CMPUT 498/501: Advanced Algorithms **Fall 2025**

Lecture 21 (Nov 24, 2025): Introduction to Streaming and Sketching

Lecturer: Mohammad R. Salavatipour *Scribe: Baxter Madore*

21.1 Introduction

Recently, there has been a large growth in applications involving data gathering and processing. These applications often occur in settings with limited access to data, or limited computational power, such as remote sensing. We introduce some tools and techniques to extract information from data and/or compress data without much loss.

Streaming There are situations where it is infeasible to store all of the data, or it is infeasible to process all of the data at the same time. For example, an airplane's sensors collect multiple terabytes of data per hour of flight. We might have access to one (or a few) passes over the data, and we have to process tha data as it arrives and make decisions without the full dataset.

Sketching/Sampling The goal of sketching is to have a compressed form of data which we can still use to answer queries and extract useful information.

Dimensionality Reduction In many applications such as medical data analysis or spam filtering, data comes in very high dimensions, with possibly thousands of features. Designing algorithms to manage high-dimensional data is difficult. One goal is to reduce dimension, for instance by projection, while approximately preserving the relevant structure/geometry.

Property Testing Checking quickly and with high probability whether a given object has a certain property. For example, checking if a matrix multiplication is correct, or if a large graph has certain properties.

Sparse Fourier Transform There are old algorithms to compute the discrete Fourier transform (DFT) of a sequence of length n in $O(n \log n)$ time. Discrete Fourier transforms have various applications, including signal/image processing, and multiplying large integers. A sparse Fourier transform is a faster way to compute the DFT when the output is sparse. For example, an image with a lot of similar pixels grouped together can be processed faster than an image of random noise.

21.2 Approximate Counting

Let's start with the simple example of approximately counting the number of events in a sequence of length N.

Suppose we have a long sequence of events. If we want a counter which can deterministically and exactly count the number of events, we will clearly need $O(\log N)$ bits. Any deterministic algorithm will need at least this much storage.

Our goal is to use exponentially less than $O(\log N)$ bits of total storage, while still getting a good approximation to the number of events.

Definition 1 *Given ε and δ, if the actual answer is n and the estimated answer that our algorithm returns is $\tilde{n}$, the algorithm is a (ε, δ) estimator if*

$$Pr[|\tilde{n} - n| > \varepsilon n] < \delta$$

Morris came up with a simple (ε, δ) estimator for event counting using $\tilde{O}(\log \log n)$ bits [**M78**].

Algorithm 1 Morris' Approximate Counting Algorithm

$X \leftarrow 0$
for each new event **do**
  increment X with probability $\frac{1}{2^X}$
end for
return $\tilde{n} = 2^X - 1$

Analysis: Let X_n be the random variable representing X after the n^{th} item, and let $Y_n = 2^{X_n}$.

Lemma 1 $\mathbb{E}[Y_n] = n + 1$

Proof of Lemma 1: The proof is by induction on n. When $n = 0$, $X_n = 0$, $Y_n = 2^0 = 1 = n + 1$.
For any $n + 1$,

$$\begin{aligned}
\mathbb{E}[Y_{n+1}] &= \sum_{i=0}^{\infty} \Pr[X_n = i] \times \mathbb{E}[2^{X_{n+1}} | X_n = i] \\
&= \sum_{i=0}^{\infty} \Pr[X_n = i](2^i(1 - \frac{1}{2^i}) + \frac{1}{2^i}2^{i+1}) \\
&= \sum_{i=0}^{\infty} \Pr[X_n = i](2^i - 1 + 2) \\
&= \sum_{i=0}^{\infty} \Pr[X_n = i](2^i) + \sum_{i=0}^{\infty} \Pr[X_n = i] \\
&= \mathbb{E}[Y_n] + 1 = n + 1 + 1
\end{aligned}$$

So $\tilde{n} = 2^X - 1$ is an estimate of n in expectation. ■
Since $\mathbb{E}[Y_n] = n + 1$, $\mathbb{E}[X_n] = \log_2(n + 1)$, so the number of bits used to store X_n is $O(\log \log n)$.

