PUBLICATION ANALYSIS FOR CHOSEN UNIVERSITIES FROM GOOGLE SCHOLAR

Rafael Marques Braga

University of Miami – School of Engineering
rxm403@miami.edu

ABSTRACT

This paper examines the acquisition and analysis of data from certain universities in Google Scholar. It looks at the implementation of a web crawler to get data from the online search engine and considers how to analyze such data. The paper also lists the difficulties with analyzing said data and proposes solutions to these issues.

***Index Terms*—** Big data, Google Scholar, web crawling, PySpark

1. INTRODUCTION

Google Scholar is one of the most accessible, leading, and complete search engines freely available as a source to the public. It has indexed information containing metadata or full text of scholarly literature (published academic articles, theses, conference papers, etc.) from countless profiles which can be organized in an immeasurable number of ways.

Google Scholar, just like Google's search engine itself, uses a web crawler, Googlebot, that scans and indexes all the useful information together [1].

This information, however, can be deemed to be very useful, if properly retrieved and analyzed. Some examples of how this data is set to be useful is by looking at correlations between authors, co-authors, different published papers or looking at patterns into publications found to be on the same topic.

With so much information indexed in Google Scholar, this is definitely in the big data analysis field as it has billions of published documents to be analyzed and information to be retrieved from.

This paper will focus on the retrieval of data from Google Scholar and the analysis of such data using big data tools such as PySpark and Google Cloud Platform in order to examine such information as fast and efficiently as possible. The main focus of the data analysis will be to find the best co-authors for each professor from different universities using the information retrieved from Google Scholar.

In order for this to be done, the goal of the paper was divided into 4 different tasks:

[1] Picking a list of 10 different university profiles from the Google Scholar website.
[2] Implementing a program to identify the top 300 professors for each university.
[3] Looking at the paper list from each professor, identifying the best co-author for the specific professor in the fastest way possible.
[4] Using the PySpark to partition the collected papers and analyze the data.

How each of these sub tasks were implemented are discussed along with some description and analysis of the data, the difficulties of such task and the optimizations that can be done when implementing and improving upon the analysis of such information.

2. IMPLEMENTATION

2.1. Task [1]

This goal of this task is to choose the universities for which its professors will be analyzed. For this, the 10 universities chosen were the following:

[1] University of Miami
[2] Florida International University
[3] Purdue University
[4] University of Florida
[5] University of Pennsylvania
[6] University of Pittsburgh
[7] Georgia Institute of Technology
[8] Stanford University
[9] Florida State University
[10] University of California, Los Angeles

Each of these universities holds a unique organization code within Google Scholar. This 20-digit number was manually retrieved from the URL of the university profile and stored in an array containing all the numbers of the listed universities. This was done so it can be easier to scrape the

information of all selected universities automatically without having to change one by one in task.

2.2. Task [2]

The main goal for task 2 is data acquisition. This is where a real-world scenario for big data analytics starts, as usually the data to be analyzed is not ready to be given or measured in a lab. Instead, when dealing with big data, technology companies like Google, Microsoft and Meta have to extract their data from all their users in order to mine the data and extract patterns.

In this case scenario, the data is to be acquired from Google Scholar.

When looking at the universities' profiles from the ones chosen in Task 1, it shows that all professors are listed in descending order when looking at their number of citations. These results are shown with 10 professors in each page, with their name, university, department, and number of citations being displayed.

For the purposes of this task, only the professor's name, number of citations and the URL for the professor was scraped, saved, and exported to a .csv file.

All the information was scrapped using the requests and Beautiful Soup library for python.

However, in order to analyze documents in a 'big data' scale, the top 300 professors' information needed to be scrapped. For this, a small program to request the next page of the Google Scholar university's profile results page had to be created to loop through the 30 first results pages (since it only shows 10 professors per page).

In the final results, the data scrapped contained exactly 10 .csv files, each containing the information (name, citations, and URL) of the top 300 professors for each university being analyzed, totaling 3,000 professors.

2.3. Task [3]

The next step is to find the best co-author for each of the 3,000 professors whose information was acquired. In order to find the best co-author for each professor, one must navigate to each professor's page and look at all the authors of the papers said professor has published.

