Academic Graphs

Walter J. Troiani Marc Parcerisa
Facultat d'Informàtica de Barcelona (FIB), UPC, 08034 Barcelona, Spain

***Abstract*—This report outlines the design and implementation of an academic property graph using data from the *Semantic Scholar API*. It covers the modeling of publications, authorship, venues, and citations, along with the evolution of the graph to include review details and affiliations. Advanced queries, reviewer recommendations, and graph algorithms are used to extract insights from the scholarly network.**

I. Introduction

This report presents a comprehensive approach to constructing and analyzing academic property graphs by leveraging data from the *Semantic Scholar API*. It outlines the design and implementation of a graph-based system for modeling scholarly publications, authorship, peer review, and citation relationships. The study details the methodology used for data extraction, transformation, and loading into Neo4j, as well as the system's evolution to incorporate review details and author affiliations. Furthermore, it explores advanced querying techniques, reviewer recommendation mechanisms, and graph algorithms like PageRank and node similarity to extract meaningful insights from the academic graph.

Section II introduces the initial graph design, downloads, transforms and loads data into *Neo4J* in the given schema, as well as explores the means in which it is altered to include new information. Section III provides queries for some of the most common problems that are solved using academic property graphs. In section IV, an algorithm is presented to automatically choose the recommended peer reviewers for a new paper being published as part of a community. Then, section V outlines the implementation of two common graph-based algorithms: *PageRank* and neighborhood-based *Node Similarity*. Finally, section VI gives a small conclusion for this report.

II. Modeling, Loading & Evolving

A. Modeling

This section centers on modeling a graph for managing research publications. We are given a set of real-world constraints and entities, including authors, papers, publication venues (conferences, workshops, journals), and the processes surrounding publication and peer review. Key constraints include the fact that each paper is published in only one venue, conferences and workshops are organized into editions held in specific cities and years, while journals publish in volumes. Papers are written by multiple authors but have only one corresponding author, and can be linked to topics via keywords, as well as to other papers via citations. Reviewers, who must be qualified authors and cannot review their own work, are assigned to evaluate submitted papers.

Based on those preconditions, and taking into account the data that will be available through the *Semantic Scholar API*, we propose the graph schema shown in figure 1. To explain it, we'll break it down in two parts:

1) Publications, citations and authorship

Publications are nodes that hold all the information that we can extract from the *Semantic Scholar API*. They have a `title`, a `url`, and a flag that signifies whether the paper is or not open access (`isOpenAccess`). Optionally, they hold the `year` they were published in, their `abstract` (if accessible), an `openAccessPDFUrl` (to download the pdf), a list of `publicationTypes`, an `embedding`, and a `tldr`. The three latter are, respectively, a list and two dictionaries which hold information about the model that generated them, as well as their value. This data cannot be stored as-is in *Neo4J*, so we encoded them in *json* format.

Authors are also nodes that hold their `name`, and their Semantic Scholar `url`. Optionally, they hold a url for their `homepage`, and their `hIndex`. The latter corresponds to the value calculated on *Semantic Scholar*'s side, and must not be confused with the *H-Index* calculated in section III.

Finally, in terms of edges, we propose three sets of edges that relate authors to publications: `Wrote`, `MainAuthor`, and `Reviewed`, which signify the obvious, and one set of `Cites` edges, which go from publications to publications and indicate that the *source* publication has one reference to the *destination* one. *Semantic Scholar* also provides a flag to identify whether a citation is or not *influential* (probably calculated using some metric on their side), and a list of *contextsWithIntent* that hold *where* and *how* the citation appears in the *citing paper*'s body, which must be stored *json*-encoded.

In this side, we also added two sets of small nodes and their respective edges: `FieldOfStudy` and `KeyWord`, the first of which are provided for each publication by *Semantic Scholar*, and the latter are inferred by the preparation script by looking at the available content of the papers, and are used for section IV. Although we could've went with only one of them, we decided to keep both, as at this point we felt we could compare them, and study their correlations, although we didn't end up using the former.

2) Publication venues and journals

Publications can be published in only one of the following: The `Proceedings` of a `Workshop` or a `Conference`, or in a *volume* (`JournalVolume`) of a `Journal`, although, when analyzing the data from *Semantic Scholar*,

we found that some of the listed publications came from sources other than those (mainly, publications in the *Arxiv*), and decided to add them to the graph design under the `OtherPublicationVenue` node set.

