Bibliobuild: a citation network visualisation and exploration tool

Philippa Gilsenan B.A. (Mod.)

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Future Networked Systems)

Supervisor: Yvette Graham

August 2021

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Philippa Gilsenan

August 31, 2021

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Philippa Gilsenan

August 31, 2021

Acknowledgements

My sincere gratitude to my supervisor, Prof. Yvette Graham, for all her guidance and support throughout a bizarre year. To my family, for their unwavering love and support, thank you (and I will stop talking about the course now). Finally, to Cathal and Igor, thank you for providing much needed laughter, and for making the long, long, hours bearable!

Philippa Gilsenan

University of Dublin, Trinity College
August 2021

Bibliobuild: a citation network visualisation and exploration tool

Philippa Gilsenan, Master of Science in Computer Science

University of Dublin, Trinity College, 2021

Supervisor: Yvette Graham

Citation networks are important sources of data that have many applications. These applications have not been fully explored, and existing tools are either hidden behind subscriptions or outdated. This dissertation, a citation network visualisation and exploration application, aims to create a modern, fast, and open tool. A micro-framework service and graph database are utilised to quickly perform data queries and transform the results into an easy to understand and explore citation network. Filters provide users with control over the citation network, allowing them to refine its contents, while statistics provide insights into the network as a whole. The AMiner dataset used contained over 5 million papers. Coupled with the dataset, the network branches out to show papers indirectly linked to the results matching the search term, creating populous citation networks to explore. The tool distinguishes inward and outward citations. This results in two ways to view the citation network, each providing different beneficial insights, one showing the foundation for the paper, the other showing the relevance of the paper in future work. A prototype citation prediction model is implemented that further utilises the data in the dataset, and is used to highlight the papers predicted to be relevant in the future. This dissertation proves that a fast and useful citation network tool can be created from open source data.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

In an academic paper, the work of others and sources of information are acknowledged through citations. While citing helps avoid accusations of plagiarism, they also direct the reader to sources of background information and sources that backup the paper's arguments. It is almost universal for academic papers to contain a number of citations. Combined, the papers and the citation relationships create a network of papers commonly referred to as a citation network.

A citation network is a directed graph displaying the citing (or referencing) relationships between papers. It is a directed graph as each relationship has a defined start and end paper. Each paper has its own node in the graph and the relationships are displayed using lines connecting the nodes. The appearance or structure of citation networks can vary significantly due to the importance or recentness of a paper, and the level of activity in its field.

Citation networks can provide a variety of insights. These networks can be used to explore a new field and to discover new papers to read. It is also a form of social network, displaying who is collaborating with who. Pattern analysis is another form of insight, and has been used to identify *citation cartels*, authors colluding to cite each other's work unnecessarily in order to inflate the number of citations, and to predict the direction or trends in fields. Another use of citation data is in bibliometrics, the statistical analysis of publications, which has applications in hiring and the assignment

of funding in academia.

However, citation networks are not without their own issues and challenges. Paper access is frequently protected behind paywalls and private companies guard their citation datasets closely, for example Google Scholar has no open API. Another difficulty is the unreliability and incompleteness of data. One such cause is in parsing the citations as there is no standard format for citations. Papers can also be indexed multiple times resulting in duplications, and citation data can quickly become stale. This can make accurate, well populated citation networks difficult to build.

1.2 Motivation

A citation network is a form of discovery and insight into papers, fields, topics, and authors. However, manually creating a citation network is incredibly time consuming, involving searching for a paper, identifying the papers it cites and that cites it, and then repeating the search process for each paper found. Subsequently, the process of visualising and studying the results of the search process introduces an additional layer of time, effort, and potential issues.

The motivation behind this project was to automate the process of building citation networks, providing users with a fast, accurate, and user-friendly application. There was a focus on providing granular visibility into the individual papers in the network, enabling the user to view the citation network as a whole, as well as studying its components. Users would be given control over the network, refining the contents of the network shown, a functionality missing from several existing solutions. Finally, a degrees of separation approach was undertaken. This would return not only the direct citing relationships of a paper, but it would branch out to return those papers indirectly connected to it. This provides the user with an increased ability to explore a network, discovering new papers, patterns, and insights.

1.3 Key terms

There are a number of terms that are used frequently throughout this dissertation. Explanations of these terms are provided here to provide understanding from the outset.

- *Inward citations* - the incoming references to a paper from other papers.
- *Outward citations* - the outward references of a paper to other papers.
- *Primary paper* - a paper that matches the search phrase.
- *Secondary paper* - a paper that has a direct relationship with a primary paper.
- *Tertiary paper* - a paper that has an indirect relationship with a primary paper.
- *Hops / degrees of separation* - the number of steps in the path between a paper and the primary paper.

1.4 Structure of dissertation

Chapter 2 presents a summary of the state of the art including the approaches to, and components in, forming citation networks as well as the different methods of visualisation. Chapter 3 outlines the design of the application and explains the reasonings behind the design decisions made. The processes undertaken in creating the application are outlined in chapter 4. An evaluation of the application is given in chapter 5. Finally, in chapter 6, a conclusion to the project is provided.

Chapter 2

State of the Art

This chapter provides a review of the state of the art and a background of challenges and opportunities in creating citation networks. The review starts with an essential component of any citation network, a source of citation data. It is followed by a review of the different database approaches of storing citation data and a survey of the various means of visualising networks. Finally, to provide a more useful application by measuring the relevance or importance of papers in a network, the current bibliometrics and existing machine learning approaches to citation prediction are investigated.

2.1 Datasets

A bibliography tool to aid users in discovering new papers requires bibliographic data. The quality and quantity of the data has a significant impact on the usefulness of the application.

Three different options for sourcing citation data were considered: building a dataset, using an API to an existing citation dataset and downloading an existing citation dataset.

2.1.1 Building a dataset

The importance of citations cannot be understated in academia. Aside from recognising the work of others, citations play a key role in the creation of bibliometrics, the analysis

and generation of statistics of publications. These metrics impact employment and funding [1].

However, there is no universally accepted and complete source of citation data. There are several large datasets managed by private companies, including Google Scholar and Microsoft Academic Service (MAS). Nonetheless, due to concerns about the accuracy of the data and the reliance on private companies, work continues on building new citation datasets.

Open-source solutions have been proposed in response to these concerns. One such semi-automatic process includes [1] which extracts and links references from manually scanned publications. It was proposed that librarians carry out the necessary labour involved in scanning the publications. This would not only require the support and time of librarians, but would not be as fast as utilising existing online stores of scans, and manual scanning would introduce a significant source of human error.

Difficulties in parsing citations

The parsing of citations themselves can be error prone which results in unreliability and incompleteness [2] in citation datasets. While referencing styles, such as the common Harvard referencing style, provide a strict structure to building references, and reputable publications require specific referencing styles to be abided by, incomplete citations are still a common occurrence.

Citations include fields such as the paper's title, author(s), journal, proceedings, volume, issue, year of publication, page range and URL. As such, a missing field or the occurrence of fields in the wrong order can result in imprecise parsing and therefore, an incorrect citation. As found in [3], a missing page number can result in the year being inaccurately identified as the page number.

This inexactness has been tackled with models, such as a heuristic model in [4], which breaks a reference into tokens and creates features based on n-grams, different capitalisations, punctuations, dictionaries containing common first and last names as well as publishers, and numerical formats including formats for year and page range.

A knowledge base approach focusing on scientific papers is taken in [3]. Various rules were created to identify whether a PDF contains a scientific paper, including looking for keywords such as ACM, IEEE or Proceedings. Due to a minimum word count

threshold, in order to disqualify PDFs predominately containing images, old scans of papers where text appears in images are ignored. This poses an issue as papers several decades old are still cited and considered valuable sources of information today. The citations in the scanned paper are checked against a knowledge base to identify the fields contained in the citation. The result of which is then analysed to match it to the most likely referencing format before applying a template specific algorithm to extract the citation metadata.

Opportunities in parsing citations

Despite the difficulties involved in accurately parsing citations, as not all citations have equal importance, developing a citation parser introduces new opportunities to examine the contexts of citations.

With access to the contents of papers, the location and frequency of the citations can be extracted. The location of a reference, for example whether in the background or the results, may speak to its importance. A source that is cited frequently can be considered more valuable or relevant to a paper than a source with a singular reference [5]. This relevance may be considered positive or negative, without the context of the citation it is impossible to know the sentiment of a reference. For instance, a disgraced paper may be regularly referenced and held up as an example. With access to the full text, sentiment analysis, a popular form of machine learning, can be applied to provide additional insight to citation networks.

While it involves considerable work to parse citations, creating a parser provides an opportunity to gather additional data about the citations. This data can later be used to examine and visualise citation relationships in new, productive ways that would not be possible when confined to the citation data contained in most existing citation datasets.

2.1.2 Datasets

There are a number of large datasets that could be utilised in lieu of building a new dataset for this project, specifically DBLP or Microsoft Academic Graph. It should be noted that, despite their widespread use, the datasets are also susceptible to reliability and completeness issues.

Microsoft Academic Graph

Microsoft Academic Graph (MAG) is one of the most utilised citation datasets. Snapshots of its citation dataset can be downloaded, which requires an Azure subscription, and are updated approximately every fortnight [6].

Its data has been used to analyse research trends in the field of computer science [7], create static rankings of articles using article-level Eigenfactors [8], enhancing author profiles by visualising the impact of an author's published work (also with article-level Eigenfactors) [9], and to visualise the impact of research [10] to name a few.

However, as announced in a blog post in 2021 [11], Microsoft is retiring its Microsoft Academic Services (MAS) after 2021 which means the data behind the MAG will no longer be updated.

DBLP

The Digital Bibliography & Library Project (DBLP) is a dataset of computer science bibliographies. It does not require a subscription and is free to download [12] in XML.

It has been used to explore venue metrics as an alternative to citations metrics in scoring authors and institutions with a Gaussian model [13], visualising the changing of focus in topics over time [14] and detecting communities from citation networks [15].

2.1.3 APIs

While no papers were found using APIs to build citation networks while researching the state of the art, a number of citation APIs do exist. A brief introduction to three APIs are given. One API in particular, Google Scholar, is frequently used as a benchmark in tests on citation dataset coverage and staleness.

Google Scholar

Google Scholar is one of the largest players in the bibliographic database arena. It crawls the internet to build its dataset. As it indexes by paper, not by journal or field, it has the widest range of papers, including papers from conferences and journals, academic papers such as theses and dissertations, as well as academic books. New papers are added to Google Scholar each week. Although, updates to existing papers

can take from six months to over a year to appear as it involves recrawling the website it indexed the paper from [16].

There is no official Google Scholar API. However, there are third party APIs, such as SerpApi [17], and Python libraries, such as Scholarly [18], that can be used to query Google Scholar and receive formatted JSON in response. It is also possible to send a generic Python request which retrieves the contents of the webpage which can be manually parsed.

Elsevier

Elsevier is a scientific publisher that indexes academic papers to provide analytical and search tools. Its citation dataset is comprised of paper metadata, including abstracts, and citation data. It contains over 82 million papers and over 1.7 billion citation relationships gathered from over 7000 publishers [19].

There is an official Elsevier API to query its dataset which requires an API key. There is institutional access which enables those tied to a university to create API keys without a subscription. However, there is a weekly quota on the number of requests that can be made, and restrictions on the number of requests per second [20].

OpenCitations

Unlike Google Scholar and Elsevier, OpenCitations is a not-for-profit organisation. Its origins began in 2010 as a one-year project. However, it has since grown to contain nearly 66 million bibliographic entries and over 1 billion citation relationships [21].

There are a number of official APIs to query OpenCitations' citation datasets. No information could be found on request quotas or throttling rates.

2.1.4 Considerations

Staleness

Two factors of staleness are considered. The first compares the API and dataset approaches. API calls return largely up-to-date data with brief periods of staleness due to caching. Datasets on the other hand are snapshots of a particular point in time and inevitably incur a larger degree of staleness. Datasets require regular updating

which introduces a new task and a possible source of error due to parsing, formatting and ingesting the new snapshots.

The second factor considers how often the datasets themselves are updated. MAS has a slower update rate than Google Scholar which means the data in the dataset is stale for longer periods of time and issues with the data, such as duplicate entries, take longer to be fixed [22].

Coverage

One study comparing the coverage of Google Scholar, MAS, and DBLP in the field of engineering ranked the sources in that order, with Google Scholar having indexed 98.96% of the 1067 studies, MAS 97.46% and DBLP 93.43% [23].

Another study corroborates this ranking, but makes an important note that while Google Scholar contains more citations than MAS, the citations in MAS originate more equally across the different disciplines, while Google Scholar citations are more focused on computer science [22].

2.2 Storing data

If utilising a dataset, it can be stored in a database in order to query the data efficiently. There are two approaches to storing networks in databases: native and non-native. In the native approach, the database is designed and optimised to store graphs from the outset. On the other hand, in the non-native approach, the database is adapted to store a graph. With a non-native approach, as the database is not optimised for graph search, performance and scalability can be affected.

Citation networks are composed of nodes, also called vertices, and edges, representing relationships between the nodes. Graphs can be directed or undirected. If the graph is directed, such as a citation graph, the edges in the graph have source and target nodes. It may be required to store properties or attributes about the nodes and edges. For example, if a node represents a paper, it may be desirable to also store paper properties such as DOI, title, or field of study.

2.2.1 Non-native approach

Relational databases are a common non-native approach. They store data in tables made up of rows and columns. Connections can be established between tables using foreign keys.

When a citation network is stored in a relationship database, the nodes are usually stored in one table and edges in another, with each row in the node table being a unique paper and each row in the edge table being a unique relationship [24]. For larger citation networks with millions of papers and citations, this can lead to lengthy tables.

Join operations are used to connect the two tables when building a citation network. This method becomes more inefficient as the size of the tables grow [24]. Another consideration is that nodes and relationships may end up stored distantly from each other in memory which can impact query latency.

2.2.2 Native approach

Graph databases are newer technologies and take a different approach to storing node and edge values. Unlike in relational databases where the relationships between the nodes are made at the run time of a query through joins, graph databases store relationships as their own entities [25]

Graph databases are operationally more extensive and expensive when storing the data. When a node is created, the node and its edges are stored next to its adjacent nodes, in a method called index-free adjacency. This results in fast retrieval times as the nodes have physical RAM addresses and point to the adjacent nodes [26].

2.2.3 Comparison of native and non-native performance

One aspect where a native graph database solution is clearly superior to a non-native solution is in the scalability of the database. In non-native solutions such as relational databases, performance noticeably decreases as the size of the database grows. The increasing size involves more complex and computationally expensive join operations as the whole table must be scanned to gather the nodes and edges required to build a citation network [24]. In contrast, due the storing of adjacent nodes in close proximity,

graph databases tend to have a much more consistent level of performance as the database grows [26].

This performance comes with an increased storage cost. In a comparison of disk space usage between a native graph solution, Neo4j, and a non-native relational solution, MySQL, it was found that the native solution required approximately double the amount of storage [24]. Using an AMiner citation dataset, it was found that to store 2 million nodes and 8 million edges it took approximately 544 MB in MySQL and nearly 1,000 MB in Neo4j [24].

Existing projects

MySQL, a relational database management system, has previously been used in many citation projects. In a project to incorporate citation visualisations into author profiles, a snapshot of the MAG dataset comprised of 127 million papers and 528 million citations was stored in a MySQL database. While the performance of the system was not discussed in detail, it was noted that to gather the data for an author profile could take several minutes [9].

Another project, which visualises the impact of an author's work using concentric circles containing those papers citing the author's papers, stored the same MAG data in a MySQL database. However, there was no mention of performance in the analysis of the system [27].

Neo4j was used in [24] to store and query the AMiner dataset, containing 2 million papers and 4 million citation relationships. The project sought to identify the topics of the papers using a Latent Dirichlet Allocation (LDA) model, and show the topic circulation over time. The propagation of the topics could be used to see if the topic was gaining or losing popularity and could help direct research.

2.3 Visualisation

The application had a focus on helping users, especially those new to a field, discover papers to read. The data in a citation network can be visualised in a wide variety of ways in order to achieve this goal. However, care needs to be taken to show enough data to be useful to the user without the volume of data or manner of displaying it

being overwhelming or cluttered.

A range of different approaches were reviewed from simplistic tabular methods to three-dimensional interactive methods. The visualisation methods proposed in the projects below primarily deal with inward citations alone, while this project utilised both inward and outward citations. This introduced additional factors to the visualisation, such as bi-directional relationships and potentially larger numbers of nodes.

2.3.1 Range of approaches

Two projects, BiblioViz [28] and PaperCube [29] reviewed a range of visualisation methods when creating their applications.

BiblioViz was devised from studying the systems arising from the InfoVis 2004 contest [28]. They decided a graphical method was the most appropriate form to display the relationships in the metadata between different papers. Furthermore, while these relationships are better grasped through visual means, having a large variety of distinct visual forms could be confusing or overwhelming for users.

The authors considered visualisation forms across four categories, namely table, network, node (minus the network) and other, before deciding on a two-pronged approach of network and table. A key factor in their decision was the pre-existing familiarity of users with these forms. Tables were utilized to highlight trending data, with the x-axis showing the year. On the other hand, network graphs were utilized to highlight relationships between papers.

