SDM LAB Document: Property Graphs

This project is organized within the specified GitHub repository [1], where all scripts pertinent to the sections outlined below are available. Additionally, the applications have been included in the provided ZIP file.

A Modeling, Loading, Evolving

A.1 Modeling

The visual representation of the graph we designed is depicted in the following figure.

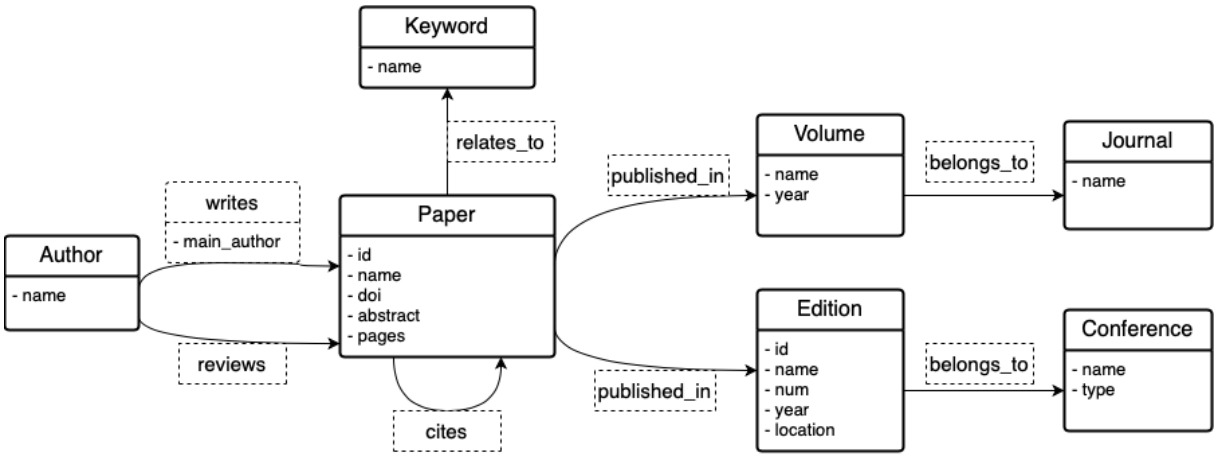

Figure 1: Designed graph

We decided to create a graph with 8 nodes. The central node is the Paper node, which contains 5 properties. We chose to exclude the year property from this node, as we already have this information on the Volume and Edition nodes, facilitating the maintenance of the graph.
The Paper node is connected by two edges to the Person node. One "writes" edge connects all the writers to their corresponding papers, and one "reviews" edge connects all the reviewers to their reviewed papers. The "writes" edge contains a boolean property called *main_author* to indicate the main author of each paper. This characteristic is dependent on the relationship, so it does not have relations with objects and does not require an additional edge.
We also have an edge "cites" that connects papers to each other. By aggregating citation counts for each paper within a conference, we can rank them based on their citation frequency. This ranking allows us to identify the top 3 most cited papers for each conference, facilitating the performance of the first query of Part B.

The Paper node is also related to the Keyword node, which is in turn related to the Topic node. Considering that a keyword can belong to multiple topics, we decided to make it a node to promote reusability. Thus, we consider that a paper has one or more keywords, and a keyword refers to one or more topics.
Ultimately, we decided to keep the Volume, Journal, Edition, and Conference nodes separate within our graph structure. While there are similarities between Volumes and Editions, as well as between Conferences and Journals, we initially explored merging them and using a *"type"* attribute to differentiate between them. However, we abandoned this idea in favor of maintaining separate paths. This decision was driven by the recognition that the attributes associated with each type can vary significantly. For example, conferences may have location information such as a city, whereas volumes do not necessarily possess this attribute. Although graph databases are schemaless and could technically allow for merging based on specific conditions, we concluded that maintaining distinct node types would optimize performance, particularly for queries focused on a specific type of entity.

A.2 Instantiating/Loading

Our data was obtained from the DBLP website [2]. The files were provided in XML format, necessitating conversion to CSV. This conversion was accomplished using a tool found in a specific GitHub repository [3]. The majority of the preprocessing was conducted in R, details of which can be found in the *markdownsmd.Rmd* script. In addition to R, a portion of the work was executed using Python, specifically in the *get_keywords.ipynb* notebook, which is elaborated upon in section C.

We encountered instances where some information was missing. Consequently, we had to either derive it from another variable (e.g., the location, edition number...) or create it (e.g., paper abstracts, review content...). Additionally, the dataset lacked citation information; therefore, we assigned between one to five citations to each paper from papers of the same community, ensuring that a paper could not cite itself. The assignment of reviewers was similarly randomized, with each paper being assigned three authors as reviewers, excluding the paper's own authors.