Although the constraints could not strictly be added, having the separation between the different kinds of publication venues allowed us to add other relations such as the `City` that the editions of the conferences or workshops (which lead to the proceedings being published) were held in.

B. Loading

After having designed the initial graph (the one on figure 1), and having taken into account the actual data that the *Semantic Scholar API* offers, this section presents our pipeline for downloading samples of said data, transforming and loading it into *Neo4J*. On top of that, some of the data needed is not avalable through it, so a small synthetic data generating script is also included in the pipeline.

1) Downloading

Multiple HTTP APIs are provided by *Semantic Scholar* to access their data. The first one we looked at was the *Datasets API*, which includes endpoints for downloading data in bulk. Hundreds of *jsonl* documents, adding up to *terabytes* of data can be downloaded through it. However, we quickly discovered an inconvenience with it: The data is not necessarily sorted, which meant that if we only downloaded a small portion of the publication data, followed by a small portion of the citation data, we could end up missing the citation data of most of the publications in our sample.

Thus, we opted for the *Graph API*, which provides a high-level way of querying the publications using their search engine, getting the list of references (and citations, which we ended up not using) of the publications, getting details about authors, and publication venue data, etc.

We created a script that first performs a search on the *Semantic Scholar*'s publications database given a string query, then downloads a list of publication ids, and then downloads details about them, their references and authors, and stores them one by one in two *jsonl* files: One with the papers details, and another with all the citations.

2) Preparing

A script was developed that receives as input a list of *jsonl* files of the same kind data (if too much data is being extracted, instead of creating one single file, the previous step will batch them into multiple files), precesses them, and prepares a set of *csv* files, one for each kind of object within the schema.

For instance, if calling the script with a *citations jsonl* input file, the script will prepare a *citations csv* file that contains four columns: *citedPaperID*, *citingPaperID*, *isInfluential* and the *json*-encoded *contextsWithIntent*.

From the *papers jsonl* input file, the rest of the nodes and edges *csv* files are created. Each *csv* will yield a single kind of object, or a single kind of relationship, that goes from a single kind of object to another one. This means that the file *nodes-cities-1.csv* will hold the information to create all the `City` nodes and nothing else, and the file *edges-ispublishedinproceedings-1.csv* will hold the information to create all the `IsPublishedIn` edges that go from a `Publication` node to a `Proceedings` node.

3) Generating

Four groups of data were not available for download from the *Semantic Scholar API*: A list of cities, the cities where each edition of the conferences and workshops were held in, who reviewed each paper, and the lists of keywords for each paper. For those, a script was developed that, given the results from the previous step, generates four new *csv* files with synthetic (random) data to supply those needs.

Generating key words given a text is a well-known and studied NLP (natural language processing) task usually solved by means of deep learning approaches. To avoid the great complexity of training such large NLP models, popular keyword models like KeyBERT or YAKE were used to overcome this problem. At the end YAKE (Yet Another Keyword Extractor) was used due to the high computational demands of KeyBERT.

YAKE is provisioned for *Python* through a light-weight library called `yake`, which was used to iterate over all the paper rows and extract keywords based on their title, and, if present, *tldr* and abstract.

4) Loading

Finally, *Cypher* provides the `LOAD CSV` interface to iterate over the rows on a *csv* file and operate with them within *Neo4J*. Thus, using it, we wrote a set of queries (one for each *csv* file) such as the following:

```
LOAD CSV WITH HEADERS FROM 'file://...' AS row
MATCH (a:Author {authorID:row.authorID})
MATCH (p:Publication {paperID:row.paperID})
MERGE (a)-[:Reviewed]->(p);
```

These iterate over each one of the rows in the *csv* file, perform all the necessary `MATCH` operations to identify the nodes necessary, and `MERGE` with the existing data to create new nodes or edges if they don't exist already.

C. Evolving

To illustrate the adaptability of graph databases, we introduce an evolution in the original model. Initially, only the reviewers assigned to each paper were tracked; now, we extend the model to capture the full review process. Each review in our new model includes a flag to signify whether it `accepted` or not the paper, the `reviewContent` as a string, and the `minorRevisions` and `majorRevisions`, which aggregate the counts of minor and major changes proposed by the reviewer. Another extension is the inclusion of author affiliations: each author is now associated with an organization, which may be a university or a company.

