LAB 1: PROPERTY GRAPHS

Mariana Mayorga Llano
Sayyor Yusupov

A. Modeling, Loading, Evolving (1st Week)

A.1 Modeling

The **Visual Representation of our Graph** is stored in file “PartA.1_graph_LlanoYusupov.png”. The data is highlighted in yellow, while the metadata is not highlighted. Nodes are colored red, relationship types - blue, the attributes in the form of “property_name**:** value” and of green color.
The application for running the queries below is stored in “PartA.2._LlanoYusupov.py”.
We listed our design decisions and their justifications as follows.

Decision 1: Edition and Conference as Nodes connected by relation ‘contains’
We added a node Edition that connects to Conference, because one conference can have multiple editions with each of them having different values for attributes (e.g. holding date, city, id). We also considered omitting this node and merging edition and conference but then editions of the same conference would be connected together and therefore it would be more computationally intensive to find all Editions of a Conference (e.g. to find editions of conference “IT Innovation”, we would have to scan through all conferences that match the name “IT innovation”).

Decision 2: Corresponding author as an attribute in relation ‘author of’
The corresponding author is a boolean attribute to the relation ‘author_of’ because it is a descriptive characteristic of every pair of an author and a paper written by him/her. This characteristic itself is dependent on the relationship, so it does not have relations with objects and does not require an additional edge since it will always be connected in the same way as the relation ‘author_of’.

Decision 3: Review as a relation and ‘assigner’ as an attribute
We considered adding Review as a node and connecting it to an additional node ‘Head Chief’ that would represent the conference chair or the journal editor (depending on the scenario) under the relation ‘assigns_to’, however, we discarded this idea and decided to represent the review object as an edge between a paper and an author (who would be different from the ones who wrote the paper itself) because the review is an object derived from the interaction of the mentioned nodes and there is no need to connect any additional relation to the review in this point. To represent the assigner of the reviewers we included the name of this person as the attribute ‘assigner’ to the relation ‘review’ under the consideration that the assigner can change for each paper.

Decision 4: Keyword as a node between Topic node and Paper node
We considered having the keywords as attributes of the relation between a paper and a topic, however, taking into consideration that a keyword can belong to multiple topics we decided instead to make it a node to promote reusability. This way, we consider that a paper has one or more keywords and a keyword refers to one or more topics.

Decision 5: Not merging ‘Volume’ with ‘Edition’ nor ‘Conference’ with ‘Journal’

Since there is a similar behavior and relations between 1) volumes and editions and 2) Conferences and Journals, we considered merging them and specifying each specific case with an attribute 'type' that would indicate if it refers to an edition or a volume. However, we discarded this idea and decided to keep both as separate paths since the attributes can be different for each case (e.g. a conference has a city and a timeframe while a volume does not). Since graphs are schemaless, we are aware that it would still be possible to merge the objects given the mentioned conditions, however, in matters of performance, we consider it is better for these objects to be labeled as different types of nodes to facilitate queries belonging exclusively to one of these options.

A.2 Instantiating/Loading

We based our data on the dataset of Engineering Publications of Brigham Young University [1]. The original dataset did not have information on whether an author is corresponding so we set it to the author with a minimum index for each paper. We also did not have information on citations, so we assigned from 1 to 20 citations to each paper randomly with the constraint that a paper cannot cite itself. The same goes to Reviewers, we randomly assigned 3 authors to each Paper, excluding the authors of the paper. Also column "Source" contained information on both Journals and Conferences but they had a high selectivity so we only left several Journals and Conferences from the dataset and assigned them randomly for all the remaining papers. Column "Volume" also had a high selectivity, so there would usually be no papers sharing the same "Volume". Therefore, we generated either 5 Volumes or 5 Editions correspondingly for each Journal or Conference. Finally, Keywords were randomly grouped to Topics.

A.3 Evolving the graph

The application for running the queries below is stored in "PartA.3._LlanoYusupov.py".
The **Visual Representation of our Graph** is stored in file "PartA.3_graph_LlanoYusupov.png" with the same format as for the graph from A1. The changes are marked yellow.

