Investigating collaboration within DCU through citation graph analysis and academic search

Anthony Reidy 18369643
Kian Sweeney 18306226

Abstract

This short paper lays out our intended idea for the last assignment of the CA4022 module. An initial study of researcher relationships within DCU's school of computing revealed little collaboration. We intend to explore this observation further. First, we plan to scrape faculty member's *Google Scholar* and *DORAS* profiles to extract additional nuggets of information using the BeatuifulSoup and Selenium libraries on PySpark. Next, we plan to explore our observation further by employing graph algorithms using GraphX. In an effort to increase collaboration, we propose an academic retrieval system, *DCU Academic Search*. Finally, we discuss the allocation of tasks to each individual group member. Overall, we believe that the results generated from this investigation will be crucial for understanding research partnerships within DCU and we hope that DCU Academic search may provide avenues for cooperation, especially for early researchers who may not know the expertise of certain faculty members.

1 Introduction

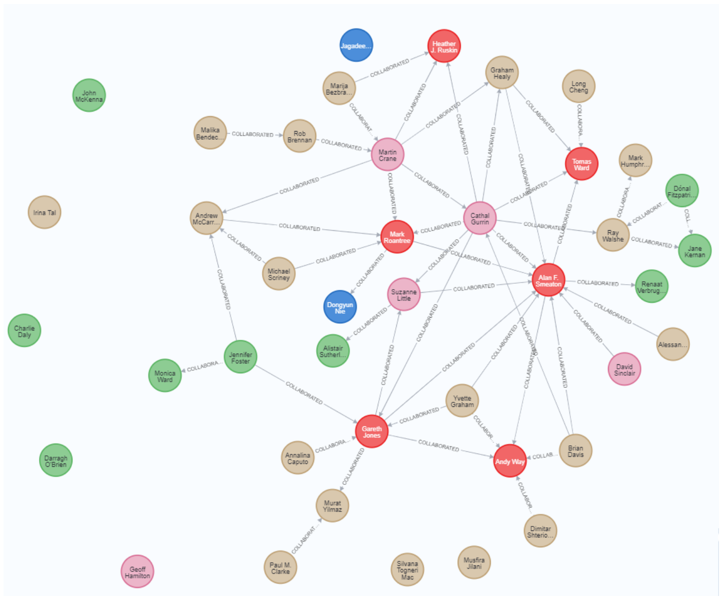

Figure 1: Graph displaying researchers that have collaborated at least once. Researchers are colour-coded by their position within the School of Computing.

Currently, there is relatively little cross collaboration between faculty members in the school of computing at Dublin City University (DCU). Figure 1 is a graph depicting current collaborative efforts between researchers. Our project is essentially composed of three formal modalities:

- Scrape data from Google Scholar and Doras
- Construct a graph and perform graph analysis
- Construct an academic retrieval system.

