Semantic Data Management

Lab 1: Property Graph

Oluwanifemi Favour Olajuyigbe
Stephanie Silva Vieira Gomes

Prof. Anna Queralt

April 2025

A.1 - Modeling

1.0.1 Graph Design

Figure 3.1 shows the design chosen to model all the requirements of the research publications graph.

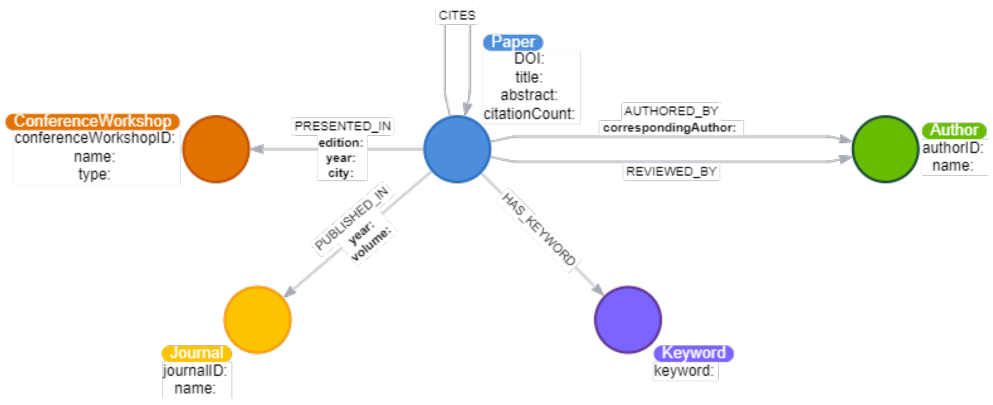


Figure 1.1: Conceptual Graph Design

Nodes

- Paper: represents research papers published in journals, conferences, or workshops. Each paper includes DOI, title, abstract, and CitationCount properties.
- Author: represents authors who have written or reviewed research papers. Each author has properties, authorID, and name.
- Journal: represents academic journals where research papers are published, with properties including journalID and name.
- ConferenceWorkshop: represents conferences or workshops where research papers are presented. Includes properties such as conferenceWorshopID, name, and type.
- Keyword: represents terms that specify topics for a research paper. It is associated with the keyword property.

Relationships

- CITES: links a paper to another paper. It represents the citation relationships between two paper nodes, where one paper node cites the other node.
- AUTHORED_BY: links a paper node to an author node to indicate the authors of a particular paper with a correspondingAuthor property that identifies which of the authors is the corresponding author for the paper.
- REVIEWED_BY: links a paper node to an author node to indicate the authors that reviewed a paper.
- HAS_KEYWORD: connects a paper node to a keyword node to indicate the keywords associated with the research paper.
- PRESENTED_IN: Links a paper node to a ConferenceWorkshop node to indicate the conference or workshop where it was presented. It has the properties edition, year, and city.
- PUBLISHED_IN: links a paper node to a Journal node to indicate the journal it was published in. It has the properties year and volume.

1.0.2 Assumptions

- For the corresponding author, since it has the same properties as the other authors, we chose to distinguish them through the property 'correspondingAuthor' of the relationship 'AUTHORED_BY'. This eliminates redundancy for the 'Author' label. This also applies to the Reviewers; once they are also authors, we chose to distinguish them by the relationship instead of adding a node.
- Only the most used properties for each label and relationship were kept. This is based on the queries in Section B and the understanding of the domain. For example, in the label 'Author' we could have the e-mail property, but since it is not used in any queries, we chose to maintain a simple design.
- The label 'Paper' has the property 'citationCount' because, based on the queries in Section B, and the nature of the domain, this property is frequently needed. For not computing it every time it is required, we stored it as a property and, for each new citation, we also update the property. We also assumed that citations are going to be created more sparsely than the demand for the 'citationCount' property.
- Conferences and workshops are represented in one single label, as they share similar properties. The 'type' property is used to distinguish between the two options.
- Proceedings are considered to be equivalent to edition, therefore, only one of them is represented in the graph.
- To optimize the data model and improve query performance, we decided to eliminate redundancy in the labels for 'ConferenceWorkshop' and 'Journal' by modeling them as unique instances. Recognizing that most queries will require information from the 'Paper-ConferenceWorkshop' or 'Paper-Journal' relationships, we prioritized minimizing the number of hops between these entities. This approach aims to enhance query efficiency without compromising overall performance.

