Lab assignment No.1

# Property Graphs

Students:
**Arina GEPALOVA**
**Chidiebere OGBUCHI**

Professor:
Anna QUERALT

2023

# Part A

## A.1 Modeling

The main node is ARTICLES. It directly connects to KEYWORDS, JOURNALS, AUTHORS, CONFERENCES.
JOURNALS and ARTICLES have one-to-many relationship => connect them with an edge PUBLISHED_IN with properties START_ID: articleID and END_ID: journalID.

KEYWORDS are located in the separate nodes, connected with ARTICLES via relationship CONTAINS. With this approach it is easier to keep many articles with the similar topic, for example, 100 articles on 2 topics we don't have to repeat keyword for each article. We just connect it with an edge CONTAINS.

Edge WRITES connects AUTHORS with their ARTICLES.

Node CONFERENCES combines idea of conferences and workshops, since it is the same for cases of using the data (parts B, C, D). Property EdtVenue represents city and EdtNumber - edition/volume.

Edge CITE_HAS_CITATION shows the relationship between 2 articles. Here we had to make sure that difference between cited article and the article that cites it is 2 years. Citation: 2017-2019; cite: 2020-2022.

Since we include the concept of review we also add a node REVIEWS that is connected with ARTICLES with the relationship REVIEWS_OF_ARTICLE. One article can have more than 1 review, therefore REVIEWS_OF_ARTICLE has properties START_ID: articleID and END_ID: reviewID.

The full list of properties:
**Articles**: articleID (int)- unique id of an article
Author (string[]) - names of the authors
Booktitle (string) -
Journal (string) - a journal, where the article is published
Year (int) - a year of writing an article
Corresponding_author (string) - the main author of the article
Abstract (string) - short summary of an article

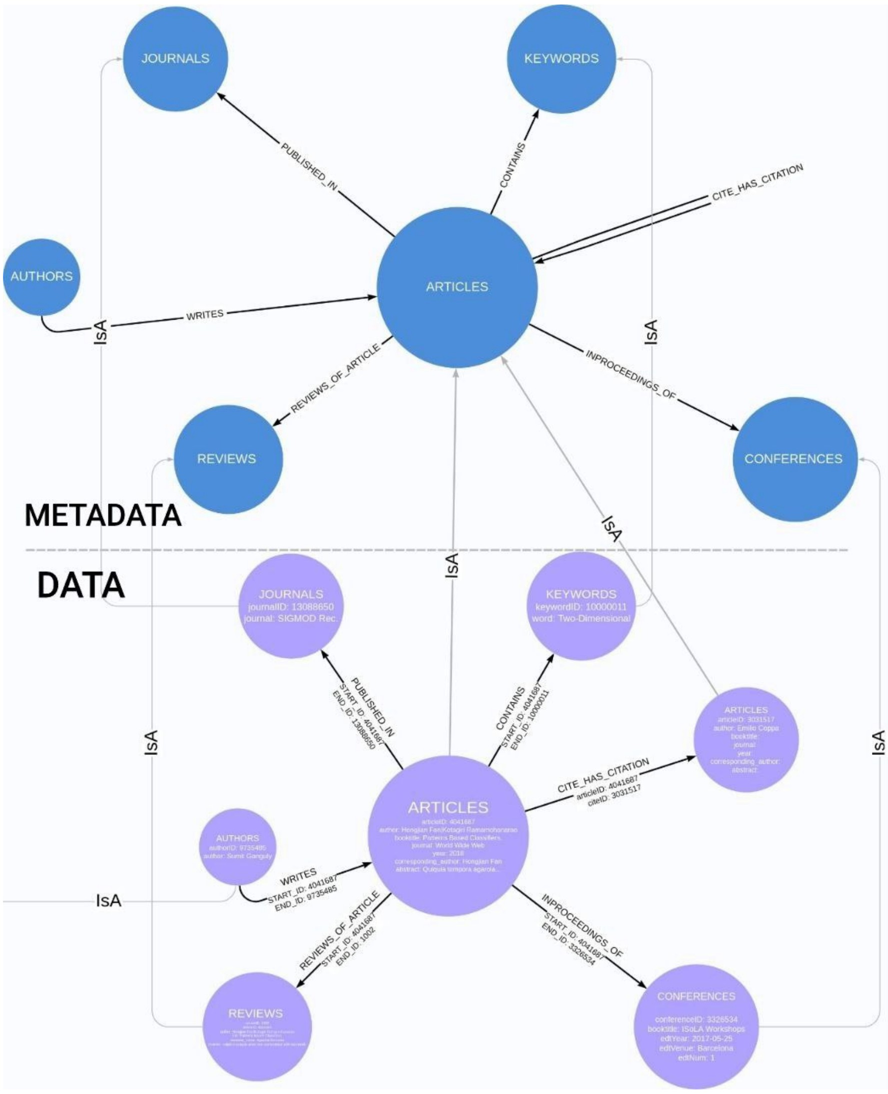
**Authors***:* authorID (int) - unique id of an author
Author (string) - name of the author

