Algorithms for data streams: Streaming models, a problem on graph streams

Maria Serna

Spring 2023

1. Data stream models
2. Graph streams
3. Sampling
4. Sliding window

The data

- Data arrives as sequence of items.
- Sometimes continuously and at high speed.
- Can't store them all in main memory.
- Can't read again; or reading again has a cost.
- We abstract the data to a particular feature, the data field of interest the label.

The data

- We have a set of n labels Σ and our input is a stream $s = x_1, x_2, x_3, \ldots x_m$, where each $x_i \in \Sigma$.
- Take into account that some times we do not know in advance the length of the stream.
- Goal Compute a function of stream, e.g., median, number of distinct elements, longest increasing sequence.

Why has it become popular?

- Practical appeal:
  - Faster networks, cheaper data storage, ubiquitous data-logging results in massive amount of data to be processed.
  - Applications to network monitoring, query planning, I/O efficiency for massive data, sensor networks aggregation, . . .
- Theoretical Appeal:
  - Easy to state problems but hard to solve.
  - Links to communication complexity, compressed sensing, embeddings, pseudo-random generators, approximation, parallel computation, . . .
- Origins in 70's but has become popular in this century because of growing theory and very applicable.

The computational data stream models

- Classical streaming model
  - The data stream is accessed sequentially.
  - The processing is done sequentially using a small working memory $O(\text{polylog } n)$.
  - Measures of complexity: number of passes over the data, the size of the working memory, the per-item processing time.
- Semi-streaming model
  - Usual for graph problems.
  - Working memory is $O(n \text{ polylog } n)$, for a graph with n vertices.
  - Enough space to store vertices but not for storing all the edges.
- Streaming with sorting
  - Allows the creation of intermediate streams.
  - Streams can be sorted at no cost.
  - Algorithms run in phases reading and creating a stream

Algorithmic goals

- Data streams are potentially of unbounded size.
- As the amount of computation and memory is limited it might be impossible to provide exact answers.
- Algorithms use randomization and seek for an approximate answer.
- Typical approach:
    - Build up a synopsis data structure
    - It should be enough to compute answers with a high confidence level.

1. Data stream models
2. Graph streams
3. Sampling
4. Sliding window

Streams that describe graphs

- G undirected
- on $[n]$ vertices
- the stream describes the edges of G
- we assume that an edge appears only once in the stream
- We want to keep a DS allowing to answer queries about a graph property
- $O(n \log n)$ memory is reasonable as we are working on the semi-streaming model.

Connectedness

- Problem: Decide whether or not the input graph G, given by a stream, is connected.
- Algorithm:
  - Maintain a spanning forest H of the seen graph
  - On query answer according to H
- G connected iff admits a spanning tree, the algorithm is correct
- 1 pass, $O(n \log n)$ memory, using a union find DS amortized $O(\alpha(n))$ per item

Estimating shortest paths

- Problem: Estimate the distance in G, given by a stream, between two given vertices.
- The algorithm builds an intermediate graph H to support queries and uses a parameter t.
- Algorithm:

```
procedure EDistances(int $n$, stream $s$, double $t$)
    $H = ([n], \emptyset)$
    while not $s.end()$ do
        $(u, v) = s.read()$
        if $d_H(u, v) \geq t + 1$ then
            add $(u, v)$ to $H$
            update distances
    On query $(u, v)$, report $d_H(u, v)$
```

Estimating shortest paths: Analysis

- 1 pass, $O(|H|\log n)$ space, $O(n)$ per edge added to H.
- $|H|$?
- Quality?

Estimating shortest paths: Analysis

- A subgraph H is a t-spanner for G iff, for all $u, v \in V$ $(V = V(G) = V(H))$,

$$d_G(u,v) \leq d_H(u,v) \leq t\, d_G(u,v)$$

- Let H be the graph computed by $\textsc{EDistances}$ on (G, t).
  H is a t-spanner.
  - H is a subgraph of G, so $d_G(u,v) \leq d_H(u,v)$, for $u, v \in V$.
  - When $d_G(u,v) = \infty$ it also holds $d_H(u,v) = \infty$
  - Let $x = v_0, v_1, \dots, v_k = y$ be a shortest path from x to y in G.
  - Pick $i \in [k]$ and let $e = (v_{i-1}, v_i)$.
    If $e \in H$, $d_H(v_{i-1}, v_i)) = 1$, otherwise, as e was not added to H, we known that in the current H' when e was read $d_{H'}(v_{i-1}, v_i) \leq t$.
  - So, $d_H(x,y) \leq t\, d_G(x,y)$ as any such H' is a subgraph of H.
- Then, $\textsc{EDistances}$ is a t-approximation

Estimating shortest paths: Analysis

The girth $\gamma(G)$ is the length of the shortest cycle in G, ∞ for acyclic G.