PaperCube was created to display citation networks with a focus on highlighting collaboration, or relationships, between authors [29]. The process of discovering where a paper lies in the general context of its field or topic can be arduous and time-consuming. The authors state this process is less intuitive when references are simply listed instead of displayed in a visual manner. A visual manner can make relationships between papers clear and does not require navigating away from the original paper and therefore losing focus on the original paper.

The project incorporated five different methods of visualising the data. Circle view, which shows a large central circle surrounded by smaller circles showing the paper's citations, and collaborators view were the most popular methods from their user study. These views are graphed as interconnected nodes. Papers per year, tree map, and paper

graph were less popular with their users due to the large amount of data displayed.

The authors concluded that multiple views were not necessarily beneficial, instead adding features to give the user more control over the view is preferable. The user study found the paper views were more beneficial than the author views, but the author view was still a popular functionality. By displaying both the paper and author networks, papers that appeared irrelevant to the user's research but are actually important to their research can be highlighted.

2.3.2 Combination approach

Similar to BiblioViz with its double view utilising a network graph and tables, ImpactVis [10] also took a combined approach. The authors also tackled the limitations and issues with measuring the impact of an author by quantitative measures such as h-index or citation count alone. For example, does the impact of the research change over time?

The authors followed the Visual Information-Seeking Mantra [30] which outlines good practices on displaying information, namely "Overview first, zoom and filter, then details-on-demand" [30]. They were also conscious that displaying large amounts of data could create clutter, devaluing the usefulness of the tool, and through informal interviews with prospective users, they learned that users wanted the ability to explore the data themselves interactively.

A matrix is displayed in the center of the ImpactVis user interface with both axes showing the year range in question. The cells of the matrix contain a coloured box with its size proportional to the number of citations and publications. Bar charts are attached to the left, top and right sides of the matrix. On the left and right sides of the matrix, bar charts display the number of publications and citations respectively by year. On the top of the matrix sits a bar chart, which shares the same x-axis as the matrix, that displays the number of citations received within each given year. ImpactVis provides a clean, uncluttered way to provide the user with multiple views of the same data.

2.3.3 Tree approach

A family tree approach was taken in [31] to create a Java applet to visualise citations. The papers that were referenced by a paper were considered ancestors and the papers citing a paper were considered descendants. The years were listed horizontally across the top of the page. Below each year, the first twenty-five characters of each paper title published in that year in the tree were displayed in a column.

The approach worked well to provide an overview of how large numbers of papers interacted. However, while written in 2012, there was no ability to zoom and the applet required Java to be installed. As large numbers of papers are displayed across a wide range of years, it is difficult to view the titles of the papers. Finally, no data aside from paper titles, authors and abstracts are visible.

2.3.4 3D approach

CyBiS, Cylindrical Biplot System, was created to visualise the results of scientific paper searches [32]. The authors also found that displaying results in a linear, text format made it difficult for users to identify correlations and other relationships between the papers returned. Similar to PaperCube [29], multiple views of the results were undesirable as users may lose focus of the current context.

CyBis is a Java web application that displays papers as spheres in a 3D cylindrical format. The publication date is represented along the x-axis with the y-axis and z-axis showing paper relevance to the search terms. This relevance is shown using the height and depth of the sphere. Colour coding was used to display the number of citations, with the darker the shade of red denoting the more citations.

The creators of CyBiS, like the creators of ImpactVis [10], followed the practices in displaying information laid out in Visual Information-Seeking Mantra [30]. The user hovers over a sphere (paper) to get a quick view of the key paper metadata, additional metadata is displayed in a table if the user clicks on the sphere. The tool also enables the user to rotate and zoom.

2.3.5 Concentric circle approach

A concentric circle approach was taken in [27] using data from Microsoft Academic Search. The authors proposed a solution to display the overall influence of an author's total body of work. Similar to ImpactVis [10] and CyBis [32], the authors note how visualisations enable the user to discover patterns and relationships in data more easily than linear or quantitative methods alone.

The center node represents an author's entire body of work. Spiralling out from the center node are all the publications which cited a piece of work in the central node (secondary nodes). The citations among these publications are shown with thin, grey lines between the secondary nodes. The authors used Eigenfactor, a measure of the importance of a journal, to scale the size of the nodes. Finally, the nodes in the circle are colour coded by field.

2.3.6 Timeline approach

A timeline approach showing the papers written and authors collaborated with for each year is proposed in VisualBib [33], a web application to create, visualise, and share bibliographies. Two datasets were used: Scopus, which required the user to have their own subscription) and OpenCitations.

The bibliography was graphed chronologically, and icons and author names were used to visualise the bibliography with dashed lines representing citations. However, paper titles are not shown in the interface by default and their number of citations is not visible. This approach is beneficial in showing an author's collaborators over time while limited in visualising elements with multiple, bi-directional relationships.

A useful feature of the application was the generation of URLs that contained query strings. This enables the sharing of exact searches as well as saving pages with bookmarks.

2.4 Graphing and relationships

An expectation from the beginning of the project was that one way the application would display citation networks in the user interface would be as graphs as they are a standard practice in displaying relationships. The nodes in a graph represent objects,

or the papers in a citation network, and the edges are the lines that connect the nodes that show relationships, such as author and reference relationships in a citation network. Graphs can be used to identify patterns, show trends over time, and discover connections between elements that would be difficult to find in textual forms.

2.4.1 Issues arising in graphing

While graphs are an intuitive way to display and read nodes and relationships, the method is not without issues. One issue is storing graphs as the choice of storage has performance and storage implications.

Five common implementations of graph data structures, namely edge list, adjacency list, incident list, adjacency matrix, and incidence matrix were compared in [34]. The efficiency of the algorithms and their space requirements were the main criteria used in the evaluations. The authors decided upon adjacency lists to use in their reference mapping tools due to its speed at finding cited and referenced papers, its relatively fast search and small space requirements. The paper also discussed issues surrounding citations, including incomplete citation databases, duplicate papers, and the best citation format for parsing.

Another issue arises when the structure or layout of the graph is suboptimal for readability and comprehension. Tessellation, when applied to graphs, is where there are no elements overlapping and no gaps in the graph. A lack of tessellation means network graphs can become difficult to read as they become more dense causing labels for nodes to start overlapping and connections between nodes become blurred.

An approach to generating tessellated network graphs in proposed in [35]. A conventional force-directed algorithm is employed to reduce clutter in the network graph, such as overlapping nodes or edge crossings. Voronoi tessellation is then applied; the area of the screen is divided into cells, allowing for a cell for each node element. To find the optimum location of each node, the following process is repeated: the barycenter of each Voronoi cell is located and the corresponding cell co-ordinates is moved to the updated barycenter. The authors also incorporated a type of anisotropic Chebyshev distance, a method to adjust the width of the Voronoi cells, to ensure the label fits within the cell.

2.5 Bibliometrics - measuring scientific impact

The data in a citation network has other uses aside from visualisation. Bibliometrics, the creation of statistics from publication data, can be calculated to provide further insights into the papers and authors in the network.

2.5.1 Existing quantitative metrics

Three common bibliometrics to measure an author's influence are the h-index, g-index and i10-index. The h-index was created in 2005 by Hirsch [36]. It calculates the value, h, where h of the author's papers have at least h citations.

G-index, an alternative to h-index, was proposed by [37] in 2006. It takes into account the distribution of citations. Articles are sorted by decreasing number of citations. The g-index is the largest unique citation count, g, where the sum of author's top g article's citations are at least g^2.

The i10-index was proposed by Google Scholar Citations (GSC) in 2011. GSC enables authors to view and explore their citations as well as adding publications to their profile. As the name suggests, i10-index measures the number of papers with 10 or more citations [38].

2.5.2 Quantitative metrics concerns and pitfalls

Gaming the metrics

Concerns around the accuracy of the indices are well reported. As Google Scholar collects data using crawlers across a broad range of websites, from publishers to universities to personal, it can result in duplicate citations.

These duplicate citations can have a favourable effect on the bibliometrics. In a study using a small dataset comprised of the body of work of 11 researchers, it was found that 8 of the 11 researchers had at least one duplicate publication in their GSC profile [39]. When duplicates were removed from the citation lists, half of the citation counts were deflated by two or more citations. Of the 11 researchers, over half had their h-index decrease by at least 1. Similarly, the i10-index of four of the researchers dropped by up to 4 points.

As previously mentioned, due to the wide catchment of Google's crawlers, multiple versions of the same article may be indexed. There is another way these duplicates can affect bibliometrics. GSC provides the functionality to merge different versions of the same article from different sources. A disreputable author may merge unconnected articles to inflate their h-index as the citations of the papers are merged in the process. This works best when the papers have little to no overlapping citations. However, consequently this manipulation approach may have an adverse effect on other metrics based on the total number of citations such as the i10-index [40].

Additionally, self-citations are not distinguished from external citations [41]. This means a researcher can frequently cite their own work in order to improve their indices. Also, the varying reputations of journals are not taken into account. One could argue that publications in prestigious journals that require strict adherence to standards and ethics with low acceptance rates should be distinguishable from disreputable publications and should be included as a weight in measuring impact.

Citation life cycle variations

Citation patterns can be vastly different depending on the field. In their study [42] of the citation cycles in four fields of economic research (applied, applied theory, econometric methods, and theory), the authors found noticeable differences between the peak number of citations, duration of peak citation levels, and active citation lifespan across the fields. Applied and applied theory overall performed much better over the three factors than econometric methods and theory. In particular, theoretical papers had the lowest peaks and durations. While econometric methods followed the same patterns as theoretical papers, successful papers excelled and tended to perform the best overall across the four fields. This posits that an index that does not take field into account would not be able to accurately state the impact or position of a researcher in their field.

The limitations of quantitative metrics such as the h-index was also discussed in [10]. The indices do not provide insight into how the researcher's impact changes over time. Researchers in fields where citation life cycles are slow burners and those in fields with short burst citation life cycles are viewed the same. Another limitation outlined was the influence of collaborators on an individual researcher. The indices do not

include factors such as the researcher's placement on the author list. As a result, the lead author and the least contributing author benefit equally from the citation count of a paper.

2.6 Machine learning approaches to relevance

The bibliometrics discussed in the previous section do not provide insight into the future. They are focused on historical data and to provide present-day insights. However, a large volume of academic work is published each year. This considerable quantity leads to questions such as: what work is groundbreaking or important, and what is the future direction of a field?

As bibliometrics play an important role in hiring and funding, the challenge of predicting the future relevance of a paper, author, or field has garnered more attention in recent years.

Machine learning models have been created that examine the different citation life cycles, and use a range of inputs, or features, to predict future citation counts. These features predominately fall into three categories, author-centric, venue-centric, and topic-centric. Due to the heterogeneity of academic fields, it is difficult to develop a one-model-fits-all solution.

2.6.1 Existing approaches

Citation life cycle approach

The rate of citation for scientific papers is generally thought to follow the following format: the citation growth rate increases for approximately the first two to three years, before levelling off and then finally declining [43].

A two stage process to predict citation counts is proposed in [43]. The authors identified six categories of citation patterns:

- *PeakInit*: papers which peak between 2 and 4 years after publication, after which experience exponential decay.
- *PeakMul*: papers which have a number of different peaks over time.

- *PeakLate*: papers which receive few citations in the first five years of publication, after which they experience a peak then exponential decay.
- *MonDec*: papers which peak in the initial year following publication, after which they either remain constant or decrease.
- *MonIncr*: papers which experience only increases in citation counts or remain constant for an extended period of time.
- *Other*: papers that did not receive at least one citation each year.

Papers were assigned to one of the six categories using a Support Vector Machine model which was trained using features across three categories: author-centric features, e.g. author productivity, venue-centric features, e.g. long term venue prestige, and paper-centric features, e.g. total number of authors. The model was then trained for each category to predict the number of citations a paper would receive. Author-centric features, in particular the average productivity of the authors, were found to be the best at predicting the number of citations.

Impact of early citers

The impact of early citers on citation prediction was examined in [44]. The experiment included 10,000 papers that were published prior to 1996. Four models were evaluated on their ability to predict long-term scientific impact (LTSI) with citation prediction. The models were linear regression, Gaussian process regression, classification and regression trees, and lastly support vector regression (SVR).

The same features were used to train all models. There were four main categories of features, including paper-centric, author-centric, and venue-centric, similar to other prediction models, as well as early-citer centric. The early-citer centric features included publication count, citation count, and co-authorship distance. As papers can have multiple authors, the highest publication and citation counts across all authors is selected for each early citing paper and then averaged to get a single value for the feature. For the co-authorship distance, the minimum co-authorship distance across the authors for each early citing paper is selected and then averaged.

Of the four models, the SVR model had the most accurate predictions. The authors found that early citers have a considerable impact on future citations and LTSI.

Early citers were classified as influential and non-influential with influential authors having high publication and citation counts. Influential early citers were found to have a negative effect while the opposite was true for non-influential early citers. One reason provided was that citations from influential early citers are potentially stealing attention. This effect was found to be stronger where the early citers are nearer to the paper's authors in the collaboration network.

Convolutional neural network approach

A convolutional neural network (CNN) to predict the citation frequency of papers as a means to measure its possible scientific impact was proposed in [45].

The current number of citations alone is not an efficient means of predicting future citations as it favours older papers. It is also not appropriate for fields with slower citation growth rates, such as the social sciences. Citation growth rates are not uniform and extreme patterns such as *sleeping beauties* can occur.

Machine learning models can produce classification outcomes, for example good or bad, or regression (numerical) outcomes. The authors chose a CNN regressor. It takes features of a bibliographic network as its input and outputs its predictions of citation counts for 10 years after the paper publication. The authors' model was evaluated on a subset of the AMiner dataset, including papers and citations from 1980 to 1985.

Similar to [43], author related features improved the prediction accuracy followed by venue related features. However, keyword features were found to have only a slight improvement on efficiency.

Link prediction problem approach

Another application of predicted citation counts is gauging the prospective impact or influence of authors. The citation of papers creates a network between the authors of the papers. This is a directed network as the edges or relationships between papers have an origin node, the paper doing the referencing and an end node, the paper being referenced. Weights can be assigned to these edges to represent a variety of factors such as number of citations the paper received, total number of citations of the authors and so on.

In [46] a novel approach is undertaken by treating the prediction process as a link

prediction problem where both links and weights are predicted. This process has an added difficulty as citation networks evolve over time. Link prediction does not factor in time, such as the age of the link. Two citation networks with an identical structure may have evolved very differently to their current states.

Two citation datasets, AMiner and HEP-Th, were used in the experiment. Twelve samples were extracted from the datasets, six from each source. The six AMiner samples contained citations from 2002 to 2009 and the six HEP-Th samples from 1992 to 2003. A dynamic metric was introduced to factor the changes in citation network trends. The metric measured the proximity of researchers. In conjunction with the dynamic metric, features such as a paper's number of citations, and the number of citations from unique researchers were used to predict citation counts.

The authors highlighted the need for feature selection to improve the accuracy of citation prediction.

2.6.2 Topic citation prediction approach

Citation prediction can have higher-level bibliometric applications. Researchers and institutions can be concerned with how a field as a whole is set to grow or decline over the coming years instead of the performance of a specific paper.

A model for predicting research topic citations was proposed in [47]. It utilised two citation datasets, MAG with abstracts from AMiner. Over 50,000 papers were involved, all of which were focused on software engineering. The authors acknowledged the focus on a singular field meant the model would not translate fully to predicting other fields. It was also noted that a topic may be winding down, but influential or fundamental papers in that topic can continue to be highly cited.

2.6.3 Feature engineering

There are a range of data points that can be used as features in a citation prediction model. However, some of these data points may have more of an impact on the accuracy of the model than others. By identifying which features are most efficient, the model can be simplified and more papers can be predicted as not all papers will have the full list of data points.

This feature selection task was addressed in [48] to see which features across four categories were best at predicting future citation count. The categories included paper features, author features, venue features, and network features, which are used in the majority of citation prediction models reviewed. The authors examined the impact the features in the categories had firstly as a whole and then individually on the citation count predictions.

Three models were used: deep neural network (DNN), multiple linear regression (MLR) and support vector machine (SVM). The citation counts were predicted for 3 and 5 years after publication. The ArnetMiner dataset, containing papers from computer science publications, was used in their experiment. A subset of the data, 10,000 papers from 2000 to 2005 with more than the average number of citations for the year published, were selected.

The SVM model was the best of the three models tested at predicting the citation counts. It was found that removing the author-centric features had the largest impact on the accuracy of the predictions, and using only the author features had similar results to using all the features. When the author features were removed individually and tested, the authors' average number of citations was most indicative of future citation count. There was a smaller, but noticeable, difference after removing venue-centric features.

2.7 Conclusion

The sections in this chapter covered the key components in the formation and structure of citation networks which greatly informed the creation of this application. The existing literature provided key insights that helped improve the performance, usefulness and design of the application, and helped avoid pitfalls.

Chapter 3

Design

3.1 Introduction

This chapter outlines the design and structure of the application, from the source of the citation data to the formation of a citation network in the user interface. It follows a chronological structure in which each subsequent section builds upon the prior section.