**Keywords***:* keywordID (int) - unique id o f a keyword
Word (string) - a keyword

**Conferences***:* conferenceID (int) - unique id of a conference
ConfName (string) -
edtYear (date) - date of a conference
edtVenue (string) - city of the conference

edtNumber (int) - number represent edition for a conference, volume for a workshop

**Journal***:* journal(ID) - unique id of a journal
journal (string) - name of a journal




# A.2 Instantiating/Loading

## DBLP Data

- 2023 DBLP data was obtained from (https://dblp.uni-trier.de/xml/)
- Data was transformed from xml into csv using a python converter accessed on github (https://github.com/ThomHurks/dblp-to-csv). This converted the dblp file into readable format for neo4j import.

## Data Wrangling

The obtained CSVs were not readily usable for the Lab exercise, thus we pre-processed them based on our graph model while using the following approaches and assumptions:

- We represent the each node and relationships as separate individual CSVs.

```
#Generated files names
files
```

```
['Mains\\articles.csv',
 'Mains\\articles_in_conferences.csv',
 'Mains\\articles_in_journal.csv',
 'Mains\\article_keywords.csv',
 'Mains\\authors.csv',
 'Mains\\authors_writes_articles.csv',
 'Mains\\author_affiliated_school.csv',
 'Mains\\citation.csv',
 'Mains\\cite_has_citation.csv',
 'Mains\\conferences.csv',
 'Mains\\journals.csv',
 'Mains\\keywords.csv',
 'Mains\\reviews.csv',
 'Mains\\reviews_in_articles.csv',
 'Mains\\school.csv']
```

- Following the order in which they were pre-processed,

○ **(:CITE NODE) --> [:CITE_HAS_CITATION]**

➔ Selected 10,000 articles from the dblp. Affixing the articles into journal J, conferences C, workshops W and others N in ratio (35:45:12:8) respectively.

➔ Further, Split the articles by 33% for citing articles and 67% for cited articles. We modified the date so that Citation articles (2017-19) have to be older than cite articles (2020-22).

○ **(:ARTICLE NODE)**

➔ Selected 10,000 articles from the dblp comprising of the cites and citation.

➔ Node contained information about the articles like authors names, year, title, book and other important information pertaining to journals & conferences .

○ **(:AUTHORS NODE) --> [:AUTHORS_WRITES_ARTICLES]**

➔ Selected 1000 random authors from dblp and assigned them to the articles, making sure that most articles have co-authors as well.

○ **(:JOURNALS NODE) --> [:PUBLISHED_IN]**

➔ Selected 8 journals from Dblp and randomly assigned to the articles published in journal.

○ **(:CONFERENCES NODE) --> [:INPROCEEDINGS_OF]**

➔ Similarly, synthetically generated 8 conferences with name, edition, venue etc and assigned randomly to inproceedings of articles.

