COMP 480/580 — Probabilistic Algorithms and Data Structure Sept 18, 2025

Lecture 7

Lecturer: Anshumali Shrivastava

Scribe By: Seyaul Kim(esk8), Iris Xue(yx66), Grace Yu(gy20)

1 Introduction

1.1 Data Streams

In modern computing, data often arrives continuously in the form of *streams* rather than as static datasets. These streams are generated by a wide range of sources at very high speeds. For instance, Google search queries, live updates on social media, financial market transactions, or internet traffic. Unlike traditional datasets, the complete size of such streams is unknown, and it is convenient to think of them as effectively infinite. The challenge is to perform meaningful analysis and critical calculations over these streams while working with very limited memory.

1.2 Applications

There are many practical applications of stream analysis. For instance, in *mining query streams*, a search engine like Google may want to quickly identify which queries are more frequent today compared to yesterday. Online platforms monitor *clickstreams* to detect sudden spikes in page visits. Social networks also rely on this type of analysis to detect and highlight trending topics by monitoring the live flow of user activity.

Beyond web-based services, stream analysis is equally critical in other domains. *Sensor networks*, for example, often consist of many sensors continuously sending data to a central controller, requiring efficient methods to process this flood of input. In telecommunications, *telephone call records* are generated as streams of data; these feeds are essential not only for generating customer bills but also for settling transactions between different companies. Likewise, monitoring *IP packets* at a network switch allows operators to gather routing information for improved performance and to detect malicious activity such as denial-of-service attacks.

1.3 Motivation

The motivation for studying streaming algorithms comes from the fact that it is infeasible to store or process the entirety of these massive or infinite data streams. Still, it is often necessary to estimate statistics (e.g., mean, median), build models (e.g., classifiers), or detect anomalies and trends as they occur. A common solution is to use *random sampling*, which provides a compact yet unbiased summary of the full dataset. With well-designed sampling, we can approximate global properties of the stream using only a small subset of items.

1.4 Sampling

Random sampling is a powerful and general tool in data analysis. The general idea is to pick a small random subset S from a much larger set of size m, and then estimate the quantity of interest using only S. The effectiveness of this approach depends on the sampling strategy, the

sample size, and the estimation algorithm employed.

A basic sampling strategy is to draw a uniform sample of size k from a set of size m:

- **With replacement:** pick a uniformly random index $i \in \{1, 2, \ldots, m\}$ each time.
- **Without replacement:** pick a single subset uniformly at random from all subsets of size k, of which there are $\binom{m}{k}$.

1.5 Streaming Model

In the *streaming model*, items arrive sequentially as a stream

$$e_1, e_2, e_3, \ldots$$

with the total number of items unknown in advance. Let m denote the number of items seen so far. The algorithm is constrained to very limited memory, typically enough to store only $k \ll m$ items. Despite this restriction, the objective is to compute useful statistics or summaries over the input data.

This leads to a fundamental question: *how can we sample uniformly from a data stream of unknown length, without storing the entire stream?* Ideally, every item should have equal probability of being selected, yet this must be achieved without prior knowledge of the total stream length.

2 Reservoir Sampling for size $k = 1$

2.1 Problem Description

The problem we would like to solve is to select a single, uniform random sample from a stream of data without knowing the length of the stream in advance. This question arises naturally in the context of streaming algorithms, where data arrives sequentially and storing the entire stream is either impossible or highly inefficient.

To describe the problem more in detail, we want every element in the stream to have an equal probability of being selected as the sample. Importantly, this should be achieved without storing the entire stream in memory, since the size of the stream may be very large or even unbounded.

The difficulty lies in the fact that the total length of the stream is not known beforehand. Without knowing how many elements will appear, it is unclear how to adjust the selection probabilities dynamically to maintain uniformity across all items.

2.2 Algorithm

The main approach to the problem of uniform sampling from a stream of unknown length is called *Reservoir Sampling*.

Here, we consider the case of selecting a single sample, i.e., $k = 1$.