A.2 - Instantiating/Loading

A set of transformations was applied to standardize and ensure that all fields were complete before integrating the data into the Neo4j graph database. The Semantic Scholar API was used to retrieve information about publications and all details related to them.

First, five relevant fields related to technology were elected, and, using the API, the first 50 publications from each field were used as a base. After locating the relevant papers, detailed metadata is retrieved, including bibliographic information, citation data, and other attributes.

Secondly, for each retrieved paper, up to 10 referenced papers are extracted to enrich the citation networks. These referenced papers undergo the same retrieval process in batches of 100 paper IDs at a time to optimize API requests.

2.0.1 Data Preprocessing

- **Handling Missing Values and Standardization**: for missing values, random information is generated according to the attribute format. For example, if the author's name is missing, a random synthetic name is created. Another example is that if a publication date is missing, a random year between 2000 and 2025 is generated.
- **Publication Classification**: The papers are categorized into "Conference", "Workshop", or "Journal" based on the type and name of the venue in which it was published. If the attribute is not clear, the default classification is "Journal". Fields such as city, edition, and volume are also extracted according to publication type.
- **Keyword Extraction**: The base keywords for each paper are extracted from the 'fieldsOfStudy' attribute. Additional keywords are inferred from the paper title using a list of predefined related terms.

- **Reference Filtering**: It was opted to retain only the existing references in the papers collection. If a paper references another publication that it was not retrieved, this reference is discarded. This is to ensure that for all references it is possible to search for all attributes of a paper.

2.0.2 Synthetic Data

- **Corresponding Author**: The first author of each paper was marked as 'True' to indicate their role as the corresponding author, while all other authors were marked as 'False'.
- **Reviewers**: For each paper, three authors were randomly selected as reviewers. In selecting the authors, it was ensured that an author could not be selected as a reviewer for their own paper.
- **References and Reference Count**: Synthetic data was generated for papers without references by assigning a random citation count between 3 and 10 and selecting a corresponding number of papers as references. For instance, if a paper was assigned a citation count of 4, four papers would be randomly chosen from a filtered list of papers that shared at least one keyword with it and were not the same paper, thus avoiding self-citations.

2.0.3 Importing Data into Neo4j

For the import process, the data was split into separate files—one for each of the five node types and one for each of the six relationship types. This approach allowed each node type to be loaded independently, ensuring that their specific attributes were correctly assigned.
For loading the data into Neo4j, the command 'LOAD CSV' was used. After establishing a connection to Neo4j, it is ensured that any existing data is cleared and then, nodes are loaded with properties. Lastly, relationships are established between the nodes and also loaded into the database.

A.3 - Evolving the Graph

3.0.1 Graph Re-Design

Figure 3.1 shows the design chosen to model all the new requirements for evolving the graph information with reviews and organization data.

