graphOnt: An Ontology Based Library for Conversion from Semantic Graphs to JUNG

Stephen Kelley*, Mark Goldberg*, Malik Magdon-Ismail*, Konstantin Mertsalov*, William Wallace† and Mohammed Zaki*
*Department of Computer Science
Rensselaer Polytechnic Institute Troy, NY 12180
{kelles, goldberg, magdon, mertsk2, zaki}@cs.rpi.edu
† Department of Decision Sciences and Engineering Systems
Rensselaer Polytechnic Institute
Troy, NY 12180
wallace@rpi.edu

***Abstract*—In this work, we present the software library graphOnt. The purpose of this library is to automate the process of dynamically extracting "interesting" graphs from semantic networks. Instructions on the extraction are fed into the library via an ontological language specification custom built for this application. A set of SPARQL queries are used to define vertices and edges in the constructed graph. Extracted graphs are returned using the JUNG framework, which offers many algorithmic and visualization options. This work allows a set of individuals analyzing the same semantic network to extract and analyze dynamically created graphs using sophisticated, specific algorithmic tools without needing to manually construct classical graphs from the data.**

I. Introduction

The recent emergence of the Semantic Web[3] as a viable paradigm for data storage and processing has resulted in large amounts of data being converted into semantically enabled formats. These formats allow data to be federated across multiple sources easily, provide a framework for expressing semantic relationships between the data, and through ontological information, are capable of being "reasoned" on. As a result, semantic technologies have begun to appear wherever large amounts of data with little structure must be stored.

Conceptually, semantic data forms a labeled multigraph. Every piece of data is expressed as a subject-predicate-object triple. Vertices in the graph correspond to subjects and objects in the data, for instance: "Jack", "Jill", "The United States of America". Edges between items express some relationship that appeared as a predicate in the data. For instance, Jack is a brother of Jill. Jill is a citizen of the United States of America. For such a small example, it is easy to comprehend the entire network. There are only 2 types of items, a person and a country, and the set of predicates expressed by edges is small, "a brother of" and "a citizen of". However, in real semantic networks, there may be many vertex types and relation types. For instance, in dbpedia[2], an effort to express various portions of the Wikipedia[1] project as semantic data, infobox data is expressed using 170 vertex types and 8000 types of relations across 22.8 million triples.

For an individual analyzing this graph, it becomes difficult to recognize any meaningful trends in the data due to its richness. Consider a Wikipedia-like system tracking information about a set of people. In addition to tracking simple biographical information, the system also tracks and stores communication between the individuals in the set. Traditionally, using a relational model for storing this data, communications would be stored in a separate database table, but here, tuples representing all of the data are stored in the same graph. The individual analyzing such a network might have difficulties separating what they are interested in from ancillary data. Visualization in particular, becomes difficult with such a rich data set, as there are many edges with various contexts within the graph.

Further, research has provided many useful algorithms that can only be run on classical graphs. Tasks such as identifying community structure and analyzing structural hierarchy are all well studied and could prove quite useful to certain types of analyses. SPARQL [7] provides an SQL-like query language to pull specific relations out of semantic data. The results of these queries can be used to manually construct graphs to be fed into implementations of graph algorithms to aid an individual in his or her analysis. However, doing so may prove cumbersome, as an analyst would need to manage and store the queries used to create graphs, the created graphs themselves, and the output of a given analysis algorithm on a specific graphs.

The goal of this software is to enable an analyst to explore semantic data in a way which allows advanced, preexisting graph algorithms to be run, but also gives the user the ability to dynamically modify the analysis to include or exclude entities or relations as he or she determines that they are interesting or uninteresting. Including this library into an algorithm implementation allows a user with a semantic dataset to track only what he or she has declared as relevant to the analysis and the output of the graph algorithm.

II. Leveraged Open Source Projects

A. Jena: Jena[4] is a commonly used, open source software framework for storing and interacting with semantic data. It provides RDF and OWL storage, supports SPARQL queries, and provides reasoning capabilities along with an interface to interact with external reasoning engines. All of these features are supplied through an easy to use API. This software was chosen as a triple store for this project due to its large user-base, simplicity of use,

Edge Predicate Modifiers

`gont:weight`	Given a real value, will weight any appearance of an edge using the modifier
`gont:edgeType`	Given either of the values "Directed" or "Undirected", constructs edges in the non-semantic graph in the appropriate manner

Fig. 1. Table showing modifiers which may be included in the edge definition.

and active development.

B. JUNG: The Java Universal Network/Graph Framework (JUNG) [5], is an open source graph library which in addition to providing the ability to manipulate graphs, offers visualization and many graph algorithms in the areas of data mining and social network analysis.

III. Meta-Ontology

The construction of a non-semantic graph from semantic data is controlled by a meta-ontology. This user provided ontology provides a framework by which information about the non-semantic graph is expressed, namely, what components of the semantic graph are to be used for vertices and how edges are to be constructed.

A. Defining Vertices: Defining vertices in the non-semantic graph ends up being quite simple. In the meta-ontology, an individual must be created of type `gont:nodeDef`. The individual then must have a property `gont:ofType` which will point to a type in the semantic data to be included in the vertex set. Multiple node definitions may exist. A user, for example, can define a non-semantic graph where the vertex set consists of writers and scientists. This is done simply by creating a second individual of type gont:nodeDef.

