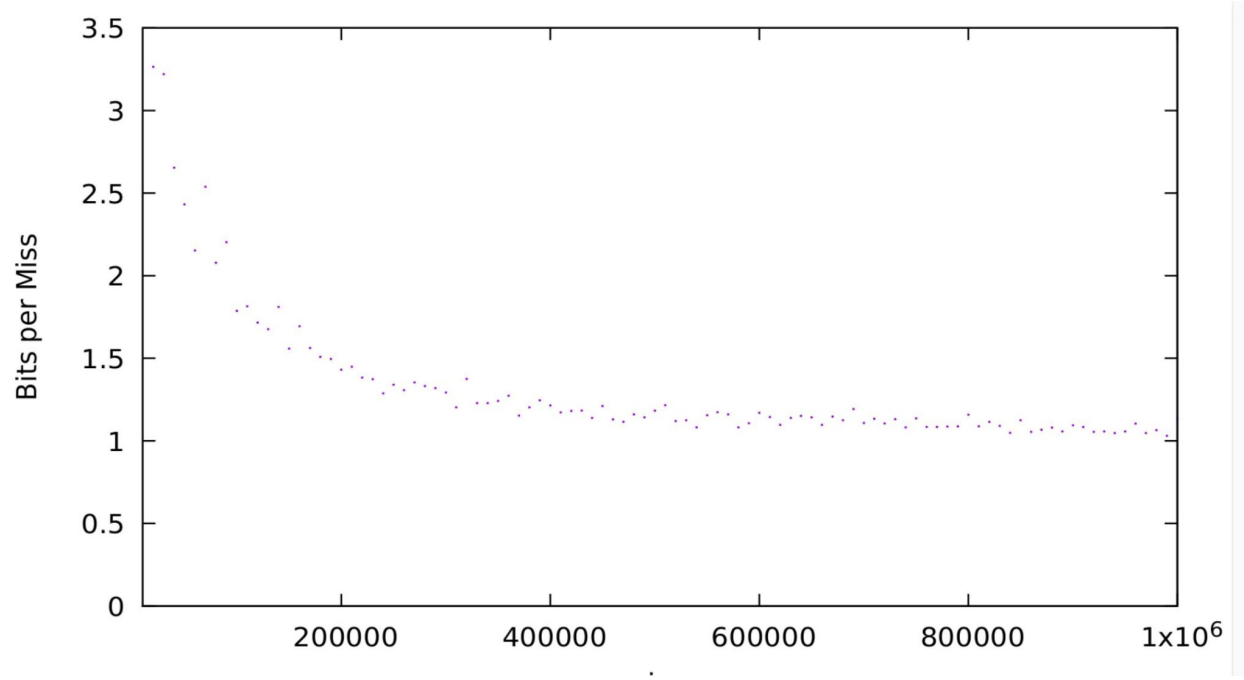
Writeup

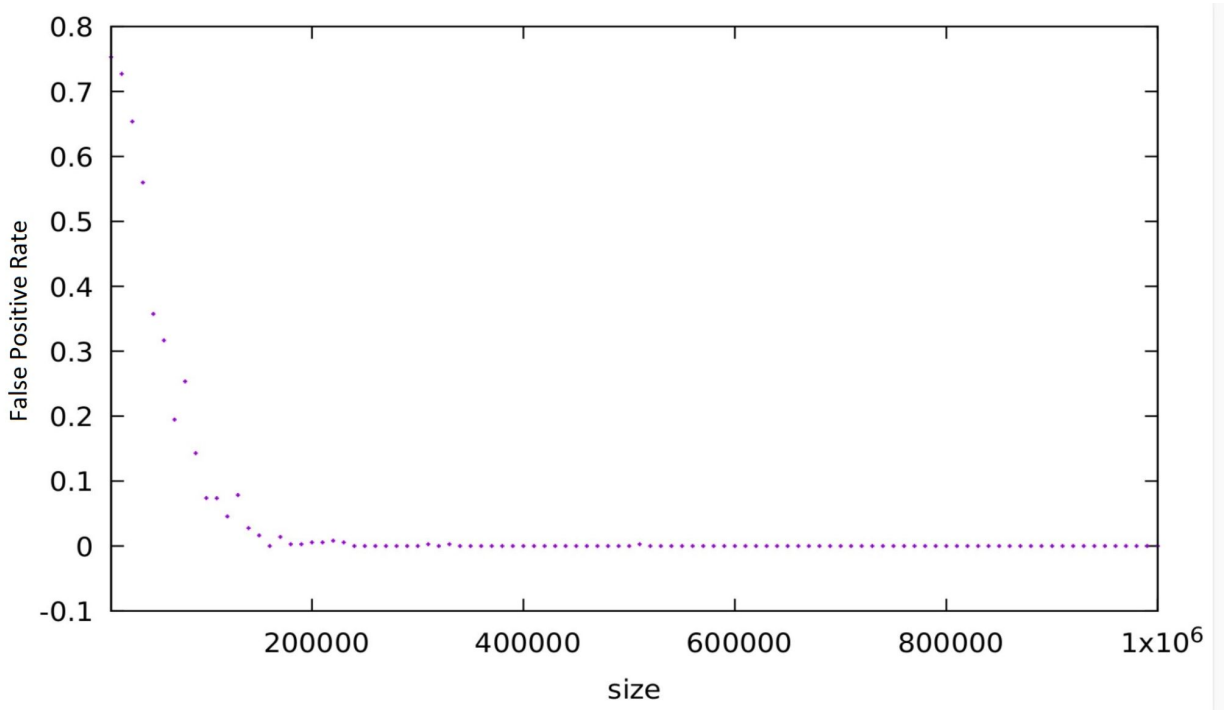
Q: How does the number of bloom filter bits examed per miss vary as the bloom filter size varies?