Alon, Hoory, Linial 2002

Let n be sufficiently large. Let G be a graph with n vertices and m edges having $\gamma(G) \geq k$, for some integer k. Then

$$m \leq n + n^{1+\lfloor \frac{k-1}{2} \rfloor}$$

Estimating shortest paths: Analysis

- When $\gamma(G) \geq k$, $m = O(n^{1+2/(k-2)})$.
- By construction the shortest cycle in the subgraph H constructed by EDISTANCES has length $t + 2$.
- Therefore, the memory used is $O(n^{1+2/t} \log n)$.
- We can 3-approximate distances using $O(n^{5/3} \log n)$ memory.

Maximum matching

- Problem: Find a maximum matching in G, given by a stream.
- The algorithm maintains a matching M.
- Algorithm:

```
procedure MMATCHING(int $n$, stream $s$, double $t$)
    $M = \emptyset$
    while not $s.end()$ do
        $(u, v) = s.read()$
        if $M \cup (u, v)$ is a matching then
            $M = M \cup \{(u, v)\}$
    On query, report $M$
```

Maximum matching: Analysis

- 1 pass, $O(n \log n)$ space, $O(1)$ ops. per item
- M is a maximal matching that provides an estimation $\hat{f}$ of the size f of a maximum matching.
- $\hat{f}$ is a 2 approximation to f.

1. Data stream models
2. Graph streams
3. Sampling
4. Sliding window

Sampling

- Sampling is a general technique for tackling massive amounts of data.
- Example: To compute the median packet size of some IP packets, we could just sample some and use the median of the sample as an estimate for the true median. Statistical arguments relate the size of the sample to the accuracy of the estimate.
- Challenge: But how do you take a sample from a stream of unknown length or from a sliding window?

Reservoir Sampling (Vitter 1985)

- Problem: Maintain a uniform sample x from a stream s of unknown length.
  The selected item should be any of the seen ones with uniform probability.
- Algorithm:
  - Initially $x = x_1$
  - On seeing the t-th element, $x = x_t$ with probability $1/t$
- Analysis:
  - 1 pass, $O(\log n)$ memory (in bits), and $O(1)$ time (in operations) per item.
  - Quality?
    What is the probability that $x = x_i$ at some time $t \geq i$?

Reservoir Sampling: Quality

- At any time step t, for $i \leq t$, $Pr[x = x_i] = 1/t$
- The proof is by induction on t.
  - Base $t = 1$: $Pr[x = x_1] = 1$.
  - Induction hypothesis: true for time steps up to $t - 1$
    - $Pr[x = x_t] = 1/t$
    - For $i < t$, $x = x_i$ only when x_t is not selected and x_i was the sampled element at step $t - 1$. By induction hypothesis we have

$$Pr[x = x_t] = \left(1 - \frac{1}{t}\right)\frac{1}{t-1} = \frac{1}{t}$$

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:
  - Initially $X = \{x_1, \dots, x_k\}$.
  - On seeing the t-th element, $t > k$, select x_t to be added to X with probability k/t.
  - If x_t is selected to be added, select uniformly at random an element from X remove it and add x_t.
- Analysis:
  - 1 pass, $O(k \log n)$ memory, and $O(1)$ time per item.
  - Quality?

    What is the probability that $x_i \in X$ at some time $t \geq i$?

Reservoir Sampling II: Quality

- At any time step t, for $i \leq t$, $Pr[x_i \in X] = k/t$
- The proof is by induction on t.
  - Base $t = k$: $Pr[x_i \in X] = 1$, for $i = 1, \ldots, k$.
  - Induction hypothesis: true for time steps up to $t - 1$
    - $Pr[x_t \in X] = k/t$
    - For $i < t$, $x_i \in X$ when x_t is not selected and x_i was in the sample at step $t - 1$, or when x_t is selected, x_i was in the sample at step $t - 1$ and x_i is not evicted.

$$\begin{aligned} Pr[x_i \in X] &= \left(1 - \frac{k}{t}\right) \frac{k}{t-1} + \frac{k}{t} \frac{k}{t-1} \left(1 - \frac{1}{k}\right) \\ &= \frac{k}{t-1} - \frac{k}{t} \frac{k}{t-1} \frac{1}{k} = \frac{k}{t-1} - \frac{1}{t} \frac{k}{t-1} = \frac{k}{t} \end{aligned}$$

1 Data stream models

2 Graph streams

3 Sampling

4 Sliding window

Sliding Windows: Replace-Sampling algorithm

- Problem: Maintain a uniform sample x from the last seen w elements.
- Algorithm:
  - Maintain a reservoir sample for the first w items in s.
  - When the arrival of an item causes an element in the sample to expire, replace it with the new arrival.
- Analysis
  - 1 pass, $O(k \log n)$ space and $O(1)$ time per item.
- Trouble: We keep an element but it might not follow the uniform distribution.

Reservoir Sampling for Sliding Windows

- Problem: Maintain a uniform sample of k items from the last w items.
- Why reservoir sampling does not work?
  - Suppose an element in the reservoir expires
  - Need to replace it with a randomly-chosen element from the current window
  - But, we have no access to past data!
  - Could store the entire window but this would require $O(w)$ memory.