Lemma 2 $\mathbb{E}[Y_n^2] = \frac{3}{2}n^2 + \frac{3}{2}n + 1$ *and* $Var[Y_n] = \frac{n(n-1)}{2}$

Proof of Lemma 2 Again, we use induction. When $n = 0$, $X_n = 0$, $Y_n = 1 = \frac{3}{2}(0^2) + \frac{3}{2}(0) + 1$ For $n + 1$, the

induction proceeds as follows:

$$\begin{aligned}
\mathbb{E}[Y_{n+1}^2] &= \sum_{i=0}^{\infty} 2^{2i}\Pr[X_{n+1}=i] \\
&= \sum_{i=0}^{\infty} 2^{2i}(\Pr[X_n=i](1-\frac{1}{2^i}) + \Pr[X_n=i-1](\frac{1}{2^{i-1}}) \\
&= \sum_{i=0}^{\infty} 2^{2i}(\Pr[X_n=i]) - \sum_{i=0}^{\infty} -\frac{2^{2i}}{2^i}\Pr[X_n=i] + \sum_{i=0}^{\infty}\frac{2^{2i}}{2^{i-1}}\Pr[X_n=i-1] \\
&= \sum_{i=0}^{\infty} 2^{2i}(\Pr[X_n=i]) - \sum_{i=0}^{\infty} -2^i\Pr[X_n=i] + 4\sum_{i=0}^{\infty}2^{i-1}\Pr[X_n=i-1] \\
&= \mathbb{E}[Y_n^2] + 3\mathbb{E}[Y_n] \\
&= \frac{3}{2}n^2 + \frac{3}{2} + 1 + 3(n+1) = \frac{3}{2}(n+1)^2 + \frac{3}{2}(n+1) + 1.
\end{aligned}$$

This also means that $\mathrm{Var}[Y_n] = \mathbb{E}[Y_n^2] - \mathbb{E}[Y_n]^2 = \frac{n(n-1)}{2}$ ■

The probability of an error within an ε factor of n can be computed using Chebyshev's inequality as

$$\Pr[|\tilde{n} - n| > \varepsilon n] < \frac{1}{(\varepsilon n)^2} \times \frac{n(n-1)}{2} \approx \frac{1}{2\varepsilon^2} \tag{21.1}$$

For any ε smaller than $\sqrt{\frac{1}{2}}$, this provides a probability greater than 1, and so a single run of the algorithm is useless for reasonable values of ε.

21.2.1 Morris+: Boosting Probability using Averages

Suppose we run r parallel copies of Morris' algorithm and find values $\tilde{n}_i$ for $1 \le i \le r$. Let $\tilde{n} = \frac{1}{r}\sum_{i=1}^{r}\tilde{n}_i$. $\tilde{n}$ is an estimator for n.

If we choose $r > \frac{1}{2\varepsilon^2\delta}$, then $\Pr[|\tilde{n} - n| > \varepsilon n] \le \frac{1}{2\varepsilon^2 r} < \delta$.

Note that because $r > \frac{2}{\varepsilon^2}$, $\Pr[|\tilde{n} - n| > \varepsilon n] < \frac{1}{4}$. The amount of space used across the parallel runs of Morris+ is $O(\frac{\log\log n}{\varepsilon^2\delta})$ bits.

21.2.2 Morris++: Using Medians to Boost Probability

The space requirements of Morris+ can be improved if we use both the average and the median to estimate n.

Let ℓ be $c\log\frac{1}{\delta}$ for some constant c, and run ℓ copies of Morris+. Suppose that we get estimates $Z_1, Z_2 ... Z_\ell$ from Morris+, and let $\tilde{n}$ be the median of these estimates. By the arguments of the previous section for Morris+ we get that for each i, $\Pr[|Z_i - n| > \varepsilon n] < \frac{1}{4}$

Let Y_i be the random variable defined as

$$Y_i = \begin{cases} 1 & \text{if } |Z_i - n| > \varepsilon n \\ 0 & \text{otherwise} \end{cases}$$

Because each run of Morris+ is independent, Y_i's are all independent, with $\Pr[Y_i] < \frac{1}{4}$ and $\mathbb{E}[\sum_{i=0}^{\ell} Y_i] < \frac{\ell}{4}$.

Note that $|\tilde{n} - n| > \varepsilon n$ only if at least half of the Z_is, that is, $\frac{l}{2}$ of them, are larger than n by εn.