○ **(:REVIEWERS NODE) --> [:REVIEWS]**

➔ Selected authors from 1000 authors previously selected and assigned 3 authors each to review a paper. We ensure that no author reviews his own paper.

➔ Generated synthetic reviews using lorem library and acceptance remarks for each reviews.

○ **{ADDING ABSTRACTS TO ARTICLES & CORRESPONDING AUTHOR}**

➔ Using the article titles and lorem paragraph, we created fake abstracts for each articles.

➔ We assume that the first author listed in each author[] array of the article is the corresponding author for that article.

➔ These properties were included as columns in the article csv.

- **(:KEYWORDS NODE) --> [:CONTAINS]**

➔ Using combined strings from the article titles and the keywords from the exercise, we randomly assigned them to the articles.

- **(:UNIVERSITIES NODE) --> [:AFFILIATED_IN]**

➔ We assume all authors are affiliated to only universities. Therefore we assign all the list of schools to authors.

Generally, the relationships had a many to many cardinality. The start id is akin to the source node id whilst end id is the target node id. However, we ensured every ':ID' for each node was unique i.e. the primary keys had no duplicates.

**Data Loading**

- **from neo4j library** we imported **GraphDatabase**. We created an importer class as well as methods for emptying the database, then creating constraint on each node before loading the node as seen in the python script. This constraint ensures that every Primary key is unique.

```
def import_csv(self, files):
    total_time = 0
    with self.driver.session() as session:
        # Delete all nodes in database
        session.run('MATCH (n) DETACH DELETE n')
        print('Deleted all nodes in database')

        #Create Constraint
        for file, label in files:
            try:
                session.run(f'CREATE CONSTRAINT {label.upper()} IF NOT EXISTS FOR (p:{label.upper()}) REQUIRE p.{label}id IS UNIQUE'
                print(f'Created constraint for {label.upper()} node')
            except Exception as e:
                print(f'Error creating constraint for {label.upper()} node')

        # Import data from CSV files
        for file, label in files:
            try:
                start_time = time.time()
                result = session.run(f'''
                LOAD CSV WITH HEADERS FROM 'file:///{file}' AS row
                WITH toInteger(row.{label}ID) AS {label}id, row
                MERGE (o:{label.upper()} {{ {label}id: {label}id }})
                SET o += row
                RETURN count(o) AS count;
                ''')
```