Sliding Windows: Backing-Sample algorithm

- Problem: Maintain a uniform sample of size k from the last seen w elements.
- Algorithm idea:
    - Maintain a backing sample B:
        - Add x_t to B with probability p
        - Remove from B all the elements that expire at time t.
    - The sample X of size k is obtained by a uniform sampling of k items from B.
- Analysis
    - 1 pass, $O((k + |B|) \log n)$ space and $O(k)$ time per item.
    - $|B|$? Should be small compared to w.
    - Quality? The algorithm might fail if $|B| < k$ at some step.

Concentration inequalities

- Usually we measure the performance of a randomized algorithm/process/experiment by computing the average.
- It is not always true that by sampling, we will get close to the expected values.
  If you draw a fair coin, with values 1 or -1 on each side, and you get as a reward the written value. The expected gain is 0. However, there is no draw that provides gain 0.
- In probability there are collections of inequalities providing bounds of the probability of being close to the mean under different hypothesis.
- We survey some such useful bounds.

Markov's, Chebyshev's and Chernoff's bounds

Markov's inequality

When X is a non-negative random variable and $a > 0$,

$$Pr(X > a) \leq \frac{E[X]}{a}$$

Chebyshev's inequality

Let X be a random variable with finite expected value μ and finite non-zero variance σ^2. Then for any real number $t > 0$,

$$Pr(|X - \mu| \geq t\sigma) \leq \frac{1}{t^2}.$$

Chernoff bounds

When applying Markov's inequality to e^{cX}, we get

Chernoff bound

For $c > 0$,

$$Pr(X > a) = Pr(e^{cX} \geq e^{ca}) \leq \frac{E[e^{cX}]}{e^{ca}}$$

Chernoff bounds

When $X = X_1 + \cdots + X_n$ and we have additional information on X_i we can get refined Chernoff bounds.

Chernoff bounds

Let $X_1, \dots, X_n$ be independent random variables corresponding to Bernoulli experiments, each variable with probability p_i of success. Let $X = X_1 + \cdots X_n$ and let $\mu = E[X] = \sum_{i=1}^{n} p_i$. Then, for any $d > 0$,

$$\Pr[X > (1+d)\mu] \leq \left(\frac{e^d}{(1+d)^{1+d}} \right)^{\mu}.$$

For any $0 < d < 1$,

$$\Pr[X < (1-d)\mu] \leq \left(\frac{e^{-d}}{(1-d)^{1-d}} \right)^{\mu}.$$

Simplified Chernoff bounds

A less accurate (but quite useful) version of Chernoff bounds is the following.

Chernoff bounds

Let $X_1, \ldots, X_n$ be independent random variables corresponding to Bernoulli experiments, each variable with probability p_i of success. Let $X = X_1 + \cdots X_n$ and let $\mu = E[X] = \sum_{i=1}^{n} p_i$. Then, for any $d \in (0,1)$,

$$\Pr[X < (1-d)\mu] \leq e^{\frac{-d^2\mu}{2}} \text{ and } \Pr[X > (1+d)\mu] \leq e^{\frac{-d^2\mu}{3}}.$$

Sliding Windows: Backing-Sample algorithm

- Problem: Maintain a uniform sample of size k from the last seen w elements.
- Algorithm:
  - Maintain a backing sample B:
    - Add x_t to B with probability $p = 2ck \log w / w$
    - Remove from B all the elements that expire at time t.
  - The sample X of size k is obtained by a uniform sampling of k items from B.
- Analysis
  - 1 pass, $O((k + |B|) \log n)$ space and $O(k)$ time per item.
  - $|B|$? Should be small compared to w.
  - Quality? The algorithm might fail if $|B| < k$ at some step.

Sliding Windows: Backing-Sample size

- Using Chernoff bounds, the size of the backing sample is between k and $4ck \log w$ with probability $1 - w^{-c}$.
- Selecting the adequate c, with high probability the algorithm succeeds in keeping a large enough backing sample.
- The bound on the space is $O(k \log w)$ with high probability.

Chain-Sampling for Sliding Windows

- Algorithm: $(k = 1)$
  - Maintain a reservoir sample for the first w items in s, but
  - whenever an element x_i is selected, choose and index $j \in [w]$ uniformly at random, x_{i+j} will be the replacement for x_i.
  - For $t > w$, when $t = i + j$, set $x = x_{i+j}$ (and choose the next replacement).
- Analysis
  - 1 pass, $O(\log n + \log w)$ space and $O(1)$ time per item.
  - Provides a uniform sample.
- For higher values of k, run k parallel chain samples.
  With high probability, for large enough w, such chains will not intersect.

Chain-Sampling: number of updates

- $k = 1$
- The algorithm perform changes only on the chain of selected items.
- The number of possible chains of elements with more than x data elements is bounded by the number of partitions of m into x ordered integer parts, which is bounded by $\binom{m}{x}$.
- Each such chain has probability at most m^{-x}.
- The probability of updating x steps is therefore at most $\binom{m}{x} m^{-x}$.
- Using Stirling's approximation we get the bound $\left(\frac{e}{x}\right)^x$.
- For $x = O(\log m)$ this is less than m^{-c}, for some constant c
- With high probability the number of updates is $O(\log m)$.