SDM Lab: Property Graph

Yi Wu
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
yi.wu@estudiantat.upc.edu

Yanjian Zhang
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
yanjian.zhang@estudiantat.upc.edu

1 Modeling, Loading, Evolving

1.1 Modeling

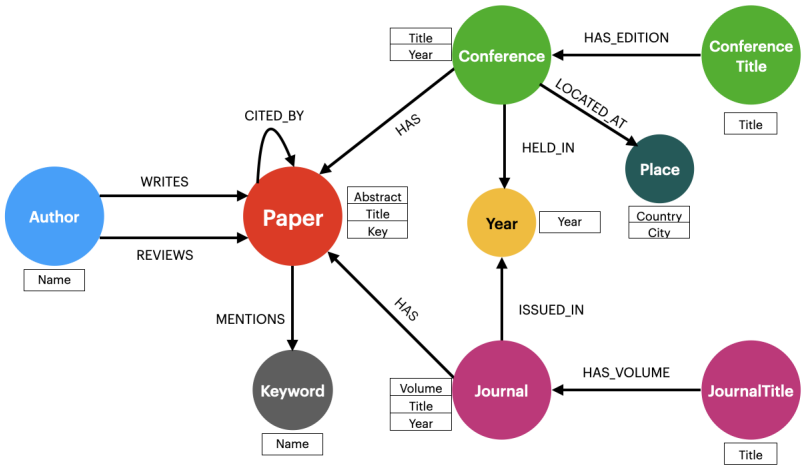

Figure 1: Model Schema

'A conference/workshop is organized in terms of editions. Each edition of a conference is held in a given city (venue) at a specific period of time of a given year'. Based on this information, we establish two nodes for conferences, one is 'ConferenceTitle', the other is 'Conference'. For an instance, a ConferenceTitle(title:IEEE) is held every year; a Conference(title:IEEE, year:2022) indicates the one that is held in 2022 specifically. They are linked with the relationship that a 'ConferenceTitle' HAS_EDITION of a 'Conference'. Similarly, node 'JournalTitle' HAS_VOLUME of node 'Journal'.

What's more, though redundancy, we add node 'Year' for both 'Conference' and 'Journal', that a 'Conference' is HELD_IN a 'Year' and a 'Journal' is ISSUED_IN a 'Year'. It is out of the consideration for effective aggregated/filtered query of certain years.

'A proceeding is a published record which includes all the papers presented in the conference/workshop'. Based on this piece of information, we found it not necessary to create a node for a proceeding. It can be covered by a one-to-many relationship HAS between conferences/workshops and papers.

What's more, a 'Paper' MENTIONS 'Keyword' which is a many-to-many relationship. And node 'Paper' is self-looped, represented by the relationship CITED_BY. A node 'Author' WRITES a 'Paper'. Also based on the information that 'Reviewers are scientists and therefore they are relevant authors (i.e., published many papers in relevant conferences or journals)',

the reviewers are treated as nodes 'Author' as well, only with the difference that they REVIEWS the 'Paper' instead of WRITES it.

Indexes are built for each node for quicker query performance. The instances data is not shown in the schema figure above, rather it is listed below.

Node	Instance
Author	"name": "E. F. Codd"
Paper	"abstract": "Est dolorem quiquia quisquam neque. Non eius velit...", "title": "MMB '99, Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, 10. GI/NTG-Fachtagung, 22.-24. September 1999, Trier, Kurzbeiträge und Toolbeschreibungen", "key": "tr/trier/MI99-17"
Keyword	"name": "Data Management"
Conference	"title": "GRAPHITE", "year": 2013
ConferenceTitle	"title": "GRAPHITE"
Place	"country": "Cyprus", "city": "March"
Journal	"volume": "TR-0263-08-94-165", "title": "GTE Laboratories Incorporated", "year": 1994
JournalTitle	"title": "GTE Laboratories Incorporated"
Year	"year": 1998

1.2 Instantiating/Loading

First, follow lab instructions, and convert DBLP data from XML to CSV, referring to this repo: ThomHurks/dblp-to-csv. See attached code A2_XMLToCSV.py.

