STREAMING ALGORITHMS

Ajinkya Potdar
Hemanga Krishna Borah

Contents

Data, data, data!!!

- Facebook by the numbers: 1.06 billion monthly active users
- Google+ by the numbers: 500m+ users, 235m of them active and 135m using the stream
- Twitter : 140 million users and sees 340 million tweets per day

Data vs Computation and Storage

- IBM Roadrunner - It has around 100 terabytes of RAM
- General computers – Few GB
- Routers – few MB (depending on the manufacturers)

Do you want to wait an hour and know the exact result?

- Finding frequency is O(n) space consuming and O(n) time consuming
- Finding MST is ~ O(m) space consuming and ~O(m+n) time consuming
- Majority Elements ~ O(n) space consuming and O(n) time consuming.

What else you might want to know

- Get me the most frequent users of my router
- Get me the estimate of the distinct users of my network
- Get me the shortest approximate relation between the two users A and B (social networks)
- Get me an approximate MST of this graph
- …. *and the queries continue*

Challenges

- Storing and indexing such large amount of data is costly.
- Important to process the data as it happens.
- Provide up to the minute analysis and statistics.
- Algorithms take only a single pass over their input.
- Compute various functions using resources that are sublinear in the size of input.

Solution to your problems – Streaming algorithms

- Within sub-linear space complexity
- Very fast computation

Bad news - Every good thing comes with a price. Only some approximation to the actual results.

Good news - there are algorithms which give you very good bounds.

Introduction-Data Streams

- Many data generation processes produce huge numbers of pieces of data, each of which is simple in isolation, but which taken together lead to a complex whole.

- Examples
  - Simple transactions of everyday life such as using a credit card, a phone or browsing the web lead to automated data storage.
  - Sequence of queries posed to an Internet search engine.

- Data can arrive at enormous rates - hundreds of gigabytes per day or higher.

Streaming Algorithms

- Input is presented as a sequence of items.
- Input can be examined in only a few passes (typically just one).
- Use limited memory available to them (much less than the input size)
- limited processing time per item.
- These constraints may mean that an algorithm produces an approximate answer based on a summary or "sketch" of the data stream in memory.

Streaming model

Input stream $a_1, a_2, \ldots$ arrives sequentially, and describes an underlying signal **A**, an one-dimensional function **A**: [1 .. N] -> R^2. Models differ on how a_i's describe **A**

- **Time Series Model:**
  - Each a_i equals A[i], and they appear in increasing order of i.
  - Anaylysed for some time window period. Discarded beyond a time duration. Also called **sliding window model**.

Streaming model…(*contd*)

- **Cash Register Model:**
  - a_i's are increments to A[j] 's. Say, $a_i = (j, I_i)$, $I_i \geq 0$.
  - $A_i[j] = A_{i-1}[j] + I_i$, where A_i is the state of the signal after seeing the i th element of the stream. (mutliple ai can update A[j] over time.

Ajinkya

- **Turnstile**
  - Here a_i's are updates to A[j]s instead of increments.
  - $a_i = (j, U_i)$ to mean $A_i[j] = A_{i-1}[j] + U_i$, where Ai is the signal after seeing the i th item in the stream.
  - U_i may be negative here.

Factors to evaluate Streaming Algorithms

- Data you can store at a time
- Computations you need to do for an input
- Number of passes you can do over the stream
- Time for query on the function A

How to evaluate Streaming Algorithms - Mathematically

- An additive definition

A streaming algorithm $\mathcal{S}$ is said to have (ε,δ) additive-approximation to the function F if we have

$$P_r[|\mathcal{S}(\sigma) - F(\sigma)| > \varepsilon] \leq \delta$$

- A multiplicative definition

A streaming algorithm is said to have () multiplicative-approximation to the function F if we have

$$Pr[|\frac{S(\sigma)}{F(\sigma)}| - 1 | > \varepsilon] \leq \delta$$

Real world problems

Problems range from simple to very complex.

- Given a stream of transactions, finding the mean and standard deviation of the bill totals.
  - Requires only few sufficient statistics to be stored.
- Determining whether a search query has already appeared in the stream of queries.
  - Requires a large amount of information to be stored.
- Algorithms must
  - Respond quickly to new information.
  - Use very less amount of resources in comparison to total quantity of data.