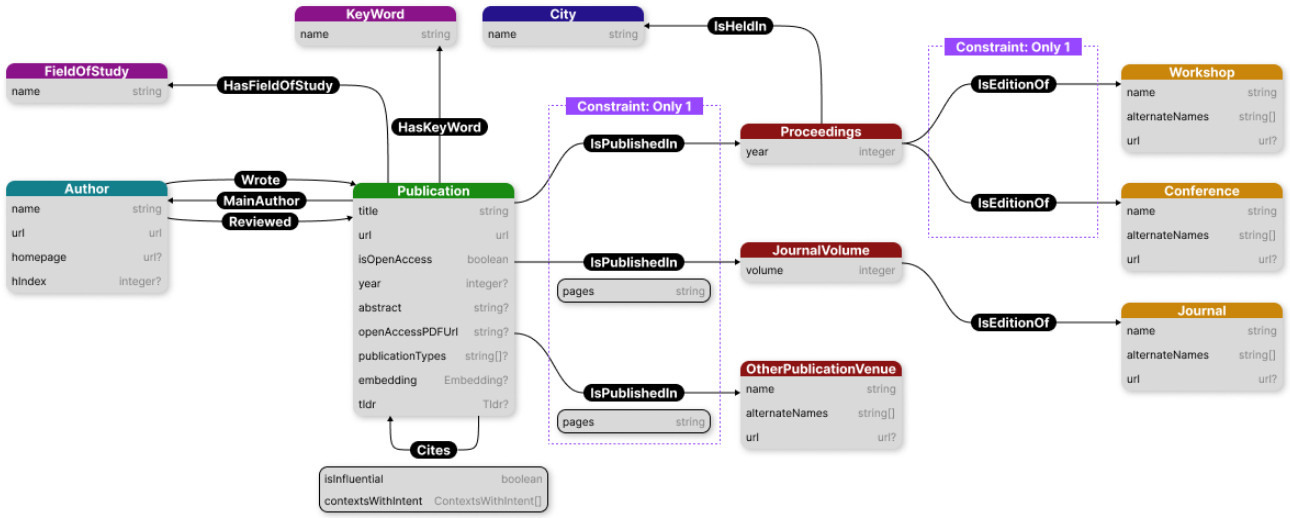

Fig. 1. Initial Graph Design. Developed by a combination of the context provided in the statement, as well as the information available through the Semantic Scholar API.

To apply the alterations, two scrpts were defined:

1) Affiliations

Luckily, *Semantic Scholar* provides the information about the affiliations of each author through its *API*. Thus, the new script simply queries *Neo4J* to retrieve the ids (*Semantic Scholar IDs*) of all the authors, and then requests the affiliation list from the *Graph API*. Finally, for ach author and affiliation, it executes a `MERGE` query such as the following to create (if it doesn't already exist) the `Organization`, and the link from the author to it.

```
MATCH (a:Author {{authorID: "123"}})
MERGE (a)-[:IsAffiliatedWith]
        ->(o:Organization {{name: "UPC"}})
```

2) Review Details

In contrast with the above, *Semantic Scholar* doesn't even provide the review edges, much less their details. Thus, a script was created that first retrieves the `ELEMENTID` of all the `Reviewed` edges in the graph, then chooses randomly the acceptance result, the minor and major revision count and, with a 90% probability, adds a *dummy* review content as a text. Finally, it performs a *Cypher* `SET` operation to update the attributes of the edges as follows:

```
MATCH (:Author)-[r:Reviewed]->(:Publication)
WHERE ELEMENTID(r)='abcd'
SET r.accepted=toBoolean(True)
SET r.minorRevisions=toInteger(15)
SET r.majorRevisions=toInteger(2)
SET r.reviewContent=None
```

After these two operations, the new graph design is showed in figure 2.

III. Querying

This section explores the retrieval of four metrics from the already loaded *Neo4J* dataset via the *Cypher* query language.

