DSC202: Knowledge Graph for Scientific Research Papers

Lila Horwitz	Mizuho Fukuda	Ashley Ho
Data Science	Data Science	Data Science
A17608597	A16310384	A16337509

1 Introduction

1.1 Background and Motivation

As students, we wanted to create an way to search for academic sources that have similar attributes to one another for research purposes. However, we also want to avoid citing overly similar papers, as we prefer diverse sources of information to get a well-rounded corpus of information for projects. For example, authors that belong to the same community and strictly cite one another may provide a narrow perspective or biased information on a topic. This project ideally would allow us to see which sources are similar to one another, which sources are cited the most, and detect communities of both papers and authors based on citations. We would also like to create a system which recommends papers that are useful for research by finding similar papers to an input paper.

2 Data Sources

2.1 Data Source

We utilized a large-scale research papers dataset sourced from Kaggle (access data source here), comprising 1,000,000 unique academic publications across a diverse range of disciplines, including computer science, mathematics, physics, and related fields. Each entry in the dataset includes a variety of metadata fields, as shown below.

Field	Description
`id`	A unique identifier for each research paper.
`title`	The title of the research paper.
`authors`	A list of author names associated with the paper.
`venue`	The journal or conference where the paper was published.
`year`	The publication year of the paper (1937–2017 in the original dataset).
`n_citation`	The number of times the paper has been cited.
`references`	A list of paper IDs that are cited by the current paper.
`abstract`	The abstract summarizing the paper's content.

2.2 Preprocessing

Given the limited computational resources available to us, we applied the following preprocessing steps using the Python Pandas package to reduce the size of the data significantly.

1. Retained only papers with more than 10 citations (n_citation > 10).
2. Filtered for papers that contain at least one reference in the references field.
3. Removed papers with empty or missing abstracts. Limited the dataset to papers published in 2010 or later (year >= 2010).
4. Excluded venues that appeared only once, keeping only those with at least two publications in the dataset.
5. Removed any paper id's in references that does not appear in the id column after processing.

In addition, to incorporate topic-level information into our analysis, we applied Latent Dirichlet Allocation (LDA) to the abstract texts. LDA is an unsupervised probabilistic topic modeling technique that assumes each document is a mixture of topics, and each topic is a distribution over words. By analyzing word co-occurrence patterns across abstracts, LDA infers a set of latent topics that best capture the thematic structure of the corpus. We configured the model to identify 15 distinct topics, and for each paper, we assigned the most dominant topic—i.e., the one with the highest probability—as its representative topic. This resulted in a new categorical feature, `topic`, with values ranging from 0 to 14, indicating the paper's primary topic assignment.

The first five rows of the resulting Pandas DataFrame is displayed below. The final data contains 134,288 unique papers, which is 13% of the original size of the data, making it more manageable for our analysis.

	id	title	year	n_citation	paper_id	abstract	topic
0	4ab3f7cd-140b-4e29-99d4-f4e8006c4f65	A Push–Pull Class-C CMOS VCO	2013	50	4ab3f7cd-140b-4e29-99d4-f4e8006c4f65	A CMOS oscillator employing differential trans...	4
1	4ab4c0a1-3c5a-44c6-bdd4-3a0618d303ca	Evaluating the accuracy of Java profilers	2010	95	4ab4c0a1-3c5a-44c6-bdd4-3a0618d303ca	Performance analysts profile their programs to...	11
2	4ab5e3f4-9b58-4fbb-9bde-ee2f2185cc61	Novel Weighting-Delay-Based Stability Criteria...	2010	266	4ab5e3f4-9b58-4fbb-9bde-ee2f2185cc61	In this paper, a weighting-delay-based method ...	1
3	4ab5e4bd-08e2-4007-825c-d34ce7cb231f	Cross-Language Information Retrieval	2010	108	4ab5e4bd-08e2-4007-825c-d34ce7cb231f	Search for information is no longer exclusivel...	11
4	4ab6c0ff-2d1a-4cb0-acdc-8ca7517cb14a	starBase v2.0: decoding miRNA-ceRNA, miRNA-ncR...	2014	315	4ab6c0ff-2d1a-4cb0-acdc-8ca7517cb14a	Although microRNAs (miRNAs), other non-coding ...	9

Figure 1: Cleaned Data

3 Methodology

3.1 Database Schemas

To facilitate our analysis, we utilized two different database structures: a relational database (**PostgreSQL**) and a graph database (**Neo4j**). Each database schema was designed to complement the other, enabling efficient data management and analysis across different types of queries.