Frequent Items Problem

- Given a stream of items, the problem is simply to find those items which occur most frequently.
- Formalized as finding all items whose frequency exceeds a specified fraction of the total number of items.
- If space is the constraint, “difficult” problem, but we solve it by approximation.
- Find all items with count ≥ φN, none with count < (φ–ε)N
  - Error $0 < \varepsilon < 1$, e.g. $\varepsilon = 1/1000$
  - Related problem: estimate each frequency with error $\pm\varepsilon N$

Motivation

- The problem is important both in itself and as a subroutine in more advanced computations.

- For example,
  - It can help in routing decisions, for in-network caching etc (if items represent packets on the Internet).
  - Can help in finding popular terms if items represent queries made to an Internet search engine.
  - Mining frequent itemsets inherently builds on this problem as a basic building block.

- Algorithms for the problem have been applied by large corporations: AT&T and Google.

Variations

- Given a stream S of n items $t_1 \ldots t_n$, the *exact ϕ-frequent items* comprise the set $\{i \mid f_i > \phi n\}$, where f_i is the frequency of item i.

- Solving the exact frequent items problem requires $\Omega(n)$ space.

- Approximate version is defined based on tolerance for error parameterized by ε.

ε-approximate Frequent Items Problem

- Given a stream S of n items, the ε-*approximate frequent items* problem is to return a set of items F so that for all items $i \varepsilon F$, $f_i > (\phi - \varepsilon)n$, and there is no $i \notin F$ such that $f_i > \phi n$.

Frequency Estimation Problem

- Given a stream S of n items, the frequency estimation problem is to process a stream so that, given any i, an f_i^* is returned satisfying $f_i \leq f_i^* \leq f_i + \varepsilon n$.

Solutions

- Two main classes of algorithms:
  - Counter-based Algorithms
  - Sketch Algorithms

- Other Solutions:
  - **Quantiles** : based on various notions of randomly sampling items from the input, and of summarizing the distribution of items.
  - Less effective and have attracted less interest.

Counter-based Algorithms

- Track a subset of items from the input and monitor their counts.
- Decide for each new arrival whether to store or not.
- Decide what count to associate with it.

Majority Problem

> [J.Alg 2, P208–209] Suppose we have a list of n numbers, representing the "votes" of n processors on the result of some computation. We wish to decide if there is a majority vote and what the vote is.

- Problem posed by J. S. Moore in Journal of Algorithms, in 1981.

Majority Algorithm

- Start with a counter set to zero. For each item:
  - If counter is zero, store the item, set counter to 1.
  - Else, if item is same as item stored, increment counter.
  - Else, decrement counter.

- If there is a majority item, it is the item stored.

- Proof outline:
  - Each decrement pairs up two different items and cancels them out.
  - Since majority occurs > N/2 times, not all of its occurrences can be canceled out.

