Open Data - Property Graph Lab

Prat Sicart, Joan
UPC Barcelona Tech
joan.prat.sicart@est.fib.upc.edu

Rubio Cuervo, Damian
UPC Barcelona Tech
damian.rubio@est.fib.upc.edu

Thursday 14th March, 2019

This document supposes the deliverable of the first assignment of the lab practices of the subject of Open Data corresponding to the Spring semester of the MIRI master degree during the course 2018/19. This deliverable is divided according to the statement provided by the lecturers and should be accompanied by the corresponding resources containing the scripts required to reproduce the output of this work.

A.1: Modelling

1. Create a visual representation of the graph you would create (in terms of nodes and edges). Use different colours to distinguish data from metadata in your graph.

In first place and analyzing the provided statement we have decided to model the data as follows. This diagram focuses on the key components of the descriptions and has been used as a first approximation to the problem. The modelled graph would look like:

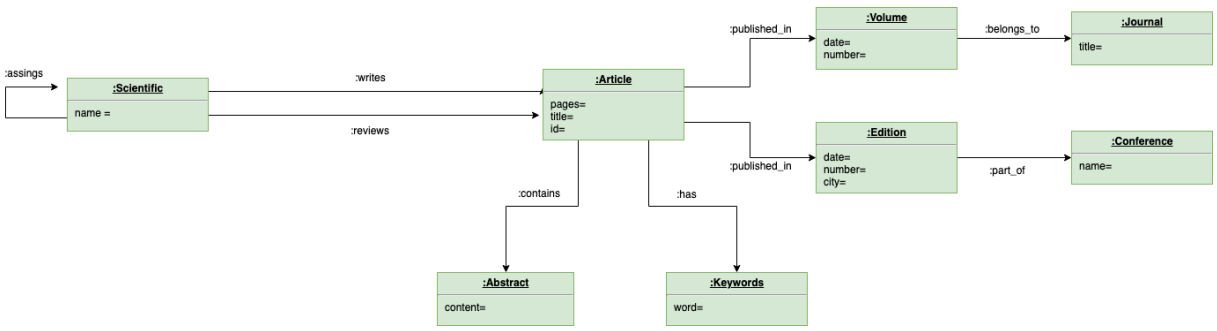

Figure 1: Graph Metadata Model

In order to provide an example of how data would be modelled in the database we have also constructed the following diagram that contains an example of what could be potential data contained in our system. An example of the data graph would look like:

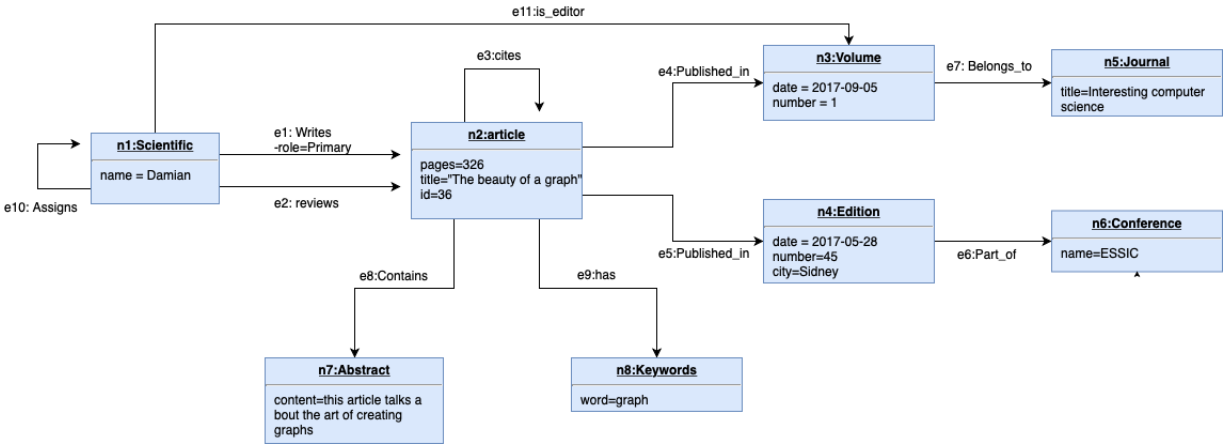

Figure 2: Graph Data Model

2. Also, justify your design decisions. Besides maintenance or reusability issues, also consider the performance of queries in Part B when creating your solution.