The algorithm works as follows:

Algorithm 1 Uniform Sample from a Stream ($k = 1$)

```
s ← ∅ ;                                              // initially no sample
m ← 0 ;                                              // number of items processed
while stream is not finished do
    m ← m + 1
    e_m ← current item
    Toss a biased coin with probability P(H) = 1/m
    if coin shows H then
        s ← e_m ;                                    // replace current sample
return s
```

Key Idea. At each step m, the new element e_m has probability $\frac{1}{m}$ of replacing the previously stored sample. This ensures that after processing m elements, every one of them is equally likely $(1/m)$ to be chosen, without requiring knowledge of the total length of the stream in advance.

2.3 Proof

Let m be the number of items in the stream: $e_1, e_2, \ldots, e_m$. Let s denote the final output of the reservoir sampling algorithm with $k = 1$. Then the probability that s equals any particular element e_j is uniform:

$$\Pr(s = e_j) = \frac{1}{m}, \quad \text{for all } j = 1, 2, \ldots, m.$$

Proof.

1. **Base case ($m = 1$).** When there is only one element e_1, the algorithm selects it with probability 1. Thus the claim holds.

2. **Induction hypothesis.** Suppose the statement is true for $m - 1$ items, i.e.,

$$\Pr(s = e_j) = \frac{1}{m-1}, \quad \text{for each } j = 1, 2, \ldots, m-1.$$

3. **Inductive step.** Consider the m-th element e_m:

   - With probability $\frac{1}{m}$, e_m replaces the current sample.
   - With probability $1 - \frac{1}{m}$, the current sample remains unchanged.

   - For $j < m$:

$$\Pr(s = e_j) = \Pr(s = e_j \text{ before step } m) \cdot \Pr(\text{not replaced at step } m).$$

   By induction, $\Pr(s = e_j \text{ before}) = \frac{1}{m-1}$, and the probability of not being replaced is $\left(1 - \frac{1}{m}\right)$. Hence,

$$\Pr(s = e_j) = \frac{1}{m-1} \cdot \left(1 - \frac{1}{m}\right) = \frac{1}{m}.$$

- For $j = m$:

$$\Pr(s = e_m) = \Pr(\text{replaced at step } m) = \frac{1}{m}.$$

Therefore, by induction, each element e_j has probability $\frac{1}{m}$ of being chosen as the sample.

3 Reservoir Sampling with Size $k > 1$

So far we considered the case $k = 1$. We now generalize the algorithm to sample k items uniformly at random from a stream of unknown length, *without replacement*.

3.1 Problem Description

Given a stream of items $e_1, e_2, \ldots, e_m$ of unknown length m, we want to maintain a reservoir S of size k such that at the end of the process, every subset of k items is equally likely to appear in S.

Formally:

$$\Pr(S = T) = \frac{1}{\binom{m}{k}}, \quad \text{for all subsets } T \subseteq \{e_1, \ldots, e_m\},\ |T| = k.$$

3.2 Algorithm

Algorithm 2 Reservoir Sampling for $k > 1$

```
S[1..k] ← ∅ ;                                          // initialize reservoir
m ← 0
while stream is not finished do
    m ← m + 1
    e_m ← current item
    if m < k then
        S[m] ← e_m ;                                   // fill initial reservoir
    else
        Generate random integer r ~ Uniform{1, 2, ..., m}
        if r ≤ k then
            S[r] ← e_m ;                               // replace a random element in reservoir
```

3.3 Proof

We want to show that after processing m items, each item e_j has probability

$$\Pr(e_j \in S) = \frac{k}{m}, \quad \text{for all } 1 \le j \le m.$$

Proof (by induction on m).

1. **Base case ($m = k$).** When the stream contains exactly k items, the reservoir is filled with all k of them. Thus, each element is included with probability

$$\frac{k}{k} = 1,$$

which matches the claim.