3.2 Citation dataset

The project required a source of citation data. It was decided to use a pre-existing citation dataset which would be stored in a database. The reasons were two-fold, the first was to improve performance by eliminating the need to call external APIs which would introduce network latency and would expose the project to request throttling and quota limitations. The second reason was to gain more control and usability over the data. An API provides restricted access to a data source, in order to build a citation network, each paper would need to be queried individually to get its citations. Subsequently, each of those citations would need to be queried individually to get their citations. In a local database, a custom query can be written which would retrieve the data in a much more efficient manner.

The AMiner citation dataset, created by [49], was selected and downloaded from [50]. The latest version at the time was v13.7 created on the 14th of May, 2021. It contains 5,354,309 papers and 48,227,950 citation relationships. The dataset's contents

comes from DBLP, a computer science bibliography database which is formatted in XML while the AMiner dataset is formatted in JSON [50].

Each paper has its own entry in the JSON file. The entry contains fields such as the paper's title, year, ISBN and DOI, as well as a UUID for the paper. It should be noted that the existence and population of these fields vary across the papers. Also included is an *author* key which contains a list of dictionaries, each containing the UUID of an author and their name. Finally, the *references* key contains a list of UUIDs of the papers it references.

A full list of the fields in the dataset and their population frequencies is provided in Appendix A.2.

3.2.1 Other data sources considered

The challenges and opportunities involved in creating a citation parser were appealing. However, it would take a considerable amount of time, and access to a significant volume of PDFs to parse enough papers to create a minimum viable citation dataset. This would result in less time to, and fewer possibilities in, visualising the citation data.

Initially, development focused on retrieving citation data from live API calls. Two APIs were tested, Google Scholar and Scopus. When using Google Scholar, requests were throttled after approximately 10 calls. Compounding the low request threshold, multiple requests were required to retrieve a paper's metadata, authors and citations. Also, the API did not have the functionality to return a paper's own references. The other API, Scopus, queries Elsevier's citation database. It requires an API key with a weekly quota of 5,000 requests for authors and 20,000 requests by title [51]. It returned fewer results than Google Scholar, and like Google Scholar it does not return a paper's own references.

3.3 Database

After the decision to use a citation dataset, the next key design decision was to determine how to store the citation dataset. The advantages and drawbacks of using native versus non-native database solutions for storing graphs are discussed in subsec-

tion 2.2.3. It was decided that a relational database would not be an optimal method of storing the citation dataset. A key reason being that as the data is not tabular, and creating a query to return a citation network would involve multi-level joins which would have performance implications.

Due to the dataset's interconnected nature, a graph database was a more appropriate choice as it performs queries on connected datasets significantly faster [52][53][54]. This is pertinent as a paper may be connected to thousands of other papers bi-directionally, with inward and outward citation relationships.

The graph database solution chosen was Neo4j [55]. It is open-source, therefore it did not require a subscription, unlike alternatives such as Amazon Neptune [56]. Critically, it is fast and supports Python, which was the planned language of the application. It is widely used, with comprehensive documentation, and an active community.

In a Neo4j database, three types of data are stored: nodes, edges, and attributes. Nodes store individual entities which are connected by edges. In this case, papers and authors would be stored as nodes while the relationships, references and authorships, would be stored as edges. Properties of the nodes and edges such as title and year for paper nodes and name for author nodes would be stored as attributes.

3.4 Service

In order to create the citation network, first the data needed to be fetched from the database and formatted. Secondly, a user interface would be required to handle user input (in order to search for a paper), and to visualise the citation network. A key design decision to make was whether to combine these two functionalities into a single application, or create a citation API service and another service to handle the web application.

3.4.1 Data operations

It was decided to use Python for the data operations in the first step. The driving factors behind this decision were familiarity with powerful Python data manipulation and analysis libraries, and the ability to connect directly to the Neo4j database with a Python driver. The speed of the data libraries was particularly important as, due to

the nested nature of the database results, several formatting processes would be needed to convert the results into a table-like structure.

It was then decided to use a micro-framework to host the data operations. A micro-framework is a minimalistic framework used to create applications or services quickly without including some functionalities such as user accounts or roles and permissions. The service would act as an API, with different endpoints to handle the different requests for data from the user interface.

3.4.2 User interface

Two options were considered for building the user interface. Web applications can also be built using Python micro-frameworks. For the data service, the endpoints would return data formatted in JSON, for the user interface the endpoints would render webpages created from HTML templates.

The other option would be to create a standalone user interface application using a JavaScript framework such as ReactJS. It would make the UI fast, the handling of interactivity easier due to states, and components of the web page could be updated without reloading the page. However, it would take longer to set up and would require maintaining two code bases, which would take development time away from the data and feature aspects of the application. As there was more of a focus on the applications of the data than the frontend element, as well as for manageability, it was decided to build the user interface using a Python micro-framework.

3.4.3 Service framework

Flask [57] was chosen as the framework for the data and web services. It is lightweight and flexible. Importantly, it can return both JSON for the data process and webpages in HTML for the visualisation process. It also meant the data and web services could be combined into a single service.

A template engine called Jinja2 [58] is used to create the webpages. A template for a page or component can be designed in a HTML script. These templates have all the same functionalities as regular HTML scripts, including importing Cascading Style Sheets (CSS) and JavaScript files from local storage and websites. However, as a Jinja2 template, if statements, for loops, while loops, variables are possible. When

Flask renders the Jinja2 template, it passes as arguments the variables necessary to populate the page. For example, if loading a page to show a detailed view of a paper, a dictionary containing the attributes of the paper could be passed as an argument and the values used to populate the template.

Jinja2 also supports template inheritance, meaning one template can inherit or import another. A base Jinja2 template was designed to contain a common structure to apply to every page. This template included the navigation bar at the top of the page, universal imports, and styling such as background color and font. This prevented code repetition or duplication, and changes to the application design could be made in one file which would be applied to every page in the application. The template for each page in the application inherited this base template.

3.5 Citation network design

One of the earliest design decisions was to display the citation network in a graph format. As covered in section 2.3, multiple papers found graphs to be the best form of displaying networks as, unlike other forms of visualisation such as tables, it enables showing multiple relationships between the nodes clearly and in a format intuitive to understand. The papers would be displayed as nodes, or dots, in the graph, and citations shown using edges, or arrows, between two nodes.

Graphs can display a range of data attributes without the use of words which frees up additional space. For example, by color-coding and scaling the size of the nodes and edges. As the graph could display a large number of papers, this meant that the relevance of the papers could be communicated to the user using node size, making it easier to identify papers of interest in the network.

3.5.1 Components of the network

Citation direction

A key design decision regarding the contents of the citation network was to only show one direction of citation, inward or outward, at a time. One reason for restricting the citation direction was to reduce clutter in the citation network graph. By including both options, the direction of the citations would need to be very clearly distinguished

in the graph. While this could be done with arrows and color-coding, the additional space this would require would make the graph harder to read. It would also result in more overlapping elements in the graph.

Another reason is that inward and outward citations provide different insights into a paper's relationships. Inward citations show the relevance of the paper as the citations are from newer papers, they can also show if the work proposed in the paper has been built upon. Outward citations show the foundation of the paper as those papers were used to form the basis of the paper's proposal. By showing the outward citations, the foundational history of the topic can be viewed going back in time. When learning about a field, papers that have the most relationships, and therefore can be argued are most relevant in the citation network, can be identified for research.

Author nodes

Author nodes are only displayed in the citation network when the user has searched by an author's name. The motivation behind this decision was to reduce the number of different node types in the graph. Each node type provides different information, and by limiting the variety of nodes shown more nodes of one type can be displayed at a time, providing a deeper level of insight into the citation network. A paper can have a single author to thousands of authors. If all the authors were displayed in the graph by default, it would reduce the number of papers that could be displayed, and as a result, it would show fewer degrees of separation.

The user would still be able to view all the papers written by a particular author by using the author search. The author nodes are displayed in the author search as there can exist multiple authors with the same name. The papers of all of these authors are returned in the results, and by showing the author nodes, the papers can be easily distinguished by author using edges in the graph.

Example of a search by paper title

A citation network is shown in Figure 3.1. The user searched by the exact title of a paper, *Peak classifier for bar code waveforms*, and selected to view the inward citations in the graph.

3.6 Future citation count prediction

Displaying a large citation network is useful for gaining insight into the activity in the network as well as making identifying patterns easier. Notwithstanding, it requires more work to judge the importance or relevance of a paper when more nodes are present. One popular way of measuring the importance of a paper is its citation count. However, comparing the citation counts of papers with different ages is not a fair comparison.

As a result, a machine learning model would be applied to predict the future number of citations for the papers. This would enable the system to rank papers for relevance more fairly, and identify recent papers that are potentially going to become relevant in their field. As the item being predicted was a paper's predicted future citation count, a regression model was appropriate for the task as they predict continuous numerical values. The other primary category of prediction, classification, was not applicable as it predicts discrete categories.

The citation network for each search is passed through the model to have the citation counts of its papers predicted. The papers with the highest citation counts are highlighted in the graph with a blue outline in order to bring them to the attention of the user.

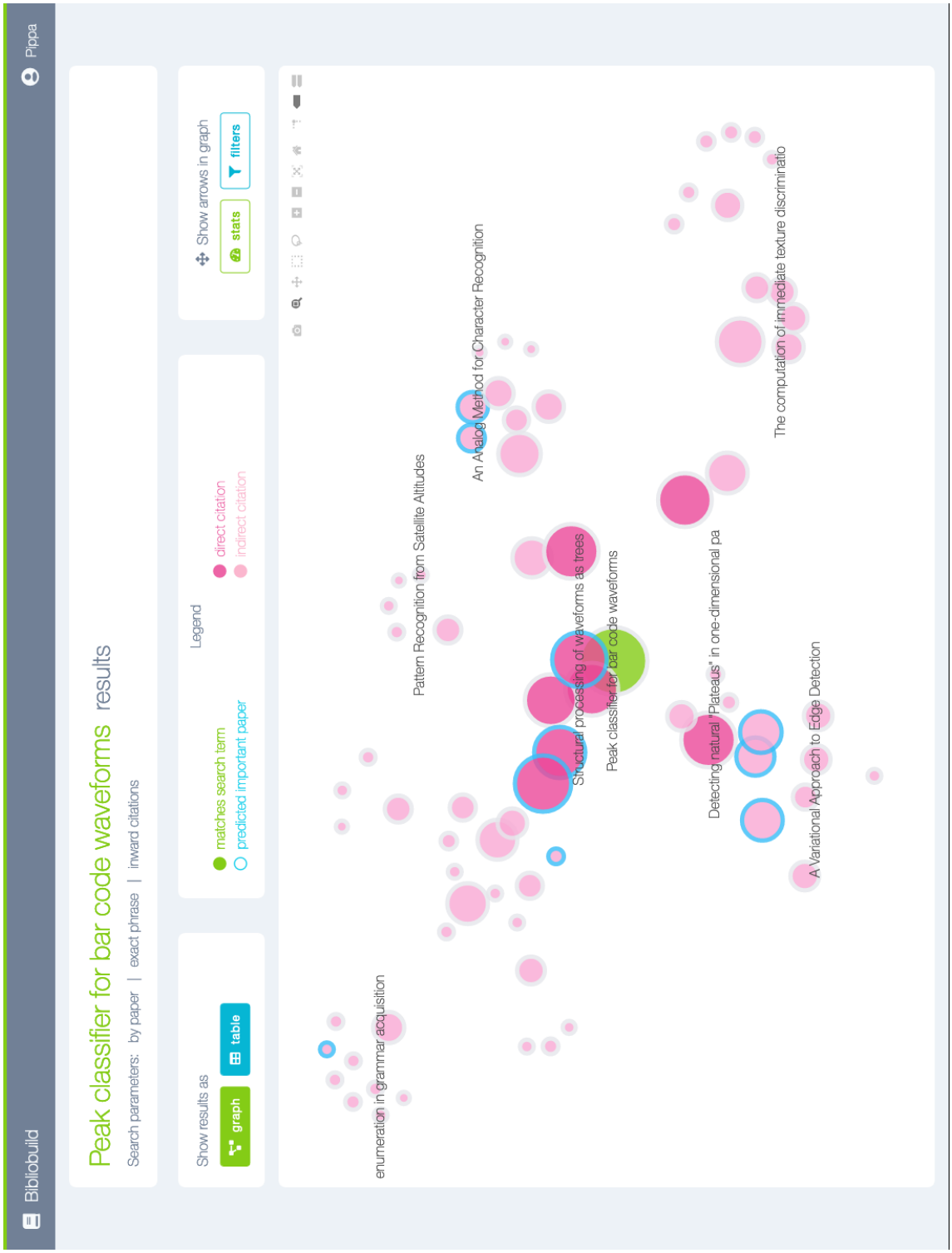

Figure 3.1: Screenshot of a citation network showing inward citations.

Chapter 4

Implementation

This chapter explains in detail the processes behind creating the application. It begins with sourcing and storing the citation data in section 4.1. After which, in section 4.2 the service to provide the application is explained. Section 4.3 explains the processes behind creating and visualising a citation network. The components of the results page are outlined in section 4.4 and the interactivity functionality in section 4.5. Finally, in section 4.6 the machine learning approach to citation prediction is explained.

4.1 Dataset and database

In a Neo4j database, both nodes in a relationship must exist in order for the relationship to be created. If uncertain whether both nodes exist, a *MERGE* query can be performed which checks for their existence, creating a node if it does not exist, before creating the relationship.

Initially, while learning and experimenting with the database, the insertion of the dataset took place using the Python driver. The ingestion was carried out in batches of 10,000 nodes until the database contained 250,000 papers. This database was used for the initial learning and testing of queries. However, this method of ingestion had performance implications.

Upon research, it was discovered that data could be ingested considerably faster using the command line tool Neo4j Admin. This would require the creation of individual CSV files for each node and relationship type. The command would ingest all

the nodes (skipping duplicates) and would then create the relationships (skipping bad relationships). Notably, this command can only be performed on an empty database.

4.1.1 Dataset formatting

In order to create the node and relationship specific CSV files, the data needed to be formatted and exported from a JSON file to CSV files. This was done in Python. The JSON file was too large (13 GB) to load in one go, causing memory issues. The *split* Unix command was used to divide the file into 36 parts. This was possible as each line in the JSON file was a different entry.

Aside from the first sub-file, all other files began with a comma. The preceding comma was removed and opening and closing square brackets were added to the start and end of the ingested text to convert it into a valid JSON list in string form. The data could then be converted to a JSON object and, from that object, a Pandas DataFrame created. Pandas [59], is a Python library for data manipulation and analysis. A DataFrame is a tabular data structure comprised of rows and columns. A DataFrame can be created from and converted to a range of data formats, including CSV files, lists and dictionaries.

Paper nodes

The id key, *_id*, could have two formats, the predominant format being a string, while occasionally formatted as a dictionary with the id accessible by the *$_oid* key. The nested id values were extracted to match the unnested string format.

Aside from id processing, the other crucial piece of formatting involved handling new line characters. These new line characters would break the data ingestion so all these characters were removed across the entire dataset. This was specifically an issue with the abstract.

The paper DataFrame column headers were then renamed so when the data was converted to CSV, the CSV headers would be in the Neo4j format. The paper nodes CSV contained multiple keys including title, ISBN, and start and end page numbers. However, only two keys were guaranteed to appear for each paper: *:LABEL* set to *Paper* and *paperId:ID* containing the UUID of the paper.

Paper relationships

The paper nodes would be connected with reference relationships. The DataFrame contained a *references* column which contained a list of papers referenced for each paper. In order to create relationships for each individual reference, each reference would need to have a different line in the CSV. The references column was *exploded* in Pandas, essentially creating a new entry in the DataFrame for each reference.

The paper references CSV contained three keys: *:TYPE* set to *REFERENCES*, *:START_ID* to the id of the paper doing the referencing, and *:END_ID* to the id of the paper being referenced.

Author nodes

The formatting of the author nodes took the longest amount of time due to the level of nesting. Each row contained a list of dictionaries with author information. This column was transformed so there was an individual row in the DataFrame for each dictionary entry. Each key in the dictionary was extracted and a new column was created to hold the key values.

The author nodes CSV contained three keys: *:LABEL* set to *Author*, *authorId:ID* set to the UUID of the author, and *name*.

Author relationships

The paper and author nodes would be connected with authored relationships. The DataFrame containing the formatted author data was used with no additional processing required. The paper-author relationship CSV contained three keys: *:TYPE* set to *AUTHORED*, *:START_ID* the id of the author, and *:END_ID* to the id of the paper.

4.1.2 Dataset ingestion

The formatting process was carried out on each of the 36 dataset files. The results of which were appended to the four CSV files. Once the CSV files were fully populated, the data could be ingested into the Neo4j database. The command used to ingest the data is available in Appendix A.1.

Step	Formatting Section	Average Duration
1	data ingestion	9.30s
2	initial formatting	1.97s
3	venue formatting	57.40s
4	paper nodes	17.78s
5	paper relationships	4.08s
6	author formatting	189.94s
7	author nodes	0.86s
8	author relationships	0.97s

Table 4.1: The average times for each section of the formatting process.