Second, extract and preprocess the data from csv files, as shown in the file named A2_Extract_Preprocess.py, A2_Citation_Filter.py, and A2_Utils.py.

Preprocessing includes actions of drop_duplicates(), dropna(), trimming whitespace, spliting a set of keywords, etc.

Extracting data is implemented according to the schema design. Conference_papers.csv are unwrapped from proceeding.csv, with attributes ['key', 'title', 'conference', 'year']. Journal_papers.csv are generated in similar way from article.csv with ['key', 'title', 'journal', 'year', 'volume']. Conferences.csv is generated from proceeding.csv as well with attributes ['title', 'year']. Journals.csv is from article.csv with properties ['journal', 'volume', 'year']. Conf_venues.csv extract places where conferences take place with ['title', 'year', 'city', 'country']), etc.

1.3 Evolving the graph

There are two changes to modify the original schema.

First, we need to adopt reviewer's decisions (accepted or not) and their textual comments for both journals and conferences. The reviewing decisions will be added as the properties of the relationship REVIEWS between 'Author' and 'Paper'. Reviews are synthetic, using random authors generating Journal_reviews.csv and Conference_reviews.csv, with columns [key;reviewer;textual_description].Then we match the column 'key' with node 'Paper"s attribute 'key', and match the column 'reviewer' with node 'Author"s attribute 'name'.

Second, the authors shall have their affiliated organizations. Data source Authors_affiliation.csv with columns [author;affiliation] is also synthetic, randomly assigning the schools from School.csv to authors. The column 'author' is then mathced that of node 'Author' property name, and the column 'affiliation' is set up as new node named 'Oraganization'. A relationship IS_AFFILIATED_TO is built to connect 'Author' and 'Organization'.

The code can be found in the attached file A3_Evolve.py.

2 Querying

2.1 Top 3 most cited papers of each conference

```
MATCH (p:Paper)-[r:CITED_BY]->(p1:Paper)
WITH p,COUNT(r) AS CitedNum ORDER BY CitedNum DESC
MATCH (ct:ConferenceTitle)-->(:Conference)-->(p)
RETURN ct.title as ConferenceTitle,collect(p.title)[..3] as Top3Cited
```

Result

ConferenceTitle	Top3Cited
"FICS"	["Proceedings 8th Workshop on Fixed Points in Computer Science, FICS 2012, Tallinn, Estonia, 24th March 2012."]
...	...

2.2 Conference community:

```
MATCH
    (a:Author)-[:WRITES]->(:Paper)<--(:Conference)<-[r:HAS_Edition]-(ct:ConferenceTitle)
WITH DISTINCT a,ct, COUNT(r) AS cnt WHERE cnt>3
RETURN ct.title as ConferenceTitle, COLLECT(a) as AuthorCommunity
```

Result

ConferenceTitle	AuthorCommunity
"SWEE"	["Yoichi Hirai", "Daniel J. McDonald", "Saso Dzeroski", "Reem Al-Nanih"]
...	...

2.3 Impact factor

$$IF_y = \frac{Citation_y}{Publications_{y-1} + Publication_{y-2}}$$

Since we need to use the calculate the number of publication on different years. We want to see how many journals have been published and how many conferences have been held in those years:

```
MATCH (j:Journal)-->(y:Year) RETURN y.year,size(collect(j)) AS
    size,collect(j.title) ORDER BY size DESC, y.year DESC
```

y.year	size	collect(j.title)
1998	7	["IEEE Data Eng. Bull.", "SIGMOD Rec.", "IEEE Trans. Knowl. Data Eng.", "ACM Comput. Surv.","Commun. ACM", "ACM Trans. Database Syst.", "VLDB J."]
1994	6	["IEEE Data Eng. Bull.", "IEEE Trans. Knowl. Data Eng.", "ACM Comput. Surv.", "Commun. ACM","ACM Trans. Database Syst.", "VLDB J."]
...	...	...