Compiling the datasets was the most time-consuming phase of the project. For a detailed breakdown, refer to the *markdownsmd.Rmd* script. Essentially, we created a separate dataset for each Node and Edge before importing them into Neo4j.

A.3 Evolving the graph

The application for running the queries below is stored in *PartA.3._ChimenoFortó.py*. Figure 2 shows the visual representation of the modified graph. The changes are highlighted in yellow.

Modification 1: Affiliation Node

We faced a choice regarding how to represent affiliations within our graph. One option was to directly attribute affiliation to each individual author node. Alternatively, we could introduce a separate 'Affiliation' node that connects to each 'Author' affiliated with it. Considering our goal of accommodating multiple authors affiliated with a single organization, we opted for the latter approach. Implementing the first option would have resulted in redundant data and compromised maintainability.

Modification 2: Reviews Edge

We deliberated on the possibility of transforming the 'Review' relationship from an edge to a node. However, we concluded that the only distinguishing feature would be the addition of two new properties indicating the content and the decision. Given that reviews are static and unique once made, there is no requirement to anticipate updates. Consequently, we defined the decision as a boolean property denoted by 'approves'.

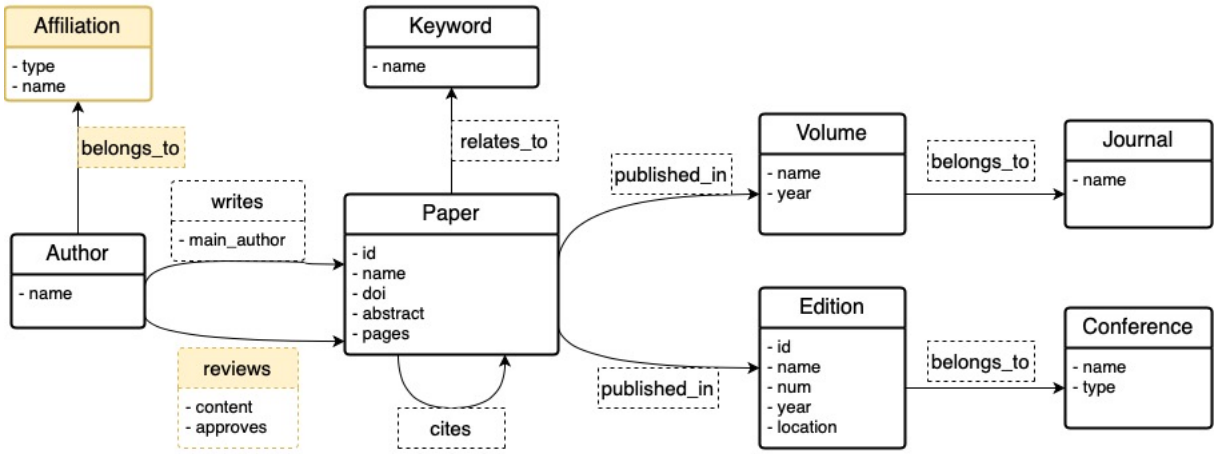

Figure 2: Modified Graph

B Querying

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

```
MATCH (c:Conference)--()--(p2:Paper)<--(p:Paper)
WITH c, p2, count(p) AS numCitations
ORDER BY numCitations DESC
WITH c.name AS Conference, COLLECT({Paper:p2.name, NumOfCitations
    :numCitations}) AS citations
RETURN Conference, citations[0..3] AS TOP3
```

```
ORDER BY Conference
```

Query 2: For each conference find its community: i.e., those authors that have published papers on that conference in, at least, 4 different editions.

```
MATCH (c:Conference)--(e:Edition)--(p:Papers)<-[:writes]-(a:
    Author)
WITH a, c, count(distinct e) AS numOfEditions
WHERE numOfEditions >= 4
RETURN a.name, c.name, numOfEditions
ORDER BY numOfEditions DESC
```