As the bloom filter increases in size, the Bits Per Miss quickly approaches 1. This means that it only took on average, a single bit to be checked before discovering if the word was certainly NOT inside the hash table.

This conclusion makes sense because as the bloom filter decreases in size, there are fewer available bits to flip. This means that the chance that the hash function returns an overlapping value that has already been flipped increases dramatically. By trying to fit the same amount of data in a smaller bloom filter, it only makes sense that collisions increase which is demonstrated by the graph.

Q: How does changing the bloom filter size affect the number of lookups performed in the hash table (False Positive Rate)?

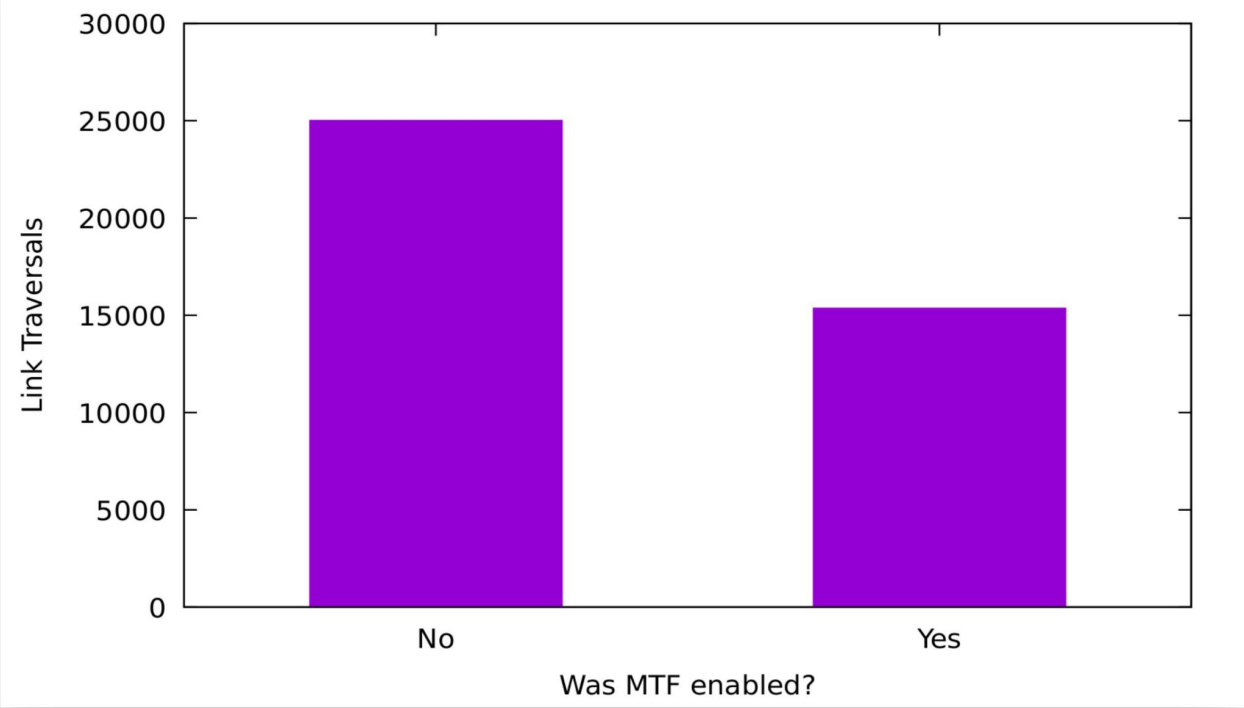


Increasing the size of the bloom filter appears to decrease the average rate of false positives. As the Bloom filter size decreases, the false positive rate approaches 1. As the bloom filter size increases, the false positive rate approaches 0.

This conclusion makes sense because of the conclusion in the previous section. A smaller bloom filter means that the chance that the hash function outputs the same index increases. When the bloom filter is sufficiently small, most of the bits inside of it will be set. This drastically increases the chance for a false positive to occur because the bits that will be checked have a much higher chance of being 1.