A. Finding the most cited papers from each conference or workshop

Given the graph design on figure 2, we provide the following query that retrieves the top 3 most cited papers of each conference or workshop:

```
MATCH (:Publication)-[c:Cites]->(p:Publication),
    (p)-[:IsPublishedIn]->(pr:Proceedings),
    (pr)-[:IsEditionOf]->(v:Workshop|Conference)
WITH p, COUNT(c) AS citations, v
ORDER BY citations DESC
RETURN v.name, COLLECT(p.title)[0..3] AS papers
```

Which returns a table such as the one in Table I. Note the usage of the function `COLLECT`, which is an aggregation function that groups rows by the keys before it, and collects them into a single list. Hence, its name.

B. Finding the community that builds a conference or workshop

Given the graph design on figure 2, we provide the following query that retrieves, for each workflow or conference, the list of authors that have published something in at least four editions of it:

```
MATCH (a:Author)-[:Wrote]->(p:Publication),
    (p)-[:IsPublishedIn]->(pr:Proceedings),
    (pr)-[:IsEditionOf]->(v:Workshop|Conference)
WITH v, a, COUNT(pr) AS editions
WHERE editions>4
RETURN v.name, COLLECT(a.name)
```

Which returns a table such as the one in Table II, which

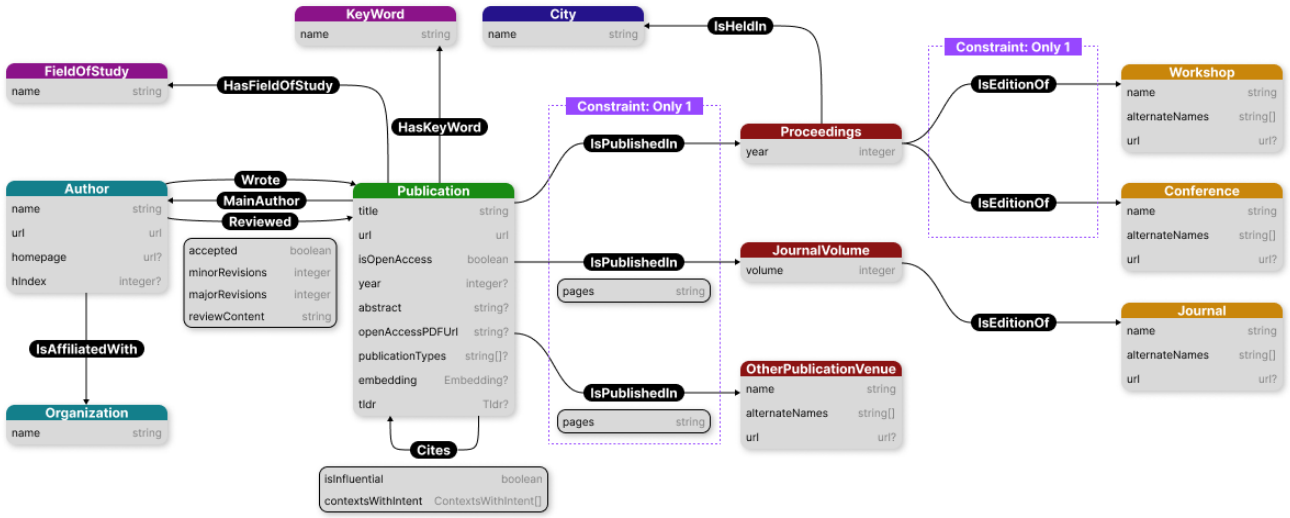

Fig. 2. Modified Graph Design. Developed by a combination of the context provided in the statement, as well as the information available through the Semantic Scholar API.

Workshop or Conference	Papers
European Conference on Machine Learning	[Text Categorization with Support Vector Machines: Learning with Many Relevant Features]
International Conference on Machine Learning	[Experiments with a New Boosting Algorithm, Rectified Linear Units Improve Restricted Boltzmann Machines]
Conference on Empirical Methods in Natural Language Processing	[Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation]
ACM Multimedia	[Caffe: Convolutional Architecture for Fast Feature Embedding]
Neural Information Processing Systems	[Practical Bayesian Optimization of Machine Learning Algorithms, Distance Metric Learning for Large Margin Nearest Neighbor Classification, PyTorch: An Imperative Style, High-Performance Deep Learning Library]
International Conference on Learning Representations	[Explaining and Harnessing Adversarial Examples, Neural Machine Translation by Jointly Learning to Align and Translate, Adversarial examples in the physical world]
Knowledge Discovery and Data Mining	[XGBoost: A Scalable Tree Boosting System]
Conference on Computer and Communications Security	[Deep Learning with Differential Privacy]
IEEE Symposium on Security and Privacy	[Towards Evaluating the Robustness of Neural Networks]

