CS 130A Bloom Filter Project: Performance Analysis and Design Choices

Vishva Arasan
CS 130A - Fall 2025

November 23, 2025

1 Introduction

The big idea behind this project was to test a specific type of data checker called a Bloom filter. Bloom Filters save a ton of memory, which is why companies love them, but the trade-off is that it sometimes gives a "maybe" answer when the item is not actually in the set, which is the false positive rate. Our main goal was to see if the math we learned in class actually matches what happens when we build the filter in code and test it with millions of items. We also wanted to figure out the best design, specifically which of the two hash functions we built works best and how many hash functions we should use. The entire simulation had a runtime of approximately 17 minutes, confirming the complexity of the workload.

2 Methods

2.1 Task 1: Hash Functions

The experiment utilized the following global parameters:

- **Universe Size (N):** $N = 2^{31} - 2$.
- **Prime (p):** $p = 2^{31} - 1$.
- **Table Size (m):** $m = 10000$.

1. **Type 1 (Polynomial):** $h_i(x) = ((a_i \cdot x + b_i) \bmod p) \bmod m$. The coefficients a_i and b_i regenerated per trial.
2. **Type 2 (Seeded Generator):** $h_i(x) = r(s_i + x)$. This was implemented using Python's `random` module, where the generator was re-seeded with `random.seed(s_i + x)` and the first generated integer `random.randint(0, m-1)` was used as the hash index.

Figure 1: Hash Function Uniformity Test (Correlated Data)

The results in Figure 1 clearly show that data choice matters significantly. The Type 1 (Polynomial) function produced visible diagonal patterns when fed even numbers correlated, indicating a failure to achieve uniformity. The Type 2 (Seeded) function, in contrast, produced a dense and uniform scatter, demonstrating a better hash. Therefore, if input correlation is possible, Type 2 is the better design choice. For random numbers, both hash functions perform consistently and show a uniform distribution.

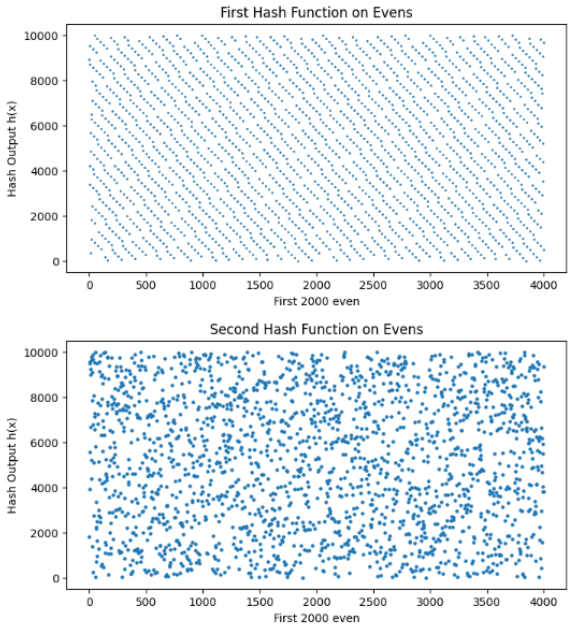


Figure 1: To address the need to compare performance with correlated data, a test was conducted using the first 2000 even numbers and mapped to a table of size $m = 10,000$.

2.2 Task 2 & 3: Bloom Filter and FPR Analysis

The Bloom filter was implemented with `Add(x)` and `Contains(x)` operations. The False Positive Rate (FPR) analysis was conducted using the following parameters:

- **Set Size (n):** $n = 10,000$.
- **Queries (Q):** $Q = 20,000$ (random elements not in S).
- **C:** $c \in \{8, 10, 12\}$.
- **Hash Count (k):** k ranged from 1 to 15.
- **Trials:** 10 trials were performed for each (c, k) pair, and the median FPR was recorded.