The relational database contains three tables that store structured metadata and large text content about the academic papers:

- `papers`: This table holds metadata for each paper, including the title, publication year, topic, and number of citations.
- `authors`: This table represents the relationship between authors and papers, with each row corresponding to an author-paper pair.
- `abstracts`: This table stores the abstract text of each paper.

The relational database is primarily used to store and retrieve structured information about papers, authors, and their abstracts. It provides efficient handling of metadata and large textual content, making it ideal for structured querying and analysis.

Next, the graph database is structured around three types of nodes and their relationships:

- `Paper`: Represents individual academic papers.
- `Author`: Represents authors of papers.

- `Venue`: Represents the venues (e.g., journals, conferences) where papers were published.

The relationships (edges) between these nodes are as follows:

- `CITED`: Indicates a citation relationship between two papers.
- `AUTHORED`: Represents the authorship of a paper, with an edge pointing from an author to a paper.
- `PUBLISHED`: Indicates the venue that published the paper.

Thus, the node-edge relationships in our graph database are:

- `Paper-[CITED]->Paper`
- `Author-[AUTHORED]->Paper`
- `Venue-[PUBLISHED]->Paper`

Below is an example of a network displaying the papers writen by a specific author, as displayed on Neo4j Desktop. The orange node represents an author and the purple nodes represent papers. The directed edges represent authorship.

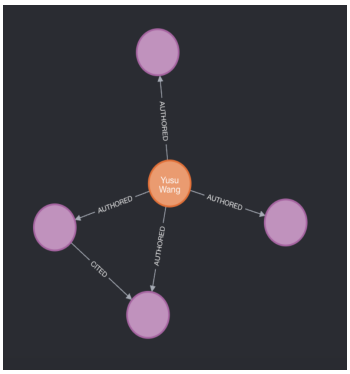

Figure 2: Example of Node-Edge in Neo4j

This schema allows for modeling complex citation networks, authorship patterns, and relationships between papers and venues. Neo4j's graph-based structure is well-suited for traversing and analyzing relationships in citation networks, offering an advantage over the relational database in tasks such as community detection, centrality measures, and network-based analysis.

3.2 Methodology

To analyze the structure of the paper network, leveraged the strengths of both Neo4j and PostgreSQL to extract meaningful insights. Neo4j allowed us to represent the paper network as a directed graph, where nodes corresponded to academic papers and edges represented citation and author relationships. This structure enabled us to apply graph-based techniques such as PageRank and Louvain modularity to efficiently traverse relationships between papers, capturing complex connections that might be difficult to model in a traditional relational database.

While Neo4j provided an efficient way to model relationships, PostgreSQL was used in parallel to perform structured queries and aggregations on the data. For each of our use case queries, we transferred data between the two databases to take advantage of both approaches. For instance, after performing computations in Neo4j, such as measuring influence scores or identifying communities, we transferred the results to PostgreSQL for further structured analysis. Likewise, data stored in PostgreSQL was sent to Neo4j when a graph-based approach was more suitable for identifying relationships within the network. This exchange between databases allowed us to take advantage of Neo4j's graph traversal capabilities while still leveraging SQL for filtering, aggregations, and comparisons with other structured data.

To facilitate interaction Neo4j and PostgreSQL, we conducted our analysis within a Jupyter Notebook environment. Using Jupyter Notebook centralized our workflow, enabling us to write, execute, and refine queries while keeping all results in one place for easy inspection. This setup streamlined the process of passing data between the two databases and performing additional transformations and analysis.