TABLE I
THE THREE MOST CITED PAPERS OF EACH CONFERENCE OR WORKSHOP IN THE DATASET.

raises the question "Who is this 'S. Levine' and how did they publish in four different editions of three different conferences?". Turns out *Sergey Levine* is a renowned author in the sectors of *Machine Learning* and *Robotics*, who racks up more than 170,000 citations across 170 articles and papers, amongst which, some of his best-ranking publications come from precisely the three conferences shown in the table.

Workshop or Conference	Authors
International Conference on Machine Learning	[S. Levine]
IEEE International Conference on Robotics and Automation	[S. Levine]
Conference on Robot Learning	[S. Levine]

TABLE II
LISTS OF AUTHORS THAT PUBLISHED IN AT LEAST FOUR EDITIONS OF CONFERENCES OR WORKSHOPS.

C. Calculating the impact factor of a journal

As defined in [1], the k-year impact factor of a journal on any given year y can be calculated as

$$\text{IF}_y = \frac{\text{Citations}_y}{\sum_{i=1}^{k} \text{Publications}_{y-i}}.$$

Given the graph design on figure 2, we provide the following query that calculates the two-year impact factor for all the journals:

```
WITH 2017 AS year
MATCH (c:Publication)-[:Cites]->(p:Publication),
    (p)-[:IsPublishedIn]->(v:JournalVolume),
    (v)-[:IsEditionOf]->(j:Journal)
WHERE c.year=year AND p.year IN [year-1, year-2]
WITH j, COUNT(c) AS citations
MATCH (p2:Publication)-[:IsPublishedIn]
                        ->(v2:JournalVolume),
    (v2)-[:IsEditionOf]->(j)
WITH j, COUNT(p2) AS publications, citations
RETURN j.name, citations, publications,
    toFloat(citations)/publications AS IF
```

Note that `year` is set as a variable at the start of the sentence,

which allows us to easily alter the year of the calculation. Also note that, for the purpose of maintaining the cleanness of the code, we are using an extra `WITH` sentence towards the end of the query. We could remove it, as all that's left of the code at that point is a `RETURN`, and we could add there the `COUNT` verb.

This query returns the results shown in table III

Journal	Cit	Pub	IF
Proceedings of the National Academy of Sciences of the United States of America	1	14	0.07143
Science	5	32	0.15625
Nature	6	31	0.19355
Journal of machine learning research	14	5	2.8
IEEE Transactions on robotics	2	17	0.11765
Review of Economics and Statistics	1	1	1.0
Journal of NeuroEngineering and Rehabilitation	1	2	0.5
Nature Materials	1	4	0.25
Advancement of science	1	1	1.0
Nature Communications	1	7	0.14286
Proceedings of the IEEE	2	6	0.33333

TABLE III
IMPACT FACTOR OF THE JOURNALS IN THE DATASET, ACCOUNTING ONLY FOR THE 1000 APPROX. LOADED PAPERS, IN THE YEAR 2017.

D. Calculating the h-index of an author

As defined in [2], the h-index of an author is defined as the maximum value of h such that the given author (or journal) has published at least h papers that have been cited at least h times.

Given the graph design on figure 2, we provide the following query that calculates the h-index of all the authors:

```
MATCH (a:Author)-[:Wrote]->(p:Publication),
    (:Publication)-[c:Cites]->(p)
WITH p, COUNT(c) AS citations, a
ORDER BY citations DESC
WITH a, COLLECT(citations) AS citation_counts
RETURN a.name, citation_counts,
    REDUCE(agg=0, x IN citation_counts |
    CASE x WHEN > agg THEN agg+1 ELSE agg END)
                                    AS hIndex
ORDER BY hIndex DESC
```

Note the use of the `REDUCE` to iterate over the ordered collection of paper citation counts, grouped per author.

This query returns the results shown in table IV, where the already acclaimed *S. Levine* appears the third.

