Faris Zahrah

Document discussing performance, and other decisions taken regarding Exact K and Inexact K searching.

Important structs

Posting list:
{termID : [IDF_t ,(docID,TF_{id}, [position 1, position 2, ….]) ,(docID,TF_{id}, [position 1, position 2, ….]) , ….] …}

This is good for accessing the IDF_t for each term which is needed to create the document and query vectors. It is not good for is accessing the frequency of the word in each document, because although the documents are sorted, we do not know what documents are there and what are not so we cannot index straight to the document we want to examine without iterating through with a search time in proportion to the number of DocID's which contain the TermID in question.

We need the frequency of the termID per docID for computing the TIDF, which is why I created the next strict I will discuss.

doc_to_term:
{docID: {termID:term-frequency},{termID:term-frequency},…},docID:{termID:term-frequency}, …} …. }

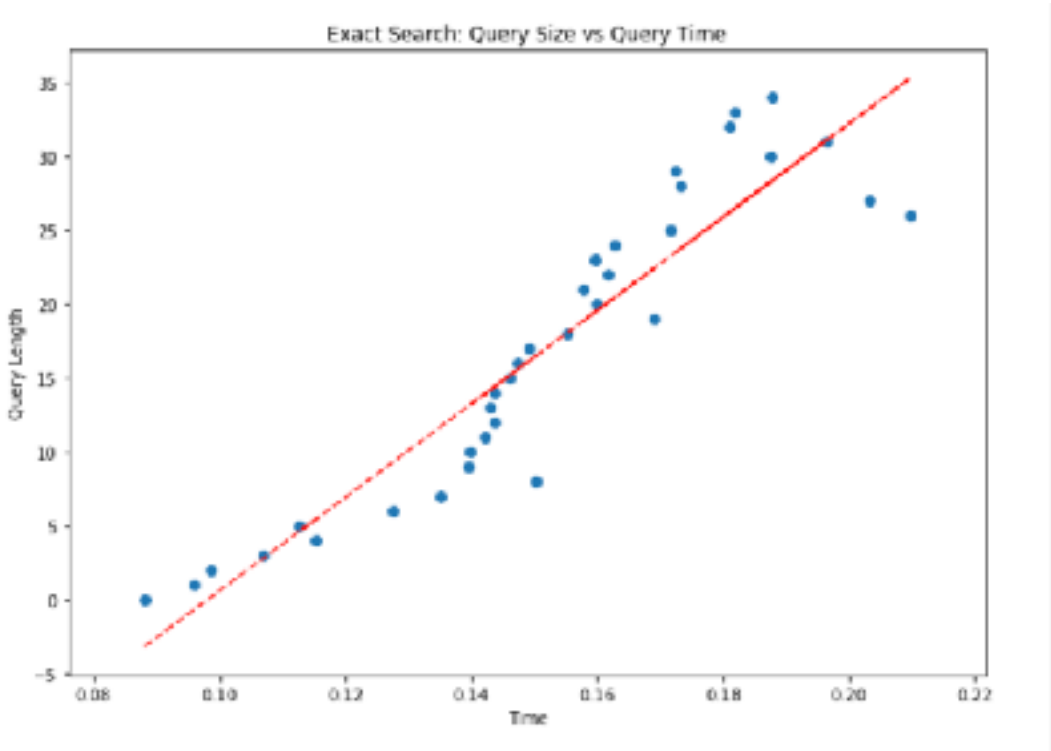
This is great for accessing all terms in each document and their frequency, because to compute IDTF the only other thing I need is the IDF_t and that I can access in constant time if I have the termID.

These two structs together make constructing query and document vectors fast. With all look up times being constant. Constructing the additional struct has an initial overhead in time but that is made up for after a few queries. The main overhead is the space, which is not an issue on a collection size such as this. It may be an overhead on a larger collection but comparing that with the time saving it provides, it seems suitable such that the memory resources are available and appropriately priced.

I will be focusing on Query search time analysis with exact search has the baseline results. After that I will briefly touch accuracy, but I am choosing to stress query search time for my analysis.

QUERY TIME ANALYSIS

Exact Search
Pretty straight forward, I find all the documents that have at least one word in common with the query and I compute the cosine similarity between the query and each document. I return K documents based on highest K scores.
The time complexity is tied to the length of the query. For every additional word in the query, it increases the total number of documents I need to compute the cosine similarity of, this is most costly when the query contains a large amount of common words.