To connect to Neo4j, we used the `neo4j` Python package along with `GraphDataScience`, which provided a convenient interface for executing graph-based queries and applying network analysis algorithms supported by the database. Similarly, we established a connection to PostgreSQL using `sqlalchemy`, which allowed us to write and execute SQL queries directly from the notebook after connecting to our local SQL server. Queries for both databases were written as strings, ensuring flexibility in modifying and re-running them as needed. Here is the code for our connection setup:

```
# neo4j
uri = "neo4j://localhost:7687"
driver = GraphDatabase.driver(uri, auth=("neo4j", password))
gds = GraphDataScience(uri, auth=("neo4j", password))

# postgres
engine = create_engine(f"postgresql://{db_user}:{db_password}@{db_host}:{db_port}/{db_name}")
```

Here is a simple example of how we passed data between the two databases and utilize `pandas` to extract the nodes representing the first five papers in our data:

```
query = "SELECT paper_id FROM papers LIMIT 5"
df = pd.read_sql(query, engine)

with driver.session() as session:
    query = f"""
        MATCH (p:Paper)
        WHERE p.id IN {df['paper_id'].tolist()}
        RETURN p.id, p.title, p.year
    """
    result = session.run(query).data()
result_df = pd.DataFrame(result)
```

To send results from Neo4j to PostgreSQL, we did the following:

```
results_df.to_sql("results", engine, if_exists="replace", index=False)
```

Our final output consisted of the results of the queries that incorporated both databases, which were converted into `pandas` DataFrames for readability and potential further analysis and visualization. This combined approach allowed us to balance the flexibility of graph traversal with the structured querying capabilities of a traditional relational database, ensuring a comprehensive analysis of the data.

4 Use-Case Queries

We implemented four use-case queries to explore and analyze the paper network:

1. Most Influential Paper Per Topic Per Year
2. Self-Citation Rate Per Topic
3. Most Frequent Topic Within Each Community
4. Paper Recommender System

4.1 Most Influential Paper Per Topic Per Year

In this use case, our goal was to identify the most influential paper within each topic for each year. To achieve this, we combined SQL for organizing the data and Neo4j's graph algorithms for measuring paper influence.

4.1.1 Motivation

Academic influence is often context dependent: the impact of a paper is more meaningful compared to its peers within the same field and period of time. By focusing on specific topics and years, this query helps identify the most influential papers within particular academic communities, rather than evaluating their impact in a broader, global context. This approach allows us to better understand how papers contribute to the evolution of research in their specific areas, identifying influential work that may not be as globally recognized but is central to certain domains.

4.1.2 Grouping Papers by Topic and Year

The first step in this analysis involved using PostgreSQL to group the papers based on their topic and year of publication. By aggregating the paper IDs for each combination of topic and year, we created distinct groups to focus on. This structured organization of the data was crucial for narrowing the scope and ensuring that each group was being evaluated individually based on its topic and the year of publication.

4.1.3 Using PageRank to Measure Influence

Once we had the papers grouped by topic and year, we used Neo4j's PageRank algorithm to rank the papers within each group by their influence. PageRank is a widely used graph-based algorithm that quantifies a paper's influence by considering both the number of times it has been cited and the significance of the papers citing it. Rather than treating all citations equally, the algorithm assigns higher weight to citations from influential papers. This recursive approach ensures that a paper cited by highly regarded sources receives a higher influence score than one cited primarily by less-referenced works. This allows us to identify papers that are influential not just by citation count, but by the quality of the citing papers as well. The algorithm assigns each paper a score that reflects its relative importance within the citation network.

4.1.4 Identifying the Top Paper

For each group (defined by a specific topic and year), the PageRank algorithm ranks the papers based on their influence score. The paper with the highest score within each group is considered the most influential for that topic and year. By applying this methodology across all topic-year combinations, we were able to determine the most influential paper for each group.

4.1.5 Results

The result of this query is a Pandas DataFrame containing one row for every topic and year combination, with the paper ID of the most influential paper of that group and its influence score as calculated by PageRank. Below are the first five rows of the resulting DataFrame:

	topic	year	paper_id	influence_score
0	0	2010	9fba111f-d65a-4c21-b02e-a8d4fbbdf92c	5.084531
1	0	2011	cac8070b-570d-4674-b334-dd96bbc223c6	3.400359
2	0	2012	7a18d786-740c-4001-b1db-95f7d7b0ae1a	2.811377
3	0	2013	f9b3409e-ab94-47f4-b926-6702900141e2	2.265877
4	0	2014	f121ff70-19ef-47d8-bc6a-61a7aa201caa	2.261410

Figure 3: Results of Use-Case Query 1

This allowed us to identify not only the most influential papers but also gain insights into how influence might vary across different topics and years.

