Incremental Graph Schema Documentation

Fatemeh Ghasemi[1]

[1] *École Normale Supérieure de Lyon, France*

March 20, 2022

1 Installation

To be able to use the clustering algorithms, you need to at first, upload your data into a database, in our project we tend to use Neo4j to manage the data. Accordingly, you need to have Neo4j in your system.
Later, to run the GUI, you need to be able to run a Django server, so we also need Django on your computer.

1.1 Installing Neo4j

Start by installing Neo4j from here.

1. Create a new database in Neo4j browser, with login information like:
   `login:neo4j` and `mdp:1234`
   And now, for example, to have LDBC database, we have to proceed as below:
2. Import the container of the LDBC's archive in `Settings` → `Open Folder` → `Import` → `ldbc_all`
3. Edit the configuration file in the `Setting` by adding these two line:

```
$ dbms.active_database=ldbc.db
$ dbms.connector.bolt.listen_address=:7687
```

4. In the Neo4j's terminal execute this A.
5. Start your database by click on `start` .

If you wanted to use another database, each time you should switch to that database in Neo4j, it means, stopping the current running database and start the new one before choosing in the web application1.

1.2 Installing Django

Django is a Python web framework, thus you need Python as well. From here you can get the latest version of Python.
If you have Windows, please follow this.
Using this you can install Django running :

```
$ git clone https://github.com/PI-Clustering/Code
```

1.3 Clone our project

We maintain a GitLab repository at https://github.com/PI-Clustering/Code.
Go to your terminal and run this command:

```
$ git clone https://gitlab.aliens-lyon.fr/
graph-db-pi/organisation
```

Now you have almost everything!

2 Run the server

After you cloned the project and had installed all the necessary tools, it's time to run the server and use it! To do so, follow these steps:

1. Go to the project directory via your terminal. (Where the *manage.py* file is.)
2. Maybe enter your preferred environment. (An environment with all the necessary tools.)
3. Now run these commands:

```
$ python manage.py makemigrations polls
$ python manage.py migrate
$ python manage.py collectstatic
```

4. And finally: `$ python manage.py runserver`
5. In your browser, go to http://localhost:8000

3 Use the web application

After loading the web application in your browser, go to the "Run Algorithm" page, and follow these steps:

1. In the first field, choose your preferred database. For example, the Covid-19 database.
2. In case of choosing one of our pre-prepared databases, you can choose the first button("Use precomputed?") to also use the pre-computed clusters and shorten the time of computation.
3. In the third field we configure the probability of having a node in the graph on which we're running the algorithm. Thus, for example if you configure it to be 80 it means that each node will appear in the graph with probability equal to 80%.
4. To give a very short description of the algorithm, it starts by clustering the graph and then clustering each cluster recursively. In the fourth field, what we're indicating is the maximum number of sub-clusters of a cluster at each step.
5. By choosing the second button, you're querying the edges between cluster as well. Our algorithm only compute the hierarchical cluster. So if you choose this button, the edges between the cluster will also appear.
6. By choosing the last check-box, two graphs will appear on the "Algorithm Performance"3.4 page. A graph, illustrating the execution time progression and the other, demonstrating the adjusted risk index!

Now by clicking on "Submit" and waiting for a bit, you'll have the final graph schema!

3.1 Results

This tab shows a visualisation of the selected graphs, along with the inferred data types. The original data has been "decorated" with meta-nodes which correspond to the inferred types. The user is able to browse and zoom the data to view the relationships between the different types, and can navigate the data to view in detail the quality of the inferred types.

3.2 Benchmarks

This tab allows the user to view statistics about the quality of particular executions. It shows a list of all executions of the algorithms performed by the user.

3.3 Add Nodes

This is used to test and then see the results of the incremental algorithm

3.4 Algorithm Performance

This tab allows the user to view statistics about the quality of the last execution. In particular, the charts give a visual representation of the evolution of the inference algorithm.

4 The Algorithms!

4.1 Base Algorithm

The main algorithm is from a previous paper written by Nicolas Mir, 2021. To summarize, it recursively divide each cluster in sub-clusters. To do so, it take a reference node, and calculate the distance of each node to the references node, and then use the Gaussian Mixture Models to generate the new sub-clusters.

4.2 New Algorithms

These are described in the second repository contained in the code repository.

Appendices

A Neo4j

This is what you should run in your Neo4j's terminal at the 4th step of Neo4j's installation.

```
./bin/neo4j-admin import --database=ldbc.db --delimiter='|'
--nodes=Comment=import/comment_0_0.csv --nodes=Forum=import/forum_0_0.csv
--nodes=Person=import/person_0_0.csv --nodes=Post=import/post_0_0.csv
--nodes=Place=import/place_0_0.csv
--nodes=Organisation=import/organisation_0_0.csv
--nodes=TagClass=import/tagclass_0_0.csv --nodes=Tag=import/tag_0_0.csv
--relationships=HAS_CREATOR=import/comment_hasCreator_person_0_0.csv
--relationships=HAS_TAG=import/comment_hasTag_tag_0_0.csv
--relationships=IS_LOCATED_IN=import/comment_isLocatedIn_place_0_0.csv
--relationships=REPLY_OF=import/comment_replyOf_comment_0_0.csv
--relationships=REPLY_OF=import/comment_replyOf_post_0_0.csv
--relationships=CONTAINER_OF=import/forum_containerOf_post_0_0.csv
--relationships=HAS_MEMBER=import/forum_hasMember_person_0_0.csv
--relationships=HAS_MODERATOR=import/forum_hasModerator_person_0_0.csv
--relationships=HAS_TAG=import/forum_hasTag_tag_0_0.csv
--relationships=HAS_INTEREST=import/person_hasInterest_tag_0_0.csv
--relationships=IS_LOCATED_IN=import/person_isLocatedIn_place_0_0.csv
--relationships=KNOWS=import/person_knows_person_0_0.csv
--relationships=LIKES=import/person_likes_comment_0_0.csv
--relationships=LIKES=import/person_likes_post_0_0.csv
--relationships=STUDIES_AT=import/person_studyAt_organisation_0_0.csv
--relationships=WORKS_AT=import/person_workAt_organisation_0_0.csv
--relationships=HAS_CREATOR=import/post_hasCreator_person_0_0.csv
--relationships=HAS_TAG=import/post_hasTag_tag_0_0.csv
--relationships=IS_LOCATED_IN=import/post_isLocatedIn_place_0_0.csv
--relationships=IS_LOCATED_IN=import/organisation_isLocatedIn_place_0_0.csv
--relationships=IS_PART_OF=import/place_isPartOf_place_0_0.csv
--relationships=HAS_TYPE=import/tag_hasType_tagclass_0_0.csv
--relationships=IS_SUBCLASS_OF=import/tagclass_isSubclassOf_tagclass_0_0.csv
```