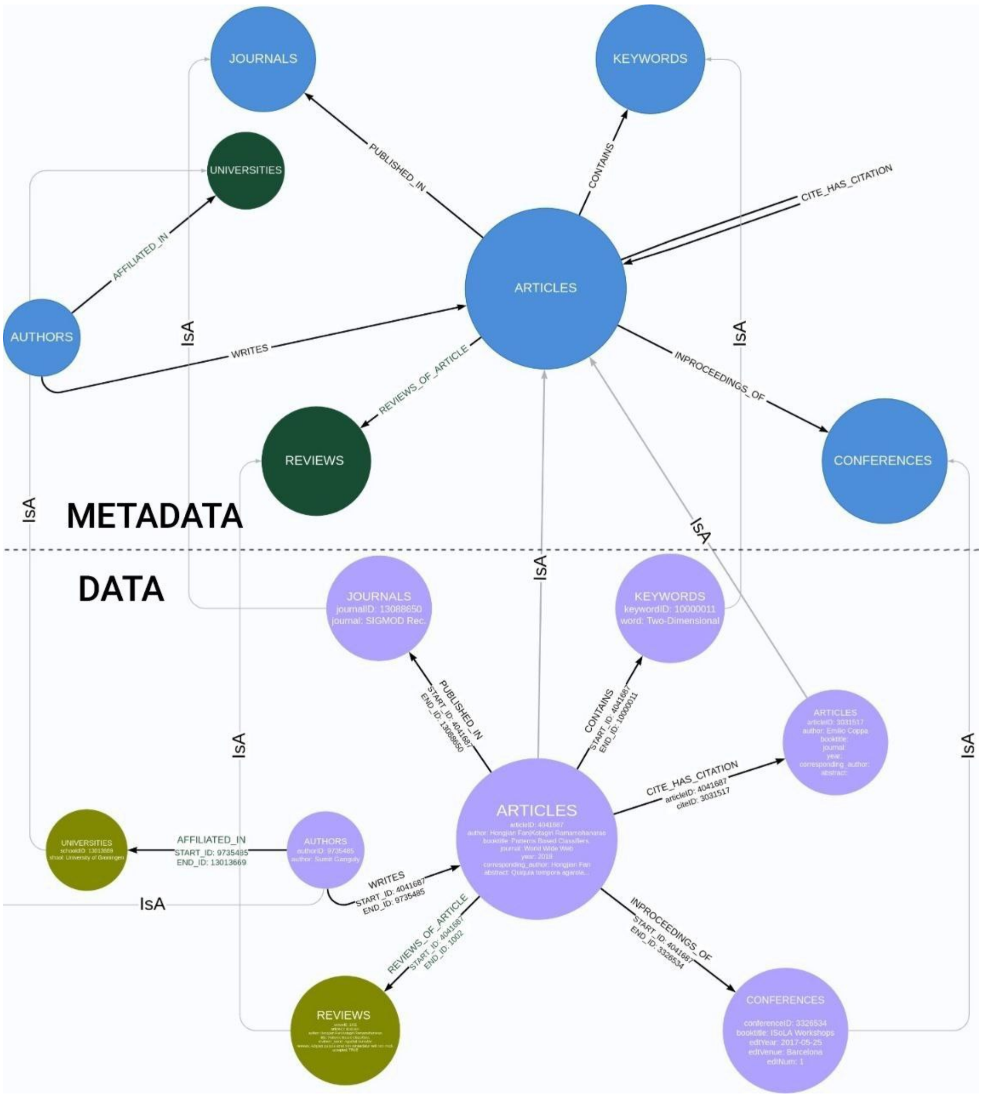
- **from py2neo** we imported **Graph.** Similar to above, we used this library to for running the queries in the lab exercises for easier display of output.

## A.3 Evolving the graph

To evolve the graph we add one new node (UNIVERSITIES) and an edge AFFILIATED_TO. UNIVERSITIES node has schoolID(int) and school(string) properties.

For REVIEWS node some new properties were added. In the initial graph we kept just the fact of a review (author who reviewed). In the evolved graph the feedback is also included, as well as the decision in order to calculate the final decision later.

Reviews (string) - review for an article
Accepted (boolean) - decision of a particular reviewer.




# Part B. Querying

All Queries were run using py2neo library and results are visible in the notebooks

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

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

```
most_cited_papers  = """
MATCH (ar:ARTICLE)<-[r:CITE_HAS_CITATION]-(a:ARTICLE)-[:INPROCEEDINGS_OF]->(c:CONFERENCE)
WITH c, a, count(r) AS citations
ORDER BY c.`ConfName:string`, citations DESC
WITH c, collect({article: a, citations: citations}) AS articles
RETURN c.`ConfName:string` AS conference, articles[0].article.`title:string` AS top_cited_1, articles[0].citations AS citations_1, articl
ORDER BY c.`ConfName:string`;"""

graph.run(most_cited_papers)
```

| conference | top_cited_1 | citations_1 | top_cited_2 | citations_2 | top_cited_3 | citations_3 |
|---|---|---|---|---|---|---|
| European Conference on Artificial Intelligence (ECAI) | Tracking control with prescribed transient behaviour for systems of known relative degree. | 10 | A combinatorial 2.375-approximation algorithm for the facility location problem with submodular penalties. | 9 | A varying terminal time mean-variance model. | 9 |
| European Conference on Computer-Supported Cooperative Work (ECSCW) | The Complexity of Logical Theories. | 11 | Improved Outdoor Localization Based on Weighted Kullback-Leibler Divergence for Measurements Diagnosis. | 9 | Stability of linear second-order time-varying differential equations via contractive polygons. | 9 |
| European Conference on Human-Computer Interaction (ECHI) | An Enhanced Histogram of Oriented Gradients for Pedestrian Detection. | 9 | Mixing logics and rewards for the component-oriented specification of performance measures. | 8 | Profit Maximization problem with Coupons in social networks. | 8 |

