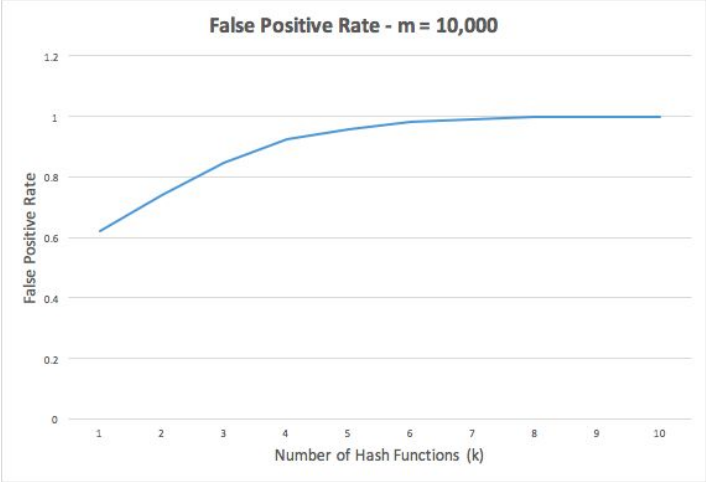
Bloom Filters

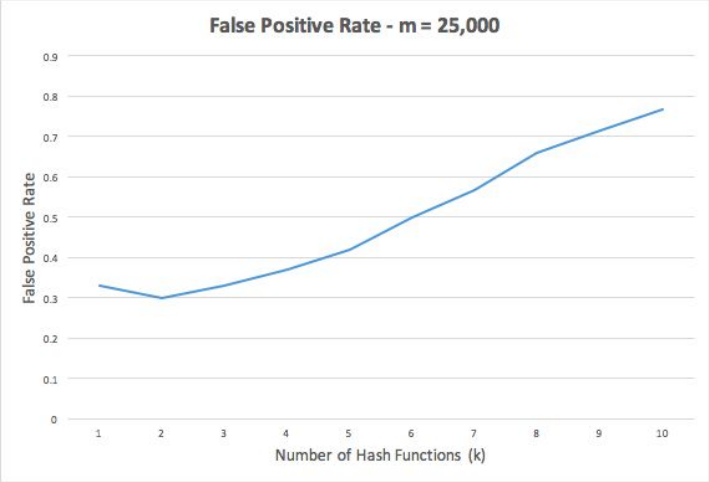
Kyler Kopacz, Jack Fergus, and Dylan Finazzo

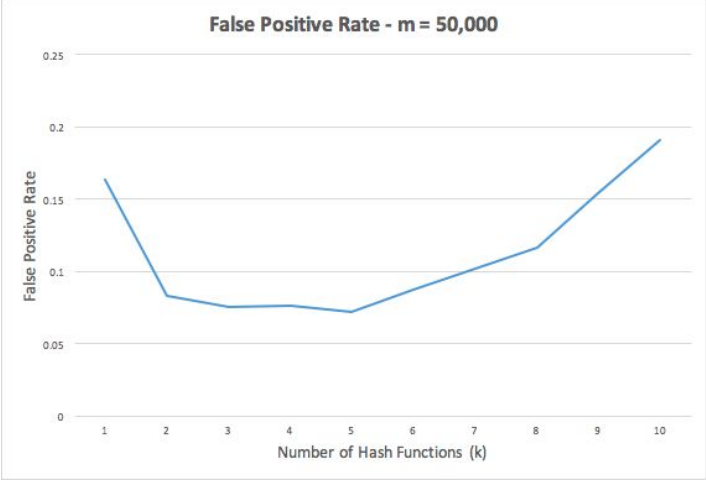
Research Question

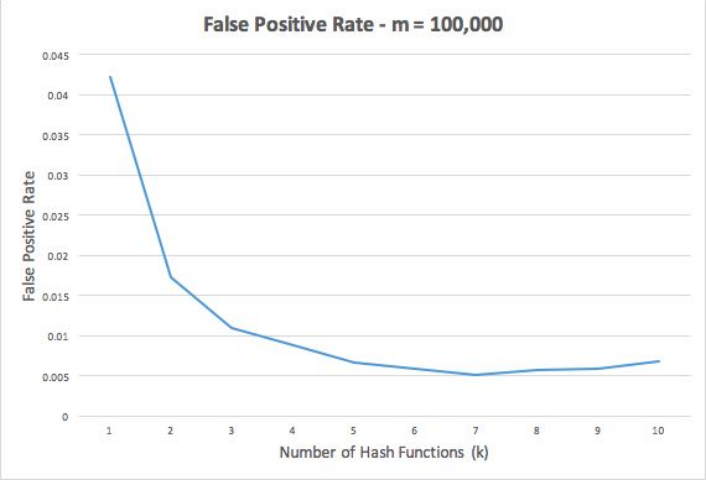
Bloom Filters offer an extremely space efficient approach to check if an element is contained in a set. However, there are ways to maximize the efficiency of a bloom filter by minimizing the chance of a filter's lookup method returning true when an element is not actually in a set. In our experimentation, we analyzed the relationship between the number of hash functions used in bloom filters with varying bit array sizes, and the false positive rate. Additionally, we analyzed the relationship between the size of a bloom filters bit array and the false positive rate while holding the number of hash functions constant. In our experiment we implemented a bloom filter, performed 10,000 insertions, and measured the total number of times our bloom filter returned a false positive. We then calculated the false positive rate by taking the number of false positives and dividing it by the total number of queries performed. In our research, we compared the relationships between the false positive rate, the number of hash functions (k), and the size of the bit array (m) with the values recommended through mathematical derivation.