2 System Architecture

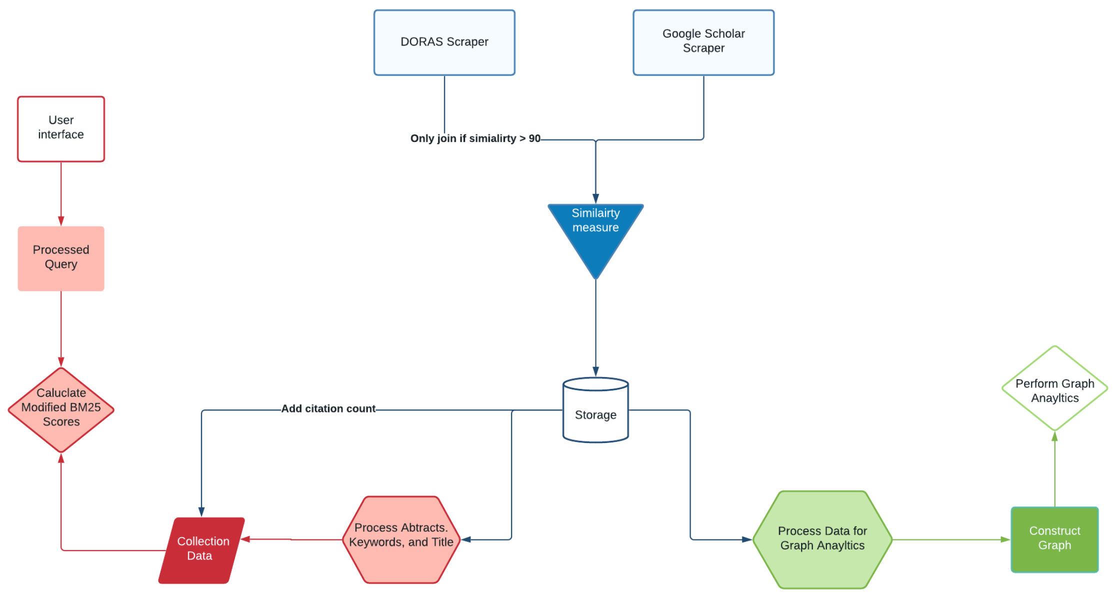

Figure 2: System Architecture

2.1 Modality 1

We plan to scrape papers listed on 41 staff member's google scholar profiles from 2011 to early November 2021. The paper title will then be used to match the same paper on Doras (DCU's open access repository) to gain additional metadata such as the keywords of a particular paper and publisher. Note, there are often slight differences between paper's title. We will employ levenshtein distance distance at a threshold of 90 to match papers. As DORAS's upload rate stands at approximately 50.1%, only about half of the papers have the enriched metadata. In addition, not every researcher in DCU has a Google Scholar and/or DORAS profile.

2.2 Modality 2

Next, multiple transformations will be performed to ensure data readiness for graph analysis. For example, a file will be manually constructed mapping a researcher to their aliases. Faculty member's names tend to vary greatly, particularly researcher's who have had long careers. For instance, Gareth F Jones, Gareth F Jones and and Gjf Jones are some of the aliases of Prof.

Gareth Jones. Data sanitation will also be performed as multiple errors were discovered when reviewing DORAS. For instance, a paper on Monica Ward's page does not credit her as an author. In addition, the number of papers (np) and citations (c) from a particular research partnership will be calculated. Two undirected graphs $G(N, E)$ will be constructed that consists of a finite set of nodes N (researchers) and a finite set of edges E. With each edge E of G, there will be an associated real number w(E), called its weight.

$$w(E) = np + 0.5(c) \tag{1}$$

The only difference between the graphs is one **will** include collaborates **from outside DCU** (*Graph1*) and one **will not** (*Graph2*). Finally, a bipartite area of interest (paper $\longrightarrow$ keyword) will be transformed into a monopartite similarity sub-graph. The graph (*Graph3*) will contain keyword $\longrightarrow$ keyword relationships where the weight, $w(E_2)$, will be:

$$w(E_2) = \text{Number of common papers} \tag{2}$$

If two keywords don't have any common papers, **there exists no relationship**. A variety of graph algorithms will be employed to model these graphs.

Triangle Count & Local Clustering Coefficient: Triangle count determines the number of triangles passing through a node in the graph. Clustering coefficient is the probability that the neighbours of a particular node are connected to each other and utilises triangle count within it's algorithm. The average clustering coefficient will be used to determine if the community's research efforts are tight knit or sparse. Additionally, the local clustering coefficient for each researcher will be calculated. This will be correlated with a particular faculty member's position in an effort to extrapolate extra meaning. This from of analysis will be applied to *Graph1* & *Graph2*

Louvian Modularity: Louvain modularity is used to find communities, hierarchies and/or evaluate different grouping thresholds. Nodes can fall into two levels (a final and intermediate community). This algorithm will be applied to *Graph1* only. Researchers who are islands in this *Graph1* form their own community and provide no new insight. Therefore, they are removed from this analysis. The results of this algorithm will be correlated with membership of the various research centres at DCU to provide additional insight.