Decision 1: Affiliation as Node
We could either write the affiliation as an attribute of each node "Author" or create a new node "Affiliation" that connects to each "Author" node that is affiliated to it. Since we wanted to have many authors being affiliated to one organization, we decided to implement the second approach. With the first approach implemented, there would have been much redundancy and maintainability would have been worse.

Decision 2: 'Review' kept as an Edge
We considered changing 'Review' from an edge to a node but realized that the only difference in its functionality is that it has the new attribute suggesting the decision. Since once a review is made there are no changes so there is no need to consider updating scenarios. We defined the decision as a boolean property "approved".

Decision 3: Relationship between 'Paper' and 'Edition' / 'Volume'
We also modified the relation 'contains' between 'Paper' and 'Volume'/'Edition', to indicate that a Paper 'applies_to' a Volume/Edition and included a boolean attribute "approved" in the relationship to indicate if the Paper was accepted or not. The attribute 'num_reviewers' is also included in the relationship to indicate how many reviewers have to approve the paper in order for it to be published.

B. Querying

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 class, collect({paper:p2.title, score:numCitations}) as citations
return class,citations[0..3] as top3 order by class
```

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:Paper)-[:written_by]->(a:Author)
with a, c, count(distinct e) as numEditions where numEditions >=4
return a.name, c.name, numEditions order by numEditions desc
```

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

```
Match (j:Journal)--(v:Volume)--(p:Paper)<-[c:cites]-(p2:Paper)
WITH j, p, toInteger(p2.year) as citationYear, toInteger(p.year) as publishingYear, count(c) as
citations
where publishingYear < citationYear AND citationYear<=publishingYear+2
with j, citationYear, collect(citations) as citationList, count(distinct p) as nPublished
with j, citationYear, nPublished, reduce(totalCitations = 0, item in citationList | totalCitations +
item) AS totalCitations
return j.name, citationYear, totalCitations/nPublished as ImpactFactor
order by j.name, citationYear
```

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

```
match (a:Author)<-[:written_by]-(p2:Paper)<--(p:Paper)
with a, p2, count(p) as paperCitations order by paperCitations desc
with a, count(p2) as papersPerAuthor, collect(paperCitations) as p2Citations
with a, papersPerAuthor, [temp in range(0,papersPerAuthor-1) where p2Citations[temp] >=
papersPerAuthor ] as h
return a.name, papersPerAuthor, last(h)+1 as indexH order by papersPerAuthor desc
```

C. Recommender

Step 1

We decided to put all keywords of this community into one topic. Initially, the keywords were randomly assigned together. To put them into one topic, we first found all keywords in our dataset that contain any of the keywords above. We obtained the following keywords:

```
['Optical data processing', 'data processing', 'data modeling', 'data storage and data querying ', 'big data',
'data management', 'data storage and data querying', 'indexing']
```

Then we removed the initial topics of these keywords:

```
match path=(k:Keyword)-[e:part_of]-(t:Topic) where k.word in [
'Optical data processing','data processing','data modeling','data storage and data querying ',
'big data','data management','data storage and data querying','indexing']
delete e
```

And assigned a single topic “Database Systems” to all of them:

```
match (k:Keyword) where k.word in [
'Optical data processing','data processing','data modeling','data storage and data querying ',
'big data','data management','data storage and data querying','indexing']
merge (t:Topic {name: "Databases"}) with k, t
merge (k)-[:part_of]->(t);
```

Step 2

None of the journals or conferences had 90% of papers relating to the “Databases” topic, so we put more DB keywords to one of the journals and one of the conferences.

```
match (j:Journal {name:"IEEE Robotics and Automation Letters"})--()--(p:Paper)
match (k:Keyword {word:"big data"})
merge (p)-[:relates_to]->(k);
match (c:Conference {name:"ASEE Annual Conference and Exposition, Conference
Proceedings"})--()--(p:Paper)
match (k:Keyword {word:"indexing"})
merge (p)-[:relates_to]->(k)
```

Now, we can confirm that 100% of papers for 1 journal and 1 conference above are related to the “Databases” community:

```
match path=(j:Journal)--()--(p:Paper)
with j, count(distinct p) as nAllPapers
match path=(j:Journal)--()--(p1:Paper)--()--(t:Topic {name:'Databases'})
with j, nAllPapers, count(distinct p1) as nDBPapers
with j, nDBPapers/nAllPapers as percDB where percDB >=0.9
return j.name, percDB;
```

We applied a similar query to Conferences, it can be seen in the file “PartC_LlanoYusupov.py”.