The formatting process took approximately 3 hours while the ingestion took under 10 minutes. It took on average 4.7 minutes to ingest, format, and convert each sub-dataset file to a CSV. As shown in Table 4.1, the formatting steps took up the bulk of that time. Unsurprisingly, formatting the author DataFrame from rows of lists of dictionaries to individual rows for each dictionary took the most time. A breakdown of the formatting times per sub-dataset file can be seen in Appendix A.3.

Ingestion issues

The formatting was initially developed and tested using the *aa* subset file and the resulting CSVs were used to test the ingestion process. This process identified formatting issues in the dataset, such as the abstracts containing new line characters and unescaped quotations. These new line characters would split the abstract text over multiple lines, but the subsequent abstract lines would then appear as new nodes. As Neo4j expects the rows to follow the same format, and contain unique identifiers which would not be present in the additional abstract lines, the ingestion process would error out. As a result, the *aa* formatting and ingesting process took several iterations to catch all the breaking characters.

Finally, due to warnings of inadequate heap size when using Neo4j admin tool, the heap size was increased from 1GB to 2GB to handle concurrent operations [60].

4.1.3 Indexes

In the initial testing of the citation database, the queries were taking significantly longer than desired. The queries were simple paper or author look-ups so more extensive queries including degrees of separation for a paper would take significantly longer than the simple look-ups. The query performance had to be improved to make the application usable.

Two indexes were added to tackle this issue. An index is a copy of part of the data in the database with the objective of making queries faster. In Neo4j, indexes can be applied on both nodes and relationships and can be on a single property of the node or edge, or multiple properties. Indexes on multiple properties are called composite indexes. There are two main drawbacks in using indexes. The copy of the data increases the storage demands of the database, and future writes are slower. The need for faster queries outweighed the corresponding increase in storage, and as the database was unchanged throughout the rest of the development, slower writes were not an issue.

Indexes created in project database

Indexes were added to the titles of paper nodes and to the names of author nodes. It was decided to index by the title and name fields instead of by the UUIDs as the user searches by paper title or author name, not ids.
The following commands were used to create the indices:

```
CREATE INDEX paperTitleIndex FOR (n:Paper) ON (n.title)
CREATE INDEX authorNameIndex FOR (n:Author) ON (n.name)
```

The indexes significantly improved the performance of the queries. In Table 4.2, the performance of two queries are compared on the non-indexed and indexed databases. One query is a look-up of a paper by its title, the other is a look-up of an author and the papers they have written.

4.1.4 Querying the database

The Neo4j database is queried using an open-source query language called Cypher. The language focuses on specifying what you want returned and not how to return the

	Field	Query	Non-Indexed Time	Indexed Time
paper	title	A solution to the problem of ...	51.95s	0.11s
author	name	Theo Pavlidis	37.13s	0.14

Table 4.2: Comparison of the query performance on the non-indexed and indexed databases.

results.

A user can search using the title of a paper or the name of an author. This can be an exact match search or a like match, where the field contains the search term. As discussed in subsection 3.5.1, inward and outward citations can provide different insights into a citation network and the user selects one of these options when searching.

An example query is shown in Listing 4.1, where a paper is searched for by its exact title and its outward citations, and the outward citations for those papers up to 3 degrees away, are returned. This query will be used in the next section to explain the structure of the Cypher queries used.

```
1  MATCH (a:Paper {
2      title: 'Experiments with Classifier Combining Rules'
3  })
4  OPTIONAL MATCH path=(a)-[r:REFERENCES]->(b)
5  CALL {
6      WITH b
7      OPTIONAL MATCH paths = (b:Paper)-[h:REFERENCES*1..3]->(c)
8      WITH c, paths, h
9      LIMIT 10
10     RETURN c, h, length(paths) as hoppies
11 }
12 RETURN a, b, r, c, h, length(path) as hops, hoppies
13 LIMIT 100
```

Listing 4.1: Cypher query to return a paper, its outward references, and the outward references of those papers up to 3 degrees away.

4.1.5 Query structure

Querying the search term

The first component of the query is the only one that changes based on the search type and match type. In this search, the user is searching by a paper title. As a result, the node type *Paper* is specified on line 1. If the search was for the name of an author, the node type would be *Author*. In the case of a LIKE search, the clause in the curly brackets is removed and a WHERE clause with wildcards is added after the closing round bracket on line 3.

Relationships

In Cypher, if a clause of the query returns no results, the final response will be null. This would pose an issue if a paper exists in the database, but it does not have any referencing relationships (secondary or tertiary nodes as outlined in section 1.3) in the specified direction. Even though the primary node, the paper, exists it would not be returned. As a result, the sub-queries to return the secondary and tertiary papers are set as optional matches, as seen on lines 4 and 7, to ensure the primary node is returned if no secondary or tertiary nodes exist.

In Cypher, nodes are specified in round brackets and relationships in square brackets. The relationship is situated in-between the known node and the result node. The direction of the relationship is signalled with arrows either side of the referencing brackets. The relationship type is set to *REFERENCES* on lines 4 and 7 as author relationships are not pertinent to retrieve the secondary and tertiary papers. By restricting the node types checked, it also improves performance. The arrow's position on the right hand side of the referencing brackets on lines 4 and 7 mean that in this case the query is looking for out-going references.

To enable hops, an asterisk followed by the minimum hop degree, then two intermediary dots, .., and lastly the maximum hop degree, is appended to the relationship type as seen on line 7.

Returning values

The tertiary query is located within a CALL clause in order to limit the number of tertiary nodes returned for each secondary node to 10. This prevents a secondary paper with a large number of tertiary nodes from constituting the majority of the search results.

The path variables initialised on lines 4 and 7, *path* and *paths*, store the route between each primary node and secondary node, and each secondary node and tertiary node respectively. The scalar function *length()* is used in the return statement to return the individual hop count for each secondary and tertiary node. The number of hops is later used for weighting and styling in the citation graph.

In order to prevent the citation graph from becoming too cluttered, a maximum of 100 nodes is returned. The *ORDER BY* sub-clause returns the nodes from closest to furthest from the primary node using the hops field so the most relevant papers to the search are returned. This ensures that secondary nodes are prioritised over tertiary nodes.

Example queries for each of the four search options are available in Appendix B.

4.2 Service

An instance of Flask was created to serve the frontend of the application and to handle requests to the Neo4j database.

4.2.1 Application routing

A Flask application can have multiple pages, created using *@route* function decorators for each page. Essentially, with routes Flask matches the URL to its corresponding function and triggers it. The Python function carries out the necessary operations to fulfil the request. At the end of the function, Flask renders the response. This response could be a JSON object or a HTML page.

4.2.2 Creating the webpages

The webpages were designed using Jinja2 templates. When rendering the templates, the values of the arguments passed could be displayed in the webpage using double curly brackets, {{ *variable* }}. Conditional statements and iteration loops could be created using similar notation. The HTML elements to be contained in the body of the statement would be preceded by {% *statement* %} and followed by {% *end statement* %}. For example, a for loop would start with {% *for x in list* %} and end with {% *endfor* %}.

Pages such as the results page had multiple components. These components had their own templates. For example, the results page included a filters panel. The panel had its own template as did the different filter input types. This modularity increased the reusability of the templates and meant the components of pages could be moved or restyled easily.

4.2.3 Application design

The pages were styled using CSS frameworks, Bootstrap [61] and Tailwind [62]. Two frameworks were utilised as Bootstrap provides structural components such as modals but has limited styling, while Tailwind provides more extensive styling functionalities but, as of the time of writing, no components.

The generic Bootstrap CSS was used. However, Tailwind enables you to customise your CSS script and configuring your own script will remove unused elements to make loading times faster. Tailwind was installed using NPM and custom stylings were added to the configuration script to provide a theme colour, button style, and font to the application. The configuration was then compiled into CSS and the script added to the project.

Form rendering and validation, for example when searching for a paper, was implemented using the Python package WTForms.

Theme colours

Tailwind has preconfigured colour palettes. Each colour in the palette contains a range of shades for that colour from light to dark. A lime green palette was selected as the

theme colour of the application. This colour would be used to highlight text, such as page headings or a paper title, and in buttons and icons. A grey colour palette was chosen for the background of the page with the navigation bar at the top of the page using a grey four shades darker.

Distinguishing the page components

The results page in particular had multiple components, including the graph, table, filters, statistics, and detailed paper view. Background colours were used to make it easier to distinguish where the components start and end. As a grey background was applied to all webpages, components were given a white background to clearly differentiate the component boundaries. The borders of the components were also rounded to soften their edges.

Distinguishing the page actions

The results page had many actions corresponding to the multiple components. These actions had different levels of importance. The primary actions of the page, such as the buttons to view the results as a graph or table, should be more noticeable than the secondary actions, for example the button to launch the statistics modal. The buttons for primary actions were given colourful solid backgrounds, while the secondary actions had a transparent white background, but with colourful borders and button text. The different stylings would make the page easier to navigate, especially for new users.

4.2.4 Connection to the database

A Neo4j Python driver was used to connect the service to the database. The connection was established upon the creation of the application, with the URI and credentials retrieved from environment variables. The same connection is used for all requests in the lifetime of the instance.

4.2.5 Query response formatting

The data returned by the Neo4j driver is a Neo4j result object. In order to access, filter, and calculate statistics on the data quicker and easier, for example in calculating

node sizing, the result object would be converted into a DataFrame. The Neo4j result object has an attribute to convert the object into a list. This list was then converted into a DataFrame.

There are seven columns in the DataFrame, one for each node type (3), each relationship type (2), as well as for both hop counts.

The list of columns returned can be seen below.

1. Primary nodes
2. Secondary nodes
3. Tertiary nodes
4. Primary-secondary relationships
5. Secondary-tertiary relationships
6. Hops (primary and secondary nodes)
7. Hoppies (tertiary nodes)

The node and relationship columns contain dictionaries, and in these dictionaries are the properties. For ease of use, the keys from these dictionaries would need to be extracted and a new column created for each key.

Primary node formatting

The primary node column stores the nodes that matched the search term inputted by the user. The node type, paper or author, depends on which type the user selected for their search. While secondary and tertiary nodes are always papers, as there are two potential node types for primary nodes, the formatting process for primary nodes was slightly different in order to handle both types.

The column containing the primary nodes is extracted from the DataFrame. A new DataFrame was created by extracting the unique keys in the dictionaries, creating a new column for each key and populating the new column with the corresponding value in the row's dictionary. The old column that contained the dictionaries is then removed from the DataFrame.

The process results in a DataFrame containing only primary nodes with a key for each property. For example, if the search was by paper title, the columns contain paper metadata such as the year, DOI, and abstract. On the other hand, if the search was by author name, the columns contain the author id and name. The number of hops is not calculated for primary nodes in the Cypher query so a default value of 0 is set for the primary nodes.

Secondary nodes

As previously mentioned, the secondary nodes contain only paper nodes. The process of extracting the node properties from the dictionaries followed the same formatting as the paper primary nodes. However, not all columns were dropped. The *hops* column applied to secondary nodes and was therefore kept. The resulting DataFrame contains the keys from the dictionaries as columns and the *hops* column.

Tertiary nodes

The tertiary node formatting differed from the secondary node formatting in one aspect. In the Cypher query *hops* refers to the number of edges between the primary and secondary nodes and *hoppies* between the secondary and tertiary nodes. Hoppies could range from 1 to 3 degrees of separation from the secondary node. In order to get the degrees of separation of the tertiary nodes to the primary node(s), the hop count of the tertiary paper's corresponding secondary paper and its hoppies value were summed.

Joining all the data

The three DataFrames, for the primary, secondary and tertiary nodes, were then merged. If the search was by paper title, all three DataFrames have the same columns and when merged, the resulting DataFrame contains a single instance of each unique key.

If the search was by author name, the author-specific columns will appear in the merged DataFrame, and secondary and tertiary node rows will have *null* set for those author keys. Also, the hop number of the papers written by the author(s), the secondary papers, is set to 0 and the hop counts for the tertiary papers are reduced by 1.

Finally, using the unique paper and author ids, duplicate rows are removed from the DataFrame. Duplicate primary and secondary nodes can appear as the response returns an entry of the primary and secondary nodes for each tertiary node. The entire formatting of the search result takes approximately 0.20 seconds.

Data formatting

General formatting is then applied to fields that are involved in node sizing or in generating the various results statistics. In the Neo4j database, some numeric data is stored as strings. Such columns include the citation count, reference count, and year columns. In order to perform operations on these fields later, the columns are cast from a string containing a float, to a float, then to an integer. Columns containing lists, including the keywords and field of study columns, are converted from a list containing a string of a list, to a list.

Paper and author nodes had their own id keys in the DataFrame, *paperId* and *authorId* respectively. In the future, to order the DataFrame by the same order of the nodes in the network, a common index column is created containing the corresponding node ids. This will later be used when creating a list of node labels from the DataFrame. If the order did not match, the wrong metadata would appear in the labels.

4.3 Creating the citation network

4.3.1 Creating the network structure

The Neo4j graph instance returned by the driver does not contain the co-ordinates of the nodes which are necessary in order to graph the citation network. While a visualisation library would be used to display the citation network in the user interface, the major libraries cannot graph networks by themselves as they cannot position the nodes in the graph. As a result, a network Python library was required to calculate the co-ordinates of the nodes in the citation network.

Networkx - storing networks

NetworkX [63] was the network library used to create the network graph co-ordinates. It has four key graph methods for storing nodes, edges, and their attributes:

- *Graph*: class for undirected graphs, multiple edges not supported.
- *DiGraph*: class for directed graphs, multiple edges not supported.
- *MultiGraph*: class for undirected graphs, multiple edges are supported.
- *DiMultiGraph*: class for directed graphs, multiple edges are supported.

During the initial development, basic graphs were created using the *Graph* class to test converting the Neo4j graph result to a NetworkX graph object. However, as a citation network is a directed graph, after the initial development the directed graph classes were tested. The directed graph classes resulted in graphs where there were a cluster of nodes in the center of the graph surrounded by an outer ring of secondary and tertiary papers. This meant the structure of how the papers branched out, the hops, from the primary nodes were lost. As a result, to keep the formation of the branching, the *MultiGraph* option was used instead, as edge directions could be later indicated in the graph using the visualisation library. *MultiGraph* was chosen over *Graph* as it is possible for two papers to cite each other and *MultiGraph* facilitated showing both edges in the network graph.

Node positioning algorithms

NetworkX has twelve node positioning algorithms which determine the co-ordinates of each node in the network. Most of the options, such as the *bipartite_layout* which positions the nodes in the graph along two straight lines, and *spiral_layout* which positions the nodes in a spiral, were not appropriate structures for a citation network.

Two node positioning algorithms for undirected graphs were tested, Fruchterman-Reingold force-directed algorithm (or spring layout) and Kamada-Kawai path-length cost-function. Both methods have a *spring* premise where nodes are either attracted to or repulsed by each other. The edges are thought of like springs, to keep those nodes connected close and repel those not connected.

The Fruchterman-Reingold algorithm attempts to position the nodes so that the edges connecting them are all approximately the same length while reducing the number of crossing edges in the graph [64]. It aims to mirror the underlying symmetry in the branching to make the layout of the graph more aesthetically pleasing.

In the Kamada-Kawai path-length cost-function [65], instead of springs existing just between adjacent nodes as in Fruchterman-Reingold, springs exist between all nodes. The length of the spring is equivalent to the graph distance, where graph distance is the shortest path, or fewest number of edges, connecting the two nodes.

The symmetry and reduced edge crossings of the Fruchterman-Reingold algorithm resulted in the most comprehensible and aesthetically pleasing graphs, and as a result, it was chosen over Kamada-Kawai. The reduced overlapping edges was important to improve the readability of the citation network in the user interface and to make viewing the relationships between nodes easier.

Network creation process

The first step in the implementation process involved creating an instance of *MultiGraph*. The nodes were then extracted from the Neo4j graph result. As nodes in NetworkX can have properties, the paper and author attributes were added as properties. Paper nodes stored properties such as the year, number of hops and number of citations. Author nodes stored the author's name and id. The nodes with their properties were then added to the *MultiGraph* instance.

The edges were then extracted from the Neo4j graph result. Edges can also store properties and the relationship types of the edges, *REFERENCES* or *AUTHORED*, were added. The individual edges were then added to the *MultiGraph* instance including their start and end nodes, relationship id, and properties.

Once the graph had been populated with all the nodes in the citation network, the positions of the nodes were created with the Fruchterman-Reingold layout function. These co-ordinates were then added to the edge attributes to connect the nodes.

4.3.2 Visualising the citation network

Graphing the citation network

While NetworkX can create and examine network structures it cannot display them. A dedicated visualisation library would be required to transform the NetworkX graph instance, containing the co-ordinates and properties of the nodes and edges, into a visible representation of the citation network in the user interface.

Initially Matplotlib, a Python visualisation library, was used to visualise the network. However, the application was later migrated to Plotly, another visualisation library, due to its ability to render interactive plots, the greater control over design elements such as axes, and its better documentation.

Network graphs can be created with Plotly using a scatter plot instance. Scatter plots consist of dots (or nodes) positioned along the X and Y axes. The nodes display the relationship between the axes values. For example, showing the share price of a company over time. A network graph can be plotted on a scatter plot using the co-ordinates generated by the node positioning algorithm in NetworkX.