2. Community authors that have published papers on that conference in, at least, 4 different editions

```
conf_comm = """
MATCH (a:AUTHOR)-[:WRITES]->(p:ARTICLE)-[:INPROCEEDINGS_OF]->(c:CONFERENCE)
WITH a, c, COUNT(DISTINCT p) AS papers, COLLECT(DISTINCT p.`year:int`) AS years
WHERE papers >= 4 AND SIZE(years) >= 4
WITH c, COLLECT(DISTINCT a.`author:string`) AS community
RETURN c.`ConfName:string`, community;"""

graph.run(conf_comm)
```

| c.`ConfName:string` | community |
|---|---|
| International Conference on Data Mining (ICDM) | ['Rogier Woltjer', 'Per Carlsson', 'Hannes Holm', 'PerOlof Bengtsson', 'Johan Eklund', 'Masatsugu Hangyo', 'Wiradee Imrattanatrai', 'Kim J. L. Nevelsteen', 'Floris Erich', 'Hisao Ogata', 'Syafril Bandara', 'Ehab Morsy', 'Kanae Akaiwa', 'Raghvendra Jain', 'Hironori Kiya', 'Simin Cai', 'Hamid Farhady', 'Ruijian An', 'Jan Erik Mostr\u00f6m', 'J. Rafid Siddiqui', 'Helena Lindgren', 'Loove Broms', 'Takuya Tsutsumi', 'Georg Hodosi', 'Frank Spie\u00df', 'Mikael Wiberg', 'Meng-Yu Kuo', 'Bob Melander', 'Jun-Li Lu', 'Hung Nghiep Tran', 'Tim Overkamp', 'Andrii Dmytryshyn', 'Karl Ljungkvist', 'Masaki Waga', 'Ahmed Ishtiaq', 'Hannes Ebner', 'Yiming Wu', 'Sameh El-Ansary', 'Thi Ngoc Dung Tieu', 'H\u00e5kan Jonsson', 'Tobias Olsson', 'Karin Danielsson \u00d6berg', 'Andrzej Pronobis', 'John W. Richardson', 'John Brumley', 'Yu Nakahata', 'Ranil Peiris', 'Aseel Berglund', 'Romain Fontugne', 'Mark A. Friedman', 'Anders Berglund', 'Tove Helldin', 'Marc A. Kastner 0001', 'Magnus Ingmarsson', 'Sanjay Ghemawat', 'Sergey Tarasenko', 'Jonathan Crusoe', 'Shahzad Saleem', 'Denis Kleyko', 'Abdelbaki Bouguerra', 'Huan Liu', 'Somchai Chatvichienchai', 'Erik Johansson', 'Wito Engelke', 'Malin Sandstr\u00f6m', 'Mo Shen', 'Muhammad Usman 0002', 'Yoshiyuki Shoji', 'Hoang-Quoc Nguyen-Son', 'Robert A. Granat', 'Fredrik Edin', 'Si-Mohamed Lamraoui', 'Kosetsu Tsukuda', 'Dieter Altenburger', 'Loic Merckel', 'Ryousuke Matsumoto', 'Thomas Weish\u00e4upl', 'Asmus Pandikow', 'Maike Paetzel-Pr\u00fcsmann', 'Namgi Han', 'Jens-Olof Lindh', 'Sergii Voronov', 'Gustav Nordh', 'Charles N. Tarimo', 'Marcel Cavalcanti de Castro', 'Kensuke Tanioka', 'Masato Shinjo', 'Gianantonio Bortolin', 'Hoc Phan', 'Islam A. K. M. Mahfuzul', 'Shotaro Kamiya', 'Yu Terada'] |

3. Using reference from https://en.wikipedia.org/wiki/Impact_factor, for the definition of the impact factor). We calculated for year 2019.