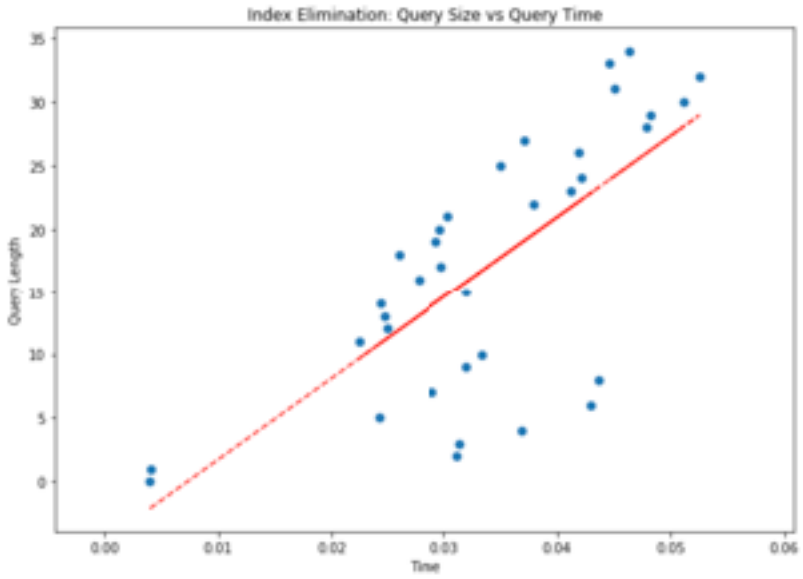


Inexact Searches

Index elimination without Champions list

This is using only the n/2 most infrequent words of the query to search. As you can see on the graph, the average search time in this situation increases much more slowly because as the query grows in size, the number of terms being queried only grows at half the rate and the terms are less frequent so there are less potential documents to score and return.

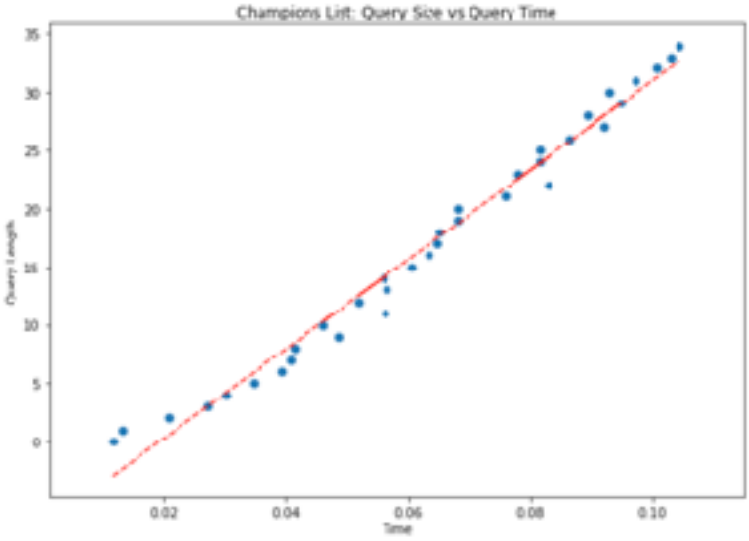
I have chose to not use the entire query if the Index elimination returns less than K results. This is because I see Index elimination as a part of a larger system and its only part is to filter the query and use the terms most infrequent in the documents.

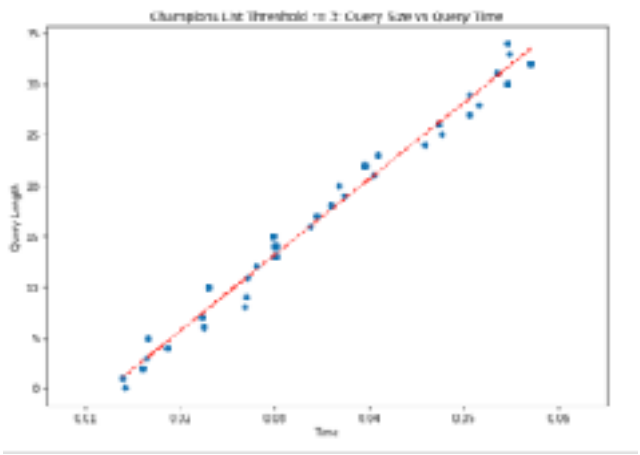


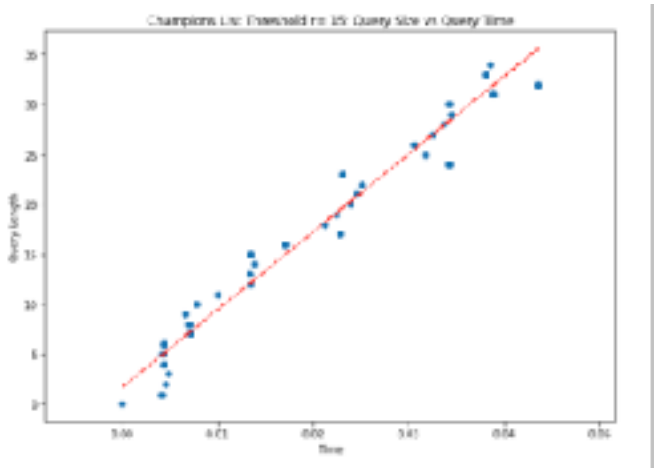
Champions List

This is using the document with the highest frequency per term in the query. The first graph is the situation where the champions list always take the top K docID per term. This shows query time is extremely correlated with the length of the query because for every additional term we are adding K more docs to compare and score, minus the overlap with prior documents. The second graph is using a frequency threshold, where all documents with a frequency higher than a constant are added to the champions list. The second and third graph have slightly higher variance which is expected because when picking a frequency threshold, some lists of common words will be very long and the lists of uncommon words will be very short. The second and third have a frequency of r=3 and r=15

I chose to not return additional items if the champions list did not return the entire k results. This decision was made because I see the champions list as the specific documents with a high frequency. The point is to only search documents with a high frequency, not to adjust r because the documents do not contain the word wit ha high enough frequency. If K documents are not returned, I think the information retrieval system should rely on other forms of inexact search to fill in the rest of the top K values.