We plan to calculate the impact factor in 1998 for "IEEE Data Eng. Bull.", "IEEE Trans. Knowl. Data Eng.", "Commun. ACM", "ACM Trans. Database Syst." and "VLDB J."

```
WITH 1998 AS target_year, ["IEEE Data Eng. Bull.", "IEEE Trans. Knowl. Data Eng.",
    "Commun. ACM", "ACM Trans. Database Syst.", "VLDB J."] AS  selected_j
MATCH (cj:Journal)-->(cp:Paper)<-[:CITED_BY]-(p:Paper)<--(j:Journal) WHERE j.title
    in selected_j AND j.year IN [target_year-1, target_year-2]
WITH j.title as j_title, count(p) as j_cite, target_year
MATCH (p:Paper)<--(j:Journal) WHERE j.title = j_title AND j.year = target_year
RETURN j.title, toFloat(j_cite)/count(p) as impact_factor, j_cite, count(p)
ORDER BY impact_factor DESC
```

The parameter target_year can be change into any year in addition to 1998. Result can be seen as follows:

j.title	impact_factor	j_cite	count(p)
"ACM Trans. Database Syst."	1.9166666666666667	23	12
"VLDB J."	1.5263157894736843	29	19
"Commun. ACM"	1.5	6	4
"IEEE Trans. Knowl. Data Eng."	0.8059701492537313	54	67
"IEEE Data Eng. Bull."	0.631578947368421	12	19

2.4 H-index

```
def h_index(driver):

    result = driver.session().run("""
       Match (cp:Paper)<-[:CITED_BY]-(p:Paper)
        WITH p, count(cp) AS citation_num ORDER BY citation_num DESC
        MATCH (p)<-[:WRITES]-(a:Author)
        RETURN a.name as name, collect(citation_num) AS author_citation
    """)
    result = list(result)
    name_index_list = []
    for record in result:
        name, cite = record["name"], record["author_citation"]
        # use enumerate, x[0] start from 0
        filted_cite = [x[1] for x in filter(lambda x: x[1] > x[0] ,enumerate(cite))]
        name_index_list.append((name, len(filted_cite)))

    sorted_name_index_list = list(sorted(name_index_list, key= lambda x: x[1], reverse= True))
    for record in sorted_name_index_list:
        name, h_index_score = record
        print(name, h_index_score)
```

Result as follows

```
Michael Stonebraker 17
Serge Abiteboul 16
Philip A. Bernstein 15
......
```

3 Graph algorithms

3.1 PageRank

Create the graph

```
CALL gds.graph.create(
    'my-native-graph',
    'Paper',
    'CITED_BY'
)
YIELD graphName, nodeCount, relationshipCount, createMillis;
```

Use PageRank algorithm

```
CALL gds.pageRank.stream('my-native-graph') YIELD nodeId,score RETURN
    gds.util.asNode(nodeId).title AS title, score ORDER BY score DESC, title ASC
```

Result

title	score
"Query Evaluation Techniques for Large Databases."	5.080053152344614
"Concurrency Control: Methods, Performance, and Analysis."	4.83307664673713
"Evolution and Change in Data Management - Issues and Directions."	4.479256397741085
...	...

Analysis

The top 3 papers in Pagerank score are published in ACM CSUR 1993, ACM CSUR 1998 and ACM SIGMOD 2000 respectively, with citation of 1991, 176 and 69 respectively.
It is consistent with our intuition: the earlier the paper published and the more citations the it has, the higher the pagerank score it owns.

3.2 NodeSimilarity

Create the graph

```
CALL gds.graph.create(
  'SimilarPapers',
  ['Paper','Keyword'],
  {
    MY_RELATIONSHIP_TYPE_NATURAL: {
      type: 'MENTIONS',
      orientation: 'NATURAL'
    }
  }
)
```

Use Node similarity algorithm

```
CALL gds.nodeSimilarity.stream('SimilarPapers')
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).title AS Paper1, gds.util.asNode(node2).title AS
    Paper2, similarity
ORDER BY similarity DESCENDING, Paper1, Paper2
```

Result

Paper1	Paper2	Similarity
"Almost" subsidy-free spatial pricing in a multi-dimensional setting."	"A Decomposition Algorithm for Optimization over Efficient Sets"	1.0
...	...	...

Rationale of the result

One explicit way to evaluate the similarity of two papers is to compare their keywords. When two sets of keywords totally overlap, which is the case of sample result(Figure 1), the similarity degree will equal to 1; semantically it shows that these two papers belong to same academic domain and focus on same topics.