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

```
year_to_int = """MATCH (a:ARTICLE)
SET a.`year:int` = toInteger(a.`year:int`);"""

graph.run(year_to_int)
```

(No data)

```
impact_factors = """MATCH (j:JOURNAL)<-[:PUBLISHED_IN]-(a:ARTICLE)
WHERE a.`year:int` >= 2017 AND a.`year:int`  <= 2018
WITH j, COLLECT(DISTINCT a) AS ARTICLE
MATCH (c:ARTICLE)<-[:CITE_HAS_CITATION]-(a)
WHERE c.`year:int` <= 2019
WITH j, COUNT(DISTINCT c) AS citations, ARTICLE
RETURN j.`journal:string` AS Journal, toFloat(citations)/toFloat(SIZE(ARTICLE)) AS TwoYearImpactFactor
ORDER BY TwoYearImpactFactor DESC;"""

graph.run(impact_factors)
```

| Journal | TwoYearImpactFactor |
|---|---|
| Inf. Technol. Dev. | 35.82887700534759 |
| Theor. Comput. Sci. | 35.26315789473684 |
| Sci. Comput. Program. | 34.35897435897436 |

4. Using reference from https://en.wikipedia.org/wiki/H-index, for a definition of the h-index metric).

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

```
h_indexes = """MATCH (a:AUTHOR)-[:WRITES]->(p:ARTICLE)
WITH a, COLLECT(DISTINCT p.`year:int`) AS years, COUNT(DISTINCT p) AS num_papers
UNWIND RANGE(1, num_papers) AS i
WITH a, years, REDUCE(s = 0, j IN COLLECT(i) | s + CASE WHEN j <= SIZE(years) AND y
RETURN a.authorID AS Author_ID, a.`author:string` AS Author, h_index
ORDER BY h_index DESC;"""

graph.run(h_indexes)
```

| Author_ID | Author | h_index |
|---|---|---|
| 9735494 | Tomoya Mori | 5 |
| 9735599 | Tatsuhisa Yamaguchi | 5 |
| 9735690 | Natthawut Kertkeidkachorn | 5 |

# Part C. Recommender

## 1. Find the research communities related by keywords

```
## Step 1.1 - Create database community node
community = """MERGE (c:community {name:'db'})
ON CREATE SET c.created_at = timestamp()
RETURN c;"""

graph.run(community)
```

| c |
|---|
| (_3500:community {created_at: 1678925201438, name: 'db'}) |

```
## Step 1.2 - Define which keywords related to the database community

comm_kw = """MATCH (c:community {name:'db'})
MATCH (kw:KEYWORD)
WHERE kw.`word:string` in ['data management', 'indexing', 'data modeling',
'big data', 'data processing', 'data storage' , 'data querying']
MERGE (kw)-[:RELATED_TO]->(c)
RETURN c;"""

graph.run(comm_kw)
```

| c |
|---|
| (_3500:community {created_at: 1678925201438, name: 'db'}) |
| (_3500:community {created_at: 1678925201438, name: 'db'}) |
| (_3500:community {created_at: 1678925201438, name: 'db'}) |