4.2 Self-Citation Rate Per Topic

4.2.1 Motivation

Self-citation occurs when authors reference their own previous work in new publications. While this can reflect an author's ongoing contribution to a specific research area, an excessive rate of

self-citation may raise concerns about the breadth of the literature being cited. By calculating the self-citation rate for each topic, we can gain insights into how authors interact with their own past work within particular academic fields. This query also helps identify topics that may be under-researched or emerging, as we infer that these fields often show higher self-citation rates. This is because fewer prior works are available to cite, leading authors to rely more heavily on their own research rather than incorporating the works of other authors.

4.2.2 Identifying Self-Citation Counts

The first step in our analysis involved querying the Neo4j graph database to identify self-citations—cases where authors cite their own papers. The Cypher query matched authors who authored papers that later cited other papers they had written. By using this query, we were able to capture the self-citations for each author, counting how often they referenced their own work in relation to other papers in the database. This query was easily executed by leveraging the `[AUTHORED]` and `[CITED]` relationships in our graph database. In contrast, performing this analysis in a relational database would be significantly more complex, as it would require self-joins or self-merging to link authors to their own citations. The graph structure of Neo4j naturally simplifies this process, allowing us to traverse the relationships between authors and papers more directly. Below are the first four rows of this Neo4j query:

	author_name	self_citations
0	Loet Leydesdorff	64
1	Rui Zhang	63
2	Dacheng Tao	63
3	Li Deng	62

Figure 4: Results of Intermediate Neo4j Query

4.2.3 Linking Authors to Topics

Once we identified the self-citations, we transferred our results to PostgreSQL to group authors by the topics they contributed to. Since authors can write papers that belong to different topics, we assigned each author a main topic to use for our self-citation rate calculation. The author's main topic is the topic that they wrote the most papers in. To do this, we used SQL to join the authors and papers tables, associating each author with the topics of the papers they authored. By applying the `RANK()` window function, we ensured that we identified the most significant topic for each author. This step allowed us to link the self-citation data to specific topics, ensuring that the self-citation rate could be calculated for each topic individually.

4.2.4 Calculating Self-Citation Rate Per Topic

After obtaining the self-citation counts for each author and their primary topic, we used SQL to calculate the self-citation rate per topic. The self-citation rate was determined by dividing the total self-citations for each topic by the number of distinct authors working in that topic. This normalization helped provide an average self-citation rate for each academic topic, offering a useful metric for comparing self-citation behaviors across different fields of study.

4.2.5 Results

The result of this query was a Pandas DataFrame containing one row for every topic and the self-citation rate for that topic. Below are the first five rows of the resulting DataFrame:

	topic	self_citation_rate
0	2	0.466766
1	5	0.411464
2	8	0.408777
3	11	0.397692
4	13	0.373600

Figure 5: Results of Use-Case Query 2

The data revealed how often authors cite their own work within specific academic communities and highlighted any trends in self-referential citation practices, and found that topic 2 has the highest self-citation rate. By analyzing this, we can gain insights into how tightly knit a research area might be, where authors are more likely to build on their own previous work within a particular topic.

4.3 Most Frequent Topic Within Each Community

We used Latent Dirichlet Allocation (LDA) to determine a topic that each document most likely belongs to within our dataset based on paper abstracts. We fit the LDA model using 15 topics, then obtained the topic distribution for each document. After assigning the most probable topic for each document, we saved the new data frame appended with a column of topic assignments. We also tried this process on the paper titles, but the results of the former demonstrated more fidelity to the set topics.

4.3.1 Motivation

The goal of this analysis was to compare the clusters generated by one of Neo4j's community detection algorithm, Louvain, with the topic categories identified using Latent Dirichlet Allocation (LDA). Specifically, we wanted to assess how well the communities detected by Louvain corresponded to thematic groupings in the text of the papers. By comparing the most frequent topics within these communities, we can understand whether papers with similar citation patterns also tend to discuss similar topics.

