Network Analysis in ArangoDB

Alessandro d'Agostino, Mattia Ceccarelli, Riccardo Scheda

January 2020

Abstract

The project aim is to develop a simple python API that let the user interact with the open-source multi-model database ArangoDB to perform fast queries on multi-partite graphs, extract and visualize sub-network which can then be analized with powerful python libraries. The study also includes results on the computational efficiency of ArangoDB's algorithms, in order to have an estimate of the API timing behaviours with large networks.

1 Introduction

ArangoDB [**?**] is a multi-model, open-source database with flexible data models for documents, graphs and key-values. It is being used in different fields: from semantic analysis to genome studies.

In the present work we employed the library `python-arango` [**?**] to import and export multi-partite network-like databases, that are defined as collections of documents (nodes) and a collection of relationships between them (edges).

In particular, a document is a python dictionary with a mandatory `_id` key assigned to a unique value. What we call an "edge" is also a python dictionary, but with an `_id`, `_source` and `_target` voices, to specify not only the linked nodes, but also the direction of said link in the case of directed graphs.

Graph Object A Graph is the combination of at least one node collection and a single edge collection, called edge definition. With a Graph two main operations are available: visualization and execution of a query.

The former is possible thanks to ArangoDB web interface, as shown in the figures below, which provide the user many options such as the starting node and the number of elements to load. By default, the starting node is random.

For the latter we relied on the poweful AQL, that is the query language implemented in arango. It is used to execute different kind of "search" through the graph, and select nodes and edges.

2 Pipeline

Now we describe the operations which lead to have a subgraph in ArangoDB. The first thing to do is read the file in gexf format. To do this we use the function *read_gexf*:

```
read_gexf(db,
          filename,
          nodes_collection_name='nodes',
          edges_collection_name='edges',
          graph_name='Net')
```

which takes the information of the nodes and the edges from the gexf file and creates the collections of the nodes and of the edges, and then returns two graphs: one of type python-arango and one of type of networkx.

Now in the Arango web interface we have a graph and the two collections of nodes and edges.

Then, we can extract a subnet from the graph with a graph traverse of python-arango, using the function:

```
first_neighbours = traverse(db,
         starting_node,
         nodes_collection_name,
         graph_name,
         direction='outbound',
         item_order='forward',
         min_depth=0,
         max_depth=1,
         vertex_uniqueness='global')
```

> Shakespear: The traverse starts from a starting node that one chooses, and compute the 1,2... first neighbours of that starting node.

This could be any traversal, any query, any sub set of nodes from the graph. this function returns a Python list containing vertex and the path crossed by the traverse.

Now we have a dict of the first neighbours of astenia and all the paths which reach that neighbours Now, having the list of the first neighbours we can obtain the subnet just using the function *subgraph* given by Networkx:

```
Nx_Sub_Net =
Nx_Net.subgraph([vertex['label']
for vertex in
first_neighbours['vertices']])
```

In this way we create the subgraph only with the nodes and the edges, so now we have to add all the attributes to each node:

```
for node in Nx_Sub_Net:
  attr = pa.get_vertex(db,
                       {'label':node},
                       'Sym_Deas')
  nx.set_node_attributes(Nx_Sub_Net,
                 {node : attr})
```

The ast thing to do is to export the subnetwork on ArangoDB, using the two functions *nx_to_arango* and *eport_to_arango*:

```
sub_net = nx.readwrite.node_link_data(Nx_Sub_Net)
sub_net = pa.nx_to_arango(sub_net, 'Sub_Net')
Sub_Net = pa.export_to_arango(db,
                  sub_net ,
                  'Sub_Net',
                  'Sub_Net_edges',
                  'Sub_Graph')
```

3 Timing

4 Results

5 Conclusions

We've produced an applicative to visualize and efficiently query a graph-like database with a simple yet resourceful programming language like python.

As shown in the results, the scaling of the ArangoDB query algorithm is linear for the two cases taken into consideration, k shortest path and neighborood search. This is promising for future application in large dataset.

Problems The main problems lie in the level of control we can gather through python. Indeed, it seems to be impossible to load and save default queries. Moreover, we couldn't manage to save graph visualization options, which could ease the accessibility.

Future Ideas We will focus on preparing the applicative for the non-programmer user.

Primarly, the readibility of the code must be improved.

We also think a Graphical User Interface (GUI) could greatly smoothen the user experience.

In the future we hope to test the applicative to a muc larger database than the one we had at disposal.