To design our graph we've procured to minimize the maintenance and at the same time improve the performance of the proposed queries. Therefore, in order to do so, we have avoided to repeat the data in the different nodes and edges to ease the maintenance of our graph database. Moreover, in function of the queries and the size of the information we've decided to extract some information into different nodes in the cases that we considered necessary, for instance, because of the size of the abstract of the article we've decided to put it apart to avoid to load it each time we query for an article. Furthermore, we've seen that many queries are related to the edition and that's why we've decided to put it as node instead of an attribute of the node conference for example. By this mean, the process of querying it speeds up, and we improve the performance of the queries in different cases, such as when we query for the editions of a conference (it's not necessary to look for all the editions, just to traverse the incoming edges of the conference).

A.2: Instantiating/Loading

1. Define the Cypher expressions to create and instantiate the solution created for the previous section.

To load our database we have decided to part from the data that can be extracted from DBLP [1] by means of the XML format. Then we have processed it with a script able to convert it from XML to CSV format so we were able to load it on the system. While we were doing the process we realized that the volume of data was to huge for the point of the task so we processed it more, splitting it and creating some reduced set of mock data. This set of data is sufficient for the development of the exercises proposed in the statement of the task. Anyway, as the lecturers suggested, we have decided to avoid loading data that were not used during the other points of the deliverable, so contents as the abstracts have not been instantiated.

In order to instantiate the data in the system we have taken advantage of the options provided by the tool. First we have decided to load the data directly from the CSV files. To speed up the process we have used the periodic commit option to perform only commits when we have reached the loading of five hundred instances. We have also decided to create indexes in some of the relevant attributes so that

[1] The Computer Science Bibliography

we can improve the performance of the queries of the upcoming points.

The script that loads our database can be found as an attachment to this document under the name: `PartA.2_PratRubio.cyp`. As it can be seen the main nodes and attributes are loaded directly from CSV files (Scientific, Article, Conference, Journal and writes) while other data (Edition, Volume, Keyword, published_in, cites and reviews) has been generated by means of Cypher code.

A.3 Evolving the graph

1. Modify the graph model created in A.1 and highlight the changes introduced. Justify your decision.

In order to store information about the reviews, such as the content, and the suggested decisions, we've decided to create a node with the label review with the attributes content and suggested decisions. Besides, it's also necessary to store in which kind of organization the author is affiliated, in order to do it, we thought about two different ways, the first one, consists on creating a node with the label organization and the attribute "type"in which we can specify weather it is an university or a company; while in the second one, the node organization does not have attributes but has a possible relation with two different nodes, one to which is linked when it is an university instance and other one for when it is a company. This second approach can suppose an improvement in the speed when querying based on the kind of organization an author is affiliated to. However in the entire project there is not any query related to this, and therefore, we have decided to choose the first approach which is more compact.

Another important modification is to include the number of reviewers that a conference or journal requires. This will be simply made by means of the addition of a new attribute in each node 'Conference' or 'Journal' indicating how many reviewers does it require. So finally, with all the modifications the graph has the following shape:

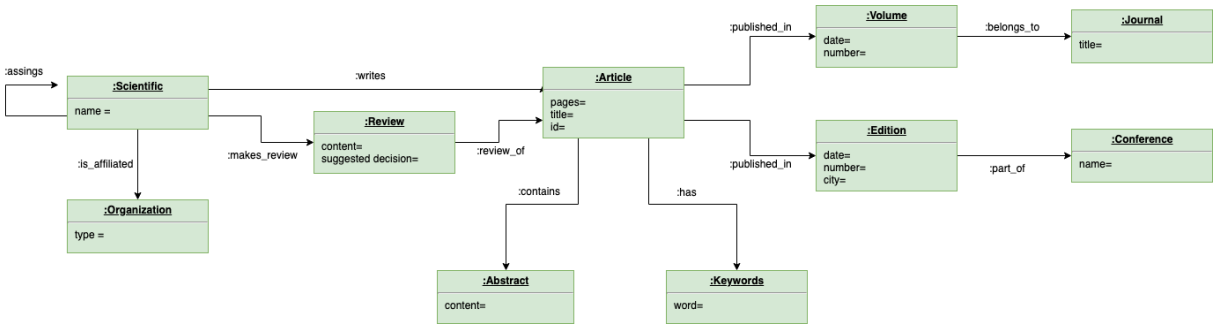

Figure 3: Evolved Graph Metadata Model

2. Write the necessary Cypher queries to transform your Neo4j graph (including data and metadata) into an updated graph aligned with the evolved model proposed in the previous item.

In our case, we have only to reshape the part corresponding to the model of the reviews. For this purpose we need to match all the current review relationships and substitute them by the proper nodes and edges. The script that performs such actions can be found as an attachment to this document under the name: `PartA.3.2_PratRubio.cyp`.

3. Write additional Cypher queries for instantiating the new concepts of your graph.

For this point we will have to instantiate the new node 'Organization' and to create the edge indicating to which organization the scientific is affiliated. This has been made according to what has been exposed in part one of this section.

In addition, we have had to initialize the attribute 'required_reviewers' for each of the 'Conference' or 'Journal' nodes. This attribute, will be instantiated with random values up to three in each case.

The script that performs such actions can be found as an attachment to this document under the name: `PartA.3.3_PratRubio.cyp`.

B Querying

In this section we have been required to design a set of queries that provide us the expected output optimizing as much as possible the disk accesses that those queries do. The queries are attached in the document `PartB_PratRubio.cyp`. Anyway, the code of each of the queries will be exposed and analyzed here.

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

The Cypher code to achieve that operation would be:

```
// H-INDEX
MATCH (s:Scientific)-[:writes]->(a1:Article)<-[:cites]-(a2:Article)
WITH s, a1, COUNT(a2) AS citations
ORDER BY s, citations DESC
WITH s, COLLECT(citations) AS list_citations
WITH s,
[i IN RANGE(0, SIZE(list_citations) - 1) WHERE list_citations[i] >= i+1] AS valids
RETURN s AS author, SIZE(valids) AS hIndex
```

Table 1: H-Index Query

This query starts by collecting all the citations that each of the articles of a scientific has and then takes advantage of the list comprehension feature to create a list of those articles where the number of citations is greater than its position when they are ordered. Counting the number of elements on that list we can obtain the H-Index of each author. This query could be extended by means of an `ORDER BY` instruction when needed.

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

The Cypher code to achieve that operation would be:

```
// MOST-CITATED
MATCH (a2:Article)-[:cites]->(a1:Article)-[:published_in]->
(e:Edition)-[:part_of]->(c:Conference)
WITH c, a1, COUNT(a2) AS citations
ORDER BY c, citations DESC
RETURN c, COLLECT(a1)[..3]
ORDER BY c.name
```

Table 2: Most-citated Query

This query starts by checking how many citations has each article of each conference taking advantage of the pattern matching and the `COUNT()` function of Cypher. Then, it sorts the articles by descending order of citations and returns the three first articles of each conference. The output of this query has been ordered by means of the conference name so it improves readability.

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

The Cypher code to achieve that operation would be:

```
// COMMUNITY
MATCH (s:Scientific)-[:writes]->(a:Article)-[:published_in]->
(e:Edition)-[:part_of]->(c:Conference)
WITH s, c, SIZE(COLLECT(DISTINCT e)) AS editions_participated
WHERE editions_participated >= 4
RETURN c, COLLECT(s)
```

Table 3: Community Query

Once again we use pattern matching to find all the conferences in which an author has participated. In this case we use the function `DISTINCT()` to avoid counting the cases on which an author takes part in two papers of the same edition. Then we filter by means of a `WHERE` statement those authors that have participated at least four times on the conference and we return them. The output of the query would be each of the conferences where we have communities together with the list of members of the community, aka the authors participating in at least four different editions.

