# A Modeling, Loading, Evolving

## A.1 Modeling

1. The metadata/schema visual representation of the graph is shown below.

A screenshot of a computer

Description automatically generated with medium confidence

2. The design decisions made are explained below.

* We decided to represent the corresponding author as an edge because such person doesn’t have any differences in properties between them and other types of authors.
* Because there is a query that distinguishes conference editions, we decided to represent it as a node, while the volume of a journal is being represented in a simpler way as a property of the edge published\_in, because we don’t have queries specifying different volumes.
* Since we will eventually query a community of keywords we decided it’s best to represent them as nodes.

We have made the following assumption.

* Since we are modeling a graph for publications, we assume that rejected articles (and therefore not published) won't be represented in this database.

## A.2 Instantiating/Loading

1. The Cypher expressions to create and instantiate the solution are in a file named load.cypher. It is being executed by a Python application attached to the project.

We obtained the base for our data from the BYU Engineering Publications in Scopus 2017-21 Kaggle dataset[[1]](#footnote-1) as a csv file, which presents data about publications and already provided real data regarding authors, publication title, publication year, volume, DOI, access link, author’s affiliations, keywords, document type (if article, conference paper or other), and publication stage. The Python script generate\_data.py processes this file with the following steps:

1. We only used articles and conference papers.
2. We discarded data that does not have one of the fields needed.
3. We removed the edition information such as year and edition number from the conference names
   1. For example,

“Proceedings - 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science, IRI 2020”

became

“Proceedings – IEEE International Conference on Information Reuse and Integration for Data Science, IRI”.

1. We parsed the author’s affiliations and generated a separate csv file with them.
2. Because we lacked sufficient papers in the same conference but in four different years (for one of the queries requested), we also synthetically created extra papers for the previous and/or next years.
3. To have conferences/journals belonging to the database community (for section D), we selected the 15 conferences/journals with more publications, and added one of the community keywords to each of them.
4. We synthetically created the citation links between articles and stored them in a separate csv file. For creating such citations, we made sure that the papers being cited were from a previous year and had at least one keyword in common with the paper citing them. A paper could have from 0 to 20 citations.
5. Since we didn’t have reviewer data, we chose up to 3 authors that have at least one keyword in common with the paper and assigned them as reviewers, saving this information in a separate csv file. We have also generated some text for the reviews themselves and stored them as a separate csv. Since we assumed all publications would be accepted in the end, there can be up to 1 reviewer rejecting each of them.

A.3 Evolving the graph

1. The modifications made on the graph model are shown below and highlighted in purple.

Timeline

Description automatically generated with medium confidence

We decided to represent the university and company that an author is affiliated to by only one node called Organization and having it differentiated between university and company by its “type” property.

We also decided to represent the review and suggested decision of a reviewer as properties of the relationship “reviews” since we thought it wouldn’t make sense to create a separate node just for a reviewer (since they are authors as well).

# B Querying

1. Find the top 3 most cited papers of each conference.

MATCH (:Publication)-[r:cites]->(cited\_paper:Publication)-[:published\_in]->(:ConferenceEdition)-[:belongs\_to]->(conf:Conference)  
WITH conf, cited\_paper, COUNT(r) AS num\_citations  
ORDER BY conf, num\_citations DESC  
WITH conf, collect(cited\_paper)[..3] AS top3\_most\_cited\_papers  
RETURN conf,  
 top3\_most\_cited\_papers[0] AS top1\_paper,  
 top3\_most\_cited\_papers[1] AS top2\_paper,  
 top3\_most\_cited\_papers[2] AS top3\_paper  
;

1. For each conference and its community: i.e., those authors that have published papers on that conference in, at least, 4 different editions.

MATCH (a:Author)-[:writes]->(p:Publication)-[:published\_in]->(ce:ConferenceEdition)-[:belongs\_to]->(c:Conference)  
WITH c, a, COUNT(DISTINCT ce) AS distinct\_editions  
WHERE distinct\_editions >= 4  
RETURN c, a  
;

1. Find the impact factors of the journals in your graph.

MATCH (journal:Journal)<-[:published\_in]-(p:Publication)<-[c:cites]-(:Publication)  
WITH journal, p.year AS year, count(c) AS citations  
MATCH (journal)<-[:published\_in]-(p:Publication)  
 WHERE p.year = year - 1 OR p.year = year - 2  
WITH journal, year, citations, count(p) AS publications  
 WHERE publications > 0  
RETURN journal, year, toFloat(citations) / publications AS ImpactFactor  
 ORDER BY ImpactFactor DESC

;

1. Find the h-indexes of the authors in your graph.

MATCH (author:Author)-[:writes]->(paper:Publication) <-[c:cites]-(:Publication)  
WITH author, paper, count(c) AS citations  
 ORDER BY author, citations DESC  
WITH author, collect(citations) AS paper\_citations  
RETURN author,  
 reduce(hindex = 0,  
 citations IN paper\_citations |  
 CASE WHEN citations > hindex THEN hindex + 1  
 ELSE hindex  
 END) AS `H-Index`

# C Graph Algorithms

We are going to apply the Page Rank algorithm in order to find the most relevant papers, and the Louvain algorithm to [INSERT REASON HERE]. Since for both we use the Publication nodes and the Cites edges, we use the same projection for both:

CALL gds.graph.create(  
 'publications',  
 'Publication',  
 'cites'  
)

### Page Rank (Centrality Algorithm)

We use the Page Rank algorithm to find the most relevant papers using the citation network. The score provided by the Page Rank algorithm is probably more relevant than the number of citations, since it takes into account the importance of the citing papers. For example, a paper that is cited 5 times by papers that nobody cites is probably less relevant than a paper that is cited by only 2 papers that a lot of papers cite.

CALL gds.pageRank.stream('publications')  
YIELD nodeId, score  
RETURN gds.util.asNode(nodeId).title AS title, score  
ORDER BY score DESC, title ASC

### Louvain (Community Detection)

CALL gds.louvain.stream('publications')  
YIELD nodeId, communityId, intermediateCommunityIds  
RETURN gds.util.asNode(nodeId).title AS title, communityId, intermediateCommunityIds  
ORDER BY title ASC

# D Recommender

1. <https://www.kaggle.com/dpixton/byu-engineering-publications-in-scopus-201721/version/1> [↑](#footnote-ref-1)