For this to be done automatically, another scrapping solution needed to be engineered. This is done by iterating through each professor's page and looking at all the papers from each professor. All the information from the papers is then scrapped and saved to another table (later exported to a .csv file). In this new table, each table entry contained the paper's name, authors, number of citations, and the URL for the paper (this is needed in case not all authors for the paper are listed at first).

A small extra scraping part had to be added in this scenario since, if there are many authors for a specific paper, not all authors are listed at first. The program can recognize this, and if this happens, a request for the paper's URL is made, and all the authors are finally extracted from that page.

After all this data extraction is done, the final data contained over 3,000 .csv files, one for each professor. Each file contained anywhere from twenty up to thousands of entries, as it would depend on how many published documents each professor had. Each file was saved with the professor's name.

2.4. Task [4]

The next natural step after all the data is acquired is to analyze it and find the best co-author for each professor. For this analysis, PySpark was used in task 4.

In order to find the best co-author, the only information needed for this analysis is the professor's name, all the co-authors for every paper, and the number of citations for every paper. This data is then loaded in to an RDD so it can be analyzed in PySpark.

Two methods were used in order to find the best co-author. The first is to find which co-author for each professor had the most citations. The second way used to analyze the data is to look at which co-author had the most publications for that specific professor without counting their number of citations.

In the first analysis scenario, for it to be efficiently done in PySpark, the coding was done similarly to the word count problem. However, in this scenario, each co-author was treated as a 'word', it being the key, and the number of citations was then mapped with it. A reduceByKey function was then called to group all the co-authors together and add all their number of citations. If the same RDD is then sorted in decreasing order, the first entries for the RDD will show the best-coauthors for the professor by showing their total of citations.

Image 1. RDD Transformations

For the second analysis, the same idea was used, with reduceByKey in the authors used as keys. However, this time, instead of the co-authors being mapped with the number of citations, a 1 was initially mapped with it, so by the final result, it would say how many papers that specific co-author had published.

3. ANALYSIS

As mentioned in section 2.3, the analysis to find the best co-author was done through the total number of citations and papers published by each author. The time that it took for the algorithm to find the co-author for each professor was measured and analyzed.

It should be noted that due to some difficulties in the data acquisition, namely, the blocking of automatic web scraping by Google and the violation of its terms of service, most professors only had their top 20 papers listed in their .csv file. This will be further discussed later.

After using PySpark to analyze the best co-author for each professor, it was found that, on average, it took around 0.69 seconds to run through all the operations and find the best co-author per professor when looking at the total number of citations.

These results only account for the top co-author. However, to print the 5 best co-authors, this had little effect on the amount of time that was taken to the calculations.

The second method was to examine the best co-author through their number of publications. The average time to find the best co-author for each professor was 0.62s when scanning only 145 professors. When the number of professors increased to 300, this time decreased to 0.61s per professor, on average. This decrease in time could have happened due to a number of reasons. The main expected reason for the average time to have decreased is that as the number of professors increased, more errors occurred when analyzing the results. These errors would end up skipping the analysis of the professor, making the average time to appear to be faster.

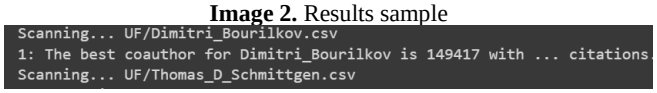
As for the calculations of the number of papers published being 0.07s faster than the total number of citations per professor, this could be due to the simpler mapping of each co-author to 1 initially rather than having to access the number of citations of each paper. This has decreased the number of shuffling functions that were called, and therefore, makes the algorithm faster.

Errors when finding the best co-author were happening at a rate of 0.6% – 3% depending on the size of the data set and the university being analyzed. This would have happened mainly due to lack of consistency in the dataset (names with accents, commas, or names written in a different alphabet). This error can easily be fixed in the future, decreasing the error rate.

It was also found that using the dataset and the specific algorithm, there was no significant difference in the processing times when using the Google Cloud platform or a local machine. This is due to the lack of parallel computing that is being used. Since there were only 20 papers per professor in the dataset, PySpark could not make much use of the partitions, almost excluding the parallel computing advantage of the equation and adding the time to access the data files from cloud storage. Some new analysis can be seen in the optimization section of the paper to further study this issue.

As for the results of the code, they were displayed as shown in Image 2. First, the best co-author is listed for each professor, and then the average time to find each co-author is shown.

