CS 6505 - Bloom Filters Project Report

Ankit Srivastava (gT ID: 902838136)

1 Theory

Bloom filters are data structures used for checking membership of an element in a set (S). The membership test can give false positive results in some cases, i.e. an element not belonging to S can pass the membership test. However, there are no false negatives, i.e. an element belonging to S can never fail the membership test.

The false positive rate of a Bloom filter can be controlled by modifying the number of bits in the Bloom filter (n), the cardinality of the set S (m), and/or the number of hash functions used by the Bloom filter (k). The ratio of the number of bits in the Bloom filter and the number of elements in the set S is denoted by $c = \frac{n}{m}$. Probability of false positives for a Bloom filter and S can be expressed as follows.

$$\text{Theoretical Pr[False Positives]} = \left(1 - e^{-k\frac{m}{n}}\right)^k = \left(1 - e^{-\frac{k}{c}}\right)^k \text{ (for large } n)$$

Using the above equation, the optimal number of hash functions for minimum false positive rate is $k = c \ln 2$.

2 Implementation

In this project, Bloom filter was implemented in C++11. Mersenne Twister pseudo-random number generator engine (`std::mt19937_64` defined in `<random>`) was used for generating random numbers. The generator is programmed to use current time as seed for getting different results every time it is used. Elements of S were chosen to be of type `size_t` for the purpose of calculating the actual false positive rate of the Bloom filter.

Given n and m, the simulator works as follows. First, it instantiates the Bloom filter with n bits and k hash functions, where k is set to $\text{round}(c \ln 2) = \text{round}\left(\frac{n}{m} \ln 2\right)$. The hash functions are implemented as $h_i(x) = (a_i x + b_i) \bmod n \ \forall k$, where a_i and b_i are chosen from $\{1, \ldots, n-1\}$ uniformly at random, using `std::uniform_int_distribution<size_t>` defined in `<random>`. Then, the simulator generates m random numbers, again using `std::uniform_int_distribution<size_t>`, and adds them to the Bloom filter. The simulator also stores all the m generated numbers in a separate set to keep track of them.

The simulator then runs n tests for determining the actual false positive rate of the Bloom filter. For each test, a number is generated and its membership in the Bloom filter is tested. If the membership test returns true, then the element is searched for in the stored set of numbers. If the element is not found in the set, then the test is marked as a false positive. The probability of false positives is then calculated as follows.

$$\text{Observed Pr[False Positives]} = \frac{\text{\# of false positives}}{\text{\# of elements tested}}$$

3 Plots and Observations

For plotting the variation of observed false positive rate with changes in the value of c and n, c was varied in the range $\{1, 2, \ldots, 10\}$ and n was varied in the range $\{1000000, 2000000, \ldots, 10000000\}$. Every experiment, for a given n and c, was repeated five times and an average of the false positive rates so observed was reported.

In the first plot, shown in Figure 1, the variation of false positive rate with changes in the value of c is shown for five different values of n. The theoretical false positive rate for corresponding values of c is shown using a thick red line. It can be seen that the theoretical as well as observed false positive rate for all the values of n decreases with increase in the value of c, which is expected since the probability of false positives will decreases as the number of bits in the Bloom filter per element of S are increased. Further, it can be seen that theoretical false positive rate is lower than all the observed false positive rates.

