1. Bigraph construction

In Commex we have a bipartite network of scholars and terms (1 scholar has multiple terms, 1 term has multiple scholars associated to it). The python code is based on a php-code from old Tinaweb so I will explain how the the original code (php-source) works.

1.1. Algorithm for Commex

- One type of nodes are the **scholars**.
- Other type of nodes are the **keywords**.

In the practice, for steps 2 and 3 we use the file get_scholar_graph.php. For the remaining steps (4 and so on) we use the file gexf_generator.php. Considering this, please observe in each step the corresponding lines of code wich are specified inside brackets] [.

1. The user selects an "ego" in order to visualize a network, in this case is a *scholar* (named *login*).
2. *keywords_ids* ⟵ Retrieve all the *keywords* related to the selected *scholar* (*login*). [47-50]
3. For each *keyword_id* on *keywords_ids*: [52-58]
    - *scholars* ⟵ Retrieve all the *scholars* associated with this *keyword_id* (firstly pushed to *scholar_array*)
4. For each *scholar* in *scholars*:[31-60]
    - *scholar_keywords* ⟵ Retrieve all the *keywords* associated with this *scholar*. [34]
    - Iterate in *scholar_keywords* and build the *termsMatrix*. [36-60]
5. *terms_array* ⟵ Retrieve all the keywords that exist in *termsMatrix*. [73-88]
6. For each *term* in *terms_array*: [106-153]
    - *term_scholars* ⟵ Retrieve all the *scholars* related to this *term*. [109-115]
    - Iterate in *term_scholars* and build the *scholarsMatrix*. [117-141]
    - Save the *term* as *nodeB* in the GEXF/JSON. [142-151]
7. *scholars* ⟵ From now, inside every bucle, we will consider only scholars that exist in *scholarsMatrix*. [155-160]
8. For each *scholar* in *scholars*: [155-241]
    - Save *scholar* as *nodeA* in the GEXF/JSON.
9. For each *scholar* in *scholars*: [248-264]
    - For each *keyword* belonging to *scholar.keywords*: [257-264]
        - Save as bipartite-edge in the GEXF/JSON with Source=*scholar*, Target=*keyword*, Weight=1

10. For each *term* in *terms_array*: [268-285]

    - *neighbors* $\longleftarrow$ Retrieve all *terms* in *termsMatrix* which are related to the occurrences of this *term*. [270-273]
    - For each *neigh* in *neighbors*: [277-284]
        - Save as type2-edge in the GEXF/JSON with Source=*term*, Target=*neigh*, Weight=$\frac{\text{occurrences of } neigh}{\text{occurrences of } term}$

11. For each *scholar* in *scholars*:

    - *neighbors* $\longleftarrow$ Retrieve all *scholars* in *scholarsMatrix* which are related to the co-occurrences of this *scholar*. [291-294]
    - For each *neigh* in *neighbors*:
        - Save as type1-edge in the GEXF/JSON with Source=*scholar*, Target=*neigh*, Weight=$jaccard(\text{occurrences of } scholar, \text{occurrences of } neigh, \text{co-occurrences of } scholar)$

1.2. Generic algorithm definition

- The scholars will be represented by *nodesA*.
- The terms will be represented by *nodesB*.

1. The user selects an "ego" in order to visualize the related network. In this case is an individual (named q) where $q \in nodesA$.

2. $B_a \longleftarrow$ Retrieve all the $b \in nodesB$ where each b is related with q.

3. For each B_{ai} in B_a:

    - $A \longleftarrow$ Retrieve all the $a \in nodesA$ associated with B_{ai} (equivalent to A[id].push(B_{ai}).

4. For each A_i in A:

    - $B_{A_i} \longleftarrow$ Retrieve all the $b \in nodesB$ where b is associated with A_i.
    - Iterate in B_{A_i} and build the *BMatrix*.

5. $B \longleftarrow$ Retrieve all the $b \in nodesB$ where b exists in *BMatrix*.

6. For each B_i in B:

    - $A_{B_i} \longleftarrow$ Retrieve all the $a \in nodesA$ where a belongs to B_i.
    - Iterate in A_{B_i}:
        - Build the *AMatrix*.
        - Save B_i as *nodeB* in the GEXF/JSON.

7. $A \longleftarrow$ Retrieve the elements in *nodesA* that exists in *AMatrix*. (A is redefined)

8. For each A_i in A:

    - Save A_i as *nodeA* in the GEXF/JSON.

9. For each A_i in A:

   - For each $b \in nodesB$ belonging to A_i:
     - Save as bipartite-edge in the GEXF/JSON with Source=A_i, Target=b, Weight=1

10. For each B_i in B:

    - $neighborsB_i \longleftarrow$ Retrieve all $b \in nodesB$ belonging to B_i.
    - For each $neighborB_{ij}$ in $neighborsB_i$:
      - Save as type2-edge in the GEXF/JSON with Source=B_i, Target=$neighborB_{ij}$, Weight=$\frac{\text{occurrences of } neighborB_{ij}}{\text{occurrences of } B_i}$

11. For each A_i in A:

    - $neighborsA_i \longleftarrow$ Retrieve all $a \in nodesA$ belonging to A_i.
    - For each $neighborA_{ij}$ in $neighborsA_i$:
      - Save as type1-edge in the GEXF/JSON with Source=A_i, Target=$neighborA_{ij}$, Weight=$jaccard(\text{occurrences of } A_i, \text{occurrences of } neighborA_{ij}, \text{co-occurrences of } A_i)$