Author	Ordered Paper Citation Counts	H Index
Wolfram Burgard	[20, 14, 11, 11, 10, 9, 9, 9, 9, 7, 7, 6, 5, 5, 5, 5, 4, 3, 3, 2, 2, 2]	9
S. Thrun	[20, 18, 17, 14, 11, 11, 9, 9, 9, 9, 9, 9, 7, 7, 6, 5, 5, 5, 4, 4, 3, 3, 1, 1]	9
S. Levine	[32, 31, 14, 14, 11, 11, 11, 9, 7, 6, 6, 5, 5, 4, 4, 3, 3, 3, 2, 2, 1, 1, 1]	8
D. Fox	[20, 14, 11, 11, 9, 9, 9, 9, 7, 7, 7, 7, 6, 5, 5, 3, 3, 2, 1, 1, 1]	8
G. Whitesides	[26, 21, 14, 11, 10, 9, 8, 4, 2]	7
R. Shepherd	[26, 21, 11, 10, 9, 8, 4, 2]	6
S. Schaal	[13, 9, 9, 7, 7, 6, 2, 2, 1]	6
P. Abbeel	[32, 31, 11, 11, 9, 6, 5, 5, 5, 5, 5, 3, 2, 2, 1]	6
J. Borenstein	[15, 14, 9, 7, 7, 4, 2]	5
F. Dellaert	[14, 11, 9, 8, 7, 5, 2]	5

TABLE IV
H INDEX OF THE TOP 10 AUTHORS IN THE DATASET, ACCOUNTING ONLY FOR THE 1000 APROX. LOADED PAPERS.

IV. REVIEWER RECOMMENDER SYSTEM

Another key point to be tackled by this work is the implementation of a recommendation system that is able to suggest promising reviewers for a given community. A community is formed around a topic and has several keywords related to it, in our example the topic is "databases" and the potential keywords are: "data management", "data processing", "data storage", "data querying", "big data".

However, this poses a challenge given the actual data model produced after section II. Therefore, the evolution of the graph schema, is once again needed in order to model a couple of new nodes, edges and properties (e.g. Communities).

Having papers already linked to `KeyWord` nodes, the recommendation algorithm is as follows. Firstly, the "Database" community will be created and associated with the following keywords: data management, indexing, data modeling, big data, data processing, data storage and data querying; which barely changes the graph.

Secondly, venues that surpass a certain percentage of papers that share keywords with the "Database" community are labeled as "DatabaseVenue". Surprisingly, using a low threshold like 10%, only 1 paper and thus venue is obtained. Thirdly, the top 100 most cited papers related by keyword to the database community from that single venue are retrieved, which results in just 1 single publication which is labeled with property "TopDatabasePaper". Lastly, we consider all authors of such paper as potential reviewers of the database community, which yields about 22 reviewers which are labelled with property "DatabaseReviewer". However, given that only one such paper exists, no matter which is the value chosen as a threshold for defining gurus, no gurus are found.

To perform such action generically for any given community, a dynamic script is provided to avoid hard-coding a specific community by parametrizing all relevant parameters inside the *Cypher* statements via *Python* f-strings. However, for the following example statement we have fixed all the paramters and the community to "Database". The data used for this purpose has been an S2 ingestion of 1000 publications and all its related information by using the query: "Database". The summarized statement to execute this recommendation algorithm with a fixed "Database" community is the following:

```
// STEP 1: Create Community and its Keywords
MERGE (c:Community {name: "Database"})
WITH c, [
  "data management", "indexing", "data modeling",
  "big data", "data processing", "data storage",
  "data querying"
] AS keywords
```

```
UNWIND keywords AS kw
MERGE (k:KeyWord {name: kw})
MERGE (c)-[:HasKeyWord]->(k);

// STEP 2: Label community-related venues
MATCH (c:Community {name: "Database"})
  -[:HasKeyWord]->(k:KeyWord)
MATCH (v)<-[:IsPublishedIn]-(p:Publication)
  -[:HasKeyWord]->(k)
WHERE v:Proceedings OR v:JournalVolume
  OR v:OtherPublicationVenue
WITH v, COUNT(DISTINCT p) AS related_papers
MATCH (v)<-[:IsPublishedIn]-(all:Publication)
WITH v, related_papers,
  COUNT(DISTINCT all) AS total_papers
WHERE total_papers > 0 AND
  (related_papers * 1.0 / total_papers) >= 0.10
SET v:`DatabaseVenue`;

// STEP 3: Rank top 100 papers in DatabaseVenue
MATCH (k:KeyWord)<-[:HasKeyWord]-(p:Publication)
  -[:IsPublishedIn]->(v:`DatabaseVenue`)
MATCH (citing:Publication)-[:Cites]->(p)
MATCH (citing)-[:HasKeyWord]->(k)<-[:HasKeyWord]
  -(c:Community {name: "Database"})
WITH p, COUNT(DISTINCT citing) AS citations
ORDER BY citations DESC
LIMIT 100
SET p:`TopDatabasePaper`;

// STEP 4: Label top reviewers and gurus
MATCH (a:Author)-[:Wrote]
  ->(p:`TopDatabasePaper`)
WITH a, COUNT(p) AS top_papers
SET a:`DatabaseReviewer`
WITH a, top_papers
WHERE top_papers >= 2
SET a:`DatabaseGuru`;
```