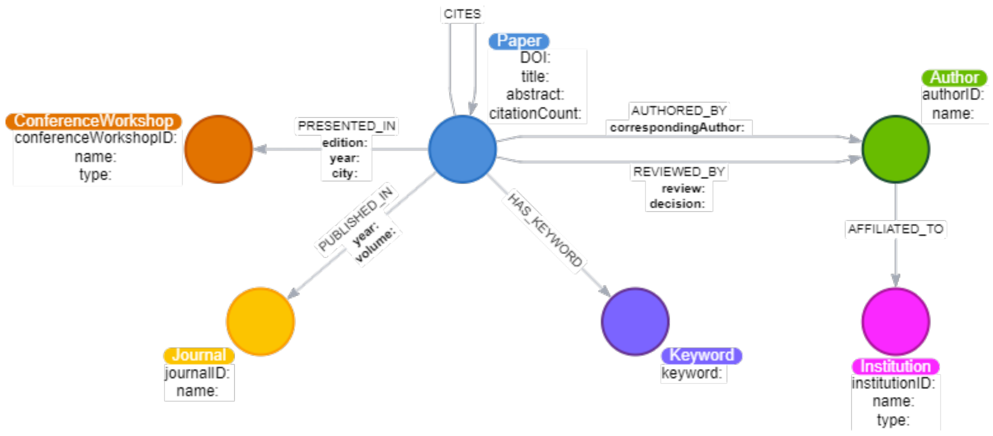


Figure 3.1: Conceptual Graph Design - Evolution of Information

Reviews

To add each reviewer's review content and their suggested decision, it was decided that the information be stored as properties of the existing REVIEWED_BY edge. To achieve this, we updated the REVIEWED_BY edge by adding two properties: 'review' to store the content of the review, and 'decision' to store the reviewer's suggested decision.

Author’s Affiliation

To create a relationship between the author and the organization they are affiliated with, it was decided that each organization should be a node, ’Institution’ since multiple authors can be associated with one institution. Furthermore, to model the type (university or company), the new node stores the needed properties to map each institution completely.

3.0.2 Data Generation and Insertion

- Each review content and decision in the ’REVIEWED_BY’ edge was populated with synthetic data. In generating the data for the decisions, we assumed that all papers were accepted for publication since they are in the Semantic Scholar database. Based on this assumption, we created a function ensuring that for each paper, the majority (more than half) of the reviewers’ decisions were "accept," while the remaining reviewers’ decisions were randomly assigned "accept," "revise," or "reject."
  Each review’s content was based on the reviewer’s decision. We created separate lists of review templates for "accept," "revise," and "reject" decisions. Then a function randomly selects an appropriate review content from the corresponding list based on the decision assigned to the reviewer.
- The review properties were added to the relationship ’REVIEWED_BY’ using Cypher commands.
- Each ’Institution’ node was populated with synthetic data. A function selects a name and a type randomly for each institution. After creating the institutions, they are randomly chosen to be associated with each author. An author can be associated with only one organization.
- Each ’Institution’ node and the relationship ’AFFILIATED_TO’ were created utilizing Cypher commands.

B - Quering the Graph

The property ’citationCount’ was used to design the queries. This allows for more efficient queries as there is no need to calculate this number for each paper. An alternative version where the property is calculated is also provided in the code. The four queries are presented as follows.

4.0.1 Query 1

```
MATCH (p:Paper)-[:PRESENTED_IN]->(cw:ConferenceWorkshop)
WITH p,cw
ORDER BY p.citationCount DESC
RETURN cw.name AS conferenceWorkshopName, collect(p.title + ' (' + p.citationCount + ')') [0..3] AS top3CitedPaper
```

Listing 4.1: Top 3 most cited papers of each conference/workshop

4.0.2 Query 2

Considering authors that have published papers on that conference/workshop in, at least, 4 different editions

```
MATCH (cw:ConferenceWorkshop)<-[presented:PRESENTED_IN]-(paper:Paper)-[:AUTHORED_BY]->(author:Author)
WITH cw.name AS conferenceWorkshopName, author, COUNT(DISTINCT presented.edition) as distinct_editions
WHERE distinct_editions > 3
RETURN conferenceWorkshopName, collect(author.name) as community
```