Displaying the node attributes

Displaying the nodes in the citation network as dots alone would not provide the user with adequate information about the papers. Information is displayed in two formats, node labels and text boxes appearing when the node is hovered upon.

If node labels were displayed for all paper and author nodes, the graph would become cluttered with the nodes, edges, and labels overlapping and becoming hard to distinguish. As a result, not all labels are immediately visible in the graph. While all primary node labels are displayed, unless there are more than 5 primary nodes, only the label of every 10th secondary or tertiary node is displayed.

An inability to view the papers behind the nodes would not be conducive to examining a citation network. As the network graph would not initially display all the node labels, a feature was added to display basic information about the node when the user hovers over the node in the graph. Aside from the paper's title, the hover text box displays the year, and hop, citation, and reference counts. A note at the bottom of the hover box informs the user that they can view additional information about the paper by clicking on the node (instead of hovering over it), this detailed paper view

is described in subsection 4.4.4. An example of the hover text box can be seen in Appendix C.2.

Node colour and size

The nodes are colour co-ordinated and their sizes scaled to help the user differentiate between nodes of different types, degrees of separation, and number of citations. The node colours were determined by node type. Primary nodes, those matching the search term, were coloured green, and all secondary and tertiary nodes were pink. The node sizes were scaled by hop count, year, and the number of inward or outward citations (depending on which the user selected in the search input).

Creating a node size scale

A base node size was calculated for each hop count using the Python *range* function, an interval count, and a maximum node size. The range function creates a sequence of numbers with each element increasing or decreasing from the previous element by a set increment. The interval, or number of elements in the range, was the maximum node size plus one. The additional hop count was due to the hop count of primary nodes being set to 0, otherwise the range would be short one value. The list of base sizes was reversed, sorted decreasingly, so the base size for a hop count could be used as the index in the list.

Weighting the year and citation counts

A *buffer* was added to the base size based on the year and citation count of the paper. As newer papers tend to have fewer citations than older papers, a basic calculation was implemented to weight the citation count by the year it was published.

Firstly, a maximum buffer size was calculated as half the difference between the base node sizes to ensure that it was easy to differentiate between nodes of different degrees of separation.

The next step involved normalising the year and number of citation columns in the DataFrame. These normalised values were individually multiplied by the buffer size to get a year buffer and a citation count buffer. The normalised year used to get the year buffer means that newer papers would be larger than older papers. The

normalised citation count means papers with more citations are larger than those with fewer citations. This balanced out the difference in citations of old and new papers. These buffers were added to the base size to calculate the final node size.

A machine learning approach to citation count prediction was later developed. The models created did not have a sufficient level of accuracy to be used in the node sizing so the approach outlined above was kept. The machine learning approach is expanded upon in section 4.6.

4.4 Results page components

The results page displays the search parameters in a panel at the top of the page. The page contains two views of the citation network, graph and table, only one view is shown at a time, as well as a detailed paper view which displays comprehensive information about a paper. There are two initially hidden components, a filter panel that filters the contents of the citation graph, and a statistics modal that provides network wide statistical insights. These components are initially hidden to provide a larger space for the citation network views. The components of the results page are outlined in more detail in the subsections below.

4.4.1 Graph view

The main component of the results page is the graph view of the citation network. It takes up the largest amount of space on the page and it is the default view of the citation network. However, in a panel above the graph there is the option to toggle between the graph and table views. The panel also contains a legend for the graph, explaining the meaning behind the colours of the nodes.

An example of the graph view of a citation network is shown in Figure 4.1. The user searched by the exact title of a paper, *A solution to the problem of touching and broken characters*, and selected to view the inward citations in the graph. The subsequent component screenshots were from the same search.

Two more graph view screenshots are available in the Appendices. In Appendix C.1, the citation network shown was created from a search for an author's name. In Appendix C.2, a search for paper titles containing 'interactive visualization' was made.

This network contains outward citations.

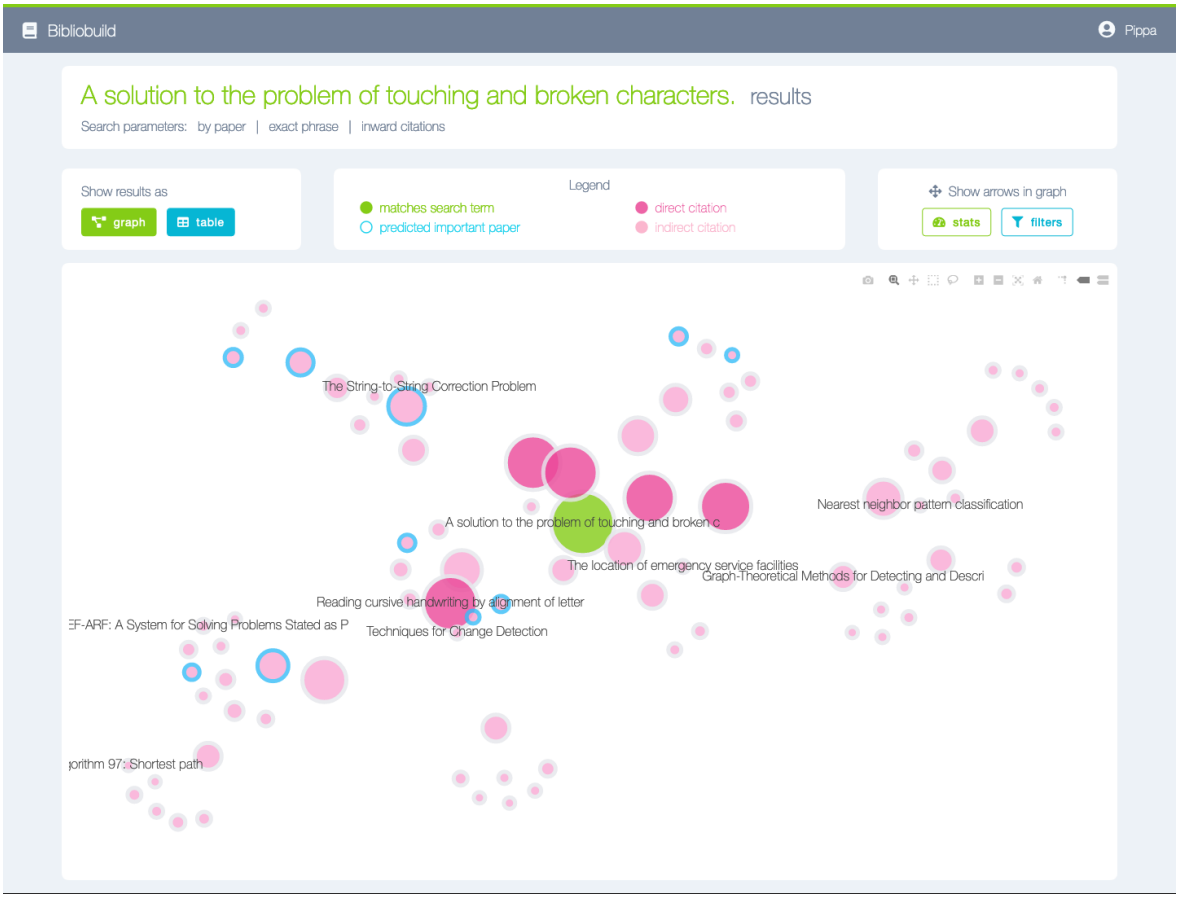

Figure 4.1: Graph view of a citation network.

4.4.2 Table view

A key benefit of displaying a citation network as a graph is the ability to clearly see the relationships between the nodes. However, the graph cannot display a large volume of data perceptibly. As a result, a table view was added to the results to display a larger volume of data.

A table view was chosen as the paper nodes contain the same keys, and the columns would make it easier to scan data. The table has five columns: title, year published, number of hops, number of citations, and number of references. The rows are ordered first by the number of hops ascending, then by year ascending. This puts the primary

nodes at the top of the table, immediately visible to the user, and the papers with the largest degrees of separation at the bottom.

On the left hand side of each paper title is a quick search link. This will open the search page in a new tab in the user's browser with the search term pre-populated to the title of the paper and the search type set to paper title.

In the number of citations and number of references columns, the largest values are highlighted in blue. This enables the user to identify the corresponding paper title at a glance.

Figure 4.2 shows the contents of the citation network as a table.

TITLE	YEAR	# HOPS	# CITATIONS	# REFS
A solution to the problem of touching and broken characters.	1993	0	17	5
The relative neighbourhood graph of a finite planar set	1980	1	921	8
The Relative Neighborhood Graph, with an Application to Minimum Spanning Trees	1983	1	140	4
Reading cursive handwriting by alignment of letter prototypes	1990	1	80	13
Direct gray-scale extraction of features for character recognition	1993	1	222	11
A shape analysis model with applications to a character recognition system	1994	1	58	10
Linear time algorithm for isomorphism of planar graphs (Preliminary Report)	1974	2	365	2
An Alternative Definition for "Neighborhood of a Point"	1975	2	53	6
Finding Minimum Spanning Trees	1976	2	390	0
The All Nearest-Neighbor Problem for Convex Polygons	1978	2	40	2
Elastic matching of line drawings.	1981	2	149	8
Localizing Overlapping Parts by Searching the Interpretation Tree	1987	2	692	16
A Method for Curve Seeking from Scattered Points	1966	3	31	2
Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters	1971	3	134	6
Linear Time Simulation of Deterministic Two-Way Pushdown Automata	1971	3	141	0
An iterative procedure for the polygonal approximation of plane curves.	1972	3	673	8
A V log V algorithm for isomorphism of triconnected planar graphs	1973	3	60	4
Computing the perceptual boundaries of dot patterns	1974	3	33	3

Figure 4.2: Table view of a citation network.

4.4.3 Summary statistics

The citation network can return up to 100 nodes. This would make approximating or manually calculating statistics from the results difficult. A sample statistic may be what are the oldest, median, and latest years of publication in the network. This can provide insight into how active the field is, or in other words, are the papers still relevant today. For example, if the latest paper in the network was published in 1992,

there has been no activity in nearly three decades.

The minimum, maximum, median, and total range (the difference between the minimum and maximum figures) values are calculated across the papers in the network for the year, citation count, and reference count. The user can click on the minimum and maximum figures to display the paper(s) behind the statistic. This makes the process of finding which paper was published most recently significantly easier than scanning the hover text of all of the nodes. The median value is used over the mean value as a small number of papers have tens of thousands of citations. As most papers in the database have less than 100 citations, these extremely cited papers drastically inflate the average citation count.

The top ten keywords across the citation network are also calculated. A list of all the keywords appearing in the papers is compiled. This list has two uses. First, the top 10 most occurring keywords are calculated, which provides an insight into what are the predominant topics in the network. Secondly, the papers in the citation network can be filtered using the keywords filter in the filter panel. The list of keyword options for this filter comes from the same list. Duplicate instances of words are removed and the list is sorted alphabetically.

Figure 4.3 shows the statistics generated on the citation network as a whole. Due to the high top citation count, the max citation figure was clicked upon to view the title behind the figure.

4.4.4 Paper view component

As outlined in subsection 4.3.2, the user is provided with basic information about the paper when hovering over the node. There were text fields that would have been too long to have in the hover text box, such as the paper's abstract. Also, as the hover text boxes are not static on the page, if links were included in the box the user might lose the hover text box while moving their mouse when attempting to click on a link. As a result, a detailed paper view component was added to display all the information about the paper in a more user-friendly format.

The user can view the paper's details by clicking on a node. As mentioned in subsection 4.3.2, users are informed of this at the bottom of the hover text box that pops up when their mouse appears over the paper's node in the graph. The paper view

Figure 4.3: The statistics modal component containing insights into the citation network.

component appears underneath the citation graph and the page automatically scrolls down so the component comes into view.

The following fields for a paper are displayed:

- Abstract
- Authors
- DOI
- Keywords
- Language
- Number of citations
- Number of references
- PDF link
- Title
- Year published

There are two links in the component. The first is a URL to an open access PDF version of the paper. However, this is only displayed when a link exists. Additionally, there is a quick search link, also implemented in the table view in subsection 4.4.2, which opens a search for the paper in a new tab in the user's browser.

Figure 4.4 shows the detailed view of a paper, including the author and the abstract.

4.4.5 Filter panel

A citation network contains nodes across different years, fields, languages, and so on. It can be beneficial to reduce the quantity of nodes displayed to examine an aspect of particular interest. For example, viewing only those papers published in recent years. This required the citation network to be filterable.
There are 7 available filters:

1. Citation count range (numerical range inputs).

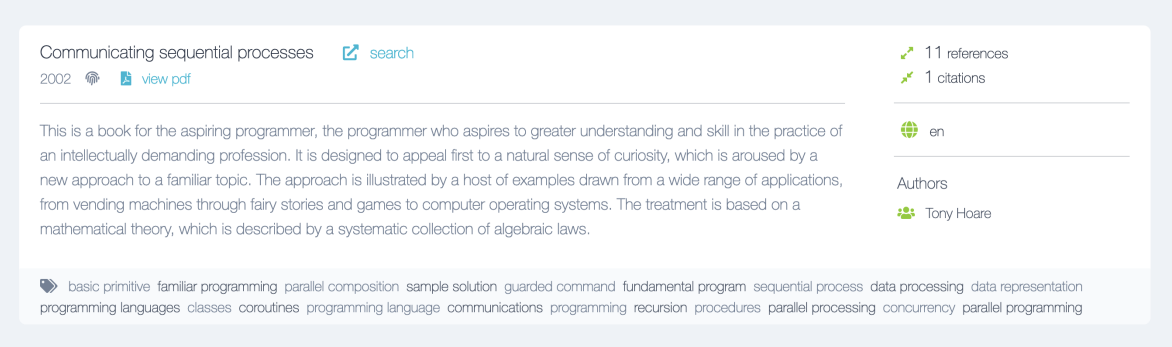

Figure 4.4: The detailed paper view component displaying all information for a paper.

2. Reference count range (numerical range inputs).
3. Year count range (numerical range inputs).
4. Hop count range (numerical range inputs).
5. Languages (a multi-select list).
6. Keywords (a multi-select list).
7. PDF available (a checkbox input).

Numerical range inputs

The citation, reference, year, and hops count filters enable the user to set maximum and minimum values, and only papers that fall within the range are returned. The default minimum and maximum values of each filter are set to the actual minimum and maximum values in the result data.

The hops count range provides the user control over the degree of branching to display in the graph. For instance, if the user wanted to view only the matches to their search term, they could select 0 as both the minimum and maximum value.

Multi-select lists

The languages filter contains a list of the unique languages the papers in the results are written in, while the keywords filter consists of the unique keywords in the papers.

The user can select multiple options from each list. The filter returns the papers that contain any of the selected options in the list.

Traditionally, papers are mostly cited by other papers in their field. However, cross-field citations do occur and create a link between the different fields. The keyword filter enables the user to view only the papers with the keywords they are interested in and so papers belonging to different fields can be isolated.

Checkbox input

The dataset contains URLs to PDFs for some of the papers. The papers are hosted on the AMiner website and are open access. The PDF available checkbox enables the user to filter out those papers that do not have a URL to display only the open access results.

Also, beneath the filters there is a reset button that will remove all the user's filters and restore the citation network to its original state.

Figure 4.5 shows the list of filters available in the panel on the left hand side.

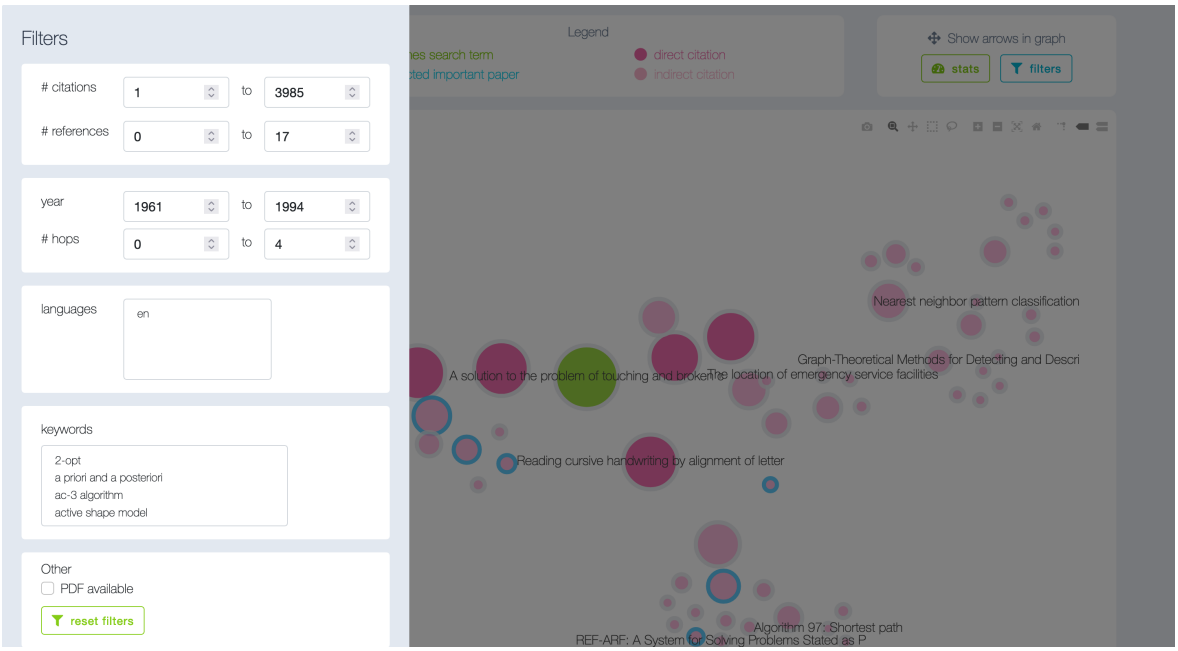

Figure 4.5: The filter panel component enabling the user to control the contents of the citation graph.