## 2. Find the conferences and journals related to the database community

Assumption: If 90% of the papers published in a conference/journal contain one of the keywords of the database community, we consider that conference/journal as related to that community.

```
related_c_j = """
//Journal
MATCH (a:ARTICLE)-[:PUBLISHED_IN]->(j1)
WITH j1, j1.`journal:string` AS Name, count(*) AS  num_papers
WHERE num_papers > 0
MATCH (j1)<-[:PUBLISHED_IN]-(a:ARTICLE)-[:CONTAINS]->(kw:KEYWORD)-[:RELATED_TO]->(com:community {name:'db'})
WITH j1,com,Name,  num_papers, count(distinct a.`title:string`) AS  num_papers_db, a.sup as type
WITH j1, com, Name, num_papers, num_papers_db, (1.0*num_papers_db/num_papers) AS db_percentage, type
WHERE db_percentage >= 0.9
MERGE (j1)-[:RELATED_TO]->(com)
RETURN Name, type,  num_papers,num_papers_db, db_percentage

UNION

//Conference
MATCH (a:ARTICLE)-[:INPROCEEDINGS_OF]->(c1)
WITH c1, c1.`ConfName:string` AS Name, count(*) AS  num_papers
WHERE num_papers > 0
MATCH (c1)<-[:INPROCEEDINGS_OF]-(a:ARTICLE)-[:CONTAINS]->(kw:KEYWORD)-[:RELATED_TO]->(com:community {name:'db'})
WITH c1,com,Name,  num_papers, count(distinct a.`title:string`) AS  num_papers_db, a.sup AS type
WITH c1, com, Name, num_papers, num_papers_db, (1.0*num_papers_db/num_papers) AS db_percentage, type
WHERE db_percentage >= 0.9
MERGE (c1)-[:RELATED_TO]->(com)
RETURN Name,type,  num_papers,num_papers_db, db_percentage;"""

graph.run(related_c_j)
```

| Name | type | num_papers | num_papers_db | db_percentage |
|---|---|---|---|---|
| Found. Comput. Math. | J | 450 | 446 | 0.9911111111111112 |
| IEEE Embed. Syst. Lett. | J | 439 | 436 | 0.9931662870159453 |
| J. Assoc. Inf. Syst. | J | 423 | 422 | 0.9976359338061466 |

## 3. Identify the top papers of these conferences/journals. We consider only for conferences.

Create a PageRank method using gds library

```
PageRank = """CALL gds.graph.project(
  'myPageRank',
  'ARTICLE',
  'CITE_HAS_CITATION'
);"""

graph.run(PageRank)
```

| nodeProjection | relationshipProjection | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| {ARTICLE: {label: 'ARTICLE', properties: {}}} | {CITE_HAS_CITATION: {orientation: 'NATURAL', indexInverse: false, aggregation: 'DEFAULT', type: 'CITE_HAS_CITATION', properties: {}}} | myPageRank | 10000 | 13397 | 87 |

Call a PageRank Algorithm method using gds stream. Identify top 100.

```
top_100 = """CALL gds.pageRank.stream('myPageRank')
YIELD nodeId, score
WITH gds.util.asNode(nodeId).articleid AS ID, gds.util.asNode(nodeId).`title:string` AS title, score
ORDER BY score DESC, title ASC
MATCH (com:community {name:'db'})
WITH ID, com, score, title
MERGE (:ID)-[:RELATED_TO]->(com)
RETURN ID, title, score
limit 100;"""

graph.run(top_100)
```

| ID | title | score |
|---|---|---|
| 6620607 | On permuted super-secondary structures of transmembrane β-barrel proteins. | 0.405 |
| 6612420 | Structure Inference for Linked Data Sources Using Clustering. | 0.405 |
| 6612436 | An Adaptive Similarity Search in Massive Datasets. | 0.34125000000000005 |

### 4. Identify the gurus

```
gurus = """
MATCH (au:AUTHOR)-[:WRITES]->(b:ARTICLE)-[:INPROCEEDINGS_OF]-
(:CONFERENCE)-[:RELATED_TO]->(com:community {name:'db'})
WITH au, COUNT(*) AS  num
WITH au, au.`author:string` AS author_name, num AS num_influential_papers,
CASE WHEN num >=2 THEN 'guru' ELSE 'reviewer' END AS reviewer_type
MERGE (au)-[:RELATED_TO {type:reviewer_type}]->(com)
RETURN author_name, num_influential_papers, reviewer_type
ORDER BY num_influential_papers DESC;"""

graph.run(gurus)
```

| author_name | num_influential_papers | reviewer_type |
|---|---|---|
| Hannes Holm | 32 | guru |
| Raihan Ul Islam | 31 | guru |
| Romain Fontugne | 31 | guru |