Query 3: Find the impact factors of the journals in your graph (see `https://en.wikipedia.org/wiki/Impact_factor`, for the definition of the impact factor).

```
MATCH (j:Journal)--(v:Volume)--(p:Papers)
WITH j, v.year AS publicationYear, count(p) AS numOfPublications

MATCH (j:Journal)--(cited_v:Volume)--(cited_p:Papers)<-
[c:cites]-(citing_p:Papers)--(citing_v:Volume)
WHERE cited_v.year = citing_v.year - 1 OR cited_v.year = citing_v
    .year - 2
WITH j, toInteger(citing_v.year) AS citationYear, count(c) AS
    numOfCitations, publicationYear, numOfPublications
WHERE  publicationYear = citationYear - 1 OR publicationYear =
    citationYear - 2

WITH j, citationYear, sum(numOfPublications) AS totalPublications
    , numOfCitations
RETURN j.name AS Journal, citationYear AS Year, numOfCitations /
    totalPublications AS ImpactFactor
ORDER BY j.name, citationYear
```

Query 4: Find the h-indexes of the authors in your graph (see `https://en.wikipedia.org/wiki/H-index`, for a definition of the h-index metric).

```
MATCH (a:Author)-[:writes]->(p1:Papers)<-[:cites]-(p2:Papers)
WITH a, p1, count(*) AS paperCitations ORDER BY paperCitations
    DESC
WITH a, collect(paperCitations) AS citationCounts
WITH a, size(citationCounts) AS papersPerAuthor, citationCounts
WITH a, papersPerAuthor, [x IN citationCounts WHERE x >=
    papersPerAuthor | x] AS h
RETURN a.name, papersPerAuthor, coalesce(size(h), 0) AS hIndex
ORDER BY hIndex DESC
```

C Recommender

In this task we create a simple recommender. Specifically, we want to create a reviewer recommender for editors and chairs.

C.1 Stage 1

Find/Define the research communities. A community is defined by a set of keywords. In order to implement this into the graph, we have decided to create a new node called **Community** and a new relation community-keyword called **associated_with**.

```
LOAD CSV WITH HEADERS FROM '{base_url}/Node_community.csv' AS row
MERGE (a:Community {{name: row.community}})
```

```
LOAD CSV WITH HEADERS FROM '{base_url}/Edge_community_keyword.csv
    ' AS row
MATCH (a:Community {{name: row.community}})
MATCH (b:Keywords {{name: row.keywords}})
MERGE (a)-[r:associated_with]->(b);
```

First, we have generated the data by designing a set of keywords that the papers could possibly have, based on a quick overview of several paper titles.

```
keywords= ["property graph", "graph processing", "data quality",
"data management","indexing","data modeling","big data", "data processing",
"data storage", "data querying", "NLP","mathematical methods", "biology","AI" ]
```

We assigned three keywords from this list to each paper, leveraging advanced NLP tasks with the help of the NLTK and sentence_transformers packages. The notebook created for this purpose is named **get_keywords.ipynb**, within which we experimented with various NLP tools. Eventually, we developed a function that assigns the three keywords with the highest "similarity" to the paper's title. As a result, each paper is associated with three relevant keywords. Next, we defined the **database community**, that is defined through the following keywords:

```
db_keywords= [ "data management","indexing","data modeling","big data",
"data processing", "data storage", "data querying" ]
```

C.2 Stage 2

Find the conferences and journals related to the database community. In order to find the conferences and journals that have 90% or more papers of the db community, which are papers that have at least one keyword on the set of data base keywords, we have done the following query:

```
MATCH (j:Journal)--()--(p:Papers)
WITH j, count(DISTINCT p) AS total_journal_papers
MATCH (j)--()--(p1:Papers)--()--(c:Community {name:"Database"})
WITH c,j, total_journal_papers, count(DISTINCT p1) AS
    db_journal_papers
WITH c,j, toFloat(db_journal_papers) / toFloat(
    total_journal_papers)*100 AS percentage
WHERE percentage >= 90
CREATE (j)-[:belongs_in]->(c)
```

We applied a similar query to Conferences, it can be seen in the script *PartC_ChimenoFortó.py*. For answering the next stage we created a new edge between the returned journals/conferences and the database community called **belongs_in**.

C.3 Stage 3

Identify the top papers of these conferences/journals.

```
MATCH (r:Community {name: "Database"})<-[:belongs_to]-(:
    Conference|Journal)<-[:belongs_to|belongs_to]-(:Edition|Volume)
    <-[:published_in|contained_in]-(cited_papers:Papers)<-[citation
    :cites]-(citing_p:Papers)-[:published_in|contained_in]->(:
    Edition|Volume)-[:belongs_to|belongs_to]->(:Conference|Journal)
    -[:belongs_to]->(r)
WITH r, cited_papers, COUNT(citation) as numcitations
ORDER BY numcitations DESC
WITH r, COLLECT(cited_papers)[..100] AS top_cited_papers
UNWIND top_cited_papers AS paper
MERGE (paper)-[:is_top_paper_of]->(r)
```

C.4 Stage 4

Identify gurus, an author of any of these top-100 papers.

```
MATCH (a:Author)-[:writes]->(p:Papers)-[:is_top_paper_of]->(r:
    Community {name: 'Database'})
WITH r, a, COUNT(p) AS numpapers
WHERE numpapers >= 2
MERGE (a)-[:is_a_guru_of]->(r)
```

D Graph algorithms

The main goal of this part is to develop our skills on using graph algorithms to query graph data. To achieve this, we will select two algorithms from those provided, apply them to our graph, and interpret their results.