Label Propagation Algorithim - Finding Super Keywords: DORAS provides a feature called uncontrolled keywords which are semantic categories representing a section and/or all of a paper. Such keywords are usually fine-grained. Keywords are also dependent on user input. Examples include game theory, software developers' personalities and language pairs. We estimate that there are 1268 unique keywords. With hundreds of thousands of publications being written every year, clustering such keywords in "Super keywords" may be an interesting topic, allowing researchers to access publications faster. Our primary desire is to answer "Can keywords be accurately grouped into larger communities that share some semantic similarity based on common papers?" This algorithm will be applied to *Graph3* only.

2.3 Modality 3

The title, keyword(s) (if available) and abstract of paper (combined together) will form the "documents" or search units of our information retrieval system. Researcher's names will be the retrieval units. These documents will undergo pre-processing such as stop word removal, and stemming. The document and the amount of citations a paper achieved will be indexed according to its author(s). We also distinguish between a search "request", as entered by the user, and the processed search “query”. The processed search “query” will undergo the same pre-processing steps as outlined earlier. Our retrieval system will use a modified version of BM25 with a citation factor added. Thus, the amount of citations a relevant paper achieved will contribute to an academic’s ranking. The matching score, $MS(j, q)$, of a document j according to query q, will be given by:

$$\left(\sum_{i=1}^{N} cfw(i) \times \frac{tf(i,j) \times (k_1 + 1)}{k_1 \times ((1-b) + (b \times ndl(j))) + tf(i,j)}\right) \times (w \times cc(j)) \tag{3}$$

where $cfw(i)$ is the collection frequency for term i, $tf(i, j)$ is the term frequency of term i in document j and $ndl(j)$ is the total number of terms in document j divide by the average number of terms in the collection grouped by document. $cc(j)$ is the number of citations garnered by document j. $k1$, w and b are constants that control the effects of tf(i, j), cc(j) and length normalisation respectively. N represents the total number of terms in a query. We plan to run a set of experiments to find the optimal values for k1, b and w.

Thus, a researcher’s(r) matching score, $RMS(r, q)$, given a query q, will be given by:

$$\sum_{k=1}^{K}(MS(j,q)) \tag{4}$$

Where $MS(j, q)$ is the matching score of a document and K represents the top 10 ranked documents produced. This is to ensure researchers who have produced a lot of papers are not over-represented.

3 Technologies

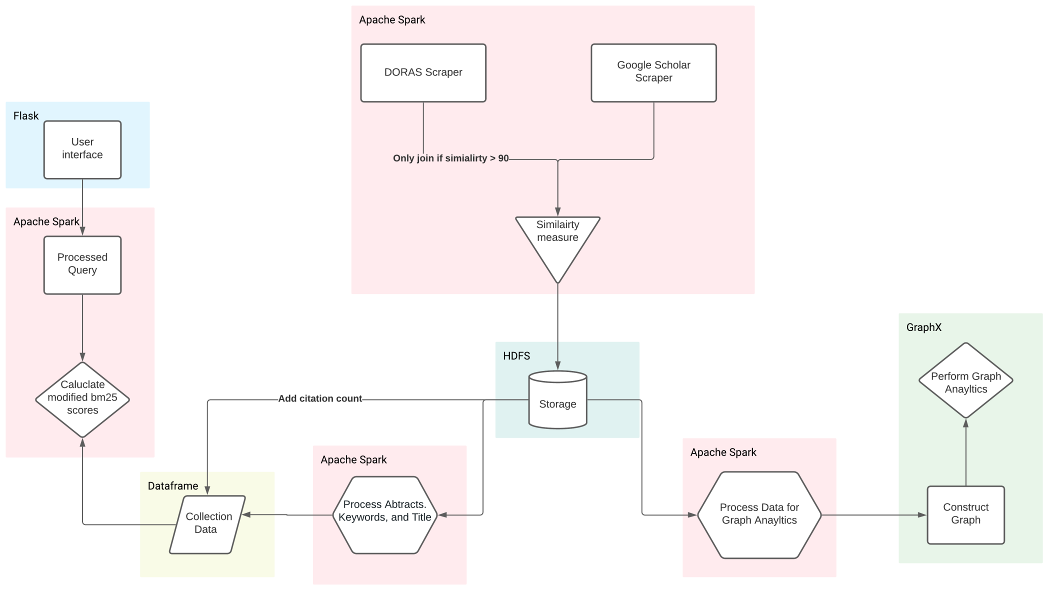

Figure 3: Potential Technologies

Figure 3 denotes the the potential technologies used in this along with association with the described modalities.