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

The Cypher code to achieve that operation would be:

```
// IMPACT FACTOR
MATCH (v:Volume)-[:belongs_to]->(j:Journal)
WITH j, COLLECT(DISTINCT(v.year)) as years
UNWIND years AS y
MATCH (a:Article)-[:published_in]->(v1:Volume)-[:belongs_to]->(j)
WHERE v1.year = y-1 OR v1.year=y-2
WITH y, j, COLLECT(a) AS publications
MATCH (citer:Article)-[:cites]->(a1:Article)
MATCH (citer)-[:published_in]->(n)
WHERE n.year = y AND a1 in publications
WITH y, j, count(citer) as cites, size(publications) as pubs
RETURN j, y, (cites * 1.0 / pubs) AS ImpactFactor
ORDER BY j, y;
```

Table 4: Impact Factor Query

This is the most complex query performed till the moment and it is divided in three steps. First of all we select all the years for which we want to know the impact factor (since we have decided to provided the impact factor for every year of every journal). Then we obtain the publications made in the two previous years by the same journal, and finally we check the number of citations that those papers receive made by papers published in that specific year. To obtain the output we divide those amounts taking into

account the floats and we rename it as the impact factor of the journal for the specific year.

C Graph algorithms

In this section we have been asked to explode the possibilities of graph theory by means of the usage of the built-in algorithms that Neo4j provides. We have selected, analyzed and executed two of those algorithms and then we provide a conclusion on the results obtained. Even though the queries can be found in the following point of this report, the code has been attached for executing purposes in the `PartC_PratRubio.cyp` file.

1. Choose two algorithms and write the Cypher queries triggering them correctly.

There are few algorithms that can extract useful data from our database. One of them is the betweenness centrality algorithm, which detects the amount of influence a node has over the flow of information in a graph. This algorithm can be very powerful in our graph to look for example which articles have a bigger impact by seeking the citation relationships, and in fact, that is exactly what we have done in the code below, retrieve the articles ordered by the impact that have.

```
// Betweenness centrality algorithm
CALL algo.betweenness.stream('Article','cites',{direction:'out'})
YIELD nodeId, centrality
MATCH (a:Article) WHERE id(a) = nodeId
RETURN a.title AS article,centrality
ORDER BY centrality DESC;
```

Table 5: The betweenness centrality algorithm

The other algorithm we have used is the strongly connected components algorithm, because it allows to find sets of connected nodes in a directed graph where each node is reachable in both directions from any other node in the same set, and that allows us to get a main idea of how our graph is structured, and more important, which elements have stronger relationships between them. For instance, in the case of the authors we can appreciate how much related they are by the sets of connected articles in function of the citations that they have between them. Next, we show the code for triggering the strongly connected components algorithm:

```
//The strongly connected components algorithm
CALL algo.scc.stream(
'MATCH (s:Scientific) RETURN id(s) as id',
'MATCH (s1:Scientific)-[:writes]->(:Article)-[:cites]->(:Article)<-[:writes]-
(s2:Scientific) RETURN id(s1) as source,id(s2) as target',
{write:true,graph:'cypher'})
YIELD nodeId, partition
WITH partition, COLLECT(nodeId) AS participants
RETURN partition, SIZE(participants) AS n_participants, participants
ORDER BY n_participants DESC
```

Table 6: The strongly connected components algorithm

2. For each algorithm used in the previous item, give a rationale of the result obtained: i.e., what data are you obtaining? provide an interpretation from the domain point of view. Your answer must show that you understand the graph algorithm chosen and its application on this domain.

From the code displayed in the "table 5: The betweenness centrality algorithm" The results we obtained are the following:

"article"	"centrality"
"Extended multi bottom-up tree transducers."	704.5071661031003
"The Clean Termination of Iterative Programs."	637.1345390342055
"Asymptotic Expansions of the Mergesort Recurrences."	621.4657888104309
"A Class of Linearly Parsable Graph Grammars."	617.5119982979353
"The Complexity of Reachability in Distributed Communicating Processes."	617.2758008607681

Figure 4: Results for the betweenness centrality algorithm

As it can be appreciated, the algorithm retrieves the nodes ordered showing first those that have a higher centrality. In the case of our model this can be translated to those articles which have a higher influence or impact in the graph flow by having a higher number of citations, consequently, the article " Extended multi bottom-up tree transducers" is the article with more impact in our graph according to the number of citations citations.