Once the bloom filter is sufficiently big, the rate does not go below 0. This is because the chance that a false positive occurs become much lower. The false positive rate cannot go below 0 so decreasing the chance that a false positive occurs when it is already such a low value means that there was not a large difference in a bloom filter. This is also shown for bloom filter sizes of 800k-1mil I was not testing with enough data

to entirely fill a bloom filter of such a size so the false positive rate is very similar between the two.



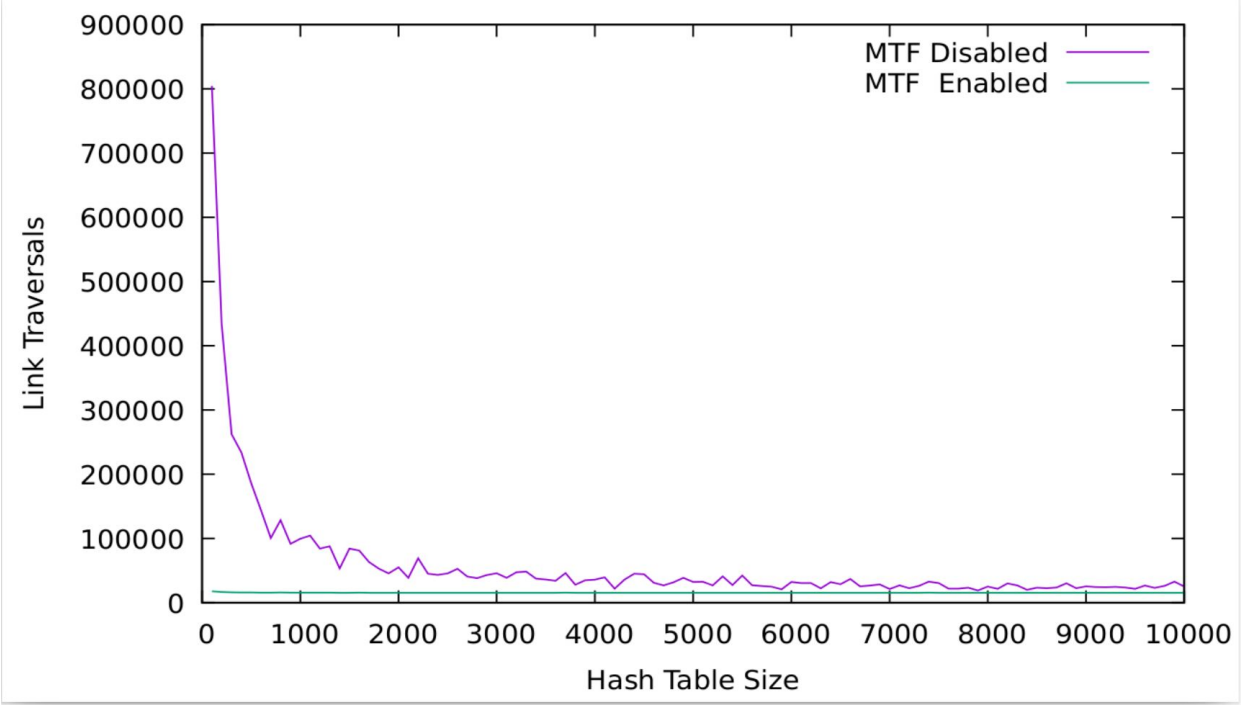
Over many different tests, the graph shows how on average, enabling Move To Front reduced the number of links that had to be traversed inside the linked list. With large sets of data, having MTF enabled meant that more common words would be quicker to access. By having the node accessed quicker, there was less time and resources spent traversing uncommon nodes.

This makes sense because an MTF action will make more common values have less linked list traversals when they are searched for, this will decrease the number of traversals required to reach the node every time it is searched which is why the total traversals is much less when MTF is used.

Something that was interesting about this is how it depends on the size of the linked list in the hash table being very long. If there was only a single node in the linked list, there would be no difference in having MTF enabled or disabled.

This point is further seen when examining the next question.

Q: How does the number of links examed vary based on hash table size?



As is shown in the plot above, having a large hash table means that the difference between having MTF enabled and disabled reduces drastically.

This also shows how having MTF enabled kept the linked lists traversals much more stable. This is interesting because if you do not have the resources to have an extremely large hash table, ensuring that the more common values are moved to the front reduces the number of traversals you need to do drastically.

When setting the hash table size when using a chained hash table, if you have a sufficiently large table, there will be very little difference between having an MTF action occurring and it not occurring. However as the hash table decreases in size, the importance of having an MTF occur begins to increase drastically.

This makes sense because as the hash table decreases in size, the length of the underlying linked lists increases. Having longer linked lists means that more links will be

traversed. By having an MTF action occur, you are making this linked list traversal much more efficient which is why the traversal difference between MTF and no MTF is so large.