**Project “Wiki Search” Report**

**Submitters:**

[tomerwe@post.bgu.ac.il](mailto:tomerwe@post.bgu.ac.il) – 311144406

[omermaim@post.bgu.ac.il](mailto:omermaim@post.bgu.ac.il) - 315720011

**GitHub repository:**

<https://github.com/tomerbv/Wiki-Search>

**Google Storage Bucket:**

<https://console.cloud.google.com/storage/browser/3111444>

**Description and graphs:**

**KEY EXPERIMENTS:**

First, we experimented with the small corpus and running our engine with google colab and built most of our structural logic based on this stage. The first major question we tackled was how to keep our indexes, to which our answer was the “hashed index”. The hashed index represents a monument for both our creativity and tunnel vision, as we built a very efficient index for getting data from the disk and realized only later that our data will be held in the RAM memory. Still, we kept it to load the relevant dictionaries from the disk when the engine boots.

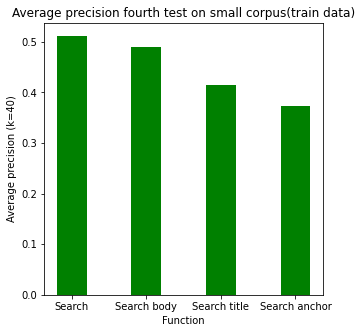
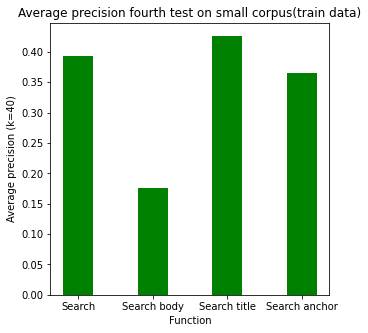
Before running the engine on the entire corpus, the only changes we made were changing how we weighted the different indexes results. The initial results were good, so we decided not to dwell on them and proceed to the whole corpus.

**As you can see from the graph below, we tested the MAP@40 value and the average query retrieval time for the query set given to us with the small corpus 5 crucial times.**

We came in testing the entire corpus with great confidence because of the results, but we later understood that the size of the real corpus changes the results drastically.

Our next key experiment was running our engine on the entire corpus using GCP. In this part we understood that we can define a instance that its remote machine has a large RAM that we can use to hold the dictionaries we created. In our first test of running the queries we saw our Map@40 drop by more than half. Here we understood that because we did not use enough important retrieval information and did not manipulate properly the corpus with LSI models, stemming, BM25 etc. our precision was hurt in such a large corpus. After that we understood we can still manipulate our calculations, so we created a new search called search\_BM25 that calculates IDF using the BM25 formula. We also used stemming on the query by adding the stemmed version of each word to the query (if it not the same as the original). Our final major experiment was to change the weights we gave to the page rank and page view of each page. Our key takeaway was that we noticed that in the correlation of results, increasing the page view weight is a better prediction than page rank. These 3 major changes helped us increase our precision by 10% (as you can see in the graph below). A good indication that our weighting was divided well for our engine is that the search method returns a higher precision than all its components. (Graph below) Our original structure played out very well in the average time retrieval for all the corpus. The strong machine of the instance even improved our time.

**As you can see from the graph below, we tested the MAP@40 value and the average query retrieval time for the query set given to us with the large corpus 5 crucial times. You can also see the difference between the precision of the search\_body with and without the BM25 calculation.**

 **After BM25 Implementation Before BM25 Implementation**

**GOOD AND BAD QUERY EVALUATIONS:**

**'migraine'**

**Average precision: 1.0**

**Results:**

ICHD classification and diagnosis of migraine, Migraine, Acephalgic migraine, Migraine-associated vertigo, Migraine Boy, Retinal migraine, Migraine (book), Hemiplegic migraine, Menstrual migraine

**Explanation:**

This is a classic example of our engines strong side that is the BM25 that takes in account rare words like migraine and their appearances in documents. In this case as “migraine” is usually more common in documents that revolve around the subject and therefore the results are very relevant.

**'Best marvel movie'**

**Average precision: 0.032**

**Results:**

Doctor Strange (disambiguation), Marvel Comics, Bret \"Hit Man\" Hart: The Best There Is, the Best There Was, the Best There Ever Will Be, Frank Cho,

Spider-Man (1994 TV series), Hobgoblin (disambiguation), Jim Starlin, Frank Miller (comics), IMDb, Marvel Girl

**Explanation:**

This is also a classic example of the downside of our engine, which is the connection between the world marvel and the word movie. Our model can give great results for movie or marvel independently but combined it lacks to find the connection between them. This could have been resolved with more advanced techniques of concepts or word embedding, that will be solved next time.

**Index Files:**

Postings\_gcp – an Inverted index for the bodies text of all relevant Wikipedia pages. (the same one made in assignment 3).

Title\_index – an Inverted index for the titles text of all relevant Wikipedia pages.

Anchor\_index – an Inverted index for the anchor text of all relevant Wikipedia pages (index for each term and the documents it has linked to).

id\_len – a dictionary of document id as key and that document’s length as value.

id\_name – a dictionary of document id as key and that document’s name as value.

pr – a dictionary of document id as key and that document’s page rank as value.

pv - a dictionary of document id as key and that document’s page view as value.

In the github repository we have a file files.txt with the names of all file located in the instance that the engine runs on.