Regarding the other algorithm, the results from the figure "Results for the strongly connected components algorithm" are the following:

"partition"	"n_participants"	"participants"
0	800	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50, 424,425,426,427,428,429,430,431,432,433,434,435,436,437,438,439,440,441,442,443,444,445,446,447,448,449,450,451,452,453,454,455,456,457,458,459,
270	1	[270]
365	1	[365]
366	1	[366]
418	1	[418]
618	1	[618]
619	1	[619]
777	1	[777]

Figure 5: Results for the strongly connected components algorithm

From the data displayed above, we can deduce according to the strongly connected components algorithm, that in function of the citations between the scientific there is a large community strongly related between them, and just a few authors that keep off from this community, most probably because they were just introduced as new writers, and don't have citations yet. This method, allows us to see that the community of authors in our database seems to be pretty endogamous and strongly related, and most of them can be found by following the citations between one and other. Just a couple of authors seem to be independent from that main community and that is represented by the fact that they are identified as from other partition.

D Recommender

In this task we are asked to create a simple recommender. We are going to use some property graph tools to build a reviewer recommender for editors and chairs. All the code developed during this point can be found on the attached document `PartD_PratRubio.cyp`.

1. For each stage, provide a Cypher statement finding the relevant data of each step and

asserting the inferred knowledge into the graph.

First of all we are required to create research communities that will be defined by the topic they talk about and some keywords they use. The following two queries allow us to create a community indexed by its name and to create the relations between that community and the keywords it uses.

```
// COMMUNITY CREATION
CREATE INDEX ON :Community(name);
```

Table 7: Community Creation Query

```
// COMMUNITY DEFINITION
CREATE (c:Community {name:'database'})
WITH c, ['data management', 'indexing', 'data modeling', 'big data',
    'data processing', 'data storage', 'data querying'] AS community_keys
UNWIND community_keys as key
MERGE (k:Keyword {topic:key})
MERGE (c)-[:related_to]->(k);
```

Table 8: Community Definition Query

Then, we need to find those conferences or journals related to that community, considering that if 90% of the papers published on them contain at least one of the keywords of the community there will be a relation between the conference/journal and the community. The query that allows us to do that would be:

```
// FIND RELATIONS TO COMMUNITY IN JOURNALS AND CONFERENCES
MATCH (c:Community)-[:related_to]->(k:Keyword)
WITH c, COLLECT(k) AS keywords
MATCH (a:Article)-[:published_in]->(v)-[:belongs_to|:part_of]->(j)
WHERE (v:Volume OR v:Edition) AND (j:Journal OR j:Conference)
WITH c, j, keywords, COLLECT(a) AS publications
UNWIND publications AS a
MATCH (a)-[:has]->(key:Keyword)
WHERE key IN keywords
WITH c, j, publications, COLLECT(a) AS using_keywords
WHERE (SIZE(using_keywords) * 100.0 / SIZE(publications)) >= 90.0
CREATE (j)-[:forms]->(c);
```

Table 9: Community Formation Query

This query takes advantage of the conditional statements that can be made in Neo4j to be able to find at the same time the relationships and labels on which we are interested (That due to our modeling are belongs_to/part_of and volume/edition) avoiding some intermediate results in that process. Then, we filter weather the paper of the conference/journal uses at least one of the keywords of the community and if they sum up to the 90% of the papers published in that conference/journal we create a new relationship between the element and the community.