Step 3

Here, for each journal and conference of the community, we are getting top-3 papers by the number of citations from papers within the community. Since we have only around 100 papers for each of the journals and conferences in our dataset, we had less papers satisfying this query and so we reduced it from top-100 to top-3.

```
with ["IEEE Robotics and Automation Letters"] as DBjs, ["ASEE Annual Conference and Exposition,
Conference Proceedings"] as DBcs
match (j1:Journal)--()--(p:Paper)<-[:cites]-(p1:Paper)--()--(j2:Journal)
match (p:Paper)<-[:cites]-(p2:Paper)--()--(c:Conference)
where j1.name in DBjs and j2.name in DBjs and c.name in DBcs
with p,j1, count(distinct p2)+count(distinct p1) as countDBCites order by countDBCites descending
with j1,collect({paper:p.title, NDBcites: countDBCites}) as cites
```

```
return j1.name as journal, cites[0..3] as top3
```

We applied a similar query to Conferences, it can be seen in the file “PartC_LlanoYusupov.py”.

Step 4
Initially, we did not have any gurus in our dataset for the “Databases” community. Therefore, we set several authors as co-authors of one more paper:

```
match (p:Paper)
match (a:Author) where a.name in ["Bowman K.E.", "Jensen C.G.”] and
p.title = "Optimization and implementation of synthetic basis feature descriptor on FPGA"
merge (p)-[:written_by {corresponding_author:True}]->(a);
match (p:Paper)
match (a:Author) where a.name in [" Crane N.", "Weidman J.E.", " Farnsworth C.B."] and
p.title = "Cross platform usability: Evaluating computing tasks performed on multiple platforms"
merge (p)-[:written_by {corresponding_author:True}]->(a);
```

This query now returns 2 gurus for the journals:

```
with ["IEEE Robotics and Automation Letters"] as DBjs, ["ASEE Annual Conference and Exposition,
Conference Proceedings"] as DBcs
match (j1:Journal)--()--(p:Paper)<-[:cites]-(p1:Paper)--()--(j2:Journal)
match (p:Paper)<-[:cites]-(p2:Paper)--()--(c:Conference)
where j1.name in DBjs and j2.name in DBjs and c.name in DBcs
with p,j1, count(distinct p2)+count(distinct p1) as countDBCites order by countDBCites descending
with j1,collect(p.title) as cites
with cites[0..3] as top3
match (p:Paper)--(a:Author) where p.title in top3
with a, count(p) as nP order by nP descending where nP>=2
return a.name, nP
```

We applied a similar query to Conferences, it can be seen in the file “PartC_LlanoYusupov.py”.
These functions can also be applied for the cases when there are multiple journals and conferences in a community. Only the arrays in the “where” clauses should be updated.

D. Graph Algorithm

1. PageRank

Measures the influence of each node within the graph, based on the number of incoming relationships and the influence of the corresponding source nodes.

Scenario
For this example we want to know which are the most influential papers in the graph, considering the number of papers citing it as well as the influence of these source nodes. This way we will not only consider the number of citations per paper but also the relevance of the papers that cited this item.

Graph Projection

This exercise only requires the ‘Paper’ nodes and the ‘cites’ relationship connecting them.

```
CALL gds.graph.project('pageRankGraph','Paper','cites')
YIELD graphName AS graph, nodeProjection, relationshipProjection
```

Function call

```
CALL gds.pageRank.stream('pageRankGraph', {maxIterations: 50,dampingFactor: 0.5})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).title AS title, score ORDER BY score DESC, score DESC
```

Parameter selection

Stream mode - To show 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 chose an intermediate number of the accepted range (0-1) to avoid sinks and spider traps that occur when the value is too high, but also not as low to guarantee that the algorithm does converge and values will not be that much similar.