Listing 4.2: Community of each conference/workshop

4.0.3 Query 3

Impact factor (year1) = Citations(year1) / Publications(year1-1) + Publications(year1-2)

```
MATCH (p:Paper)-[pi:PUBLISHED_IN]->(j:Journal)
WHERE pi.year IN [2020, 2019]
WITH j, collect(p.paperDOI) as papersIDs
WITH j, papersIDs, size(papersIDs) as journalPublications

UNWIND papersIDs as papersID
MATCH (citingPaper:Paper)-[:CITES]->(citedPaper:Paper {paperDOI:papersID})
MATCH (citingPaper)-[citingPub:PUBLISHED_IN|PRESENTED_IN]->()
WHERE citingPub.year IN [2021]
WITH j, journalPublications,COUNT(citingPaper.paperDOI) AS journalCitations

RETURN j.journalID as journalID,j.name as journalName, 2021 as impactYear,
journalCitations, journalPublications, (toFloat(journalCitations)/
journalPublications) AS impactFactor
ORDER BY impactFactor DESC, journalName ASC
```

Listing 4.3: Impact factor of each journal

4.0.4 Query 4

The h-index is calculated by finding the maximum position where the number of papers with citations is greater than or equal to that position, to identify the point at which an author has h papers with at least h citations each. The h-index is calculated by finding the maximum position where the number of papers with citations is greater than or equal to that position, to identify the point at which an author has h papers with at least h citations each.

```
MATCH (paper:Paper)-[:AUTHORED_BY]->(author:Author)
WITH author, paper.citationCount as citation
ORDER BY author.authorID, citation DESC
WITH author, collect(citation) as citations
WITH author, citations, range(0, size(citations)-1) as indices
WITH author, max([i in indices WHERE i < size(citations) AND citations[i] >=
 i+1 | i+1]) as hindex
RETURN author.authorID as authorID, author.name as authorName, size(hindex)
as hindex
ORDER BY hindex DESC
```

Listing 4.4: H-index of each author

C - Recommender

For this section, it was defined that each community would be represented by a new node 'Community '. The other needed elements were designed as relationships from the existing nodes to the newly created node. Each relationship aims to represent the inferred knowledge from the queries and store it in the graph.

5.0.1 Query 1

```
CREATE (c:Community {name: 'Technology'});

MATCH (k:Keyword)
WHERE k.keyword IN ['Machine Learning', 'Big Data', 'Natural Language
Processing', \
'Artificial Intelligence','Graph', 'Computer Science', 'Cloud', 'Data Mining
', 'Computer Vision', 'Cybersecurity']
MATCH (c:Community)
MERGE (k)-[:DEFINES]->(c)
```

Listing 5.1: Define 'Technology' Community and its Keywords

5.0.2 Query 2

```
MATCH (p:Paper)-[:HAS_KEYWORD]->(k:Keyword)-[:DEFINES]->(c:Community {name:'
Technology'})
WITH c, collect(p.paperDOI) AS papersID
MATCH (j:Journal)<-[:PUBLISHED_IN]-(p2:Paper)
WITH j, c, CASE WHEN p2.paperDOI IN papersID THEN 1 ELSE 0 END AS
journalPaperCommunity
WITH j, c, (toFloat(sum(journalPaperCommunity)) / count(*)) AS
percentangePapersCommunityJournal
WHERE percentangePapersCommunityJournal >= 0.90
MERGE (j)-[:RELATED_TO]->(c);

MATCH (p:Paper)-[:HAS_KEYWORD]->(k:Keyword)-[:DEFINES]->(c:Community {name:'
Technology'})
WITH c, collect(p.paperDOI) AS papersID
MATCH (cw:ConferenceWorkshop)<-[PRESENTED_IN]-(p3:Paper)
WITH cw, c, CASE WHEN p3.paperDOI in papersID THEN 1 ELSE 0 END as
cwPaperCommunity
WITH cw, c, (toFloat(sum(cwPaperCommunity)) / count(*)) as
percentangePapersCommunityConferenceWorkshop
WHERE percentangePapersCommunityConferenceWorkshop >= 0.90
MERGE (cw)-[:RELATED_TO]->(c)
```

