x) = Ax$** for random **A**
- Easy to maintain **Ax** under insertions/deletions

… with a caveat!

… with a caveat!

- Can’t afford to store a sketch matrix **A**

… with a caveat!

- Can't afford to store a sketch matrix **A**
- Need to generate it on the fly using small amount of randomness (bounded independence, Nisan's generator etc.)

… with a caveat!

- Can’t afford to store a sketch matrix **A**
- Need to generate it on the fly using small amount of randomness (bounded independence, Nisan’s generator etc.)
- But let’s not worry about it!

Taxonomy of linear sketches

Taxonomy of linear sketches

- **$O(1 / \varepsilon^2)$** for the **L_2** norm

Taxonomy of linear sketches

- **$O(1 / \varepsilon^2)$** for the **L_2** norm
- **$O(\log n / \varepsilon)$** for **ε**-Heavy Hitters (will see in a moment)

Taxonomy of linear sketches

- **$O(1 / \varepsilon^2)$** for the **L_2** norm
- **$O(\log n / \varepsilon)$** for **ε**-Heavy Hitters (will see in a moment)
- **$O(\log n / \varepsilon^2)$** for **$\varepsilon$**-Heavy Hitters w.r.t. **L_2** norm (stronger)

Taxonomy of linear sketches

- **$O(1 / \varepsilon^2)$** for the **L_2** norm
- **$O(\log n / \varepsilon)$** for **ε**-Heavy Hitters (will see in a moment)
- **$O(\log n / \varepsilon^2)$** for **ε**-Heavy Hitters w.r.t. **L_2** norm (stronger)
- **$O(1 / \varepsilon^2)$** for **L_p** norm for **$0 < p < 2$** (as opposed to **$p > 2$**)

Taxonomy of linear sketches

- **$O(1 / \varepsilon^2)$** for the **L_2** norm
- **$O(\log n / \varepsilon)$** for **ε**-Heavy Hitters (will see in a moment)
- **$O(\log n / \varepsilon^2)$** for **ε**-Heavy Hitters w.r.t. **L_2** norm (stronger)
- **$O(1 / \varepsilon^2)$** for **L_p** norm for **$0 < p < 2$** (as opposed to **$p > 2$**)
- Graph sketches etc.

Point queries

Point queries

- Receive **n** updates “insert/delete **k**”, where **$1 \leq k \leq u$**

Point queries

- Receive **n** updates "insert/delete **k**", where $\mathbf{1 \le k \le u}$
- **Want:** find all elements with frequencies $\mathbf{> \varepsilon n}$

Point queries

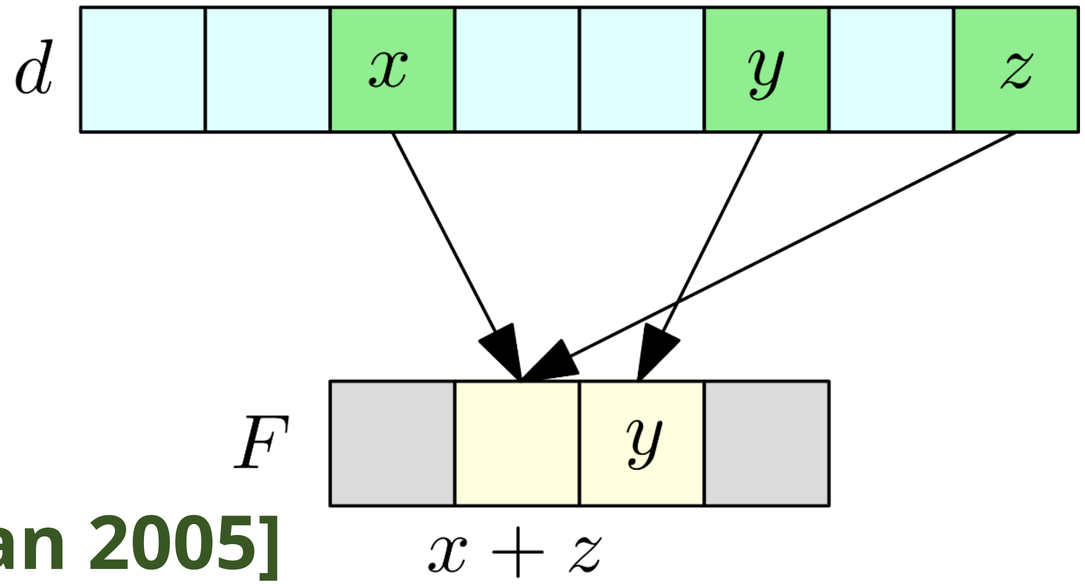
- Receive $\mathbf{n}$ updates "insert/delete $\mathbf{k}$", where $\mathbf{1 \le k \le u}$
- **Want:** find all elements with frequencies $\mathbf{> \varepsilon n}$
- **Point queries:** for $\mathbf{1 \le k \le u}$ estimate $\mathbf{x_k}$ up to $\mathbf{\pm\, \varepsilon \, \|x\|_1}$ w.h.p.