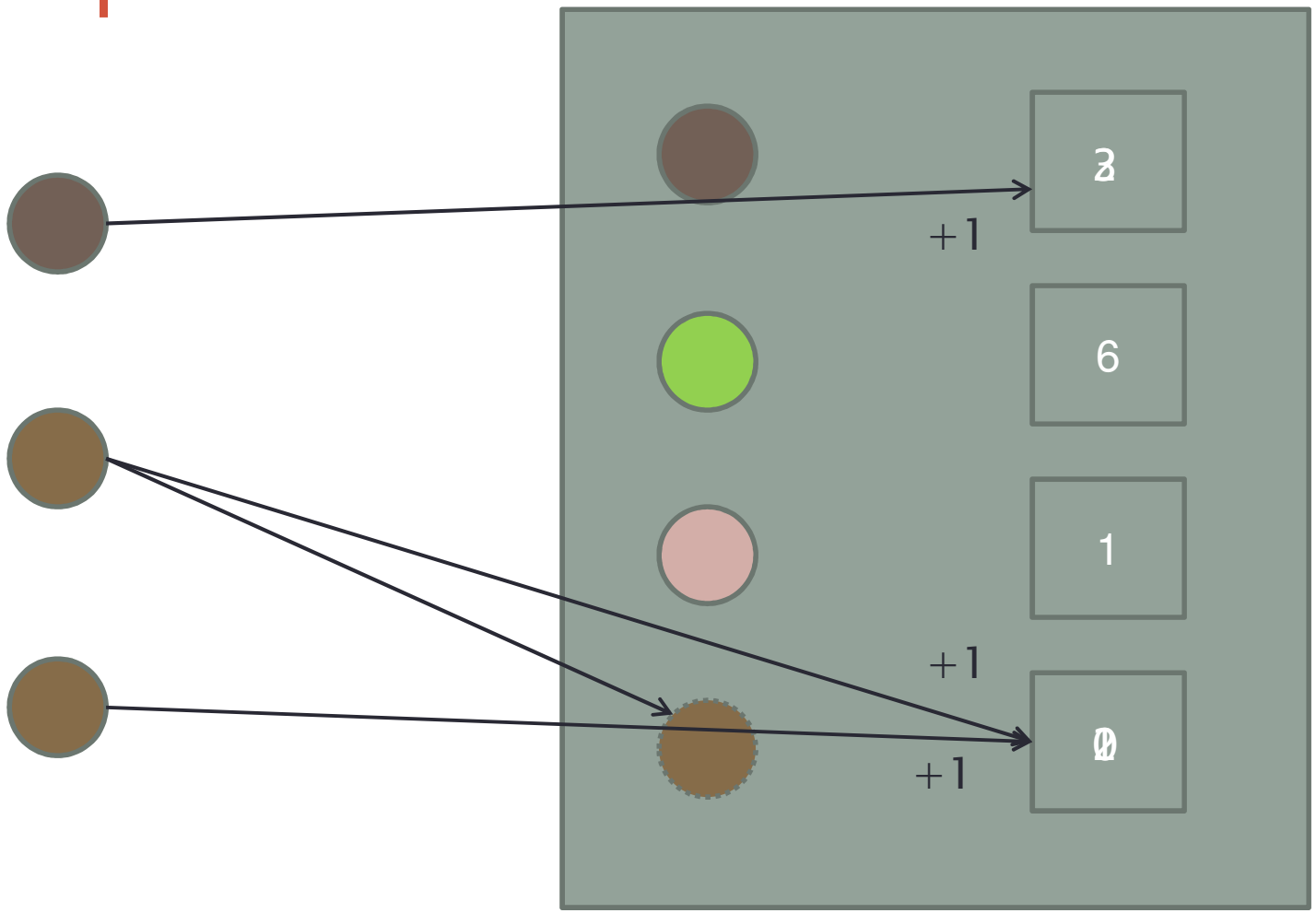
The Frequent Algorithm

- First proposed by Misra and Gries in 1982.
- Finds all items in a sequence whose frequency exceeds a 1/k fraction of the total count.
- Stores k-1 (item, counter) pairs.
- A generalization of the Majority algorithm.