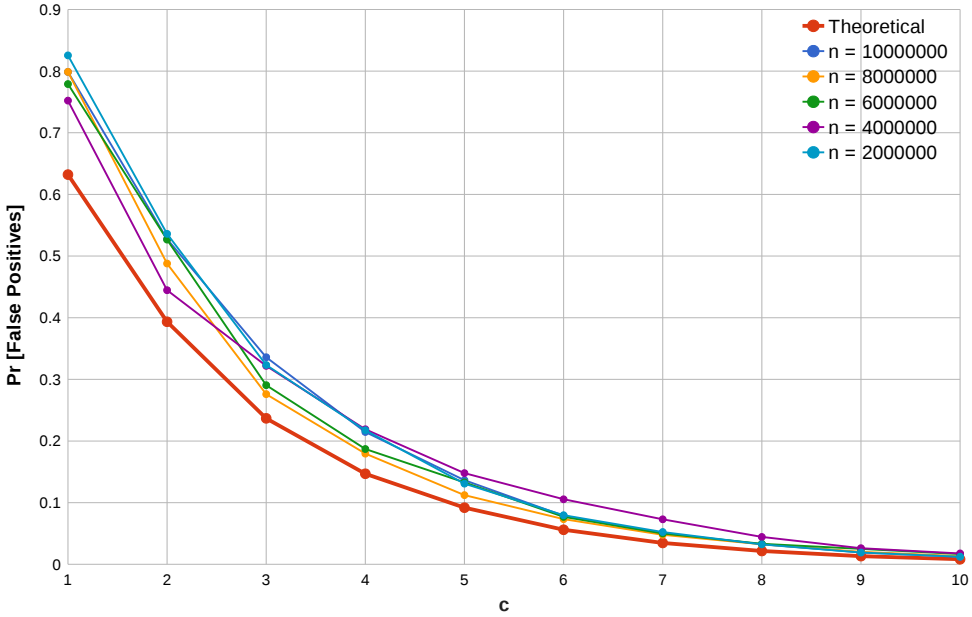

Figure 1: False Positive Rate vs. c

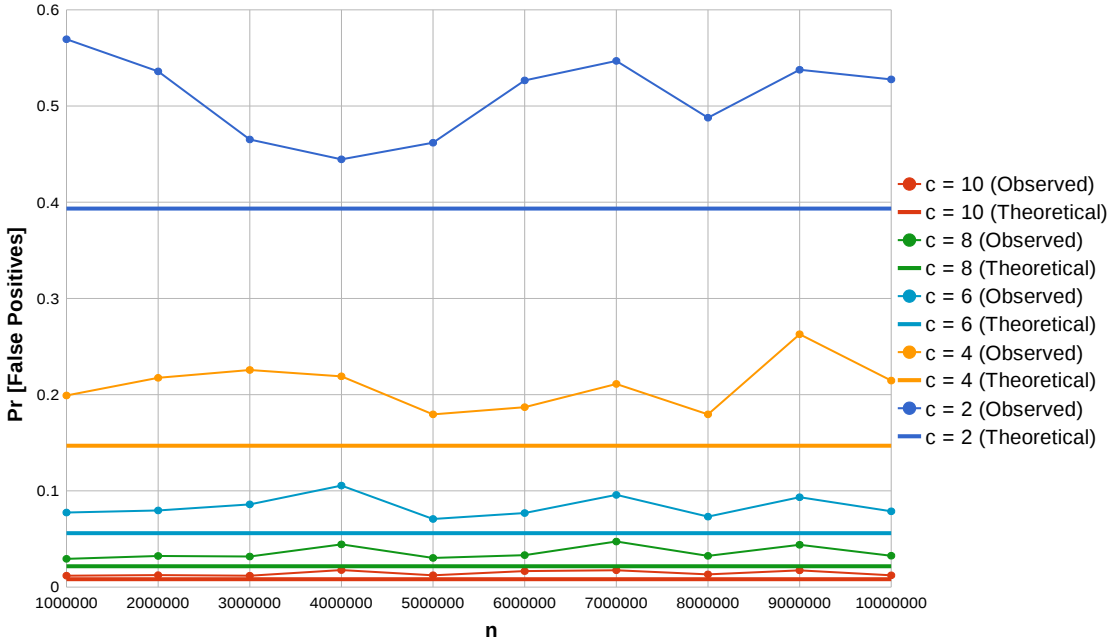

Figure 2: False Positive Rate vs. n

The second plot, shown in Figure 2, shows the variation of false positive rate with changes in the value of n for five different values of c. Observed false positive rates are plotted using lines and points, while the theoretical false positive rate for the corresponding value of c is plotted using a thick line of the same color, parallel to the x-axis. It can be seen that the observed false positive rate remains almost with increase in the value of n. Also, as observed in Figure 1, it can be that observed false positive rates are higher than the theoretical false positive rates.

4 Conclusion

As part of this project, Bloom filter was implemented and the false positive rate for different parameter values was calculated and plotted for the implementation, along with the corresponding theoretical false positive rate. The observed false positive rate was greater than the theoretical false positive rate in all cases. However, on varying the parameters, the observed rate followed the same trend as the theoretical rate in all cases.