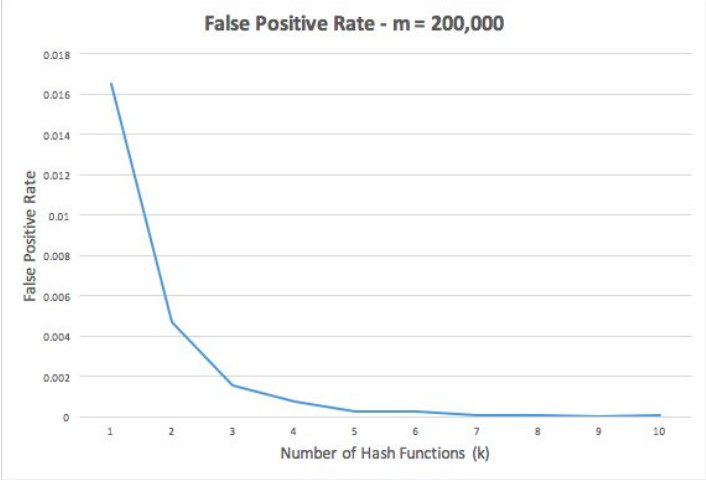
Results

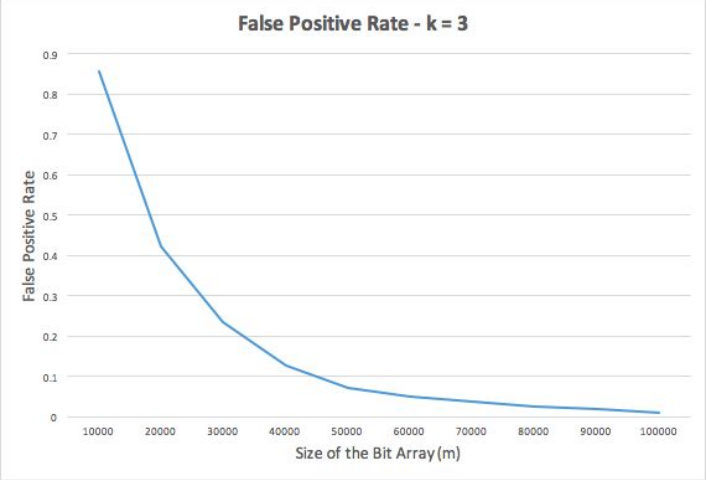

False Positive Rate - m = 10,000
False Positive Rate
Number of Hash Functions (k)


False Positive Rate - m = 25,000
False Positive Rate
Number of Hash Functions (k)


False Positive Rate - m = 50,000
False Positive Rate
Number of Hash Functions (k)


False Positive Rate - m = 100,000
False Positive Rate
Number of Hash Functions (k)


False Positive Rate - m = 200,000
False Positive Rate
Number of Hash Functions (k)


False Positive Rate - k = 3
False Positive Rate
Size of the Bit Array (m)

Discussion

As we can see from the graphs, increasing the number of hash functions for a smaller size array (m = 10,000 - 50,000) past a certain point leads to a higher false positive rate. This is because more bits are hashed than what is necessary, so inserting more than a certain number of values will fill the array with 1's causing every lookup to return positive. For the larger arrays (m = 200,000), increasing the number of hash functions decreases the false positive rate, at least to a certain point. If we kept increasing the number of hash functions past 10 however, we would eventually see the false positive rate increase again, similar to m = 100,000. From the final graph (k = 3), we see that as the size of the array increases, the false positive rate decreases. We expect the graph to continue to decrease asymptotically, but if we were to add a few more hash functions, it would decrease at a faster rate. As we increase the size of the array, we get more possible spaces for the entries to hash to, which in turn decreases the likelihood of two entries hashing to the same spot.

Additionally, in the first 5 graphs, we can see that the k value that achieves the lowest false-positive rate is the "optimal" number calculated by the following equation:

$$k = \frac{m}{n} * ln(2)$$

This result is what we anticipated to see. The lowest false positive rate was achieved when the number of hash functions was closest to the calculated k values from the above formula. For example, when m = 100,000, m/n = 100,000/10,000 = 10 and 10(ln2) = 6.93. In this case we would expect to see our Bloom Filter with 7 hash

functions return the lowest false positive rate, and that is exactly what the graph shows.

We know the expected number of inputs and our desired false positive probability, so we can deduce the necessary size of our bit array using the following formula:

$$m = -\frac{n * ln(P)}{ln(2)^2}$$

For example, if we insert 10,000 elements and want our false positive rate to be .001, the necessary array size will be 143,775.