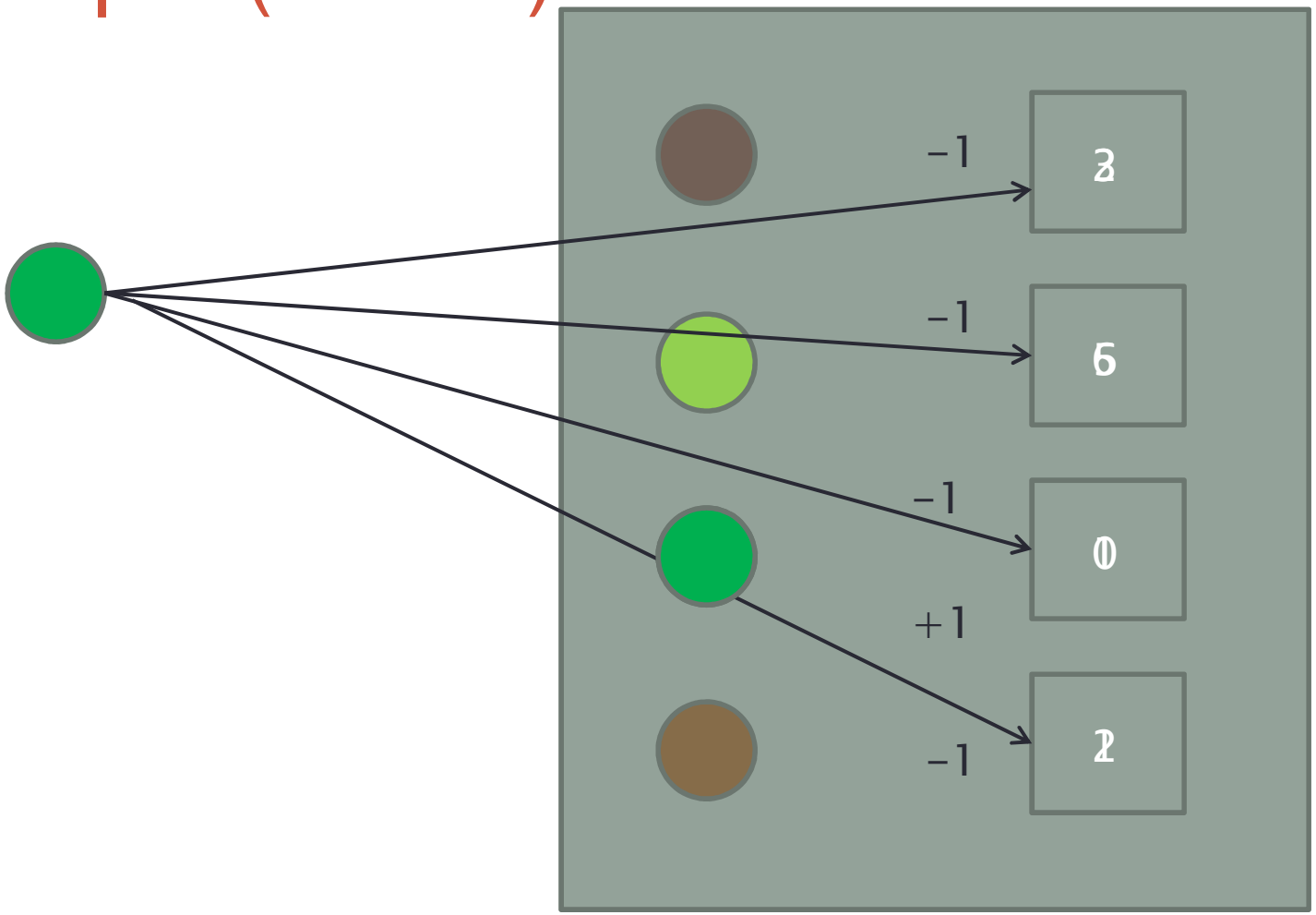
The Frequent Algorithm (contd.)

- For each new item:
  - Increment the counter if the item is already stored.
  - If <k items stored, then store new item with counter set to 1.
  - Otherwise decrement all the counters.
  - If any counter equals 0, then delete the corresponding item.

Example



Example (contd.)



Pseudo Code

Algorithm 1: FREQUENT(k)

```
n ← 0;
T ← ∅;
foreach i do
    n ← n + 1;
    if i ∈ T then
        c_i ← c_i + 1;
    else if |T| < k–1 then
        T ← T ∪ {i};
        c_i ← 1;
    else forall j ∈ T do
        c_j ← c_j – 1;
        if c_j = 0 then T ← T \ {j};
```