Listing 5.2: Define Journals/Conferences/Workshops related to the Technology Community

5.0.3 Query 3

```
MATCH (citingPaper:Paper)-[]->(:Journal|ConferenceWorkshop)-[r:RELATED_TO
]->(c:Community {name: 'Technology'})
MATCH (citedPaper:Paper)-[]->(:Journal|ConferenceWorkshop)-[r:RELATED_TO]->(
c:Community {name: 'Technology'})
MATCH (citingPaper)-[:CITES]->(citedPaper)
WITH c, citedPaper, count(DISTINCT citingPaper) as citations
ORDER BY citations DESC
LIMIT 100
MERGE (citedPaper)-[:TOP_100]->(c)
```

Listing 5.3: Define top 100 papers from the Technology Community

5.0.4 Query 4

```
MATCH (a:Author)<-[:AUTHORED_BY]-(p:Paper)-[:TOP_100]->(c:Community {name: '
Technology'})
WITH a, c, count(DISTINCT p) as quantityTop100Papers
MERGE (a)-[:POSSIBLE_REVIEWER]->(c)
WITH a, c, quantityTop100Papers
WHERE quantityTop100Papers > 1
MERGE (a)-[:GURU]->(c)
```

Listing 5.4: Define Gurus and Possible Reviewers of Technology Community

D - Graph Algorithms

6.0.1 Page Rank Algorithm – Identifying Influential Research Papers

The Page Rank Algorithm is a centrality algorithm used to estimate the importance of a node by evaluating the number of incoming connections from neighboring nodes, while also considering both the influence of those neighbors and their out-degree centrality.

In the context of research publications, the page rank algorithm is used to identify the most influential papers based on how frequently other papers have cited them. The assumption is based on the fact that the more a paper is cited, the greater its impact on the research community. The algorithm also ensures that a node is not only ranked based on direct citations but also on the influence of the citing nodes. Hence, it will be very useful in finding groundbreaking research papers in a research domain, recommending highly cited papers for researchers to explore, and ranking publications based on their research impact.

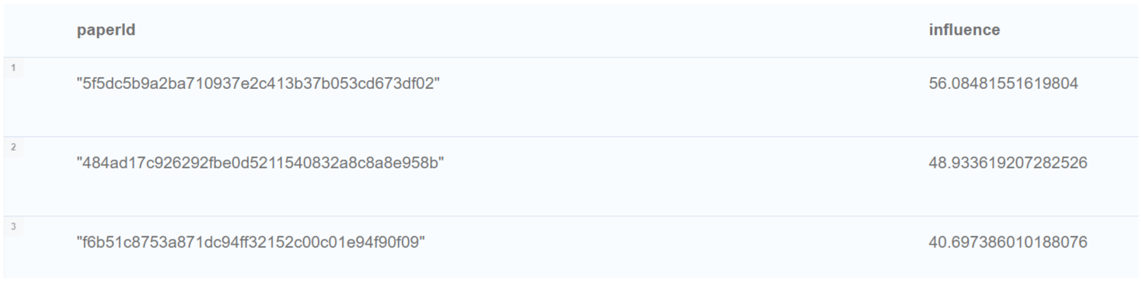
Identify Most Influential Papers

```
CALL gds.graph.project(
    'paperCites',
    'Paper',
    'CITES'
);

CALL gds.pageRank.write.estimate('paperCites',
    {writeProperty: 'pageRank'}
);

CALL gds.pageRank.stream('paperCites')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).paperDOI AS paperId, score as influence
ORDER BY influence DESC
LIMIT 10;
```

Listing 6.1: Using the Page Rank Algorithm to Identify most influential papers

The gds.graph.project call creates an in-memory graph projection named 'paperCites' with Paper nodes and CITES relationships, where a (:Paper)-[:CITES]->(:Paper) relationship represents one paper citing another. The gds.pageRank.write.estimate call then estimates the required memory needed for the operation. Finally, we run the PageRank algorithm to compute the PageRank score for each paper and then sort the results in descending order to return the top influential papers.