Results

Here are top-5 papers by Page Rank:

	Title	**Score**
1	"Investigating influences on first-year engineering students' views of ethics and social responsibility"	1.78700597554
2	"Solid-state membranes formed on natural menisci"	1.74813704160
3	"Assessing the environmental impact and payback of carbon nanotube supported CO2 capture technologies using LCA methodology"	1.69845487755
4	"Changes in perceptions of ethical climate among undergraduate engineering students"	1.69603275808
5	"Antenna Loss and Receiving Efficiency for Mutually Coupled Arrays"	1.64741185642

2. Betweenness Centrality

Often used to find nodes that serve as a bridge from one part of a graph to another.

Scenario

Considering that the citations mentioned in a paper should be relevant for the subject of the paper itself, we want to find the paper with the most unusual subjects of study based on the use of sources of information that are not commonly seen together.

As an example, imagine we have a paper that benchmarks databases used for mobile apps. In this case we could find citations to papers related to databases and others related to software development. This example, in a real life scenario should have a low score here since databases and software development is typically seen together. On the other hand, if we had a paper that studies if there is any correlation between people's perspective of art given their religious beliefs by using clustering algorithms, we could find citations from papers related to theology, arts and data science, which are three subjects not usually seen together. Therefore, this paper would have a high score in this scenario.

Graph Projection

```
CALL gds.graph.project('betweennessGraph', 'Paper', 'cites')
YIELD graphName AS graph, nodeProjection, relationshipProjection
```

Function call

```
CALL gds.betweenness.stream('betweennessGraph', {samplingSize: 2000, samplingSeed: 891})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).title AS title, score ORDER BY score DESC
```

Parameter selection

Stream mode - To show the score for each node.

samplingSize - A larger sample size will generally lead to more accurate estimates, that why we chose a high value for this setting.

samplingSeed - Setting the samplingSize to the node count of the graph produces exact results, which is what we did in our graph that has 891 nodes.

Results

We are showing top-5 results below.

Title	**Score**
Modeling effects of annealing on coal char reactivity to O2 and CO2 based on preparation conditions	46.521862176097
Strategies for Removing Common Mode Failures from TMR Designs Deployed on SRAM FPGAs	36.5251735453822
Haptic Shape-Based Management of Robot Teams in Cordon and Patrol	35.1809168455473
Municipal wastewater treatment by semi-continuous and membrane algal-bacterial photo-bioreactors	33.084813808113

3. Node Similarity

Node Similarity Algorithm is used to measure how similar a pair of nodes are based on the nodes they are connected to.

Scenario

We decided to apply this algorithm to find for each author top-5 authors with whom they usually work on writing papers. In other words, the higher the similarity index between, the co-authorship of more papers they share in common and it's more likely that they will work together in the future again.

Graph Projection

```
CALL gds.graph.create.cypher( 'NodeSimilarity',
'MATCH (n) where n:Author or n:Paper RETURN id(n) AS id',
'MATCH (n:Author)<-[w:written_by]-(m:Paper) RETURN id(n) AS source, id(m) AS target')
yield graphName as graph, nodeQuery, nodeCount as nodes, relationshipCount as rels
```

Function Call

```
CALL gds.nodeSimilarity.stream('NodeSimilarity', {topK: 5, similarityCutoff: 0.5})
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).name AS Author1, gds.util.asNode(node2).name AS Author2,
similarity ORDER BY similarity descending, Author1, Author2
```

Parameter Selection

We set "topK" to 5 since we wanted to get top-5 co-authors. We are interested in very similar co-authors so we set "similarityCutoff" to 0.5 to ignore any co-authors with lower similarity. Thus, if an author had a top-3rd co-author with a similarity of 0.4, s/he will not be included.

Results

Here are most similar authors for author "Aarsnes U.J.":

Author1	Author2	similarity
" Aarsnes U.J."	" Behounek M."	1.0
" Aarsnes U.J."	" Cayeux E."	1.0
" Aarsnes U.J."	" Detournay E."	1.0
" Aarsnes U.J."	" Gandikota R."	1.0
" Aarsnes U.J."	" Harmer R."	1.0

These are the authors with whom "Aarsnes U.J." wrote all of his papers.

References

1. David. (2021). BYU Engineering Publications in Scopus 2017-21. Retrieved March 14, 2023 from https://www.kaggle.com/datasets/dpixton/byu-engineering-publications-in-scopus-201721.