Analysis

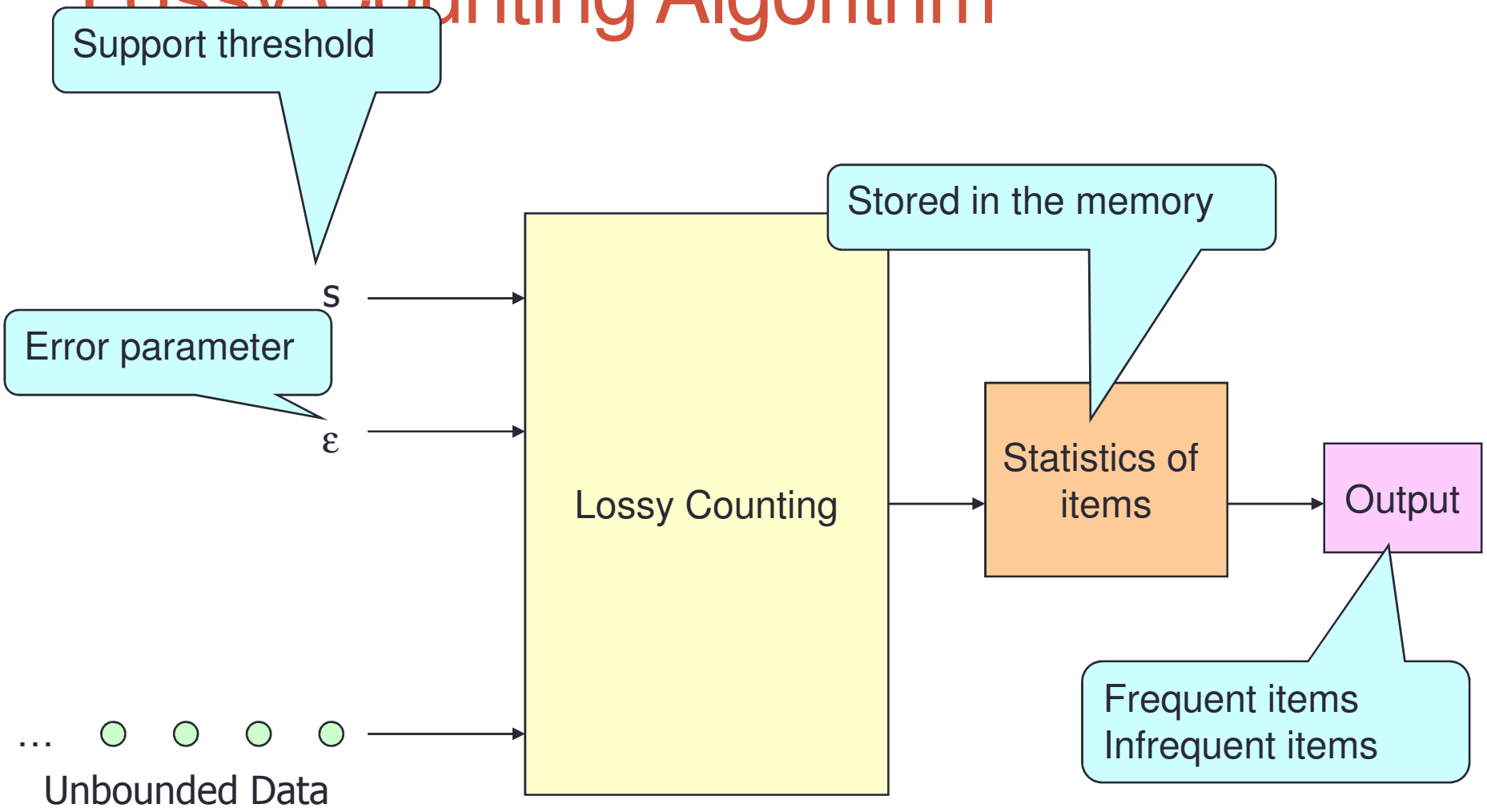
- Time cost involves:
  - O(1) dictionary operations per update.
  - Cost of decrementing counts.
    - Can be performed in O(1) time.

- Also solves the frequency estimation problem if executed with $k=1/\varepsilon$.

