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 [1] 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` [2] 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.

Pipeline

```
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:

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

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.

2 Timing

3 Results

4 Conclusion

References

[1] "Arangodb." `https://www.arangodb.com/`. open-source multi-model database.

[2] joowani, "python-arango." `https://github.com/Joowani/python-arango`. Python driver for ArangoDB.