Count-min sketch

Count-min sketch

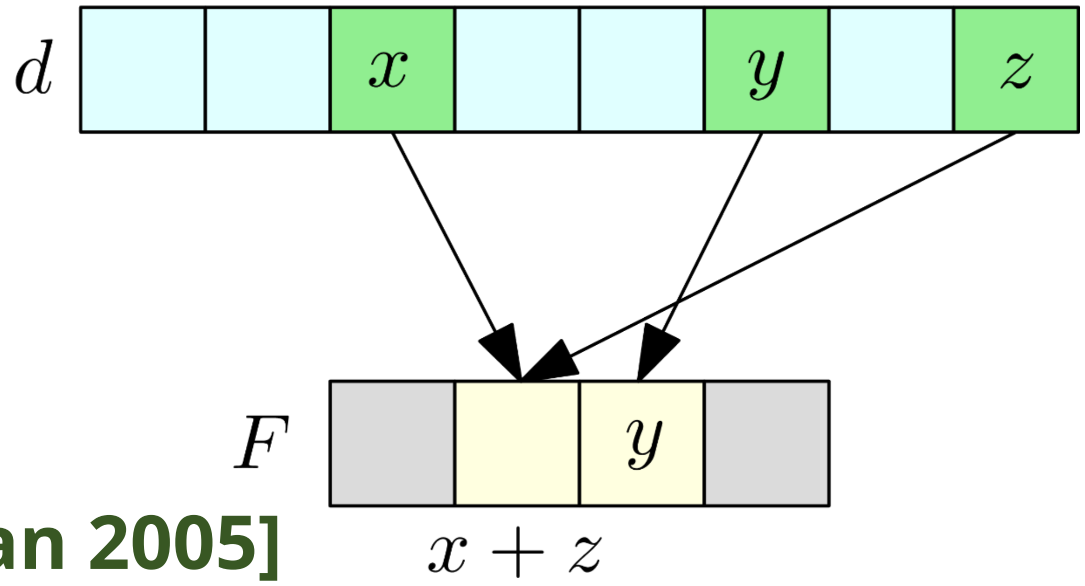
- **Idea: [Cormode, Muthukrishnan 2005]**
  - For a hash **h: [u] → [t]**, store sums for each of the **t** bins

Count-min sketch



- **Idea: [Cormode, Muthukrishnan 2005]**
  - For a hash **h: [u] → [t]**, store sums for each of the **t** bins

Count-min sketch



- **Idea: [Cormode, Muthukrishnan 2005]**
  - For a hash **h: [u] → [t]**, store sums for each of the **t** bins
  - Repeat for **s** hashes
  - For a point query **k**, take the **minimum** of the sums for **h(k)**
- **t=1/ε, k=O(log(1/δ))**

Count-min sketch ctd.

Count-min sketch ctd.

- Linear sketch

Count-min sketch ctd.

