Network Analysis in ArangoDB

Alessandro d'Agostino, Mattia Ceccarelli, Riccardo Scheda

2020
January

Abstract

In this project we develop a Python module NEAAR in order to perform network analysis on ArangoDB graphs using the Python library Networkx.

ArangoDB is an open-source native multi-model database for graphs and documents. It's useful for graph visualization.

The main problem of the project was to make an API between arangoDB and the Python library Networkx in order to do some operations on the graphs of arangodb through Networkx.

To create a graph in ArangoDB, we need some two types of documents, one for the nodes and one for the edges. A single document contains all the attributes of one node or of one edge. A set of documents is a collection. So to create one simple graph we need two collections, one for the nodes and one for the edges.

The starting point is the creation of a networkx graph using the function

```
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=db, starting_node='emicrania',
         nodes_collection_name='Sym_Deas',
         graph_name='Sym_Net',
         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.