4.5 Client-side interactivity

JavaScript is used to provide client-side interactivity, essentially handling all clicks and inputs of the user.
A summary of interactivity provided includes:

- Handling input changes.
- Toggling between the graph and table views.
- Displaying the filter side bar and statistics modal.
- Displaying the detailed paper view.
- Filtering the nodes in the citation network graph.
- Displaying the papers behind the figures in the statistics modal.
- Adding paper titles as annotations to the citation network graph.

4.5.1 Filtering the citation graph

It is fast and clean to update the Plotly graphs in Python. However, this would involve sending requests to the server each time a filter is changed. The data behind the citation networks would need to be cached to avoid repeatedly querying the database with the same query every time a single user changed their filters.

Plotly has a JavaScript version. This meant that operations such as filtering the nodes in the graph could be handled on the client-side instead of sending a new request to the server to redraw the graph every time a filter was changed. In other words, the graphs are initially created in Python Plotly and are later updated in JavaScript Plotly.

Plotly graphs have a *transform* attribute which can take a list of filters to filter the graph by. Basic filters, where there is one option to filter or search for and the column to search in is made up of single, unnested values, are simple to add.

The data behind the graph is contained in the *data* key. A Plotly graph can contain multiple sub-graphs, called traces, layered on top of each other. The index 0 is used to get the trace containing the node data. The data for the edges, which connect the nodes, are contained in another trace in the plot.

4.5.2 Creating filters for different data types

As covered in subsection 4.4.5, there are three generic types of filters: boolean, range, and select inputs.

The range filter involved adding one to two filters to the transform list. One filter for the minimum value for the field, another for the maximum value for the field. However, both filters were not always required. For example, if the user changed the minimum value, a filter would be added to set the minimum value for that field. If the user did not change the maximum filter, a maximum filter would not be required and, as a result, only one filter would be added to the transform list.

The multi-select filter took a list of the selected values, and checked to see if any of those selected values appeared in the field being filtered for each node. The field being filtered could have different data types. The language field in the data contained single string values while the keyword field was made up of arrays of strings.

As the transform filters do not support filtering for an array of values or in an array of values, a new key was created in the data to enable a multi-select filter. The value of the key would store an ordered list of booleans for whether each node contained any of the selected options. The values of this new key were generated by looping through the field's values and adding *true* if there is an intersection between the selected input values and the node's values, otherwise adding *false* to the new key's list of values. The multi-select filter could then be added to the transform list by setting the value being filtered for to *true*, and data being filtered to the list of booleans.

Resetting the filters

A reset filters button was added to the filter panel to reset the citation graph and the input filters back to their original values. The graph was reset to its original structure by setting the transforms list to an empty list. However, after doing so, the filters in the UI would still contain the old selected values.

The input values in the user interface were reset to their original values individually in JavaScript. The value of the multi-select inputs were set to null to uncheck all selected values, and the value of the checkbox was set to false, to become unchecked. In order to reset the range filters back to their original states, default value attributes were added on input creation so the original value could be retrieved from the input

element and then set as the current value.

Figure 4.6 shows the results of a hop filter on the citation network. The maximum hop count was set to 2. Also, the graph lines were enabled, using the toggle above the statistics and filter buttons. The unfiltered graph can be viewed in Figure 4.1 for comparison.

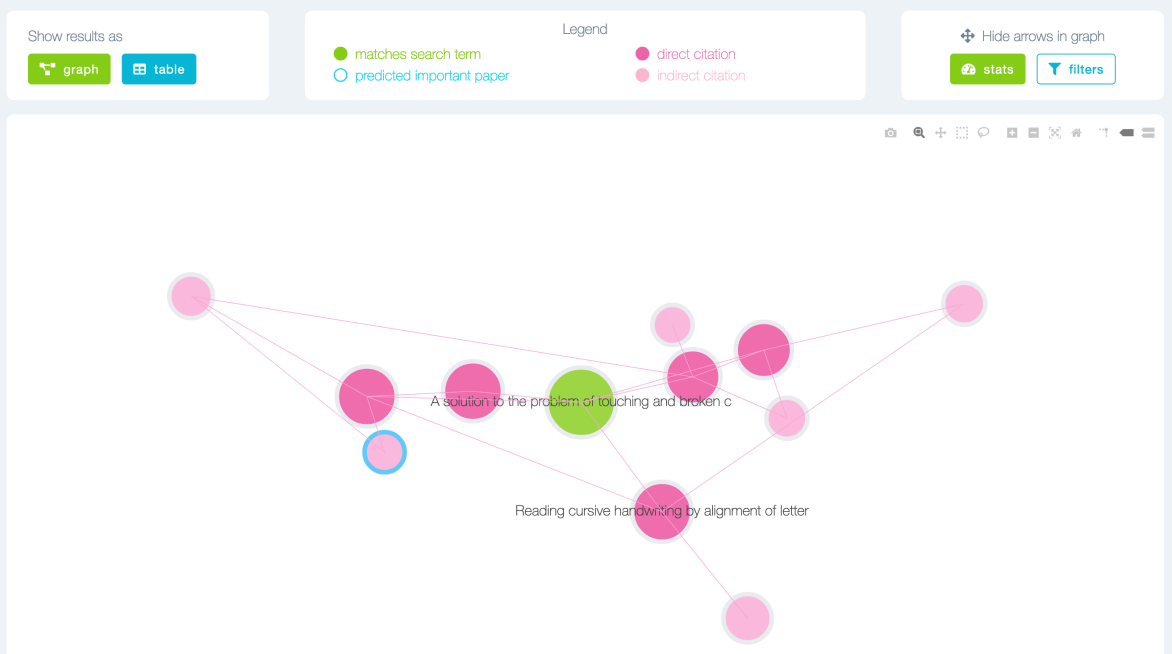

Figure 4.6: Example of a filtered graph where the maximum number of hops is 2.

Trial of an analytic dashboard library

Due to prior experience with JavaScript, handling the client-side interactivity was not an unfamiliar task. However, one key aspect that would have benefited from dedicated third-party libraries were the operations involving the citation network graph.

An alternative to the protracted filtering of the graph data in JavaScript would be to use Plotly Dash [66]. Dash is based on Plotly, Flask, and ReactJS, and is used to create analytic applications.

The application is a dashboard page that can contain one or more components. The components can be different visual means including tables, charts, and filters. Instead of handling the clicks on the components and the filter changes with JavaScript listeners

and manually filtering and redrawing the page, Dash uses callbacks. The callbacks are function decorators that call a Python function. The data behind the components can be easily updated in the function using Pandas, or the graph type changed or restyled.

As Flask is used behind Dash, it would require running an application instance of Flask to handle all the non-dashboard routes, and a Dash instance to serve a citation dashboard. The Flask instance would be passed as an input to the Dash instance. Both instances would be required as a plain Flask service couldn't serve the Dash endpoints, and the Dash service can only serve the dashboard endpoints.

Dash was incorporated into the project mid-way through. This involved a restructuring of the project to make both the Flask and Dash instances compatible. Each user would need their own citation network dashboard containing only the results of their search. However, Dash does not handle the underlying dataset of the dashboard changing very well. If another user were to run a search, the underlying data of the other users would be updated as well. It was then decided not to use Dash in order to facilitate multiple users using the application at a time.

4.6 Machine learning approach to citation prediction

As mentioned in the design in section 3.6, a citation prediction model would be implemented to further utilise the citation dataset and provide additional features to the user. An overview of the process to create the model is as follows, the first task was to create a training dataset containing a range of features that could be used as inputs to the model. The next step involved choosing an assortment of models and testing them to determine the most accurate model. Once the model was decided, a process called feature engineering could then take place. This involves trialling the various features and applying different pre-processing steps to the features to determine the permutation with the highest accuracy.

4.6.1 Creating the training dataset

Papers published between 2005 and 2015 were chosen as their citation rate would have passed their peak and would have begun to level off. In order to perform natural

language processing on the text fields, only papers written in English were included. Those papers made up the vast majority of the citation database, and would mean that commonly occurring words with little value, called stop words, and tokenisation, reducing a word to its root, would only need to concern one language. Finally, the paper must have non-null values for its abstract, title, venue name, reference count, citation count, and keywords.

Many of the papers reviewed in section 2.6 had success with author- and venue-centric features. As a result, the most populated venue field, the name, is included in the list of paper attributes retrieved for the training dataset.

The author features were more complicated to retrieve. Two items were desired, the average and maximum number of author citations for the paper. This required, for each paper, getting its authors and all the papers they have written. The average and maximum number of citations the author's papers have received is calculated, before the average and maximum number of citations across all authors is calculated for the paper. The logic behind these fields is that new papers by authors whose published papers frequently receive a large number of citations are more likely to receive a higher number of citations than those of authors with few or no citations.

Cypher query

A Cypher query was written to retrieve all the required data for the training dataset for a given year. A Python script ran this Cypher query for each year in the 2005 - 2015 range and stored the output to a JSON file.

Formatting data

The years were individually ingested and formatted before being merged. Similar to the process mentioned in section 4.2.5, data had to be extracted from dictionaries to unnest the results to make operations such as filtering and sorting easier and faster. Papers with null values for the author features, the average and maximum number of author citations for the paper, were removed. Later, the script was rerun to remove papers with more than 50 citations as papers with thousands of citations were extreme outliers. Finally, a random sample of 10,000 papers per year were selected. These year samples were combined and stored in JSON to form the final training dataset.

4.6.2 Model selection

Once the training dataset had been generated, the model selection and evaluation process could begin. Four regression models across different levels of complexity and approaches were evaluated. These models, linear regression, ridge regression, linear support vector regression, and k-nearest neighbors regression, are outlined in the following four subsections. The Python *scikit-learn* library was used to implement these models [67].

Linear regression

Linear regression assumes there is a linear relationship between the inputs of a model (the features) and the output. It attempts to find a line that best fits this relationship in order to make predictions. The scikit-learn *LinearRegression* model uses ordinary least squares, the most common method of estimating output values.

Linear regression was the simplest model chosen as it does not have any hyperparameters. Hyperparameters are model inputs that are used to control how the model is trained. In order to find the hyperparameter value that yields the most accurate predictions, a range of values for the hyperparameter are selected. A model is trained using each value and the accuracies of the models are compared to determine the optimal value. As the linear regression flavour used had no hyperparameters only one instance of the model had to be run in the model evaluation.

Ridge regression

Ridge regression is an extension of linear regression. Unlike linear regression, it has a regularisation hyperparameter. A model can have several inputs, however, not all inputs will have the same contribution to the prediction, as some inputs will be more influential in creating accurate predictions. In order to weight the inputs by their influence, each input is assigned a value called a coefficient. A cost function reduces the coefficient values of the inputs that contribute the least to the prediction, essentially giving them a penalty. The size of the penalty is determined by the regularisation hyperparameter.

Linear support vector regression

Support vector regression (SVR) is a regression implementation of the support vector machine (SVM) classification process. SVMs are used when the dataset is not immediately linearly separable. Unlike the previous models covered, it takes a multidimensional approach by adding a new dimension (a z-axis). The dataset can be linearly separated along this z-axis. When transformed back to the initial plane, the line appears as a circular boundary between the classes. In SVR, the boundary line, instead of separating classes, is used to estimate the continuous output values. A penalty hyperparameter is also used in linear SVR.

Nearest neighbours regression

Nearest neighbours regression (k-NN regression) does not take a linear approach to making predictions. Instead, it assumes similar data points will appear in proximity of each other. Therefore, in order predict the output of another data point, the data point's k-nearest neighbours are located. In regression, the output values of these neighbours are averaged to predict the data point's output value. In k-NN regression, the hyperparameter used is k, the number of neighbours to use in the prediction calculation.

Training the models

The models were trained and evaluated with the top performing model being used in the subsequent feature engineering. All models were trained using the same dataset with 100,000 data points and the same features. Only the numerical features were considered in the initial model training. These features were the age of each paper, and the average and maximum number of author citations for each paper. This was due to time constraints as three of the models would be trained multiple times and including textual features would have added a significant amount of time. The inputs were normalised to scale their values between 0 and 1. A range of hyperparameters were chosen for each model, aside from linear regression. The values selected are shown in Table 4.3.

Model	Hyperparameter Values
Linear regression	N/A
Ridge regression	0.001, 0.01, 0.1, 1, 10
Linear support vector regression	0.001, 0.01, 0.1, 1, 10
Nearest neighbours regression	2, 6, 10, 20, 25, 30

Table 4.3: The hyperparameters tested for each model.

Model results

As the models were trained, the training durations and the mean squared error results were logged. Mean squared error (MSE) is a metric used to evaluate regression models. It measures the squared difference between predicted and actual output values for each training data point, and averages these values to get a single figure. In using MSE, the smaller the value the better as it means a smaller error.

The results of the model training are presented in Table 4.4. The k-NN models out-performed all the other models having the lowest MSEs, while the Linear SVR models had the highest errors of all the models. As a result, the k-NN model which took 25 neighbours into account, which had the smallest errors at 94.57 and therefore, the highest accuracy, would be used in the subsequent feature engineering. While the models were compared using MSE, the appropriateness of MSE as a metric is evaluated in subsection 5.3.2.

4.6.3 Feature engineering

The model would be trained using text and numerical features. There were three numerical features, the age of the paper, and the average and maximum number of author citations for the paper and five text features, the abstract, field of study list, keywords list, paper title, and venue published.

The numeric and text fields would be processed, or transformed, individually to determine the most accurate processing settings for each feature before finally training the model using all eight features.

	Model	Hyperparameter	Time to Train	MSE
1	KNeighborsRegressor	25	0.22s	94.5732
2	KNeighborsRegressor	20	0.23s	95.2758
3	KNeighborsRegressor	30	0.21s	96.3170
4	KNeighborsRegressor	10	0.21s	102.7988
5	KNeighborsRegressor	6	0.2s	108.4233
6	Ridge	1	0.0s	127.6799
7	Ridge	10	0.0s	127.9446
8	Ridge	0.1	0.0s	127.9769
9	LinearRegression	N/A	0.02s	130.1210
10	Ridge	0.01	0.0s	130.5524
11	Ridge	0.001	0.01s	132.5357
12	KNeighborsRegressor	2	0.23s	140.9298
13	LinearSVR	1	0.06s	148.4468
14	LinearSVR	0.1	0.04s	150.8688
15	LinearSVR	0.01	0.04s	152.9121
16	LinearSVR	10	0.19s	153.7204
17	LinearSVR	0.001	0.04s	162.1879

Table 4.4: Results of the initial model training.

Numerical features

The numerical features were normalized, or scaled, between 0 and 1 using Scikit-learn's *MinMaxScaler*.

Text features

The text features needed to be transformed from strings into a matrix of numbers as text cannot be inputted into the models directly. This was done using Scikit-learn's *TfidfVectorizer*. There are two components to the transformation. The first is the vectorizer which converts the text into the matrix of word, or term, counts. Each word is given its own count in the matrix for each data point. Secondly, *tf-idf* which stands for term frequency–inverse document frequency, is used to measure how important a term is within a document in a collection of documents. Term frequency measures the number of appearances of a word in a document. Inverse term frequency is used to reduce the noise of terms that appear across the collection of documents, to bring

Feature	Maximum Frequency	Minimum Frequency	MSE
abstract	0.5	1	133.69
fields of study	0.5	0.15	139.99
keywords	0.7	0.1	135.03
title	0.8	1	131.97
venue name	0.6	1	114.96

Table 4.5: The final transformation settings for each text feature.

attention to words that are frequent in the document, but not throughout the collection. A tf-idf score is given to each term.

For each text feature, a set range of maximum and minimum document frequencies were tested. The maximum frequency would ignore words that appeared in over a certain percentage of the papers in the training dataset, while the minimum frequency ignores words that appear in less than a certain percentage of papers. While the *stopword* attribute of the tf-idf vectorizer was set to ignore English stop words in all tests, the maximum frequency setting acts as a corpus-specific stop word filter. The maximum frequency, *max_df*, was tested first with no minimum frequency restrictions. The range of values tested were decimals that are treated as percentages: 1.0, 0.9, 0.8, 0.7, 0.6, 0.5. The first value, 1.0, is the default.

Once completed, the max_df setting with the most accurate MSE was selected, and used while testing the minimum frequencies, the *min_df* attribute. The range of minimum percentages tested were 0.05, 0.1, 0.15, 0.2. The default of 1 was not included as it was used in the maximum frequency test. The most accurate minimum percentage was then selected to have the best transformation settings for the feature. This process was carried out for each text feature, the results of which are displayed in Table 4.5.