# Part D. Graph algorithms

### 1. Closeness Centrality Algorithm

Closeness centrality is a way of detecting nodes that are able to spread information very efficiently through a graph. For our case, we determine highly connected articles in the community base of CITE_HAS_CITATION.

```
## Call previous algorithm method using gds stream

closeness = """CALL gds.beta.closeness.stream('myPageRank')
YIELD nodeId, score
WITH gds.util.asNode(nodeId).articleid AS ID,
     gds.util.asNode(nodeId).`title:string` AS title,
     score
ORDER BY score DESC, title ASC
MATCH (com:community {name:'db'})
WITH ID, com, score, title
MERGE (:ID)-[:RELATED_TO]->(com)
RETURN ID, title, score
limit 100;"""

graph.run(closeness)
```

| ID | title | score |
|---|---|---|
| 6619440 | "A la Burstall" Intermittent Assertions Induction Principles for Proving Inevitable Ability Properties of Programs. | 1.0 |
| 6620955 | "Almost stable" matchings in the Roommates problem with bounded preference lists. | 1.0 |
| 6611966 | "Postéditorial": Les Universités d'Aix-Marseille sur les routes de la soie. | 1.0 |

### 2. Louvain Algorithm

The Louvain method is an algorithm to detect communities in large networks. We use the Louvain to create a community of articles that contain same keywords.

```
myCommunity = """CALL gds.graph.project(
    'myCommunity',
    'ARTICLE',
    {
        CONTAINS:{
            orientation: 'UNDIRECTED'
        }
    },
    {
        nodeProperties: 'year:int'
        //relationshipProperties: ''
    }
);"""

graph.run(myCommunity)
```

| nodeProjection | relationshipProjection | graphName | nodeCount | relationshipCount | projectMillis |
|---|---|---|---|---|---|
| {ARTICLE: {label: 'ARTICLE', properties: {`year:int`: {defaultValue: null, property: 'year:int'}}}} | {CONTAINS: {orientation: 'UNDIRECTED', indexInverse: false, aggregation: 'DEFAULT', type: 'CONTAINS', properties: {}}} | myCommunity | 10000 | 0 | 156 |

```
## Call a previously made algorithm method using gds stream

louvain = """CALL gds.louvain.stream('myCommunity',{ includeIntermediateCommunities: true })
YIELD nodeId, communityId, intermediateCommunityIds
WITH gds.util.asNode(nodeId).`title:string` AS title,
     communityId,
     intermediateCommunityIds,
     gds.util.asNode(nodeId).sup AS type,
     gds.util.asNode(nodeId).`year:int` AS year
WHERE type = 'C' and year = 2019
RETURN communityId, title;"""

graph.run(louvain)
```

| communityId | title |
|---|---|
| 21 | Simulating a trust-based service recommender system for decentralised user modelling environment. |
| 24 | EDARD: efficient data access based on rumour dissemination in wireless sensor networks. |
| 26 | Computationally perfect compartmented secret sharing schemes based on MDS codes. |

### 3. Triangle Algorithm

A triangle is a set of three nodes where each node has a relationship to the other two. It detects community of articles that are cohesive. In our case, we obtain articles that have CITE_HAS_CITATION edges in the same conference.

```
# Call a previously made algorithm method using gds stream
triangle = """CALL gds.triangleCount.stream('myCommunity')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).articleid AS name
ORDER BY triangleCount DESC;"""

graph.run(triangle)
```

| name |
|---|
| 4038748 |
| 4039839 |
| 4041135 |

**References:**


- DBLP data (2023). Accessed from (https://dblp.uni-trier.de/xml/) at 28th Feb 2023.
- Dblp-to-csv. Accessed from github (https://github.com/ThomHurks/dblp-to-csv) at 28th Feb 2023.
- Neo4j Manual. Accessed on https://neo4j.com/docs/graph-data-science/
- Property graph SDM. https://learnsql2.fib.upc.edu/moodle/
- Creating a Recommender System with a Graph DB (Neo4j) (2019). Access from github https://iprapas.github.io/posts/neo4j-recommender/