4.3.2 Creating Communities Using Louvain

We used Neo4j's Louvain community detection algorithm to identify communities of papers based on their citation relationships. The Louvain method generates communities by first assigning each paper to its own community. It then iteratively merges communities in a way that maximizes the density of connections within each group while minimizing connections between groups. This process helps uncover clusters of papers that are closely related through citation patterns, which may indicate similar research interests or shared areas of focus.

4.3.3 Identifying the Most Frequent Topic per Community

Once the communities were identified, we exported the results into SQL for further analysis. We focused on the top 20 largest communities based on the number of papers in each community created by Louvain. For each community, we calculated the most frequent topic by counting the number of occurrences of each topic within that group. We used the `RANK()` window function in SQL to identify the topic that appeared most often within each community.

4.3.4 Results

The result of this query was a Pandas DataFrame containing one row for every community identified by Louvain and the most frequent topic for that community. Below are the first seven rows of the resulting DataFrame:

	communityId	top_topic
0	27353	7
1	27707	10
2	38226	7
3	43552	10
4	46081	10
5	48783	3
6	49004	2

Figure 6: Results of Use-Case Query 3

The results of this query provide insight into how well the community structure generated by Louvain corresponds to the most common research topics within each community. By comparing the communities detected algorithmically with the most frequent topics, we can gain a better understanding of the relationships between citation networks and research themes. The top 3 communities identified by Louvain align closely with the top 3 most populated topics, suggesting that the community detection algorithm is effectively grouping papers based on shared research interests.

4.4 Paper Recommender System

4.4.1 Motivation

The goal of this use case was to develop a simple yet effective content-aware recommendation system for academic papers, leveraging both citation network structure and paper metadata. By using community detection results from the Louvain algorithm, we aimed to identify thematically related papers and recommend the most relevant ones to a given input paper. This hybrid approach allows us to generate recommendations that reflect both structural proximity in the citation graph and topical similarity within communities.

4.4.2 Building Community Nodes for Recommendation

To support recommendation functionality, we first used the output of Neo4j's Louvain community detection algorithm (as described in 4.3.2) to create a new set of community nodes in the graph. Each paper node was then linked to its corresponding community using a `[:IN_COMMUNITY]` relationship. This structure enables efficient traversal from a given paper to other papers within the same community, which we incorporate into our recommendation pipeline.

4.4.3 Calculating Similarity Scores

In Neo4j, given an input paper x, we compute a relevance score for a paper p_i within the database using the formula:

$$\text{relevance}(p_i) =$$

$$\mathbf{2}[\text{same community}] + \#\text{ shared authors} + \mathbf{1}[x \text{ cites } p_i] + \#\text{ papers cited by both } p_i \text{ and } x$$

The five most relevant papers are selected based on the highest relevance scores.

4.4.4 Display Recommended Papers

The selected paper IDs are passed to SQL, where we retrieve the corresponding titles, publication years, authors, and abstracts to present the results in a user-friendly format. The recommended papers are ordered descending by their relevance score, then ascending by the absolute difference between the publication year of the paper and the input paper.

4.4.5 Results

The final output is a ranked list of five recommended papers that are most relevant to the input paper, along with their key metadata. Below is an example of the result from the recommendation system for a given input paper. Note that some of the recommended papers may look the same. This is likely

due to errors in the original data (assigning different paper id's for the same paper), or the same paper being published in different venues etc. We decided to ignore these cases for the purpose of our analysis

input paper:

```
Auto-Context and Its Application to High-Level Vision Tasks and 3D
Brain Image Segmentation (2010)

authors:  Zhuowen Tu, Xiang Bai

The notion of using context information for solving high-level vision
and medical image segmentation problems has been increasingly realized
in the field. However, ...
```