- Linear sketch
- Enough to have **2**-independent hash functions

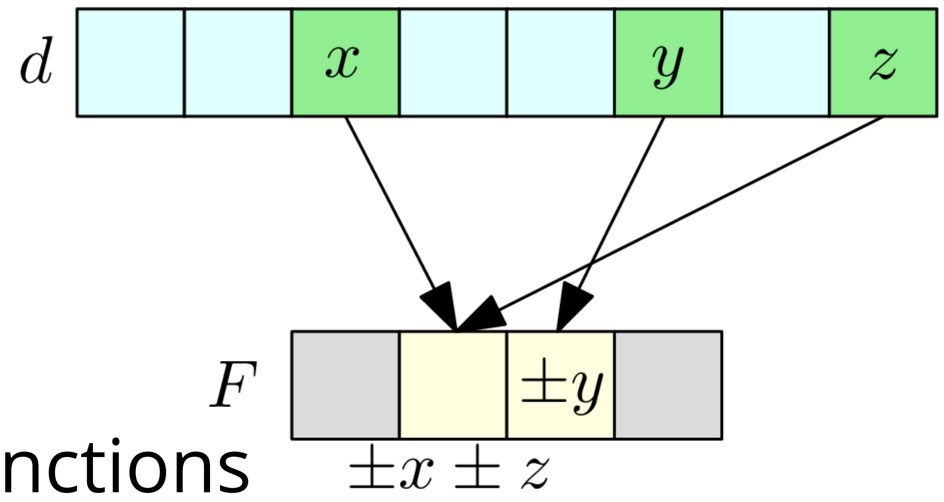
Count-min sketch ctd.

- Linear sketch
- Enough to have **2**-independent hash functions
- Important to have **fast** hash functions, lots of research in this direction

Count-min sketch ctd.

- Linear sketch
- Enough to have **2**-independent hash functions
- Important to have **fast** hash functions, lots of research in this direction
- A stronger version: for $\mathbf{1 \le k \le u}$ estimate $\mathbf{x_k}$ up to $\pm\, \varepsilon\, \|x\|_2$ w.h.p. **[Charikar, Chen, Farach-Colton 2002]**
  - Combine elements with random signs
  - Need range of size $\mathbf{t=1/\varepsilon^2}$

Count-min sketch ctd.



- Linear sketch
- Enough to have **2**-independent hash functions
- Important to have **fast** hash functions, lots of research in this direction
- A stronger version: for **$1 \le k \le u$** estimate **x_k** up to **$\pm\ \varepsilon\ \|x\|_2$** w.h.p. **[Charikar, Chen, Farach-Colton 2002]**
  - Combine elements with random signs
  - Need range of size **$t=1/\varepsilon^2$**

Graph sketching

Graph sketching

- Can we sketch anything else besides vectors?

Graph sketching

- Can we sketch anything else besides vectors?
- Yes! **Graph sketching** **[Ahn, Guha, McGregor 2012]**

Graph sketching

- Can we sketch anything else besides vectors?
- Yes! **Graph sketching** **[Ahn, Guha, McGregor 2012]**
- Want to maintain an **n**-node undirected graph under edge insertions/deletions in **$o(n^2)$** space and then compute something useful

Graph sketching

- Can we sketch anything else besides vectors?
- Yes! **Graph sketching** **[Ahn, Guha, McGregor 2012]**
- Want to maintain an **n**-node undirected graph under edge insertions/deletions in **$o(n^2)$** space and then compute something useful
- Finding connected components?

Connectivity

Connectivity

- **[Ahn, Guha, McGregor 2012]:** there are linear sketches of size **n polylog(n)** for connectivity (and other problems)

Connectivity

- **[Ahn, Guha, McGregor 2012]:** there are linear sketches of size **n polylog(n)** for connectivity (and other problems)
- Even cooler: gives a way to compress (linearly) each row of an adjacency matrix into **polylog(n)** words

Connectivity

- **[Ahn, Guha, McGregor 2012]:** there are linear sketches of size **n polylog(n)** for connectivity (and other problems)
- Even cooler: gives a way to compress (linearly) each row of an adjacency matrix into **polylog(n)** words
- The plan:
  - Design a “classical” algorithm that is amenable to sketching
  - Implement it using sketches

Classical algorithm

- Repeat **O(log n)** times:
  - For every node find an arbitrary neighbor
  - Collapse subgraphs formed by these pairs

Vector representation

Vector representation

- For every node **v** store an $\mathbf{n^2}$-dimensional **{-1, 0, 1}**-vector $\mathbf{a_v}$

Vector representation

- For every node **v** store an **n^2**-dimensional **{-1, 0, 1}**-vector **a_v**
- For every set of nodes the support of the sum of **a_v**'s corresponds to the outgoing edges

Vector representation

- For every node **v** store an **n^2**-dimensional **{-1, 0, 1}**-vector **a_v**
- For every set of nodes the support of the sum of **a_v**'s corresponds to the outgoing edges
- **Remains: linear** sketch for finding a coordinate in the support of a vector
  - Linearity is crucial

