Semantics of Expert Finder Network

August 24, 2014

# Background

To make expert finder networks requires displaying nodes of concepts and authors and edges of concept to concept, concept to author and author to author. Edges have weight that is determined by the number of papers represented by the edge. The three types of edges are:

1. Concept to Concept. Count the papers in which both concepts appear, regardless of author
2. Concept to Author. Count papers by the author on the concept
3. Author to Author. Count the papers in which the two authors are both listed as authors of the paper, regardless of concept

These data are difficult to compute in real-time. We envision an interface where the user can “recenter” a network by clicking on an author or concept to redraw the network around the selected author or concept. All possible edges of concept-to-concept, author to author, and concept to author must be known in advance, along with the number of papers associated with each. The actual papers can be fetched in real-time when the user selects a particular edge or concept, or author.

Pairwise counts are currently computed off-line using Python, resulting in a “concordance”. The concordance data structure contains all the information needed[[1]](#footnote-1) for generating of the desired networks, but slicing the concordance in real-time cannot be done from the web to Python. A “pure web” architecture is needed in which Ajax makes SPARQL calls, returning JSON which can be transformed into the structures needed for the user interface. All should be under a model-view-controller architecture managed by AngularJS. Interface updates must be done in interactive time (1-2 seconds). The semantics and pre-calculations described in this document are designed to support the expert finder architecture.

# Semantics

The table below shows the triples that are needed.

|  |  |  |  |
| --- | --- | --- | --- |
| Subject | Predicate | Object | Note |
| curi | rdf:type | skos:Concept | Existing |
| curi | rdfs:label | “concept name” | Existing |
| curi | *ufVivo:pubCount* | “number of publications” | New |
| euri | rdf:type | ufVivo:Edge | New |
| euri | *ufVivo:hasConcept* | curi | New |
| euri | *ufVivo:hasPerson* | puri | New |
| euri | *ufVivo:pubCount* | “number of publications” | New |
| auri | rdf:type | foaf:Person | Existing |
| auri | rdfs:label | “author name” | Existing |
| auri | *ufVivo:pubCount* | “number of publications” | New |

# Methods for Generating Assertions

1. pubCount for concepts can be updated at any time. A simple query generates the counts for each concept, a construct produces the required statements. All previous concept pubCount statements are deleted. All new statements are added.
2. The same process can be used for authors. Generate the counts, use a construct to generate the statements. Drop all the existing statements. Add all the new statements.
3. To update the links, a similar process is used enabled by the concordance
   1. All the link statements and all statements with the links as subjects can be dropped.
   2. The concordance is used to generate new links for concept-concept pairs, author-author pairs and concept-author pairs. For each pair, a link, two end points and a count is generated.
   3. The new links are added to VIVO

These steps can be execute after each paper ingest.

# Using the New Assertions

The new assertions make it very simple to generate co-concept, co-author and concept-author maps centered on any particular concept or author. The following steps are followed.

1. Given a concept, use Ajax to execute the SPARQL query from a web page to retrieve the desired elements related to the concept. For concept x, retrieve all triples with x as subject plus all with x as object. This retrieves all links.
2. As a subquery, retrieve all links associated with the concepts, leading to co-occuring concepts, authors, and links.
3. As a sub-subquery, retrieve all links associated with the above, as desired to make a co-concept map, co-author-map or concept-author map.
4. The resulting objects and links are returned as JSON to the web page. This should be fast (less than a second).
5. Use Javascript in the call back function to reshape the JSON into the format required by D3 according to the rimming parameters currently in place as determined by Angular. This should be very fast (less than a tenth of a second).
6. Display with D3. This should be very fast (less than a tenth of a second).

1. Author-author pairings are not currently in the concordance, but this is a straightforward addition using the logic already in place for concept-concept and concept-author pairings. [↑](#footnote-ref-1)