Neo4j offers similarity algorithms like Node Similarity, K-Nearest Neighbors, Jaccard Similarity, Cosine Similarity, Pearson Similarity, etc. The mechanism of Node Similarity algorithm is that it computes pair-wise similarities based on the Jaccard metric, also known as the Jaccard Similarity Score, similar to Jaccard Similarity algorithm but has a application focus on pair nodes.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Where set A is a set of Keyword nodes mentioned by Paper A, and set B is a set of Keyword nodes mentioned by Paper B. The similarity degree between Paper A and B is measured by using sets A and B's intersection cardinality divided by their union cardinality.

4 Recommender

Users are firstly asked to type in a set of keywords, like ["Data Storage", "Data Processing", "Data Modeling"], then the list is treated as an input which will be called by cypher in the format of $list to interact with Neo4j in Python.

4.1 Find the research communities

```
MERGE (c:Community{keywords_list:$list})
WITH c MERGE (k:Keyword{name:c.keywords_list[$i]})
WITH c,k MATCH (c),(k) CREATE (c)-[:CONTAINS]->(k)
```

Create node 'Community' with property keywords_list, and the value of the property comes from the user's input list. Each element from the keyword list should overlap with the name of node 'Keyword', otherwise a new 'Keyword' node will be established. Then connect them with a relationship called CONTAINS.

4.2 Find the conferences/journals related to the community

```
MATCH (c:ConferenceTitle)-->(:Conference)-[h1:HAS]->(:Paper)
WITH c, COUNT(h1) AS cnt1
MATCH (c)-->(:Conference)-[h2:HAS]->(:Paper)-->(:Keyword)<--(cm:Community)
WITH c,cm,cnt1,COUNT(*) AS cnt2 WITH c,cm,cnt1,cnt2*100/cnt1 AS ratio WHERE ratio>90
WITH c,cm MATCH (c),(cm) CREATE (cm)-[:RELATED_TO]->(c)
```

The aim is to connect node 'ConferenceTitle' with node 'Community' under required conditions. First compute the total number of papers of a ConferenceTitle, cnt1, from a path yielding from 'ConferenceTitle' to 'Paper'. Secondly under same 'ConferenceTitle' extend the path to 'Community', and compute the total number of papers from same ConferenceTitle that belong to a Community, cnt2. If cnt2 covers at least 90% of cnt1, a relationship RELATED_TO from 'ConferenceTitle' to 'Community' will be built.

4.3 & 4.4 Find the top 100 papers, potential reviwers, gurus of the community

```
CALL gds.graph.create.cypher(
'top100papers',
'MATCH (n:Paper) RETURN id(n) AS id',
'MATCH (n:Paper)-[r:CITES]->(m:Paper),
    (c:Community)-[:RELATED_TO]->()-[:HAS_Edition|:HAS_VOLUME]->()-[:HAS]->(n),
    (c:Community)-[:RELATED_TO]->()-[:HAS_Edition|:HAS_VOLUME]->()-[:HAS]->(m)
    RETURN id(n) AS source, id(m) AS target')
YIELD graphName AS graph, nodeQuery, nodeCount AS nodes, relationshipQuery,
    relationshipCount AS rels
CALL gds.pageRank.stream('top100papers') YIELD nodeId,score
WITH gds.util.asNode(nodeId).name AS name, score ORDER BY score DESC, name ASC
MATCH (p:Paper{name:name}),
(c:Community)-[:RELATED_TO]->()-[:HAS_Edition|:HAS_VOLUME]->()-[:HAS]->(p)
RETURN/WITH c.name as Community, collect(p)[..100] as Top100papers //run for 4.3
MATCH (a:Author)-[w:WRITES]->(p:Paper) WHERE p in Top100papers
RETURN Community, collect(a) AS Potential_reviewers //additional cypher for 4.4
MATCH (a:Author)-[w:WRITES]->(p:Paper) WHERE p in top100list WITH  c, a, COUNT(w)
    AS Top100_times WHERE Top100_times>1
RETURN Community,collect(a) AS Gurus //additional cypher for 4.4
```