

COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2026.

Lecture 6

In-Class Quiz 2

Question 1: Consider independent random variables $X_1, \dots, X_n$, each taking value -1 with probability $1/2$ and 1 with probability $1/2$. Let $X = \sum_{i=1}^n X_i$. What are $\mathbb{E}[X]$ and $\text{Var}[X]$?

Question 2: Use Chebyshev's inequality to prove an upper bound on $\Pr[|X| \geq 5\sqrt{n}]$. If you wanted to prove a tighter bound, what concentration bound(s) taught in class could you potentially use?

- Problem Set 1 is due this Friday..
- My office hours this week will be **Wednesday**
10:30am-11:30am

Lecture Pacing:

- Way too Fast – ?
- A Bit too Fast – ?
- Just Right – ?
- Too Slow – ?
- Way Too Slow – ?

Last Time

Last Class:

- Exponential concentration bounds
- Bernstein inequality and the Chernoff bound
- Connection to the central limit theorem.
- Application to random hashing – $O(\log n)$ maximum load per bucket when hashing n items into n buckets.

This Class:

- Bloom filters: random hashing to maintain a large set in small space.

Chernoff Bound Analysis

We showed that when hashing n items into n buckets of a hash table, the maximum load is $O(\log n)$ with high probability? How did we come to that answer?

- Let $X_i = \sum_{j=1}^n X_{i,j}$ be the number of items in bucket i .
- $\mathbb{E}[X_i] = 1$ and by a Chernoff bound, for any δ ,

$$\Pr[X_i > 1 + \delta] \leq \Pr[|X_i - 1| > \delta] \leq 2e^{-\frac{\delta^2}{2+\delta}} \approx e^{-\delta}.$$

- By a union bound, $\Pr[\max X_i > 1 + \delta] \leq n \cdot e^{-\delta}$.
- So, e.g, to have $\Pr[\max X_i > 1 + \delta] \leq \frac{1}{100}$ we need to set:

$$n \cdot e^{-\delta} \leq \frac{1}{100} \implies e^{-\delta} \leq \frac{1}{100n} \implies \delta = \ln(100n) = O(\log n).$$

Bloom Filters

Approximately Maintaining a Set

Want to store a set S of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

Goal: support $insert(x)$ to add x to the set and $query(x)$ to check if x is in the set. Both in $O(1)$ time. What data structure solves this problem?

- Allow small probability $\delta > 0$ of false positives. I.e., for any x ,

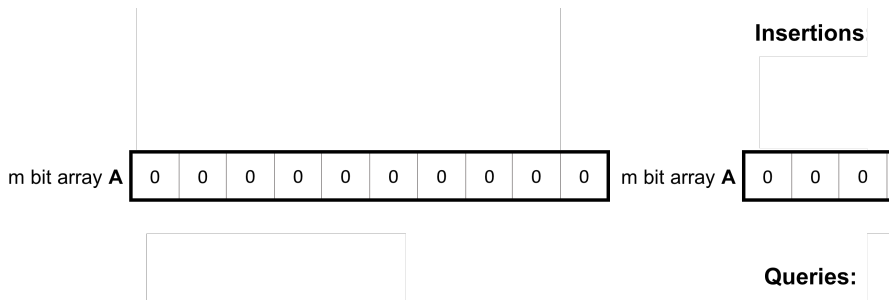
$$\Pr(query(x) = 1 \text{ and } x \notin S) \leq \delta.$$

Solution: Bloom filters (repeated random hashing). Will use much less space than a hash table.

Bloom Filters

Chose k independent random hash functions $h_1, \dots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

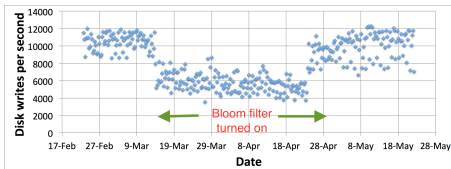
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[h_1(x)] = \dots = A[h_k(x)] := 1$.
- *query*(x): return 1 only if $A[h_1(x)] = \dots = A[h_k(x)] = 1$.



No false negatives. False positives more likely with more insertions.

Applications: Caching

Akamai (Boston-based company serving 15 – 30% of all web traffic) applies bloom filters to prevent caching of ‘one-hit-wonders’ – pages only visited once fill over 75% of cache.



- When url x comes in, if $query(x) = 1$, cache the page at x . If not, run $insert(x)$ so that if it comes in again, it will be cached.
- **False positive:** A new url (possible one-hit-wonder) is cached. If the bloom filter has a false positive rate of $\delta = .05$, the number of cached one-hit-wonders will be reduced by at least 95%.

Applications: Databases

Distributed database systems, including Google Bigtable, Apache HBase, Apache Cassandra, and PostgreSQL use bloom filters to prevent expensive lookups of non-existent data.

Movies

	5			1	4				
		3					5		
				4					
		5							5
	1			2					

Users

- When a new rating is inserted for $(user_x, movie_y)$, add $(user_x, movie_y)$ to a bloom filter.
- Before reading $(user_x, movie_y)$ (possibly via an out of memory access), check the bloom filter, which is stored in memory.
- **False positive:** A read is made to a possibly empty cell. A $\delta = .05$ false positive rate gives a 95% reduction in these empty reads.

More Applications

- **Database Joins:** Quickly eliminate most keys in one column that don't correspond to keys in another.
- **Recommendation systems:** Bloom filters are used to prevent showing users the same recommendations twice.
- **Spam/Fraud Detection:**
 - Bit.ly and Google Chrome use bloom filters to quickly check if a url maps to a flagged site and prevent a user from following it.
 - Can be used to detect repeat clicks on the same ad from a single IP-address, which may be the result of fraud.
- **Digital Currency:** Some Bitcoin clients use bloom filters to quickly pare down the full transaction log to transactions involving bitcoin addresses that are relevant to them (SPV: simplified payment verification).

Analysis

For a bloom filter with m bits and k hash functions, the insertion and query time is $O(k)$. How does the false positive rate δ depend on m , k , and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0? $n \times k$ total hashes must not hit bit i .

$$\begin{aligned}\Pr(A[i] = 0) &= \Pr(h_1(x_1) \neq i \cap \dots \cap h_k(x_k) \neq i \\ &\quad \cap h_1(x_2) \neq i \dots \cap h_k(x_2) \neq i \cap \dots) \\ &= \underbrace{\Pr(h_1(x_1) \neq i) \times \dots \times \Pr(h_k(x_1) \neq i) \times \Pr(h_1(x_2) \neq i) \dots}_{k \cdot n \text{ events each occurring with probability } 1-1/m} \\ &= \left(1 - \frac{1}{m}\right)^{kn}\end{aligned}$$

Analysis

How does the false positive rate δ depend on m , k , and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

$$\begin{aligned}\Pr(A[\mathbf{h}_1(w)] = \dots = A[\mathbf{h}_k(w)] = 1) \\&= \Pr(A[\mathbf{h}_1(w)] = 1) \times \dots \times \Pr(A[\mathbf{h}_k(w)] = 1) \\&= \left(1 - e^{-\frac{kn}{m}}\right)^k \quad \text{Actually Incorrect! Dependent events.}\end{aligned}$$

n : total number items in filter, m : number of bits in filter, k : number of random hash functions, $\mathbf{h}_1, \dots, \mathbf{h}_k$: hash functions, A : bit array, δ : false positive rate.