In order to find top collaborations to the community we then process our data to find those one hundred papers with the biggest Page Rank coming from the citations of papers of the community that have been published in events (being that a conference or a journal) that form up the community. The code to perform such action would be:

```
// FIND TOP 100 PAPERS
MATCH (a:Article)-[:published_in]->(x)-[:belongs_to|:part_of]->
    (y)-[:forms]->(c:Community)
WHERE (x:Edition OR x:Volume) AND (y:Conference OR y:Journal)
WITH c, COLLECT(a) AS articles_from_community
CALL algo.pageRank.stream('articles_from_community',
    'cites', {iterations:20, dampingFactor:0.85})
YIELD nodeId, score
MATCH (art:Article)
WHERE id(art)=nodeId
WITH c, art, score
ORDER BY score DESC
WITH c, art LIMIT 100
CREATE (art)-[:is_top_from]->(c);
```

Table 10: Top Papers Identification Query

This query can be explained by dividing it in three parts. First of all we perform a pattern matching over the database with some filters to obtain all the articles that have been published in conferences or journals that form part of the community. Then, we use this collection to perform Page Rank over them considering the number of citations they have among other papers of the same community and finally we order the result by its Page Rank score and create a relationship between the 100 first elements and the community were we indicate that it is an article that is top in the community.

Finally, we want to detect what we consider to be a good or top reviewer. To perform such a task we consider that an author is a potential reviewer of the community if he has wrote at least one article among the top one hundred articles of the community. We are also going to consider that an author is a guru of the community if he has wrote at least two of the top one hundred papers of that community. The queries that help us to identify such authors are:

```
// FIND REVIEWERS
MATCH (s:Scientific)-[:writes]->(a:Article)-[:is_top_from]->(c:Community)
MERGE (s)-[:is_reviewer_of]->(c);
```

Table 11: Reviewers Identification Query

```
// FIND GURUS
MATCH (s:Scientific)-[:writes]->(a:Article)-[:is_top_from]->(c:Community)
WITH s, c, COLLECT(a) AS articles
WHERE SIZE(articles)>=2
MERGE (s)-[:is_guru_of]->(c);
```

Table 12: Gurus Identification Query

It must be said that we have taken a design decision where an author can be both a reviewer and a guru of a conference. This has not been a trivial decision since been the author a guru it could be implicitly inferred that he is also a reviewer, but we have considered that specifying both relationships at the same time eases future data extractions from the database and does not overload the model.

As a summary it can be said that executing all these steps we would have built a model that is able to recommend authors as reviewers fro a community considering their performance and relevance on topics related to that community. The resulting model after all the steps would be:

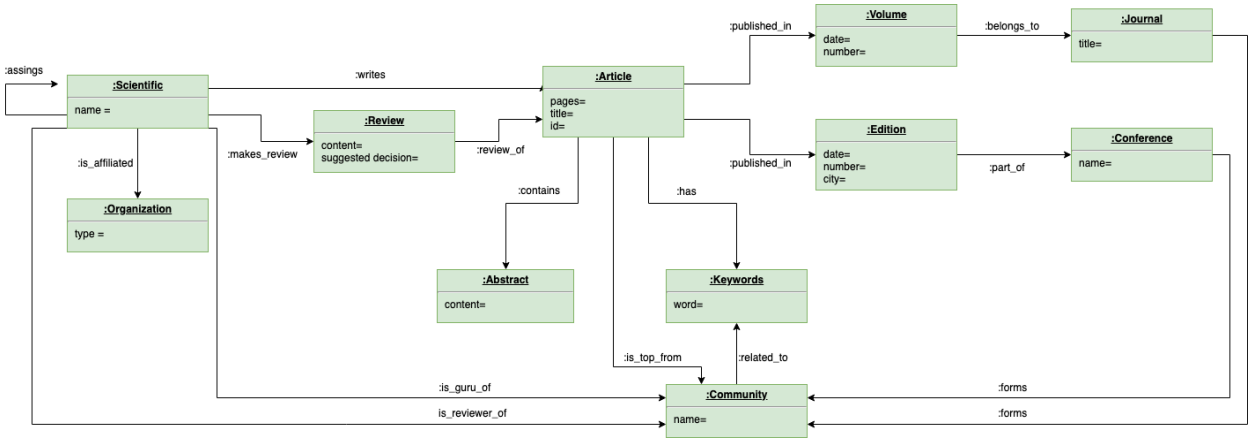

Figure 6: Graph Model with Communities