Combining the features

As different transformations were to be applied to each feature, a *Pipeline* tool was used to apply the custom settings for each feature before training the model in one go. This negates the need to fit and train the model to each feature individually. A baseline model was trained containing all of the features. However, as some features may not contribute to the accuracy of the model, or may reduce the accuracy of the model,

Features Removed	MSE	Accuracy Change
no features removed	117.4027	N/A
abstract	116.3936	improved with removal
age	118.0543	disimproved with removal
fields of study	114.2065	improved with removal
keywords	119.5785	disimproved with removal
avg. author citation count	116.5596	improved with removal
max. author citation count	115.8625	improved with removal
title	120.1386	disimproved with removal
venue name	135.0993	disimproved with removal

Table 4.6: Model accuracy having removed each feature.

Features Removed	MSE	Accuracy Change
no features removed	114.9687	N/A
age	115.8157	disimproved with removal
keywords	112.9933	improved with removal
title	115.0972	disimproved with removal
venue name	135.2885	disimproved with removal

Table 4.7: Model accuracy having removed each feature - iteration 2.

the model was trained a further 8 times, each time holding out a different feature to discover the effect on the accuracy.

The removal of a feature was said to improve the accuracy of the model if the MSE decreased, while if the MSE increased with the omission of a feature, the feature was said to improve the accuracy of the model. As seen in Table 4.6, removing the abstract, fields of study, and both author fields had a positive impact on the accuracy of the model.

The process was repeated two more times, each time removing the features where the accuracy improved when they were omitted. The results of the second iteration are shown in Table 4.7, and the third in Table 4.8. Only the keyword feature had a negative impact on the accuracy from the second iteration. As a result, it was removed before the third iteration. As the error increased in the third iteration when any of the features were omitted, no more features were removed.

Features Removed	MSE	Accuracy Change
no features removed	114.1638	N/A
age	117.8458	disimproved with removal
title	115.7669	disimproved with removal
venue name	132.3162	disimproved with removal

Table 4.8: Model accuracy having removed each feature - iteration 3.

4.6.4 Final results and model deployment

The final model was trained using the age, title, and venue name features, and had a MSE of 114.1638. As the ideal MSE figure is close to zero, this prototype model did not have a sufficient level of accuracy to have the predictions be incorporated into the application as numerical values. The results of the model and the metric used are evaluated in subsection 5.3.1.

Model deployment

However, it was discovered that the results of one the models had surprising accuracy in identifying the papers with the top citation counts. This was found while checking did the same papers appear in the lists of papers with the top 10 citation counts in the actual and predicted results. A basic model using the title and keywords correctly predicted a top 10 citation count for, on average, 6.5 of the top 10 papers in the test values. As a proof of concept test, this model was saved using the Python serialiser *pickle*. The papers in the citation network were passed through the model, and the papers with the top 10 citations were outlined in blue in the citation graph to highlight those papers to the user.

Chapter 5

Evaluation

There are two aspects to the application's evaluation, the performance impact of adding indexes to the citation database, and the completeness of the database results versus that of a citation API, and two aspects of the citation prediction evaluation, the accuracy of the model and the metric used.

5.1 Performance impact of the indexes

The performance impact of the two indexes applied to the database were tested by running a series of queries on a non-indexed version and an indexed version of the database, and comparing their durations. As outlined in subsection 4.1.3, the *title* property was indexed in paper nodes and the *name* property was indexed in author nodes.

5.1.1 Paper index

A random sample of 100 titles were used in the paper index test. Each title was queried three times. The first time was a simple look-up query that returned only the paper's metadata. The second query built upon the first query but also returned the direct inward citations to the paper. Finally, the third query, which built upon the second query, returned the inward citations for the paper up to 4 degrees away. No limit was set on the maximum number of papers that could be returned in order to measure a range of citation network sizes.

Query attributes, such as query type, paper title, database version (whether the index was present), and duration were all logged. After the completion of all queries, the log file was parsed in Python. The log entries were grouped by query type and database status. The following statistics were calculated for each group: minimum, maximum, and mean durations, and standard deviation.

The results of the paper index test can be seen in Table 5.1. As expected, the indexed queries were several times faster than the unindexed queries. This is particularly noticeable in the advanced queries, where the average time using the indexed database took less than a second, while with the non-indexed database, it took on average 1 minute and 20 seconds. The non-indexed queries also had a much larger standard deviation, especially the basic queries. The minimum and maximum times of the basic queries had a 6.6 minute difference. On the other hand, the indexed times had a range of under 2 minutes.

Interestingly, the basic queries had the highest mean duration, as well as the biggest standard deviations for both the non-indexed and indexed databases. One reason for this is that after the basic queries, Neo4j caches the results. As the queries build upon each other, the subsequent queries only need to retrieve the nodes for the additional degree(s) of separation from the previous query, which are adjacently stored in memory allowing for fast retrieval.

The longest unindexed query, which took 7.76 minutes (466.80 seconds), was to get the citation network for *Improving rule extraction from neural networks by modifying hidden layer representations*, returning the inward citations up to 4 hops away. The network contained 347 nodes and 359 relationships. The same query using the indexed database took 0.44 seconds. In summary, the queries on the indexed database were faster and their performance more consistent.

5.1.2 Author index

A random sample of 50 author names were used to test the author index. The sample size was reduced due to the total duration of the non-indexed paper queries taking several hours. Another factor was that unlike in the paper test, the basic author query would contain a relationship, and it was predicted this would further increase the runtime of the tests.

Query Level	Indexed	# Queries	Min.	Mean	Max.	Std.
basic	no	100	42.38	83.82	466.80	65.58
basic	yes	100	0.02	6.48	113.55	20.84
intermediate	no	100	4.56	68.83	180.00	35.37
intermediate	yes	100	0.02	0.05	0.21	0.03
advanced	no	100	42.56	80.89	223.75	48.90
advanced	yes	100	0.04	0.99	14.33	2.37

Table 5.1: Comparison of the test paper query durations on a non-indexed database vs an indexed database.

As with the paper index test, the 50 authors were queried in three different ways. The first got the author metadata and the author's papers. The second returned the same data as the first, as well as the direct inward citations of the author's papers. Finally, the third query got the author, their papers, and the papers' inward citations up to 4 degrees away. The log files were formatted and the statistics generated using the same process as the paper test.

The results of the author index test can be seen in Table 5.2. As with the paper index test, the database version with the index had a far faster and more consistent performance. One interesting difference between the non-indexed database performance across the two node types was that the author queries had a far wider range of times, with a significantly smaller minimum time and a significantly larger maximum time.

The longest unindexed query, which took 17.76 minutes (1066.92 seconds), was the basic query to get the papers written by *Yingchun Zha*, interestingly the advanced query to get the author's papers and the inward citations up to 4 hops away took 7 seconds. The network contained 2 nodes and 1 relationship. The same query using the indexed database took 0.07 seconds.

5.2 Completeness of the citation results

The completeness of the application's search results was then evaluated. The goal of this evaluation was to see if there were citations for a paper in the database that were not returned by the query. The test was conducted by comparing the results of a Cypher query, that returns all inward citations for a paper, to a list of citations for the

Query Level	Indexed	# Queries	Min.	Mean	Max.	Std.
basic	no	50	0.47	45.60	1066.92	159.69
basic	yes	50	0.02	12.57	303.28	45.46
intermediate	no	50	5.16	7.42	14.75	2.31
intermediate	yes	50	0.01	0.12	0.76	0.17
advanced	no	50	5.31	10.22	75.81	12.30
advanced	yes	50	0.02	3.20	47.07	9.63

Table 5.2: Comparison of the test author query durations on a non-indexed database vs an indexed database.

same paper from a citation API.

5.2.1 Citation API selection

The citation APIs outlined in section 2.1.3 were tried to ascertain which were possible options for the test. The citation API selected as the baseline was OpenCitations. Its REST API has an endpoint that returns a list of citations for a given DOI [68].

Google Scholar was not a viable option due to severe request throttling, and institutional access to Elsevier did not include permission to use its citation endpoint.

5.2.2 Data for evaluation

The evaluation dataset consisted of 100 random paper DOIs retrieved from the Neo4j database. The only restriction was the papers had to have a non-null DOI. This was to enable accurate matching of papers between the database and OpenCitations citation results. The papers had between 1 and 103 citations.

Retrieving the citation data from the Neo4j database

The Cypher query to retrieve the data is shown in Listing 5.1. Each test paper was queried individually and the results written to a JSON file.

```
MATCH (a:Paper {{doi:'{doi}'}})
OPTIONAL MATCH (a)<-[r:REFERENCES]-(b)
RETURN a.title, a.doi, b.doi, b.title
```

Listing 5.1: Retrieve the citations for a paper by its DOI.

Retrieving the citation data from OpenCitations

The API's *citations* endpoint was individually queried using the same 100 DOIs as used in database queries. An example request is `https://opencitations.net/index/coci/api/v1/citations/10.1002/adfm.201505328`. Following the same process as in the database test, the results of the requests were stored in a JSON file.

5.2.3 Evaluating the application's completeness

For each paper, the citations from both sources were compared. Two lists were created, one containing the citations only present in the database results and the other, the citations only present in the OpenCitations results.

The DOIs of the citations in the latter list were then queried in the database. If the DOI was present in the database, it was predicted one of two scenarios had occurred. First, the Cypher query did not identify the citing relationship between the two papers, and as a result, the citing paper was missing from the results of the query. The second scenario was that the citation dataset ingested into the database did not contain the citing relationship between the two papers.

The 100 papers combined had 1,645 citations. Of these 1,645, 19 papers that cite one of the 100 test papers are present in the database, and appear in the OpenCitations results, but not in the database results.

The 19 missing citations were spread across 13 papers. Nine test papers were missing 1 citation in the database results, 3 test papers were missing 2, and one test paper was missing 4 citations. As the missing citations did not appear as inward citations to the 13 papers, a reversed query, where the outward citations were obtained for the 19 papers, was written to discover why the relationships were not identified.

The reversed query returned a list of the paper's references. These lists were checked to see if they contained any of the test papers. Twelve of the missing papers had an

empty reference list. Six of the missing papers had a populated references list but the test papers were not in them. Finally, one 'missing paper' was a self-citation.

In summary, the application returned all the citations it had in its database.

5.2.4 General observations of the results

General statistics were calculated to compare the number of citations from each data source. A breakdown of the total number of citations can be seen in Table 5.3. Overall, OpenCitations returned 128 more citations than the database. An interesting discovery was that there were only 671 citations that appeared in the results of both datasets, where the citations were for the same paper.

Upon further inspection, it was discovered that a paper could have multiple DOIs. For example, the paper *FORMLESS: scalable utilization of embedded manycores in streaming applications* has the DOI 10.1145/2248418.2248429 in the database, but the DOI 10.1145/2345141.2248429 is returned from OpenCitations. The DOIs were resolved using the online tool at `https://www.doi.org/` and both resolved to the same paper, however, published in different journals. The database DOI referred to the version published in *LCTES '12: Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems*, and in OpenCitations, to the version published in *ACM SIGPLAN Notices, Volume 47, Issue 5*.

In order to determine were there more occurrences of two different DOIs being used, and if so to see its impact on the number of common citations, a regular expression was written to find more occurrences. This was possible as the characters to the left of the forward slash, and the digits after the last full stop, were identical in both DOIs. However, only the above scenario was found.

As the database returned 128 fewer citations than OpenCitations, it was desired to see whether this was due to the database returning fewer citations on average per paper, or was it down to the database containing only a small fraction of a few test papers' total citations. In other words, was the dataset's coverage smaller across all papers, or did a handful of papers contribute to the figure.

In Table 5.4, it can be seen that only 23 test papers had the same number of citations from both sources. Forty-four papers had more citations from the database than

	Count
Total number of citations	1,645
Total number of Neo4j citations	1,094 (423 unique)
Total number of OpenCitations citations	1,222 (551 unique)
Total number of common citations	671

Table 5.3: Breakdown of citation counts in completeness evaluation dataset.

Papers	Count
Sources had equal citations	23
Neo4j database had more citations	44
OpenCitations had more citations	33
Total	100

Table 5.4: Completeness evaluation sources with the most citations by paper.

OpenCitations, 11% higher than the number of test papers with more OpenCitations citations (33 papers).

5.3 Evaluation of the machine learning approach

In this evaluation the process to train the citation prediction model and the accuracy of the results are reviewed. The metric used in measuring the accuracy is also evaluated.

5.3.1 Citation prediction model

The machine learning model undertaken was a prototype, not a fully fledged model. The MSE of the final model, 114.1638, is not accurate enough to be implemented into the application. There are a few changes that could have been made in the testing process, as well as a few areas of future work.

There are a number of machine learning model types that were not tested due to time constraints. This includes recurrent neural networks (RNN) which would have been interesting to test, especially in working with the text features. The untested models could be far more accurate at predicting citation counts.

The training dataset itself had 100,000 entries. A larger dataset could have been beneficial, especially with the sizable number of features when including all of the text feature transformations. More papers could capture the relationships between the different input features and the predicted field better. A random sample of data was used for the dataset, while the number of entries were equal for each year, further research could be applied to ensure the dataset is a good representation of the citation dataset.

Multiple other statistics could be generated from the database and used as features. However, the process would need to run more permutations of the features to find the combination of features with the highest accuracy. Another area that could improve accuracy is testing more feature transformation settings, including different scalers for numerical features and count limits on the vocabulary size, different stop word libraries, and different tokenisation and preprocessing methods for text features.

5.3.2 Mean squared error metric

The mean squared error (MSE) metric was used throughout the model selection and feature engineering process as a judgement of accuracy. However, this metric is not without its faults. The MSE score is not unit-free, meaning that its value does not remain constant when the underlying distributions are changed. This means the value can be improved by altering the scale of the prediction distribution [69].

For example, the ideal MSE value is as close to zero as possible. In the steps undertaken in the implementation of the model, the value being predicted, the number of citations, was not normalised. The lowest MSE recorded was 94.5732. However, if the y of the model, the field being predicted, is normalised, the MSE plummets to 0.0479. Despite this considerable improvement in the MSE score, it did not result in a improvement in the predictions.

5.4 Summary

This chapter evaluated the design decision to implement two indexes in the database and tested the completeness of the database functionalities to query and locate citation relationships. The indexes added greatly improved the performance of the application

and was a valuable design decision. The database queries were successful in returning all the citations it contained for a paper. Missing citations, where a paper that cited another was not returned in the results, was due to the relationship not being included in the dataset.

The citation prediction model is still in its infancy. Changes to the testing process and areas of future work are suggested. Finally, the metric used throughout the process and its drawbacks are identified.

Chapter 6

Conclusion

6.1 Conclusion

A citation network exploration application was implemented in this work. An AMiner citation dataset was ingested into a Neo4j graph database. Indexes were added to both nodes types, paper and author, to improve performance. A micro-service was created that takes user search parameters, retrieves the citation data for those parameters, and returns an interactive citation network. The citation network can be viewed in graphical and table form. Users can filter the data in the graph, and can view statistics about the network. Users have a choice of search methods, by paper title or by author name, by matching the exact term or partial matches, and lastly by inward or outward citations.

The application is fast. Database queries, to return a citation network including papers up to four hops away, take on average less than a second. The entire search process, from pressing the go button on the search page to the results page rendering, takes approximately 1.07 seconds. This process includes formatting and visualising the citation network data, compiling the statistics, and building the filters.

The results are complete, as assessed in subsection 5.2.3. The 5 million papers and 48 million citations mean the citation networks are not sparse, and with the degrees of separation even papers with a single citation can have a populous network graph.

The application aims to aid the user in discovery. The nodes in the citation graph can be filtered by year, number of citations, number of references, hops, keywords,

languages, and by PDF availability simultaneously.

Finally, with the prototype citation prediction model, the predicted top papers in the network are calculated and highlighted in the graph. At a glance, the user can see papers of potential relevance without the need to view each node in the graph individually.

6.2 Future work

There are four key areas for future work, the citation prediction model, expanding the search functionality, PDF parsing, and expanding the citation database.

The citation prediction model is the primary area of opportunity. There are a number of actions that can be taken to improve the accuracy of the model. These actions have been outlined in subsection 5.3.1. Also, the predicted citation counts have further applications, including predicting the future relevance of authors and fields.

In the current version of the application, it is possible to search by paper title and author. A beneficial enhancement would be to enable searching by topic, where the resulting citation network would show the most important papers, and those papers predicted to be important, for that topic. One use of this search feature would be in identifying the best papers to read when exploring a field for the first time, or in predicting the direction of the field.

The AMiner dataset contains links to PDFs for a number of the papers. This introduces the opportunity to examine the context behind the citation. Sentiment analysis and statistics could be performed on those PDFs to provide further insight into a citation network. For example, if a source is cited regularly in a paper, it can be argued it is more important than a source that is cited once. As a citation is not always positive, by performing sentiment analysis on the context of the citation, it can be guessed whether the offerings of a paper are viewed positively or negatively. Papers that receive particularly positive or negative mentions could be highlighted in the network.

Finally, the dataset utilised contained 5 million papers, predominately from the field of computer science. In order to become useful to students and academics in other fields, either a new dataset containing papers across multiple fields could be used, or additional field-specific datasets could be merged with the AMiner dataset,

to expand the coverage of the citation database. This would have the added benefit of incorporating more cross-field citations that can identify interesting collaborations between different fields.