V. Graph Algorithms

Now that the structural construction of the research graph is complete, the remainder of this work focuses on raw data analysis to uncover latent insights from its topology. To facilitate this, we employ the Graph Data Science (GDS) library provided by Neo4j, which enables scalable execution of graph algorithms directly over projected subgraphs. Among the suite of available algorithms, two were selected due to their analytical relevance in a scholarly publication context: **PageRank** and **Node Similarity**.

PageRank is a centrality algorithm that quantifies the relative importance of each publication within the citation graph. It operates under the intuition that a paper is influential if it is cited by other influential works. By streaming scores over the citation subgraph, this method provides a citation-weighted ranking of papers. The results offer an interpretable metric akin to classical bibliometrics like impact factor and h-index but derived from the topological structure of the citation network itself.

Node Similarity, on the other hand, is used to measure the affinity between publications. Specifically, we compute similarity between nodes (constrained to publications only) using a Jaccard similarity metric over the set of neighbors in the graph that includes both `Wrote` and `Cites` relationships. This yields a numerical indicator of how similar two papers are in terms of their authorship and citation patterns, highlighting collaborative clusters and thematic proximity in the graph.

All graph projections and algorithm executions were handled via Python using the official Neo4j driver, and the resulting scores (e.g., PageRank values, node similarities) were exported to tabular format for downstream analysis. Full code, results, and reproducibility instructions are available in the project repository [3].

VI. Conclusions

In this paper, we presented a methodology for modeling, loading, and evolving academic graphs, focusing on research papers, authors, conferences, and their interconnections. The framework incorporates various aspects such as the publication process, citation relations, review assignments, and topic associations, by leveraging data from the *Semantic Scholar API*.

The proposed model allows for efficient querying and evolution of academic graphs. Future work would focus on refining the graph structure to incorporate more nuanced aspects of the academic ecosystem, including collaboration networks, author productivity, and citation dynamics. Moreover, additional data sources could be integrated to enhance the completeness and accuracy of the graph.

However, this work has several points of improvement that could be refined in posterior efforts. Firstly, keyword extraction could be enhanced by the usage of the paper embeddings provided by the *API*. Secondly, the graph sizes utilized throughout this study were significantly smaller than what was initially expected, in the order of $10^3 - 10^4$ at most, rather than the initial goal of 10^5. The reason behind this is the huge amount of edges in this kind of graphs (popular publications have a humongous number of citations and references), and the limitations associated with the *S2 API key* used and the lack of bulk retrieval endpoint for references of the API which slowed down the process greatly.

We believe that such an academic graph has been proven to be a valuable tool for both researchers and institutions to analyze the landscape of scientific publications and collaborations.

References

[1] T. I. for Scientific Information (ISI), "The impact factor," 1994. [Online]. Available: https://clarivate.com/academia-government/essays/impact-factor/

[2] J. E. Hirsch, "An index to quantify an individual's scientific research output," *Proc Natl Acad Sci U S A*, vol. 102, no. 46, pp. 16 569–16 572, Nov. 2005.

[3] W. Parcerisa, Troiani, "Languagedetection," 2025. [Online]. Available: https://github.com/AimbotParce/MDS-MUD-LanguageDetection