Using Chernoff bounds:

$$\begin{aligned}
\Pr[|\tilde{n} - n| > \varepsilon n] &\leq \Pr[\sum_{i=0}^{n} Y_i \geq \frac{\ell}{2}] \\
&\leq \Pr[|\sum_{i=0}^{n} Y_i - \mathbb{E}[\sum_{i=0}^{n} Y_i]| > \frac{\ell}{4}] \\
&\leq \Pr[\sum_{i=0}^{n} Y_i \geq 2\frac{\ell}{4}] \\
&\leq (\frac{e}{2^2})^{\frac{\ell}{4}}
\end{aligned}$$

With $\ell = c \log \frac{1}{\delta}$, this probability is less than δ for large enough c, and the total space required for estimation is $O(\frac{\log(1/\delta)}{\varepsilon^2} \log\log(\frac{n}{\varepsilon\delta}))$.

21.3 Estimating Frequency Moments

Suppose we have a stream of data $\sigma = a_1, a_2, ..., a_m$, where each $a_i \in \{1, 2, ..., n\}$.

The frequency vector $f = (f_1, f_2, ..., f_n)$ for σ has as its elements f_i the number of times that i appears in σ.

For example, if $\sigma = 3, 2, 1, 2, 3, 4, 5, 3, 5, 3, 2, 4, 5, 3, 2, 2$, then $f = (1, 5, 5, 2, 3)$.

Definition 2 *Given a sequence σ, the k^{th} moment of σ, $F_k(\sigma)$ is defined as*

$$F_k(\sigma) = \sum_{i=1}^{n} f_i^k$$

$$\begin{aligned}
F_0(\sigma) &= \sum_{i=1}^{n} f_i^0 = \sum_{i:f_i>0} 1 = \text{number of distinct elements in } \sigma \\
F_1(\sigma) &= \sum_{i=1}^{n} f_i = m = \text{number of elements in } \sigma \\
F_\infty(\sigma) &= \max_i f_i = \text{maximum frequency of any element in } \sigma
\end{aligned}$$

We also define $0^0 = 0$.

Morris++ works for frequency moments when $k = 1$. Next we will show an algorithm for $k = 0$, that is, the number of distinct elements in σ. If $d = |\{i : f_i > 0\}|$, no deterministic algorithm can find d in sublinear space.

21.3.1 Hashing Review

k-universal: A family $\mathcal{H}$ of hash functions $h : \mathcal{U} \to T$ is (strongly) k-universal if for any distinct elements $x_1, ..., x_k \in \mathcal{U}$ and k values $\alpha_1, ..., \alpha_k \in T$,

$$\Pr[\bigwedge_{i=1}^{k} h(x_i) = \alpha_i] = \frac{1}{n^k} = \frac{1}{|T|^k}$$

These k elements are said to be k-wise independent.

Definition 3 *A collection of random variables $x_1, x_2..., x_n$ are k-wise independent if for any subset of k of them,*

$$Pr[x_1 = a_1 \wedge x_2 = a_2 \wedge ... \wedge x_k = a_k] = \prod_{j=1}^{k} Pr[x_j = a_j]$$

If $x_1..x_n$ are pairwise independent and $Y = \sum_{i=1}^{n} X_i$ then

$$\text{Var}[Y] = \sum_{i=1}^{n} \text{Var}[X_i]$$

If $x_1..x_n$ are Bernoulli, then

$$\text{Var}[Y] \leq \sum_{i=1}^{n} \mathbb{E}[X_i^2] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \mathbb{E}[Y]$$

The first fact follows from the fact that each pair of X_i values are independent and have no covariance, and the second is because the variance of a sum of independent random Bernoulli variables, npq for some $p \in [0, 1], q = 1 - p$, is always less than their expectation np.

21.4 AMS Algorithm for $F_0(\sigma)$

We describe an algorithm for estimating the number of distinct elements in a collection σ. This algorithm is due to Alon, Matias, and Szegedy [**AMS96**], based on earlier work by Flajolet and Martin [**FM85**].

The big picture idea of the algorithm is that we put each distinct item randomly between 0 and 1. On expectation if there are t distinct elements, then the interval $[0, 1]$ should be divided into chunks of size roughly $\frac{1}{t+1}$. We use a 2-universal hash function to place items randomly. This is the rough outline, the following definitions will allow it to work formally.

