Contents

1 Intro

This project was an introduction to distributed file systems and blockchain-based storage of data. Project allows users to insert groups of papers, at which point, metrics such as most citations in a conference are updated. Users can interact with these metrics, and can even see the historical changes of which paper corresponds to the highest metric.

To run:

```
git clone https://github.com/harshasomisetty/citation_chain
cd citation_chain
python block_finder.py
python server.py
```

Then go to localhost:5000, and click around on conference links, year links, and select metrics to find the paper with the highest metric ranking in that group.

2 Architecture

2.1 Current Architecture

The overall data structure storing all the metrics is a list of lists. Essentially, an overarching list stores the metrics and list of conferences. Each list of conferences then store a list of overall metrics for the conference, and list

of years. Each list of years stores overall metrics for the year, and a list of papers.

The process of inserting papers is as follows.

First, "block finder" reads in the dataset of papers as batches. The papers in these batches are inserted in the appropriate list, and the list metrics to be recalculated are marked.

Next, each marked year list's papers are gathered as a pandas df (the doi is references against the total list of papers, and the appropriate metrics are gathered), resorted, and the top paper for each metric is stored in the metric section of the year. This process is repeated for the conference (all the papers across all the conferences' years are gathered and metrics are collected), and for the entire data base (all papers across all conferences are gathered and metrics collected). Each new batch insertion thus increases the block size, and for historical purposes, all the blocks are stored in a json object in "blockspace.json"

Finally, the frontend reads the latest block, and displays all possible conferences and years in conferences. All the possible metrics are shown, and each click to a metric reads the block and accesses the appropriate metric for either metrics per entire data set, metrics per conference, and metrics per year in conference.

2.2 Previous architecture

Previously, the top papers only were stored, and a makeshift filesystem was constructed after reading the block.

First block finder reads in the data set, and according to a predetermined pattern of inserting documents into the application, creates and inserts blocks of papers into the current chain.

Each block itself is a JSON object, that contains metrics as keys, and values as a greatest-to-least ordered list of papers according to that metric.

Each creation step then involved inserting new papers metrics into the lists, while maintaining orderliness.

Finally, after all the blocks are created, the init directories by block method reads in the latest block, and recreates a file structure that mirrors the data in the block. Specifically, the first paper in a metrics list is clearly the greatest metric, so that paper would be linked to by the corresponding metric link in the root directory. Then, sub-directories are created for each unique conference, and according to the stored papers, sub-directories of years are created for the conferences. Each subdirectory has links to the paper represented by the metric.

3 Collection of Research Paper Metrics

3.1 APIs used for collecting DOIs:

There are many research papers related to different domains. But for our project we are considering only a limited number of conference papers related to the Computer Science field. The efficient way to distinguish between research papers is the DOI. A DOI, or Digital Object Identifier, is a string of numbers, letters and symbols used to uniquely identify an article or document, and to provide it with a permanent web address (URL). A DOI will help the reader easily locate a document from the citation. Think of it like a Social Security number for the article you're citing it will always refer to that article, and only that one. While a web address (URL) might change, the DOI will never change. So the first step in our project is to collect a list of DOIs which we want to show on our webpage and build a ranking system on those dataset. There are numerous ways to get a list of DOIs from the internet, but we used two specific APIs to automatically populate the DOI array.

1. IEEE provides us with the API which can be used to Query and retrieve metadata records including abstracts for more than 5 million documents in IEEE Xploreő including Journals, Conference Proceedings, Books, Courses and Standards. So we used the API to retrieve a list of DOIs with a few keywords as the search parameters. For example , if you need a list of 100 DOIs which are related to the Computer Vision field from the years 2015 to 2017, we can write a simple query request="`https://ieeexploreapi.ieee.org/api/v1/search/articles?index_terms=software&max_records=100&start_year=2015&end_year=2021&start_record=1&apikey=yourapikey`" x=requests.get(request). Now x has the list of DOIs along with their information.

Link : `https://developer.ieee.org/docs`

1. Scopus also provides us with a huge database of research papers information. They have a large number of research papers related to many different fields. We can simply download the list of papers DOI for different conferences for different years which is in the form of a csv file. Using pandas we convert that into a dataframe and only extract the DOI column and append to our original list of DOIs.

Link : `https://www.scopus.com/sources.uri`

3.2 APIs used to collect metrics

Now that we have a list of DOIs, we can extract a lot of information about the papers using DOI as an index which distinguishes one paper from another. We classified different information related to a research paper to be a metric for our ranking system. There are different APIs available to achieve this task:

- Arxiv `https://arxiv.org/help/api/index`
- Crossref `https://github.com/fabiobatalha/crossrefapi`
- Springer `https://dev.springernature.com/example-metadata-response`
- SemanticScholar `https://www.semanticscholar.org/product/api`
- Scopus `https://dev.elsevier.com/api_docs.html`
- IEEE `https://developer.ieee.org/docs`

But for our specific use case we used Scopus and Semantic Scholar APIs which gave us an almost consistent dataset.