The query execution returns the top ten most influential papers, displaying their PaperId and PageRank influence scores. Figure 6.1 shows a sample of the data obtained.



	paperId	influence
1	"5f5dc5b9a2ba710937e2c413b37b053cd673df02"	56.08481551619804
2	"484ad17c926292fbe0d5211540832a8c8a8e958b"	48.933619207282526
3	"f6b51c8753a871dc94ff32152c00c01e94f90f09"	40.697386010188076

Figure 6.1: Papers with their pagerank scores

6.0.2 Node Similarity Algorithm - Identifying Similar Authors

The Node Similarity Algorithm is an algorithm used to find similar nodes based on common relationships.

In the context of research publications, the Node Similarity algorithm finds authors with similar research contributions based on shared papers. This is particularly useful in academic networks for identifying potential research collaborators within a domain, recommending author networks for educational partnerships, and understanding researchers who contribute to similar areas of study.

Identify Similar Authors based on shared papers

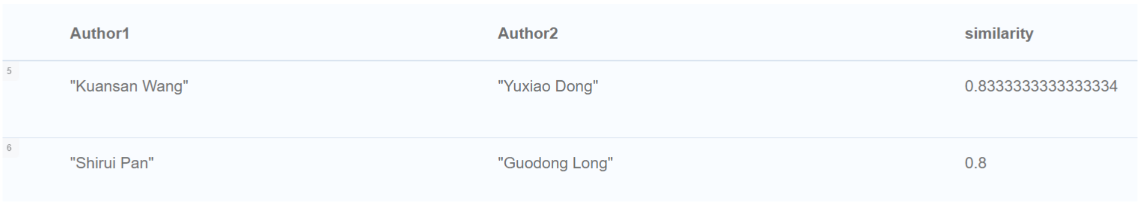
```
CALL gds.graph.project('similarAuthors',
    ['Author', 'Paper'],
    {AUTHORED_BY:
        {orientation:'REVERSE'}
    });

CALL gds.nodeSimilarity.write.estimate('similarAuthors',
    {
        writeRelationshipType: 'SIMILAR',
        writeProperty: 'score'
    });

CALL gds.nodeSimilarity.stream('similarAuthors',
    {
        topK:1,
        degreeCutoff:2
    })
YIELD node1, node2, similarity
WHERE similarity < 1
RETURN gds.util.asNode(node1).name AS Author1,
  gds.util.asNode(node2).name AS Author2, similarity
ORDER BY similarity DESC;
```

Listing 6.2: Using the Node Similarity Algorithm to Identify similar Authors

The gds.graph.project call creates an in-memory graph projection named 'similarAuthors'. The AUTHORED_BY relationship is reversed to analyze which authors have worked on which papers. The gds.nodeSimilarity.write.estimate call then estimates the required memory needed for the operation. Finally, we run the Node Similarity algorithm to compute similar author pairs, using the following parameters:

- topK is set to 1 to limit the output to the single most similar author pair for each node, reducing redundancy.
- degreeCutoff is set to 2 to only consider authors who have authored at least two papers, ignoring those with only one paper.
- The results are filtered to exclude trivial cases where similarity equals 1, ensuring the algorithm highlights meaningful author relationships.

The query execution returns pairs of similar authors along with their similarity scores, ranking them according to their degree of collaboration. Figure 6.2 shows a sample of the data obtained.



	Author1	Author2	similarity
5	"Kuansan Wang"	"Yuxiao Dong"	0.8333333333333334
6	"Shirui Pan"	"Guodong Long"	0.8

Figure 6.2: Similar authors pairs with similarity scores