For fib25, the same procedure is to be applied, and the command to be entered is :

```
./bin/neo4j-admin import --database=fib25.db --nodes=Neurons=Neuprint_Neurons_fib25.csv --
```

B Algorithm

Algorithm 1 Incremental Scheme

procedure ADD_NODE($node, cluster$)
    Add the node to the dictionary of node of the cluster
    Increment the number of total modification made
    if number of modification / number of nodes < 0.1 **then**
        $labs \leftarrow$ label of $node$
        for lab_set **in** cutting value of $cluster$ **do**
            if $lab\_set \subset labs$ **then**
                $fils \leftarrow$ subcluster corresponding to the cutting value
                ADD_NODE_REC($fils, node$)
            end if
        end for
    else
        Transform the data into treatable data
        Reapply the general algorithm to recompute the while cluster
    end if
    return $cluster$
end procedure

procedure ADD_NODE_REC($node, cluster$)
    Add the node to the dictionary of all nodes of the subcluster $cluster$
    $d \leftarrow$ distance between $node$ and the reference node of $cluster$
    $cuts \leftarrow$ array containing the value where GMM has separate the different clusters
    if $cuts = [\,]$ **then** ▷ That means that we have reached the smallest subclusters, there are no more after
        return $cluster$
    end if
    $cuts[0] \leftarrow 0$ ▷ As the reference node may be fictive, we add 0 to be sure that we will find a place for the node
    $i \leftarrow$ position of d in the sorted array $cuts$
    $sub\_cluster \leftarrow i$-th subclusters of $cluster$
    return ADD_NODE_REC($node, sub\_cluster$)
end procedure

Bibliography

Nicolas Mir Angela Bonifati, Stefania Dumbrava (2021). "Hierarchical Clustering for Property Graph Schema Discovery". In: *International Conference on Extending Database Technology (EDBT)*.

Algorithm 2 Exact incremental Scheme (outline, as issues remains (like computing line 20, but in good progress of being resolved efficiently))

1: **procedure** EXACT_INCREMENTAL_SCHEME($node, cluster$)
2:     $labs \leftarrow$ label of $node$
3:     $sets\_labels \leftarrow cluster.get\_setes\_labels()$
4:     **for** set_labels **in** $sets\_labels$ **do**
5:         **if** $set\_labels \subset labs$ **then** ▷ We act only on the subclusters where the node will have an influence
6:             $sub\_cluster \leftarrow$ the subcluster of $cluster$ corresponding to the cutting value set_labels
7:             EXACT_INCREMENTAL_SCHEME_REC($node, sub\_cluster$) ▷ Changing $sub\_cluster$ changing the variable $cluster$
8:         **end if**
9:     **end for**
10:     **return** cluster
11: **end procedure**

12: **procedure** EXACT_INCREMENTAL_SCHEME_REC($node, cluster$)
13:     Add $node$ of the dictionnary of nodes of $cluster$
14:     $nodes \leftarrow$ the set of the node of $cluster$
15:     $new\_reference\_node \leftarrow$ MAX_LAB_PROPS($nodes$)
16:     $measure \leftarrow$ array of the distance of all nodes to $new\_reference\_node$
17:     $prediction \leftarrow$ ITER_GMM($measure$) ▷ Calcul de dans quel cluster doit se retrouver chaque noeud selon la GMM
18:     $sets\_nodes \leftarrow$ the list of the set of nodes, each set corresponding to the node of one subcluster according to $prediction$
19:     **for** set_node **in** $sets\_nodes$ **do**
20:         **if** set_node is the set of an already computed cluster **then**
21:             Add the corresponding computed cluster to the subclusters of $cluster$
22:         **else**
23:             Add $rec\_clustering(set\_node)$ to the subclusters of $cluster$ ▷ It is not the good format of data but it gives the idea and simplify
24:         **end if**
25:     **end for**
26: **end procedure**

Algorithm 3 Pruning the data (modification of the former procedure)

procedure TO_FORMAT($similarities\_dict, nodes$)
    $data \leftarrow \emptyset$
    $pre\_ecrasage \leftarrow \min_{node \in nodes} \lfloor \log_{10}(d_2[node]) \rfloor$
    for $node \in nodes$ **do**
        $amount \leftarrow \left\lfloor \frac{nodes[node]}{10^{\max(0, pre\_ecrasage-2)}} \right\rfloor$
        Add $amount$ times $similarities\_dict[node]$ to data
    end for
    return data, $\max(0, pre\_ecrasage - 2)$
end procedure