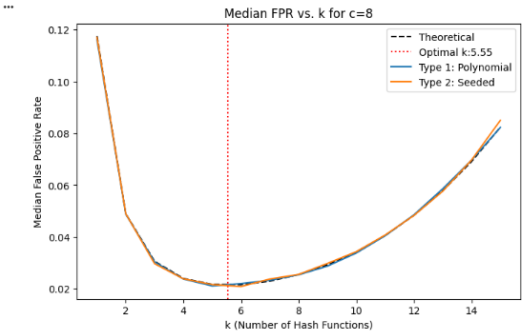
The theoretical values for comparison were derived from the standard Bloom filter formulas in class:

- **Theoretical FPR:** $f = (1 - e^{-k/c})^k$
- **Optimal Hash Count (k_{opt}):** $k_{opt} = c \cdot \ln(2)$

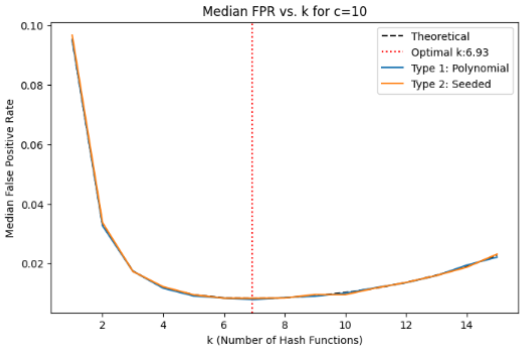
3 Results: FPR vs. Hash Count (k)

The experimental median FPR is plotted against the number of hash functions (k) for $c = \{8, 10, 12\}$.

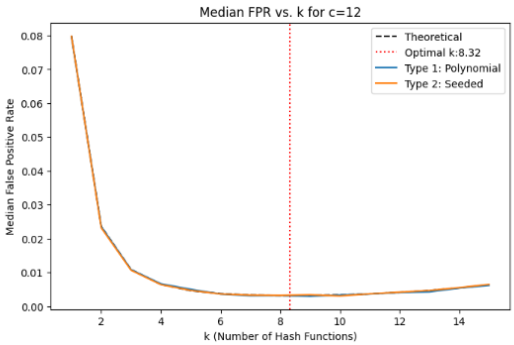
1. **Experimental vs Theory:** In Figures 2-4, the Type 1 (Polynomial) and Type 2 (Seeded) experimental curves are visually indistinguishable from the theoretical prediction. This strongly confirms that the model accurately predicts the filter's performance.
2. **Optimal k:** The characteristic U-shape of the FPR curve is present in all plots. The minimum FPR for each experimental curve occurs at the integer k closest to the theoretical k_{opt} (e.g., $k = 6$ for $c = 8$, $k = 7$ for $c = 10$).



(a) Median FPR vs. k for $c = 8$. Theoretical optimal $k \approx 5.55$.



(b) Median FPR vs. k for $c = 10$. Theoretical optimal $k \approx 6.93$.



(c) Median FPR vs. k for $c = 12$. Theoretical optimal $k \approx 8.32$.

Median False Positive Rate (FPR) vs. k (number of hash functions) for different values of c.

3. **Hash Function Comparison:** Despite the Type 1 (Polynomial) function's failure on correlated data Figure 1, both hash types yielded virtually identical results when used within the Bloom filter with *random* input data. This demonstrates that for inputs that are truly random and uniformly distributed, both hash functions are highly effective.

4 Conclusion

The experiments confirm that Bloom filters are an incredibly reliable tool when implemented correctly. The primary conclusion is that the performance of a Bloom filter, measured by the median False Positive Rate, aligns almost perfectly with the theoretical model, $f = (1 - e^{-k/c})^k$. The optimal number of hash functions k_{opt} was consistently validated as the point that minimizes the FPR. While the Polynomial hash (Type 1) is computationally simpler, its vulnerability to correlated data makes the Seeded Generator (Type 2) the superior and safer choice for any real-world application where the nature of the input data cannot be guaranteed to be perfectly random.