recommended papers:

```
Hyperspectral Unmixing with Robust Collaborative Sparse Regression
(2016)

authors:  Chang Li, Xiaoguang Mei, Yong Ma

Recently, sparse unmixing (SU) of hyperspectral data has received
particular attention for analyzing remote sensing images...



Robust Sparse Hyperspectral Unmixing With $ell_{2,1}$ Norm (2017)

authors:  Chang Li, Xiaoguang Mei, Yong Ma

Sparse unmixing (SU) of hyperspectral data have recently received
particular attention for analyzing remote sensing images, which aims at
finding the optimal subset of signatures to best model the mixed pixel
in the scene...



Alzheimer's disease diagnostics by adaptation of 3D convolutional
network (2016)

authors:  Ayman El-Baz, Ehsan Hosseini-Asl, Robert S. Keynton

Early diagnosis, playing an important role in preventing progress and
treating the Alzheimer's disease (AD), is based on classification of
features extracted from brain images...



Alzheimer's Disease Diagnostics by a Deeply Supervised Adaptable 3D
Convolutional Network (2016)

authors:  Ayman El-Baz, Ehsan Hosseini-Asl, Georgy L. Gimelfarb

Early diagnosis, playing an important role in preventing progress and
treating the Alzheimer's disease (AD), is based on classification of
features extracted from brain images...
```

```
Exploiting Domain Knowledge in Aspect Extraction (2013)

authors:  Arjun Mukherjee, Bing Liu, Zhiyuan Chen

Aspect extraction is one of the key tasks in sentiment analysis. In
recent years, statistical models have been used for the task...
```

This approach takes advantage of both graph-based community structure and textual features derived from abstracts, resulting in meaningful recommendations that reflect shared research interests. Since our dataset includes papers published in 2010 or later, the recommender is effective for papers published from 2011 onward.

5 Conclusion

5.1 Results

Our final product, a recommendation system which accepts a single paper as input and outputs five recommended papers that are similar to the input, achieves our goal of extracting documents that are similar to an input document.

The queries we ran prior to our recommendation system offer more insight on the network structure and node attributes such as importance of papers, authors that overly-cite their own work, and community detection. These queries are vital to understanding the underlying composition of the graph and questioning the influence of a node (paper) on other nodes. Graphical representation and graph-based techniques such as PageRank are so vital to understanding the network because it requires more than just unweighted edges to determine the influence of a paper. Because of our use of PageRank and Louvain algorithms, we can understand why the graph determines some nodes as more important than others.

The graphical representation of this network enables us to answer our original questions about the network and be able to offer clear explanations for the results of our queries.

5.2 Limitations

Ideally, we would be able to use data from the full citation network rather than our limited dataset. However, because of the limit on number of observations in Neo4j, we had to cut down on the number of observations.

Additionally, the number of topics in LDA is determined by each user, so had we chosen a different number of topics, our results may have appeared differently. The number of topics was up to our discretion and although we chose what seemed most logical, there is no way of determining if it was the optimal number of topics.

5.3 Future Improvements

Having access to more information about the papers, for example what fields they were published in, may help improve this project. In addition, extracting the underlying social relationships between authors may contribute to our understanding of the network. This might take more data, such as social network communication or authors' previous education, but would help us understand why communities are formed within the network.

5.4 Lessons Learned

Through our analysis utilizing both PostgreSQL and Neo4j, we gained valuable insights into the advantages and challenges of integrating relational and graph databases for academic paper network analysis. Our key takeaways are as follows:

5.4.1 Strengths and Trade-offs of Database Models

One of the most significant lessons learned was recognizing the complementary strengths of relational and graph databases. PostgreSQL excelled in handling structured queries, aggregations, and filtering large datasets efficiently, while Neo4j proved invaluable for exploring citation networks and computing graph-based metrics such as PageRank and community detection. However, transferring data between the two databases introduced overhead, requiring careful optimization to ensure efficiency.

5.4.2 Workflow Efficiency and Query Optimization

Using Jupyter Notebook as our primary analysis environment streamlined our workflow by centralizing data queries, transformations, and visualizations. However, we observed that query execution time varied significantly based on query structure and database indexing. Learning how to optimize SQL and Cypher queries was crucial for improving performance, especially when working with large datasets.

Overall, this project demonstrated the power of integrating relational and graph databases for academic network analysis, while also underscoring the importance of careful data management, query optimization, and computational efficiency.