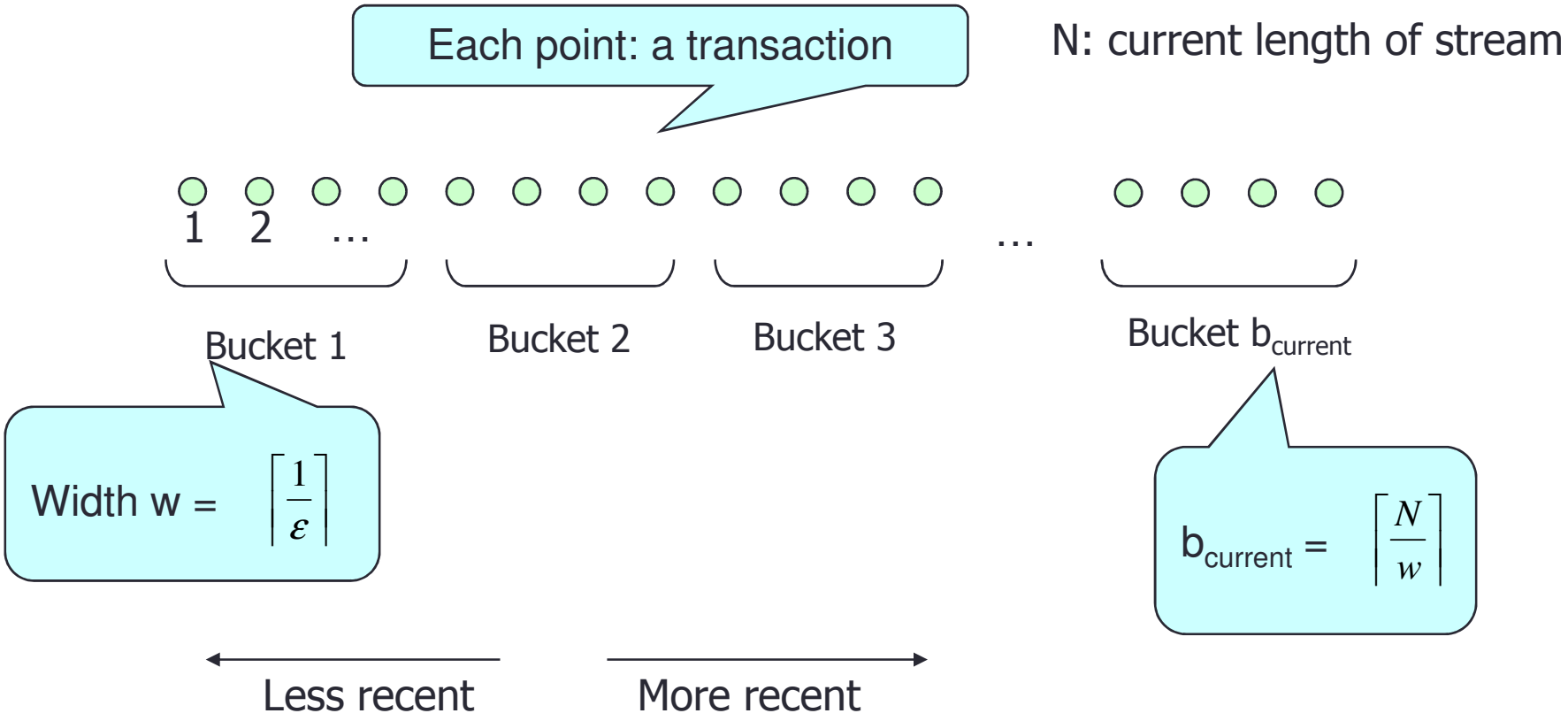
Lossy Counting Algorithm

- Proposed by Manku and Motwani in 2002.
- Uses space $1/\varepsilon \log(\varepsilon N)$.

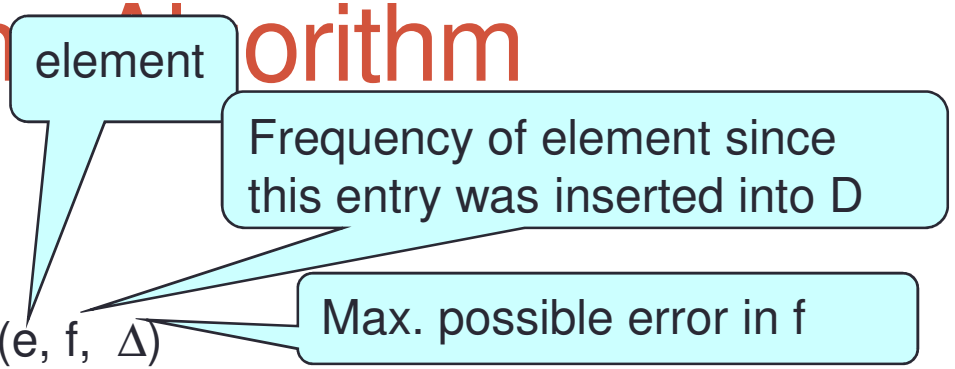
Lossy Counting Algorithm



Lossy Counting Algorithm



Lossy Counting Algorithm



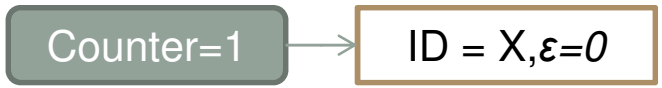
1. D: Empty set
   - Will contain (e, f, Δ)
2. When data e arrives,
   - If e exists in D,
     - Increment f in (e, f, Δ)
   - If e does not exist in D,
     - Add entry (e, 1, $b_{current}$-1)
3. Remove some entries in D whenever
   $N \equiv 0 \bmod w$
   (i.e., whenever it reaches the bucket boundary)
   The rule of deletion is:
   (e, f, Δ) is deleted if
   $f + \Delta <= b_{current}$
4. **[Output]** Get a list of items where
   $f + \varepsilon N >= sN$

SpaceSaving Algorithm

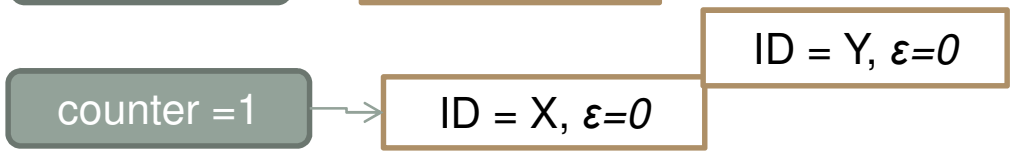
- Introduced by Metwally et al. in 2005.
- Store k (item, count) pairs.
- Initialize by first k distinct items and their exact counts.
- If new item is not already stored, replace the item with least count and set the counter to 1 more than the least count.
- Items which are stored by the algorithm early in the stream and are not removed have very accurate estimated counts.

