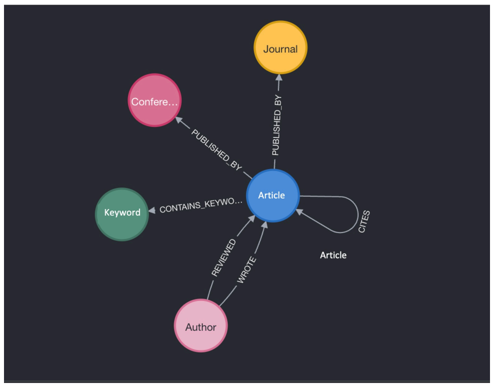
Big Data management for Data Science

Angelo Di Gianvito

A.1 Modelling

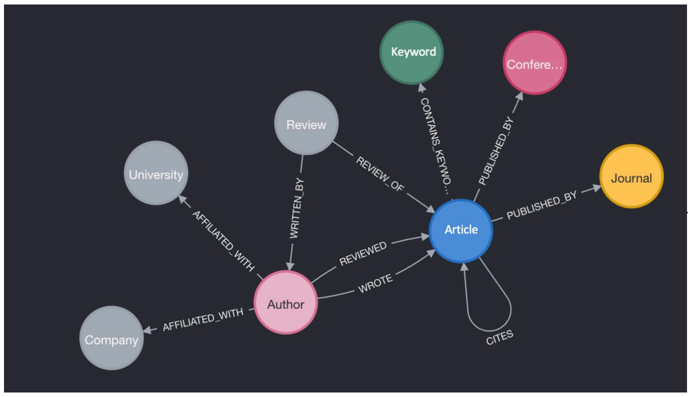


We have designed a scheme for modeling the relationship between research articles using graph data. Below is the justification of the relationship between the nodes.
author - wrote - article: This relationship connects authors to their works, helping with the computation of metrics like the i10-index.
article - published_by - journal/conference: This relationship enables the easy retrieval of all articles published in a specific journal or conference, which is essential for queries such as finding the top-cited papers within a particular conference. This will be important for finding influential papers within a conference, and can be used to locate important articles.
article - contains_keyword - keyword: Keywords are vital for tagging articles with relevant topics, aiding in the efficient search and categorization of papers based on their content. This is an important part of the schema and also helps to relate articles which have similar keywords aswell
author - reviewed - article: This relationship helps us understand the reviewers for each article. This is necessary for us to understand how peer reviews have worked, and track the peer review relationships between academics.
article - cites - article: This self-referencing relationship is crucial for calculating impact factors and understanding the citation network of papers. It's also useful to understand when we would want to account for self-referencing, to ensure we don't have any issues when querying the data.

A.2 Instantiation/Loading

We decided to use the real BDLP (computer science bibliography) data but synthetically generated the citations as the format in the real data differed from our desired format. Note that we only used a sample of 10,000 articles due to computational constraints.

A.3 Evolving the Graph



To adjust the graph to match the new requirements we will add some news nodes; the university, review and company.

Implementing review nodes along with their associated relationships allows for a more detailed and organized way of handling reviews. By creating separate review nodes, each review can be uniquely identified and handled separately, rather than having reviews considered in the article or author nodes. This new approach prevents us from adding extra attributes to the author node and so is a cleaner approach overall.

Introducing company and university nodes and affiliated_with relationships helps us to track authors' affiliations, which is useful for understanding cross institutional relationships in the network. By explicitly modeling affiliations, we avoid ambiguity in author data and can be confident about the links between institutions and the authors of the papers.

B Querying

```
// QUERY 1.
MATCH
(citedArticle:Article)<-[:HAS_CITATION]-(article:Article)-[:PUBLISHED_BY]->(conference:
Conference)
WITH conference, article, COUNT(citedArticle) AS citationCount
ORDER BY conference.id, citationCount DESC
WITH conference, COLLECT([article, citationCount]) AS topPapers
RETURN conference.name AS conference_name,
    topPapers[0][0].title AS paper1, topPapers[0][1] AS num_cites1,
    topPapers[1][0].title AS paper2, topPapers[1][1] AS num_cites2,
    topPapers[2][0].title AS paper3, topPapers[2][1] AS num_cites3;

// QUERY 2.
// Find authors who have published papers in at least 4 different editions of the same conference
MATCH (conf:Conference)
WITH conf.name AS conferenceName, collect(conf.edition) AS editions
WHERE size(editions) >= 4

// Match authors who have published papers in the specified conference editions
```

```
MATCH (aut:Author)-[:WROTE]->(art:Article)-[:PUBLISHED_BY]->(conf:Conference)
WHERE conf.name = conferenceName AND conf.edition IN editions

// Group authors by conference
WITH conferenceName, aut, count(distinct conf) AS editionCount
WHERE editionCount >= 4

// Return the conference name and the community of authors
RETURN conferenceName AS Conference, COLLECT(aut.name) AS Community;

// QUERY 3:
MATCH (j:Journal)<-[:PUBLISHED_BY]-(art:Article)
OPTIONAL MATCH (citingArt:Article)-[:HAS_CITATION]->(art)
WITH j, COUNT(art) AS totalPapers, COUNT(citingArt) AS totalCitations
WHERE totalPapers > 0
RETURN j.name AS Journal, (toFloat(totalCitations) / totalPapers) AS ImpactFactor
ORDER BY ImpactFactor DESC;

// QUERY 4.
MATCH (author:Author)-[:WROTE]->(article:Article)
OPTIONAL MATCH (article)<-[:HAS_CITATION]-(citingArticle:Article)
WITH author, article, COUNT(citingArticle) AS numCitations
ORDER BY author.name, numCitations DESC
WITH author, COLLECT(numCitations) AS allCitations
WITH author, SIZE([index IN RANGE(0, SIZE(allCitations) - 1) WHERE allCitations[index]
>= index + 1]) AS h_index
RETURN author.name AS Author, h_index AS HIndex
ORDER BY HIndex DESC;
```

C Graph Algorithms

We implemented both the Betweeness and PageRank algorithms in Neo4j, below we describe them both and the results from our database. The Betweeness algorithm is used to identify nodes in a graph that act as intermediaries or bridges between other nodes. This can act as an indication of the importance or significance of an article in a community of articles. It may identify articles that bridge the knowledge gap between two related areas in academia. Generally, it can be used to find articles of significance. Running the algorithm on Neo4j the output returns the 20 most influential articles in the graph based on their Betweeness ranking, helping to determine what the most important and influential papers in this academic field are. Below you can see the 20 most influential articles from our data.