L_0-sampling

L_0-sampling

- Linear sketch for a vector that would allow to sample (once) uniformly a coordinate from the support

L_0-sampling

- Linear sketch for a vector that would allow to sample (once) uniformly a coordinate from the support
- Idea: **[Jowhari, Saglam, Tardos 2011]**
  - Subsample at all rates
  - Use **sparse recovery** to recover the (small) support of a subsample

Connectivity: a wrap-up

Connectivity: a wrap-up

- Need a fresh sketch at every iteration
    - The probability of failure is over the sketch construction

Lower bounds

Lower bounds

- Can stream/sketch **L_2** norm using **$O(1 / \varepsilon^2)$** words

Lower bounds

- Can stream/sketch L_2 norm using $O(1 / \varepsilon^2)$ words
- What about other L_p?

Lower bounds

- Can stream/sketch **L_2** norm using **$O(1 / \varepsilon^2)$** words
- What about other **L_p**?
- **[Indyk 2000]:** the same bound for **$0 < p < 2$** (unlike insertions-only, **L_1** is not obvious at all)

Lower bounds

- Can stream/sketch **L_2** norm using **$O(1 / \varepsilon^2)$** words
- What about other **L_p**?
- **[Indyk 2000]:** the same bound for **$0 < p < 2$** (unlike insertions-only, **L_1** is not obvious at all)
- **Not so for $p > 2$! [Alon, Matias, Szegedy 1995], [Bar-Yossef, Jayram, Kumar, Sivakumar 2002], [Indyk, Woodruff 2005]**

Lower bounds

- Can stream/sketch L_2 norm using $O(1 / \varepsilon^2)$ words
- What about other L_p?
- **[Indyk 2000]:** the same bound for $0 < p < 2$ (unlike insertions-only, L_1 is not obvious at all)
- **Not so for $p > 2$! [Alon, Matias, Szegedy 1995], [Bar-Yossef, Jayram, Kumar, Sivakumar 2002], [Indyk, Woodruff 2005]**

Need around $d^{1-2/p}$ words for d dimensions!

Lower bounds

- Can stream/sketch L_2 norm using $O(1 / \varepsilon^2)$ words
- What about other L_p?
- **[Indyk 2000]:** the same bound for $0 < p < 2$ (unlike insertions-only, L_1 is not obvious at all)
- **Not so for $p > 2$! [Alon, Matias, Szegedy 1995], [Bar-Yossef, Jayram, Kumar, Sivakumar 2002], [Indyk, Woodruff 2005]**

  Need around $d^{1-2/p}$ words for d dimensions!
- How can one even hope to prove a lower bound like this?

Communication complexity

Communication complexity

$x \in \{0,1\}^n$ $\qquad$ $y \in \{0,1\}^n$

$$f(x, y)$$

Disjointness

- $f(x, y) = \exists i : (x_i = y_i = 1)$
- Any *randomized* communication protocol requires $\Omega(n)$ communication
- A conceptual proof using **information complexity**

[Bar-Yossef, Jayram, Kumar, Sivakumar 2002]

Estimating max-norm

- Stream of elements from **{1, …, d}**, **maximum frequency**
- **[Alon, Matias, Szegedy 1995]: 1.99**-approximation requires $\Omega(d)$ space
- Alice and Bob want to solve **disjointness**
  - Alice performs her updates
  - Alice sends the memory to Bob
  - Bob performs his updates
  - Max-frequency **1** vs. **2**
- Even one-way lower bound for disjointness is enough **(easy!)**

Lots of practical work

Lots of practical work

- **HyperLogLog** for counting distinct elements **[Flajolet, Fusy, Gandouet, Meunier 2007]**

Lots of practical work

- **HyperLogLog** for counting distinct elements **[Flajolet, Fusy, Gandouet, Meunier 2007]**
- **1-bit minwise hashing** for estimating similarity between documents **[Konig, Li 2008]**

Lots of practical work

- **HyperLogLog** for counting distinct elements **[Flajolet, Fusy, Gandouet, Meunier 2007]**
- **1-bit minwise hashing** for estimating similarity between documents **[Konig, Li 2008]**
- …