Definition 4 *For any $t \in \mathbb{Z}^+$, let zero(t) be the largest power of 2 that t is divisible by. Equivalently, zero(t) is the number of trailing zeros in the binary representation of t. As examples, zero$(37) = 0$, zero$(80) = 4$, and zero$(2048) = 11$.*

Assume that we have sufficiently many uniformly distributed numbers. If we had d distinct elements in σ, then for each $x \in \sigma$, we expect one x to have zero$(x) \geq \lfloor \log(d) \rfloor$. Intuitively, if we have $d \geq 2^{\lfloor \log(d) \rfloor}$ uniformly randomly distributed elements, we should expect at least one element to have each of the d possible moduli, mod $\lfloor \log d \rfloor$, including 0. To achieve almost uniform distribution of values we use hashing.

Algorithm 2 AMS Distinct Elements Estimator

$\mathcal{H} \leftarrow$ 2-universal hash family $[n] \to [n]$
select h from $\mathcal{H}$
$Z \leftarrow 0$
for each element in σ **do**
  $Z \leftarrow \max\{Z, \text{zero}(h(a_i))\}$
end for
return $\tilde{n} = 2^{Z+\frac{1}{2}}$

Let S be the set of distinct elements in σ. Define $X_{r,i}$ for all $i \in \{1, 2, .., n\}$ as

$$X_{r,i} = \begin{cases} 1 & \text{if zero}(h(i)) \geq r \\ 0 & \text{otherwise} \end{cases}$$

.

Define $Y_r = \sum_{i \in S} X_{r,i} = |\{x \in \sigma : \text{zero}(h(x)) \geq r\}|$. Y_r is the number of distinct elements x with at least r trailing zeros in the binary representation of $h(x)$.

$$\mathbb{E}[X_{r,i}] = \Pr[\text{zero}(h(i)) \geq r] = \Pr[2^r \text{ divides } h(i)] = \frac{n/2^r}{n} = \frac{1}{2^r}$$

.

Therefore, $\mathbb{E}[Y_r] = \sum_{i \in S} \mathbb{E}[X_{r,i}] = \frac{d}{2^r}$.

Assuming that d is a power of 2, $\mathbb{E}[Y_r] = \mathbb{E}[X_{r,i}] = \frac{d}{2^{\log d}} = 1$

Now we compute the variance of Y_r. Since h is 2-universal, we have 2-wise independence, and

$$\text{Var}[Y_r] = \sum_{i \in S} \text{Var}[X_{r,i}] \leq \sum_{i \in S} \mathbb{E}[X_{r,i}^2] = \sum_{i \in S} \mathbb{E}[X_{r,i}] \leq \frac{d}{2^r}$$

Using Chebyshev's inequality:

$$\Pr[Y_r > 0] = \Pr[Y_r \geq 1] \leq \mathbb{E}[Y_r] \leq \frac{d}{2^r}$$
$$\Pr[Y_r = 0] = \Pr[|Y_r - \mathbb{E}[Y_r]| > \frac{d}{2^r}] \leq \frac{\text{Var}[Y_r]}{(d/2^r)^2} \leq \frac{2^r}{d}$$

Suppose Z' is the last value of Z in the algorithm, and the output is $2^{Z'+\frac{1}{2}} = \tilde{d}$. We claim that $\tilde{d}$ is close to d. Let a be the smallest integer such that $2^{a+\frac{1}{2}} \geq 3d$.

$$\Pr[\tilde{d} \geq 3d] \leq \Pr[Z' \geq a] = \Pr[Y_a > 0] \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}$$

Similarly, let b be the largest integer such that $2^{b+\frac{1}{2}} \leq \frac{d}{3}$.

$$\Pr[\tilde{d} \leq \frac{d}{3}] \leq \Pr[Z' \leq b] = \Pr[Y_{b+1} = 0] \leq \frac{2^{b+1}}{d} \leq \frac{\sqrt{2}}{3}$$

This algorithm returns a $\tilde{d}$ such that $\frac{d}{3} \leq \tilde{d} \leq 3d$ with probability no less than $1 - \frac{2\sqrt{2}}{3} \approx 0.057$, so this is a $(3, 0.057)$-estimator for $F_0(\sigma)$. We could improve this to being a $(3, 1-\delta)$-estimator by repeating the algorithm $O(\log \delta)$ times and picking the median, which would take $O(\log n \log \frac{1}{\delta})$ total space.