Example

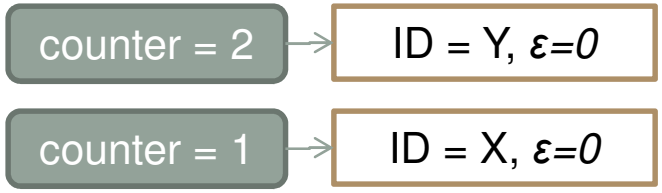
- Assuming $m = 2$, and $A=\{X, Y, Z\}$, Stream $s = \{X, Y, Y, Z\}$.
- Step 1: (e=X)



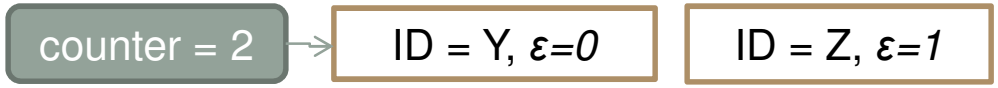
- Step 2: (e=Y)



- Step 3: (e=Y)



- Step 4: (e=Z)



Pseudo Code

Algorithm 3: SpaceSaving(k)

$n \leftarrow 0$;
$T \leftarrow \emptyset$;
foreach i **do**
    $n \leftarrow n + 1$;
    if $i \in T$ **then** $c_i \leftarrow c_i + 1$;
    else if $|T| < k$ **then**
        $T \leftarrow T \cup \{i\}$;
        $c_i \leftarrow 1$;
    else
        $j \leftarrow \arg\min_{j \in T} c_j$;
        $c_i \leftarrow c_j + 1$;
        $T \leftarrow T \cup \{i\} \setminus \{j\}$;

Analysis

- The space required is O(k).
- The time cost involves cost of:
  - the dictionary operation of finding if an item is stored.
  - finding and maintaining the item with minimum count.
- Simple heap implementations can track the smallest count item in O(log k) time per update.
- As in other counter based algorithms, it also solves frequency estimation problem with $k=1/\varepsilon$.

Counter Algorithms Summary

- Counter algorithms very efficient for arrivals-only case
  - Use O(1/ε) space, guarantee εN accuracy
  - Very fast in practice (many millions of updates per second)
- Similar algorithms, but a surprisingly clear "winner"
  - Over many data sets, parameter settings, SpaceSaving algorithm gives appreciably better results

Sketch Algorithms

- Sketch refers to a class of algorithm that represents a large data set with a compact summary, typically much smaller than the full size of the input.