PySpark: PySpark is our primary technology for this application as it enables scalable analysis. The large amount of data to be extracted and prepared for our application further promotes the use of PySpark. We plan to separate our scrapers across individual worker nodes, enabling parallel scraping of data. PySpark has an in-built levenshtein distance function which will be used to calculate title similarity. The python libraries BeatuifulSoup and Selenium will aid us in scraping. For the graph pre-processing, we plan to use the transformations operators offered by spark's data-frames. For the academic retrieval pre-processing, we will also use the aformentioned transformation functions. In addition, stemming, lemmatization and the removal of punctuation, and stop words will be aided by Spark NLP's annotators.

HDFS: Hdfs will be used to store data from the scrapers. We chose HDFS over simply leaving it in Spark data-frames as we are more proficient in this technology. Also as our dataset will be static after scraping, we want to take advantage of HDFS'S distributing storage before we proceed to develop modulaties one and two.

GraphX: We estimate *Graph1* to contain 4,647 nodes and 131,694 edges. *Graph2* will have 41 nodes and 5,337 edges. While *Graph3* will consist of *Graph3* will contain 1,268 nodes 60,046 and edges. The size of the graphs require us to consider scalability issue in the (graph) analytics itself. One of the big advantages of GraphX over other graph processing systems and graph databases such as neo4j, is its ability to treat the underlying data structures as both a graph, using graph concepts and processing primitives, and also as separate collections of edges and vertices that can be mapped, joined, and transformed using data-parallel processing primitives. Furthermore, GraphX contains in-built functions for all the graph algorithms we intend to use.

Dataframe: Dataframe is chosen to store *DCU academic search*'s collection and system of storage we will use on PySpark. Dataframe enables fast execution when computing matching scores due to distributed computation ability. Thus, quicker query response times will become a feature of our app.

Flask: Flask will serve the user interface. We examined two popular python web frameworks, Django and Flask. As this is our first experience of web development, Flask was chosen as it is more light-weight compared to Django, enabling quicker uptake. Flask also comes with a built-in development server and a fast debugger compared to Django. Finally, integrated Unit testing is a feature of flask that we believe will be beneficial to ensure the robustness of our overall system.

4 Distribution Of Work

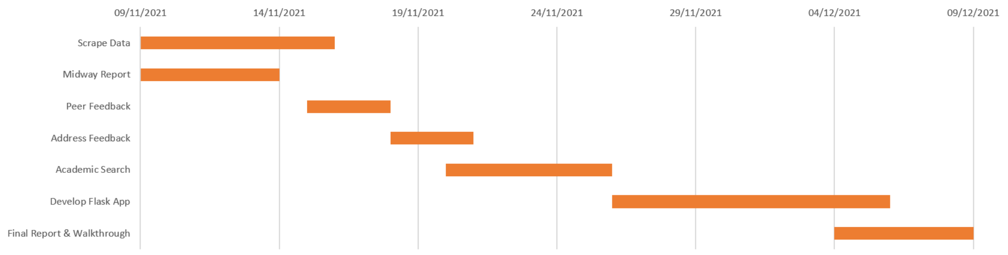

Figure 4: Proposed Work timeline

As per figure four, this is our proposed work timeline for this project and what we intend to undergo. Starting off we intend to scrape our data online using Selenium and BeautifulSoup libraries. We intend to split this in a 50:50 way with Kian taking the scraping of google scholar and Tony undertaking the scraping of doras. We also intend to evenly distribute our tasks regarding our graph analysis and use of graph algorithms for analysing our data. The final development stage of our project will again be split in an equal manner. For developing our flask app Tony will focus on the development of the backend for our DCU academic search app while Kian will focus on the front end.

References