1. Scopus also provides us with a Python-based API-Wrapper to access Scopus database: pybliometrics. pybliometrics is an easy to use Python library to pull, cache and extract data from the Scopus database. More information available here : `https://pybliometrics.readthedocs.io/en/stable/` . pybliometrics provides one class per Scopus API Access Point. We used classes like:

   - scopus.AbstractRetrieval([identifier,]) - Interaction with the Abstract Retrieval API.
   - scopus.AffiliationRetrieval(aff_{id}[,]) - Interaction with the Affiliation Retrieval API.
   - scopus.AuthorRetrieval($\text{author}_{\text{id}}$[, refresh,]) - Interaction with the Author Retrieval API.

   AbstractRetrieval is used to extract all the information regarding the research paper. AffiliationRetrieval is used to extract all the information about the affiliations which are related to the research paper. AuthorRetrieval is used to extract all the information about the authors from the research paper.

2. Semantic Scholar also provides the RESTful Semantic Scholar Academic Graph (S2AG) API as a service to the global research community. The API is a reliable on-demand source of data about authors, papers, citations, venues, and more that allows linking directly to the corresponding page on semanticscholar.org for more information. Paper and author lookup are available from their Academic Graph service. More information available here :

`https://www.semanticscholar.org/product/api`. The /paper/search endpoint will perform a keyword search for papers. Eg: `http://api.semanticscholar.org/graph/v1/paper/search?query=literature+graph` There is a python library available on GitHub that aims to retrieve data from Semantic Scholar API. `https://github.com/danielnsilva/semanticscholar` It programmatically accesses paper and author data from the API.

3.3 Metrics Used

#	Metric	API	Description
1	Citations	Semantic Scholar	Number of citations
2	References	Scopus	Number of references
3	Authors	Scopus	Number of authors in the research paper
4	Pages	Scopus	Number of pages
5	Citation Velocity	Semantic Scholar	Citation Velocity is the average rate at whi
6	NumOfAuthorsofAffiliations	Scopus	If a research paper has 3 affiliations , it will
7	NumOfDocumentsofAffiliation	Scopus	Sum of all papers published by the affiliatio
8	NumOfDocsbyAuthors	Scopus	Sum of all papers published by the all the a
9	hIndexAuthors	Scopus	ă Sum of all the h index of authors from th
10	PlumX	Scopus	PlumX Metrics provide insights into the wa
11	Page Rank	Semantic Scholar	ranking method is based on citation networ

3.4 Page Rank Algorithm

Thousands of research papers are published every year and these papers span various fields of research. For a new researcher, it becomes a very difficult task to go through the entire repository of research papers in order to determine the important ones. The term important is subjective but it can be assured that a research paper that is popular will be important in most cases. There can be several ways of determining whether a research paper is important depending on the field of work, conference of publication, etc. Research papers cite other research papers from which they derive

inspiration and there exists a well connected graph structure among the network of research papers. The importance of a research paper is directly proportional to the number of research papers that cite it. We have used this concept in our algorithm. We will build the citation network defined as a graph, with each research paper representing a node and the citations representing the edges in the graph, the edges being directed ones, directed from the citing node to the cited node. The first step uses paper citations and creates a data structure mapping each paper to its inlinks and outlinks. The next step is to implement the iterative PageRank algorithm. The iterative PageRank algorithm starts with initializing all the candidates to a constant value, generally unity and then it iteratively modifies each candidate's score depending on the score of the candidates that point towards it. It stops when all the candidate scores converge, i.e. become constant. The PageRank algorithm is based on the fact that the quality of a node is equivalent to the summation of the qualities of the nodes that point to it. In this case, quality refers to the score of the research paper.

4 Conferences, Year and Number of Research Papers

Conference
2015 29th Brazilian Symposium on Software Engineering
2015 International Conference on Interactive Technologies and Games
2019 SoutheastCon
2020 IEEE International Symposium on Antennas and Propagation and North American Radio Scie
ACIRS
ACIT
ACM Comput. Surv.
ACM Comput. Surv.
ACM Comput. Surv.
ACM Comput. Surv.
ACM Comput. Surv.
ACM Comput. Surv.
ACM Comput. Surv.
ACM Computing Surveys
ACM Computing Surveys
ACM Computing Surveys
ACM Computing Surveys
AIKE
APEIE
ASE
ASONAM
AST
ASYU
AUTEEE
ApPLIED@PODC
ArXiv
ArXiv
BCD
Big Data Min. Anal.
BigComp
BigMM
C2I4
CCAI
CCDC
CCDC
CEI
CIBDA
CICED
CICN
CIMPS
CISTI
CNNA
COMPSAC
CPP