Bibliography

[1] A. Lauscher, K. Eckert, L. Galke, A. Scherp, S. T. R. Rizvi, S. Ahmed, A. Dengel, P. Zumstein, and A. Klein, "Linked open citation database: Enabling libraries to contribute to an open and interconnected citation graph," in *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries*, JCDL '18, (New York, NY, USA), p. 109–118, Association for Computing Machinery, 2018.

[2] L. Gao, X. Qi, Z. Tang, X. Lin, and Y. Liu, "Web-based citation parsing, correction and augmentation," in *Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '12, (New York, NY, USA), p. 295–304, Association for Computing Machinery, 2012.

[3] Z. Guo and H. Jin, "Reference metadata extraction from scientific papers," in *2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 45–49, 2011.

[4] I. G. Councill, C. L. Giles, and M.-Y. Kan, "Parscit: an open-source crf reference string parsing package," in *LREC*, 2008.

[5] X. Liu, J. Zhang, and C. Guo, "Full-text citation analysis: Enhancing bibliometric and scientific publication ranking," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, (New York, NY, USA), p. 1975–1979, Association for Computing Machinery, 2012.

[6] "Frequently Asked Questions about Microsoft Academic Graph." `https://docs.microsoft.com/en-us/academic-services/graph/resources-faq`. Retrieved July 2021.

[7] S. Effendy and R. H. Yap, "Analysing trends in computer science research: A preliminary study using the microsoft academic graph," in *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, (Republic and Canton of Geneva, CHE), p. 1245–1250, International World Wide Web Conferences Steering Committee, 2017.

[8] I. Wesley-Smith, C. Bergstrom, and J. West, "Static ranking of scholarly papers using article-level eigenfactor (alef)," *arXiv*, 06 2016.

[9] J. Portenoy and J. D. West, "Visualizing scholarly publications and citations to enhance author profiles," in *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, (Republic and Canton of Geneva, CHE), p. 1279–1282, International World Wide Web Conferences Steering Committee, 2017.

[10] Y. Wang, C. Shi, L. Li, H. Tong, and H. Qu, "Visualizing research impact through citation data," *ACM Trans. Interact. Intell. Syst.*, vol. 8, Mar. 2018.

[11] "Next Steps for Microsoft Academic – Expanding into New Horizons." `https://www.microsoft.com/en-us/research/project/academic/articles/microsoft-academic-to-expand-horizons-with-community-driven-approach/`. Retrieved July 2021.

[12] "dblp: computer science bibliography." `https://dblp.org/`. Retrieved July 2021.

[13] L. Keselman, *Venue Analytics: A Simple Alternative to Citation-Based Metrics*, p. 315–324. IEEE Press, 2019.

[14] L. Abazi-Bexheti, A. Kadriu, and M. Apostolova, "Word cloud analytics of the computer science research publications' titles over the past half century," in *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pp. 887–892, 2020.

[15] S. Yudhoatmojo and M. Samuar, "Community detection on citation network of dblp data sample set using linkrank algorithm," *Procedia Computer Science*, vol. 124, pp. 29–37, 01 2017.

[16] "Google Scholar Search Tips." https://scholar.google.com/intl/us/scholar/help.html. Retrieved August 2021.

[17] "Google Scholar API — Scrape Google Scholar - SerpApi." https://serpapi.com/google-scholar-api. Retrieved July 2021.

[18] "Scholarly - PyPI." https://pypi.org/project/scholarly/. Retrieved August 2021.

[19] "How Scopus works: Information about Scopus product features." https://www.elsevier.com/solutions/scopus/how-scopus-works. Retrieved August 2021.

[20] "Elsevier Developer Portal." https://dev.elsevier.com/api_key_settings.html. Retrieved August 2021.

[21] "OpenCitations - Download." https://opencitations.net/download. Retrieved August 2021.

[22] J. Ortega and I. Aguillo, "Microsoft academic search and google scholar citations: Comparative analysis of author profiles," *Journal of the American Society for Information Science and Technology*, vol. 65, 06 2014.

[23] R. Fatima, A. Yasin, L. Liu, and J. Wang, "Google scholar vs. dblp vs. microsoft academic search: An indexing comparison for software engineering literature," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 1097–1098, 2020.

[24] T. Nguyen and P. Do, "Managing and visualizing citation network using graph database and lda model," in *Proceedings of the Eighth International Symposium on Information and Communication Technology*, SoICT 2017, (New York, NY, USA), p. 100–105, Association for Computing Machinery, 2017.

[25] Y. Unal and H. Oguztuzun, "Migration of data from relational database to graph database," in *Proceedings of the 8th International Conference on Information Systems and Technologies*, ICIST '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[26] H. Vyawahare, P. Karde, and V. Thakare, "A hybrid database approach using graph and relational database," in *2018 International Conference on Research in Intelligent and Computing in Engineering (RICE)*, pp. 1–4, 2018.

[27] J. Portenoy, J. Hullman, and J. West, "Leveraging citation networks to visualize scholarly influence over time," *Frontiers in Research Metrics and Analytics*, vol. 2, 11 2016.

[28] Z. Shen, M. Ogawa, S. Tee Teoh, and K. Ma, "Biblioviz: A system for visualizing bibliography information," in *In Proceedings of Asia-Pacific Symposium on Information Visualization*, vol. 60, pp. 93–102, 2006.

[29] P. Bergström and D. C. Atkinson, "Augmenting the exploration of digital libraries with web-based visualizations," in *2009 Fourth International Conference on Digital Information Management*, pp. 1–7, 2009.

[30] B. Shneiderman, "The eyes have it: a task by data type taxonomy for information visualizations," in *Proceedings 1996 IEEE Symposium on Visual Languages*, pp. 336–343, 1996.

[31] J. Matejka, T. Grossman, and G. Fitzmaurice, "Citeology: Visualizing paper genealogy," in *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, (New York, NY, USA), p. 181–190, Association for Computing Machinery, 2012.

[32] G. Costagliola and V. Fuccella, "Cybis: A novel interface for searching scientific documents," in *2011 15th International Conference on Information Visualisation*, pp. 276–281, 2011.

[33] A. Dattolo and M. Corbatto, "Visualbib: Narrative views for customized bibliographies," in *2018 22nd International Conference Information Visualisation (IV)*, pp. 133–138, 2018.

[34] C. Wang, S. Xu, L. Chen, and J. Li, "The application of graph algorithms: A reference mapping tool," in *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 1–6, 2015.

[35] R. Ishida, S. Takahashi, and H. Wu, "Interactively uncluttering node overlaps for network visualization," in *2015 19th International Conference on Information Visualisation*, pp. 200–205, 2015.

[36] J. Hirsch, "An index to quantify an individual's scientific research output," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, pp. 16569–72, 11 2005.

[37] L. Egghe, "Theory and practice of the g-index," *Scientometrics*, vol. 69, 04 2006.

[38] "Google Scholar Citations Open To All." `https://scholar.googleblog.com/2011/11/google-scholar-citations-open-to-all.html`. Retrieved August 2021.

[39] G. Doğan, I. Şencan, and Y. Tonta, "Does dirty data affect google scholar citations?," in *Proceedings of the 79th ASIS&T Annual Meeting: Creating Knowledge, Enhancing Lives through Information & Technology*, ASIST '16, (USA), American Society for Information Science, 2016.

[40] R. Bevern, C. Komusiewicz, R. Niedermeier, M. Sorge, and T. Walsh, "H-index manipulation by merging articles: Models, theory, and experiments," *Artificial Intelligence*, 07 2015.

[41] L. Leydesdorff, "Caveats for the use of citation indicators in research and journal evaluations," *Journal of The American Society For Information Science and Technology*, vol. 59, pp. 278–287, 01 2007.

[42] V. Anauati, S. Galiani, and R. Gálvez, "Quantifying the life cycle of scholarly articles across fields of economic research," *Economic Inquiry*, vol. 54, pp. n/a–n/a, 11 2015.

[43] T. Chakraborty, S. Kumar, P. Goyal, N. Ganguly, and A. Mukherjee, "Towards a stratified learning approach to predict future citation counts," in *IEEE/ACM Joint Conference on Digital Libraries*, pp. 351–360, 2014.

[44] M. Singh, A. Jaiswal, P. Shree, A. Pal, A. Mukherjee, and P. Goyal, "Understanding the impact of early citers on long-term scientific impact," in *Proceedings of*

the 17th ACM/IEEE Joint Conference on Digital Libraries, JCDL '17, p. 59–68, IEEE Press, 2017.

[45] J. Xu, M. Li, J. Jiang, B. Ge, and M. Cai, "Early prediction of scientific impact based on multi-bibliographic features and convolutional neural network," *IEEE Access*, vol. 7, pp. 92248–92258, 2019.

[46] E. Bütün, M. Kaya, and R. Alhajj, "A supervised learning method for prediction citation count of scientists in citation networks," in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, ASONAM '17, (New York, NY, USA), p. 952–958, Association for Computing Machinery, 2017.

[47] S. Sarkar, R. Lakdawala, and S. Datta, "Predicting the impact of software engineering topics: An empirical study," in *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, (Republic and Canton of Geneva, CHE), p. 1251–1257, International World Wide Web Conferences Steering Committee, 2017.

[48] X. P. Zhu and Z. Ban, "Citation count prediction based on academic network features," in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pp. 534–541, 2018.

[49] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: Extraction and mining of academic social networks," in *KDD'08*, pp. 990–998, 2008.

[50] "Citation Network Dataset: DBLP+Citation, ACM Citation network." `https://www.aminer.cn/citation`. Retrieved June 2021.

[51] "Introduction to Scopus APIs." `https://www.elsevier.com/de-de/events/scopus-community-anwendertreffen?a=917179`. Retrieved July 2021.

[52] S. Almabdy, "Comparative analysis of relational and graph databases for social networks," in *2018 1st International Conference on Computer Applications Information Security (ICCAIS)*, pp. 1–4, 2018.

[53] M. Singh and K. Kaur, "Sql2neo: Moving health-care data from relational to graph databases," in *2015 IEEE International Advance Computing Conference (IACC)*, pp. 721–725, 2015.

[54] P. Shi, G. Fan, S. Li, and D. Kou, "Big data storage technology for smart distribution grid based on neo4j graph database," in *2021 IEEE 4th International Conference on Electronics Technology (ICET)*, pp. 441–445, 2021.

[55] "Graph Database Platform — Graph Database Management System — Neo4j." `https://neo4j.com/`. Retrieved July 2021.

[56] "Amazon Neptune Pricing." `https://aws.amazon.com/neptune/pricing/`. Retrieved August 2021.

[57] "Welcome to Flask." `https://flask.palletsprojects.com`. Retrieved July 2021.

[58] "Jinja - Jinja Documentation." `https://jinja.palletsprojects.com`. Retrieved July 2021.

[59] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference* (Stéfan van der Walt and Jarrod Millman, eds.), pp. 56 – 61, 2010.

[60] "Memory configuration - Operations Manual." `https://neo4j.com/docs/operations-manual/current/performance/memory-configuration/`. Retrieved August 2021.

[61] "Bootstrap - The most popular HTML, CSS, and JS library in the world.." `https://getbootstrap.com/`. Retrieved August 2021.

[62] "Tailwind CSS - Rapidly build modern websites without ever leaving your HTML.." `https://tailwindcss.com/`. Retrieved August 2021.

[63] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11 – 15, 2008.

[64] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Softw. Pract. Exper.*, vol. 21, p. 1129–1164, Nov. 1991.

[65] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Inf. Process. Lett.*, vol. 31, p. 7–15, Apr. 1989.

[66] "Dash Overview." `https://plotly.com/dash/`. Retrieved August 2021.

[67] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[68] "REST API for COCI, the OpenCitations Index of Crossref open DOI-to-DOI references." `https://opencitations.net/index/coci/api/v1#/citations/{doi}`. Retrieved August 2021.

[69] Y. Graham, "Improving evaluation of machine translation quality estimation," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, (Beijing, China), pp. 1804–1813, Association for Computational Linguistics, July 2015.

Appendix A

Dataset

A.1 Dataset ingestion

```
neo4j-admin import \
    --database=neo4j \
    --skip-duplicate-nodes \
    --skip-bad-relationships \
    --nodes=imports/papers.csv \
    --nodes=imports/authors.csv \
    --relationships=imports/paper_refs.csv \
    --relationships=imports/author_refs.csv
```

Listing A.1: The command to ingest the CSV files into the Neo4j database.

A.2 Dataset fields

Field	Populated	% Populated	Not Populated	% Not Populated
abstract	3864877	72%	1489432	28%
doi	4001707	75%	1352602	25%
fos	5003384	93%	350925	7%
isbn	842450	16%	4511859	84%
issn	2101353	39%	3252956	61%
issue	2011224	38%	3343085	62%
keywords	4182722	78%	1171587	22%
lang	4726363	88%	627946	12%
n_citation	4775108	89%	579201	11%
n_references	5354309	100%	0	0%
page_end	4469228	83%	885081	17%
page_start	4608990	86%	745319	14%
pdf	652343	12%	4701966	88%
title	5353918	100%	391	0%
volume	2853562	53%	2500747	47%
year	5354295	100%	14	0%

Table A.1: The fields in the citation dataset and their populations.

A.3 Dataset formatting times

	Subset File	Number of Nodes (at end)	Formatting Duration (mins)
1	aa	149999	4.55
2	ab	150000	4.53
3	ac	150000	4.87
4	ad	150000	4.80
5	ae	150000	4.73
6	af	150000	4.78
7	ag	150000	4.80
8	ah	150000	4.32
9	ai	150000	4.08
10	aj	150000	4.18
11	ak	150000	4.73
12	al	150000	4.80
13	am	150000	4.58
14	an	150000	4.73
15	ao	150000	4.85
16	ap	150000	4.28
17	aq	150000	4.40
18	ar	150000	5.33
19	as	150000	4.87
20	at	150000	4.57
21	au	150000	4.32
22	av	150000	3.62
23	aw	150000	4.35
24	ax	150000	5.05
25	ay	150000	4.95
26	az	150000	4.83
27	ba	150000	5.00
28	bb	150000	6.50
29	bc	150000	6.55
30	bd	150000	6.42
31	be	150000	5.00
32	bf	150000	4.25
33	bg	150000	4.33
34	bh	150000	4.20
35	bi	150000	4.55
36	bj	104310	2.83
		5354309	169.53

Table A.2: The formatting times for each subset of the citation dataset.

Appendix B

Cypher Queries

B.1 Searches by paper title

Exact match query

```
MATCH (a:Paper {
    title: 'Experiments with Classifier Combining Rules'
})
OPTIONAL MATCH path=(a)-[r:REFERENCES]->(b)
CALL {
    WITH b
    OPTIONAL MATCH paths = (b:Paper)-[h:REFERENCES*1..3]->(c)
    WITH c, paths, h
    LIMIT 10
    RETURN c, h, length(paths) as hoppies
}
RETURN a, b, r, c, h, length(path) as hops, hoppies
LIMIT 100
```

Listing B.1: Search for exact paper title.

Where contains query

```
MATCH (a:Paper)
WHERE a.title =~ '.*The relationship between.*'
OPTIONAL MATCH path=(a)-[r:REFERENCES]->(b)
CALL {
    WITH b
    OPTIONAL MATCH paths = (b:Paper)-[h:REFERENCES*1..3]->(c)
    WITH c, paths, h
    LIMIT 10
    RETURN c, h, length(paths) as hoppies
}
RETURN a, b, r, c, h, length(path) as hops, hoppies
ORDER BY hops, hoppies
LIMIT 100
```

Listing B.2: Search for paper titles containing search term.

B.2 Searches by author name

Exact match query

```
MATCH (a:Author {name: 'Yun Xu'})
OPTIONAL MATCH path=(a)-[r:AUTHORED]->(b)
CALL {
    WITH b
    OPTIONAL MATCH paths = (b:Paper)-[h:REFERENCES*1..3]->(c)
    WITH c, paths, h
    LIMIT 10
    RETURN c, h, length(paths) as hoppies
}
RETURN a, b, r, c, h, length(path) as hops, hoppies
ORDER BY hops, hoppies
LIMIT 150
```

Listing B.3: Search for exact author name.

Where contains query

```
MATCH (a:Author)
WHERE a.name =~ '.*Yun.*'
OPTIONAL MATCH path=(a)-[r:AUTHORED]->(b)
CALL {
    WITH b
    OPTIONAL MATCH paths = (b:Paper)-[h:REFERENCES*1..3]->(c)
    WITH c, paths, h
    LIMIT 10
    RETURN c, h, length(paths) as hoppies
}
RETURN a, b, r, c, h, length(path) as hops, hoppies
ORDER BY hops, hoppies
LIMIT 150
```

Listing B.4: Search for author names containing search term.

Appendix C

Application Screenshots

C.1 Screenshot of search by author

Please see next page for screenshot.

C.2 Screenshot of search where title contains search query

Please see two pages ahead for screenshot.

Figure C.1: Screenshot of a citation network following a search by author name.

Figure C.2: Screenshot of a search for paper titles containing 'interactive visualization', returning outward references, with the year filtered for 2000-2016.