Algorithms for data streams: Streaming models, graph streams

The data

The data

- Data arrives as sequence of items.
- Sometimes continuously and at high speed.

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.

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

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 $x_i \in \Sigma$.

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.

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:

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, ...

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, ...

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 computational data stream models

- Classical streaming model
  - The data stream is accessed sequentially.
  - The processing is done sequentially using a small working memory $O(\text{polylog } n)$.

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.

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

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.

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.

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.

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.

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

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 that allows 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.

Connectedness

- Problem: Decide whether or not the input graph G, given by a stream, is connected.
- Algorithm:

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

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,

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

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,

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

Maximum matching

- Problem: Find a maximum matching in G, given by a stream.

Maximum matching

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

Maximum matching

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

Maximum matching

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

1: **procedure** MMatching(int n, stream s, double t)
2: $M = \emptyset$
3: **while** not $s.end()$ **do**
4: $(u, v) = s.read()$
5: **if** $M \cup (u, v)$ is a matching **then**
6: $M = M \cup \{(u, v)\}$
7: On query, report M

Maximum matching: Analysis

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.
  Because, at least one vertex of each edge of M must be matched by an edge in a maximum matching.

Sampling

Sampling

- Sampling is a general technique for tackling massive amounts of data.

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.

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)

Reservoir Sampling (Vitter 1985)

- Problem: Maintain a uniform sample x from a stream s of unknown length.

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.

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:

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 $1/t$

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:

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.

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 $i \leq t$, $Pr[x = x_i] = 1/t$

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.

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.

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:
  - Initially $X = \{x_1, \ldots, x_k\}$.
  - 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.

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:
  - Initially $X = \{x_1, \ldots, x_k\}$.
  - 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:

Reservoir Sampling II

- Problem: Maintain a uniform sample X of size k from a stream of unknown length.
- Algorithm:
  - Initially $X = \{x_1, \ldots, x_k\}$.
  - 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.

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 $i \leq t$, $Pr[x_i \in X] = k/t$

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.

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.

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

Reservoir Sampling for Sliding Windows

- Problem: Maintain a uniform sample of k items from the last w items.

Reservoir Sampling for Sliding Windows

- Problem: Maintain a uniform sample of k items from the last w items.
- Why reservoir sampling does not work?

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!

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.

Chain-Sampling for Sliding Windows

Chain-Sampling for Sliding Windows

- Algorithm: $(k = 1)$

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).

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.

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.

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: total time

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along the chain of selected items.

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along 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}.

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along 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 $\left(\frac{e}{x}\right)^x$.

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along 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

Chain-Sampling: total time

- $k = 1$
- The algorithm perform changes only along 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 $O(\log m)$.