Image 2. Results sample



The last result that was given was that the average number of papers for the best co-author was 5.88 papers, when looking at the best co-author by the number of papers published.

4. DIFFICULTIES

4.1. Web Scraping at scale

The first major difficulty with this project is web scrapping at scale using Google. Google's terms of service clearly prohibit "the sending of automated queries of any sort to our system without express permission in advance from Google." [2]. This is done so it does not overload their servers with traffic. Getting caught with automatic requests to Google's search engine gets the IP flagged, leading to the need to bypass Captchas to continue the search and scrapping. This was a huge difficulty in scraping the web as new APIs needed to be implemented in order to get through proper amounts of data and actually have results with huge sizes. If all data as foreseen had been captured, it is expected that Google Cloud Platform would have performed a lot faster than a local machine.

Another solution to overcome this problem is to use proxy servers. However, this is a paid solution to the problem which also slows down the crawling by a significant margin.

Having to crawl through multiple pages also resulted in getting flagged by Google and slowed down the data acquisition by a lot, since only 10 professors are shown per page in the university's profile page and only 20 papers are shown at once in the professor's profile page. For example, when a professor has over 1000 papers, multiple clicks have to be simulated. This leads to a very slow web crawling time as human speeds of navigating through web pages needs to be simulated or it leads to Google flagging and 'blacklisting' certain IP addresses due to the high number of requests to its servers.

4.2. Data inconsistency and pre-processing

Another huge difficulty to overcome in this task was the data inconsistency. When dealing with big data, especially from the web, there are multiple ways the same data can be displayed.

For example, for the co-author Valarie A Zeithaml, she had her name published in multiple papers; however she had her name listed as VA Zeithaml, V Zeithaml, Valarie A Zeithaml, among others. These slight changes are hard to overcome.

The best solution for this is to either match the names by the last names or by using algorithms in order to match similar names.

4.3. PySpark

For PySpark, the biggest difficulty was that using the data acquired, the best co-author was always the professor. To overcome that, a filter function was used to filter out all the authors which had the same last name as the professor, since that name could also be written in multiple ways. However, this increased the computational time since another 2 shuffling functions had to be added.

Another huge difficulty of PySpark was to make the algorithm be as efficient and fast as possible in order to find the best coauthor. For this, the number of shuffling functions had to be decreased as much as possible. This problem could also be further improved if the number of .csv files was decreased. However, then another column of data indicating the professor for each co-author would have to be added, creating the possibility of increasing the calculation time.

5. OPTIMIZATIONS AND FURTHER STUDIES

There are a lot of factors that could be used to further improve and optimize this project. The first one is increasing the number of studies made using Google Cloud Platform - GCP. By optimizing this algorithm to be used with GCP and parallel computing, plus with better management of data, it could drastically decrease the computational times. This is because file access and calculations would be faster when using top of the line processors and databases, which are available when using GCP.

Another further study that could be done for this project would be to compare the time it takes to find the best co-author using usual computing resources. For example, one could create an algorithm only using Python and its libraries without any big data mining resource. This could also be done using only queries in SQL. This would allow a time comparison for the calculations using python, SQL and PySpark.

Lastly, one could also look at correlations of the data and do some further data mining with the information acquired. By looking at the correlation with quantity of papers published vs. number of citations, it could easily give more insight of who could be highlighted as a better co-author for a specific professor.

6. CONCLUSION

In sum, this paper showed how big data can be acquired from the internet in a real-world scenario in order to be analyzed. Running through the analysis of the results, it could be seen how big data tools can be used to help the faster processing of big data algorithms and how tools such as PySpark were used for data processing. This paper also looked at the difficulties of processing such large amounts of data due to its data inconsistencies, lack of optimization, and difficulties getting ahold of such amounts of data in a limited amount of time. However, it can be concluded that all these difficulties can be overcome, and big data analytics is still critical to get vast amounts of information mined and analyzed.

7. REFERENCES

1. [1] R. Vine, "Google scholar," Journal of the Medical Library Association, Jan-2006. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1324783/. [Accessed: 01-May-2022].

2. [2] R. Vine, "Google scholar," Journal of the Medical Library Association, Jan-2006. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1324783/. [Accessed: 01-May-2022].