D.1 PageRank Centrality Algorithm [4]

The PageRank algorithm measures the importance of each node within the graph, based on the number incoming relationships and the importance of the corresponding source nodes. The underlying assumption roughly speaking is that a page is only as important as the pages that link to it.

Contextualization:
In this example, our objective is to identify the most influential papers within a graph. We aim to determine this influence by considering not only the number of citations each paper has received but also the significance of the papers citing it. Therefore, our analysis will encompass both the quantity of citations per paper and the relevance of the papers that have cited it.

Graph Projection:

```
CALL gds.graph.project('pageRankGraph', 'Papers', 'cites')
```

Run the algorithm in stream mode:

```
CALL gds.pageRank.stream('pageRankGraph', {maxIterations: 50,
    dampingFactor: 0.75}) YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS title, score
ORDER BY score DESC, title DESC
```

Parameter selection

- stream execution mode: Return the score for each node
- maxIterations: We gave a big value in order to let the algorithm run multiple times considering that papers have multiple citations.
- dampingFactor: We choose a value slightly lower than the default to mitigate the occurrence of sinks and spider traps associated with excessively high values. However, we avoid selecting a value too low to ensure algorithm convergence

Results:

Title	Score
Complete Families of Invariant Distributions	4.49
Sample Complexity of Composite Likelihood.	3.94
Imagining CLP(A, equiv alpha beta)	3.72
Fixpoint evaluation with subsumption for probabilistic uncertainty	3.68
Implementierung von Algorithmen zut Kompaktifizierung von Programmen...	3.51

Interpretation:
The node 'Complete Families of Invariant Distributions' in the Papers section is the most influential on our graph, not only due to its high number of citations but also because of the significance of the papers citing it.

D.2 Node similarity Algorithm [5]

The Node Similarity algorithm compares a set of nodes based on the nodes they are connected to. Two nodes are considered similar if they share many of the same neighbors.

Contextualization:
In this example, our objective is to identify, for each author, the top 5 authors with whom they usually collaborate on the same papers. In other words, the higher the similarity index between authors, the more papers they share in common as co-authors, indicating a greater likelihood that they will work together again in the future.

Graph Projection:

```
CALL gds.graph.project('nodeSimilarityGraph',
    ['Author', 'Papers'],
    'writes')
```

Run the algorithm in stream mode:

```
CALL gds.nodeSimilarity.stream('nodeSimilarityGraph', {topK: 5,
   similarityCutoff: 0.3})
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).name AS Author1, gds.util.asNode(
   node2).name AS Author2, similarity
ORDER BY Author1, similarity DESC
```

Parameter selection

- topk: We set topK to 5 since we wanted to get only 5 co-authors per author.
- similarityCutoff: We set it to 0.5 to ignore any co-authors with lower similarity.

Results:

Author1	Author2	Similarity
A. Battermann	Eyal Arian	0.5
A. Lauren Crain	Carol R. Thrush	1.0
A. Lauren Crain	Brian C. Martinson	0.5
A. Philip Dawid	Hui Guo	0.5
Aapo Hyvärinen	Michael Gutmann	0.4

Interpretation:
A. Lauren Crain and Carol R. Thrush have a similarity score of 1.0. This indicates that these two authors have a high level of similarity, possibly because they have collaborated extensively or have worked on similar topics. The rest of the showed authors have a moderate level of similarity between them.

References

[1] Marc Fortó Alicia Chimeno. PropertyGraphs_SDM. `https://github.com/marcforto14/PropertyGraphs_SDM/tree/main`, 2024.

[2] DBLP raw data. `https://dblp.uni-trier.de`.

[3] Thom Hurks. dblp-to-csv. `https://github.com/ThomHurks/dblp-to-csv`, 2020.

[4] Neo4j. PageRank. `https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/`.

[5] Neo4j. Node Similarity Algorithm. `https://neo4j.com/docs/graph-data-science/current/algorithms/node-similarity/`.