2. **Induction hypothesis.** Suppose after processing m items ($m \geq k$), each element is in the reservoir with probability

$$\Pr(e_j \in S) = \frac{k}{m}, \quad \text{for all } j = 1, \ldots, m.$$

3. **Step $m+1$.** Now process the $(m+1)$-th item e_{m+1}:

   - With probability $\frac{k}{m+1}$, random integer $r \leq k$, the new item is chosen. In that case, e_{m+1} enters the reservoir.
   - With probability $1 - \frac{k}{m+1}$, the reservoir remains unchanged.
   - For $j = m+1$:
   $$\Pr(e_{m+1} \in S) = \frac{k}{m+1}.$$
   - For $j \leq m$: The probability that e_j is still in the reservoir is the probability it was in the reservoir after m steps, times the probability it was not replaced at step $m+1$:
   $$\Pr(e_j \in S \text{ after } m+1) = \frac{k}{m} \cdot \left(1 - \frac{1}{m+1}\right).$$
   Simplifying:
   $$\frac{k}{m} \cdot \frac{m}{m+1} = \frac{k}{m+1}.$$

Thus, by induction, each item e_j has probability $\frac{k}{m}$ of being in the reservoir after m steps, completing the proof.

4 Weighted Reservoir Sampling (WRS)

4.1 Description of Problem:

Weighted reservoir sampling differs from standard reservoir sampling by offering the ability to give certain items more weight to be chosen. It preserves the mathematical guarantees of reservoir sampling (one-pass and low memory usage) while respecting the weight distribution, meaning each item's probability of selection is proportional to its weight relative to the total weight of all items processed so far.

4.2 Practical Application Examples:

1. When you need to ensure certain groups are represented proportionally

2. When recent items might have higher importance than older ones

3. When you have known biases in your data stream, you can attach weights to counteract them.

4.3 Key Assumptions for Algorithm:

1. The data stream has n items, where n is unknown or very large.
2. For item i, where i ∈ {1,2,...,n}, i has a weight w_i where $w_i > 0$.
3. You are selecting a random sample of k items, where $k < n$.

With the above assumptions, the probability of selecting an item i should be proportional to its weight w_i.

4.4 Efraimidis-Spirakis Algorithm:

Algorithm 3 Efraimidis-Spirakis Algorithm

$R \leftarrow \emptyset$; `// Initialize empty reservoir`
$i \leftarrow 0$; `// Item counter`
while *stream is not finished* **do**
  $i \leftarrow i + 1$
  $w_i \leftarrow$ weight of current item
  Generate random number $u_i \sim \text{Uniform}(0, 1)$
  $k_i \leftarrow u_i^{(1/w_i)}$; `// Calculate key`
  if $|R| < k$ **then**
    $R \leftarrow R \cup \{(i, k_i)\}$; `// Add item to reservoir`
  else if $k_i >$ *smallest key in* R **then**
    Remove item with smallest key from R
    $R \leftarrow R \cup \{(i, k_i)\}$; `// Replace item with smallest key`

The final reservoir R contains the weighted random sample. This algorithm can be implemented in O(n log k) time using a min-heap.

4.5 Correctness of Weighted Reservoir Sampling

Given positive weights $w_1, \ldots, w_n$ and independent $u_i \sim \text{Unif}(0, 1)$, the algorithm sets $k_i := u_i^{1/w_i}$ and keeps the item with the largest key. We show

$$P(\text{item } i \text{ is kept}) = \frac{w_i}{\sum_{j=1}^{n} w_j}.$$

Proof. Let

$$s_i := \frac{-\ln u_i}{w_i}.$$

Then for $x \geq 0$, $P(s_i > x) = P(u_i < e^{-w_i x}) = e^{-w_i x}$, so $s_i \sim \text{Exp}(w_i)$ and the s_i are independent. Moreover $k_i = e^{-s_i}$, hence $\arg\max_i(k_i) = \arg\min_i(s_i)$.

For independent exponentials,

$$P(s_i = \min_j s_j) = \int_0^\infty \left[w_i e^{-w_i x}\right] \prod_{j \neq i} e^{-w_j x}\, dx = \int_0^\infty w_i e^{-(\sum_j w_j)x}\, dx = \frac{w_i}{\sum_{j=1}^{n} w_j}.$$

Therefore the algorithm selects item i with probability proportional to its weight.