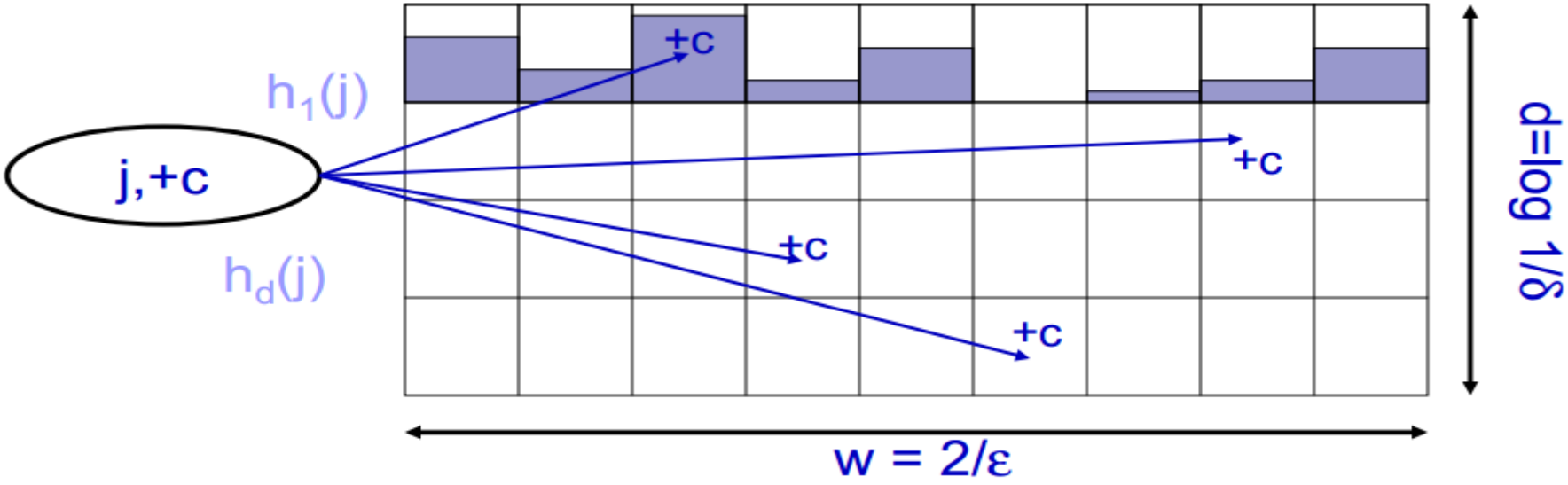
- Given an input of N items (x_1;x_2 : : :x_N), each item x (where x is drawn from some domain U) is mapped via hash functions into a small sketch vector that records frequency information.

- Thus, the sketch does not store the items explicitly, but only information about the frequency distribution. Sketches support fundamental queries on their input such as point, range and inner product queries to be quickly answered approximately.

Count Min Sketch

- The CM sketch consists of a two-dimensional t × k array of counters, plus t independent hash functions, each chosen uniformly at random from a 2-universal family.
- For each input element, we read a token, resulting in an update of f_j, we update certain counters in the array based on the hash values of j (the “count” portion of the sketch).
- When we wish to compute an estimate fj , we report the minimum of these counters.
- The values of t and k are set, based on ε and δ, as shown below

Count Min Sketch Structure



- Each entry in vector x is mapped to one bucket per row.
- Estimate x[j] by taking mink CM[k,hk(j)]
  - Guarantees error less than ε||x||1 in size O(1/ε log 1/δ)
  - Probability of more error is less than 1-δ

What is the trick?

- Since all the elements shall definitely hash to some value, and the co-domain of the mapping is lesser that the domain, there shall be collision.
- But, the resulted collisions shall stay within some bounds based on the hash-function.
- We always return the minimum value for the elements hashed values. This is because we want the estimate for the least collisions.
- The intuition is that the collisions shall only result in a bounded limitation on the quality of the results.

Simple intuition for Graph Algorithms

- Given n vertices {1,2, ...,n}, stream = $<(u_1, v_1), (u_2, v_2), ..., (u_m, v_m)>$, with each $(u_i, v_i) \in [n]$

- Graph streams are of the ”semi-streaming model” :: O(n) space for computation.

Connectedness Problem

Initialize : F ← φ, where F is a set of edges;

Process (u, v):

1 if F ∪ {(u, v)} '*does not contain a cycle'* then

2 F ← {(u, v)} ∪ F ;

Output : if |F| (*The number of edges*) = n – 1 then

Yes

else

No

Union-Find data structure.

References

[1] B. Boyer and J. Moore. A fast majority vote algorithm. Technical Report ICSCA-CMP-32, Institute for Computer Science, University of Texas, Feb. 1981.

[2] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In ACM PODS, 2004.

[3] G. Manku and R. Motwani. Approximate frequency counts over data streams. In International Conference on Very Large Data Bases, pages 346–357, 2002.

[4] Amit Chakrabarti ,Dartmouth CollegeCS85: Data Stream Algorithms Lecture Notes, Fall 2009

[5] PHD COURSE ON STREAMING ALGORITHMS (http://cs.au.dk/~gerth/stream11/)

References *contd…*

[6] Data Stream Algorithms by S. Muthu Muthukrishnan (http://www.cs.mcgill.ca/~denis/notes09.pdf)

[7] Finding Frequent Items in Data Streams by Moses Charikar, Kevin Chen, and Martin Farach-Colton3

[8] Sketch Algorithms for Estimating Point Queries in NLP by Amit Goyal and Hal Daume III

[9] Data Stream Algorithms, Lecture Notes, Fall 2009 by