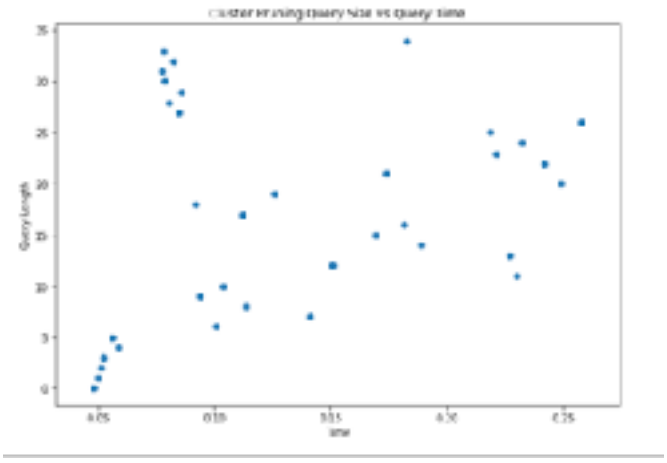
Cluster Pruning

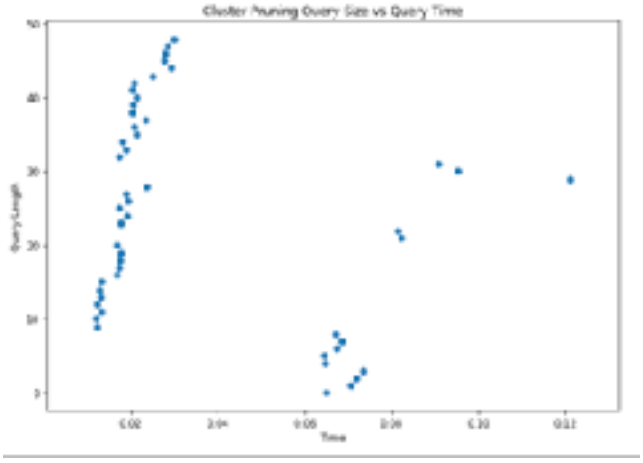
This is expensive to create these clusters. For N-sqrt(n) values we need to find the closest leader. This is sqrt(n)*(n-sqrt(n)) different vector and cosine score calculations. Roughly ~8,450 for our collection. This is far more than we do in any other inexact queries at run time, good thing this is only at index-build time. But this increases my index build from roughly .6 seconds to 5.55 seconds which is huge, this is 9.25x the time.

In the current implementation of build clusters, I have it set that each follower is tied to two leaders in order to increase the scope of documents that will be searched upon query. The current implementation of search clusters requires a minimum of two clusters to be searched. similar to the build cluster implementation choice, this is intended to increase the minimum number of documents being search in order to get more accurate results to the user. The trade off here is between search time but the advantage to clustering is that most of the work is preprocessing, once clusters have been made they are relatively cheap to search(at worst case a cluster is n-sqrt(n) but the expected size of a cluster is must smaller ~sqrt(n)).

These charts makes a great deal of sense because some clusters have a lot of terms and others have very little, so the time to search a cluster may vary widely. The graphs below show no direct correlation between query length and query time. The key factor here Is the number of followers the closest leader of the query has.

As we increase the minimum number of clusters we search, we expect the results to smooth out because we expect to get a better distribution of the number of followers each leader has and there for get a smoother result over multiple queries.





QUERY ACCURACY ANALYSIS

Briefly I will touch on the accuracy of inexact searches with exact search being the baseline. I will be using output.txt to analyze the accuracy of the inexact results.

Inexact Searches

- **Term in correct order, each 10% because k=10, so 100/10 = 10%**
- **Accuracy score for terms out of order: one minus position difference divided by total relevant results.**

The champions list used here has a threshold frequency of r=5

Query: Long distance from Bagdad

Index Elimination Accuracy:98%
Champions List Accuracy: 100%
Cluster Pruning Accuracy: 0%

Query:which stipulates that Britain

Index Elimination Accuracy:59%
Champions List Accuracy: 10%
Cluster Pruning Accuracy: 12%

Query:hello world

Index Elimination Accuracy:10%
Champions List Accuracy: 19%
Cluster Pruning Accuracy: 42%

Query: TWELVE-YEAR STRUGGLE AGAINST COMMUNIST GUERRILLA

Index Elimination Accuracy:92%
Champions List Accuracy: 0%
Cluster Pruning Accuracy: 62%

Query: BRITISH PROTECTORATES OF SARAWAK

Index Elimination Accuracy:90%
Champions List Accuracy: 10%
Cluster Pruning Accuracy: 100%

Query: CARRYING HUGE PAPER DRAGONS

Index Elimination Accuracy:39%
Champions List Accuracy: 10%
Cluster Pruning Accuracy: 25%