B. Defining Edges: Defining edges in the non-semantic graph is similar to the process for defining vertices. In the meta-ontology, an individual must be created of type `gont:edgeDef`. The individual must have property `gont:SPARQL` which points to a string consisting of a correct SPARQL query returning pairs of items in the graph. These items need not necessarily be of a type included in the defined vertex set using the above method, but only edges connecting 2 items which have been defined as vertices will appear in the graph. Edges are by default unweighted and undirected, but may be modified using the predicate set listed in Figure 1.

IV. Design

The library attempts to remove any knowledge of semantic technologies from the algorithmic side. This should enable individuals with no experience in semantic web technologies the ability to ensure that their algorithms will function properly with semantic data. A programmer intending to use the library need only create an instance of the library and offer it either the names of 2 persistent Jena models or files containing RDF graphs. One of these graphs is the data that will be processed while the other is a file consisting of the previously defined meta-ontology.

The library first ensures that the proper RDF graphs is loaded into Jena. It then queries for all individuals in the meta-ontology of type `gont:nodeDef`. These individuals are then queried to collect all of the `gont:ofType` values. The data ontology is then queried based on the `gont:ofType` values and all matching URIs are returned. These URIs are then added as vertices in the initial JUNG graph.

Next, the library issues a query to the meta-ontology for all individuals of type `gont:edgeDef`. For each matching individual, the library will query the data for all pairs matching `gont:SPARQL`. Based on the existence of any edge modifier predicates, edges will be added to the JUNG graph in weighted or unweighted and directed or undirected forms as appropriate. This process continues until all individuals of `type gont:edgeDef` have been examined.

The JUNG graph can then be returned to the algorithm implementation for processing. The performance of the library is not significantly different from manually extracting edges from the semantic data using SPARQL, as it simply replicates the queries that would be done manually in such an extraction. The running time of graph construction varies heavily with the complexity of the SPARQL queries used to construct the graph's edges.

V. Example

Using a subset of the RDF data from the dbpedia project and ontological data from the YAGO project [8], a simple example of the usefulness of this library can be constructed. The infobox dataset from dbpedia, using information collected from infoboxes in Wikipedia, consists of almost 32 million triples. The YAGO ontology provides classifications for a huge number of the subjects in the dbpedia dataset. One of these classifications indicates that a person is a writer. We take this set and use it to construct the vertex set in the graph we would like to process. In order to do this, in the meta-ontology file, we add the following information in N3 format.

```
:Node1 a gont:nodeDef;
  gont:ofType "Writer" .
```

This indicates that the individual Node1 is of type `gont:nodeDef` which is the type reserved for declaring types to include in the vertex set. The `gont:ofType` property indicates that the vertex set should be composed of any individual which is of the YAGO class `Writer`.

Now, suppose someone wishes to perform a series of analyses on this network. A commonly asked, very useful question would be which authors in this set are the most influential? That is to say, which authors have exerted the most influence on other authors. Examining the RDF data, dbpedia's infobox data contains a relation *influences* which indicates that one author has declared that another author is one of his influences. As an example, the data states that James Joyce was influenced by Oscar Wilde. Since influence is not necessarily a bidirectional relationship, we would like to create directed edges in the graph representing this

Analysis 1		Analysis 2
Edgar Allan Poe		Edgar Allan Poe
Charles Dickens		H. P. Lovecraft
Victor Hugo		Jules Verne
William Shakespeare		Mark Twain
Voltaire		Clark Ashton Smith
Walter Scott		Fritz Leiber
Fyodor Dostoevsky		William S. Burroughs
Leo Tolstoy		Robert Bloch
Gustave Flaubert		Richard Matheson
James Joyce		H. G. Wells

Fig. 2. The highest ranked authors in the 2 sample analyses.

interaction. This can be accomplished via the following change to the meta-ontology file in N3 format

```
:Edge1     a gont:edgeDef;
  gont:SPARQL " SELECT DISTINCT ?c ?d WHERE {
                  ?c j.1:influences ?d
                }";
  gont:edgeType "Directed" .
```

Here, we can see the SPARQL query which returns pairs of individuals linked by the predicate `j.1:influences`. Also, the relation is declared to be directed since there exists a statement containing the predicate `gont:edgeType` pointing to the value `Directed`. Now, with the proper statements in the meta-ontology, any algorithmic ranking algorithm can be run. In this case, JUNG contains an implementation of the PageRank[6] algorithm. The individuals with the highest ranking are given in Figure 2. This set consists of many of the world's significant, classical writers.

Now, suppose the individual analyzing the data would like to modify the meaning of edges in the graph; perhaps the ranking shown in Figure 2 was already known. The library can be used to create a different underlying graph composed of information which does not explicitly exist in the RDF graph. Many of the writers in the data have the predicate `genre`, which is used to specify which area(s) of literature the writer is known for. The individual engaging in our hypothetical analysis would like to find a ranking of individuals who have had a lasting effect within their genre. However, rather than performing the above analysis with the restriction that any edges must link individuals within the same genre, our hypothetical analyst is interested in a 2-hop set of influences. In this case, an edge would be placed between Person A and Person B are known for the same genre and if Person A was influenced by Person C and Person C was influenced by Person B. As an example, Stephen King was influenced by H. P. Lovecraft who was influenced by Edgar Allan Poe. Since King and Poe both are known for their work in the horror genre, they would be linked.

The vertex set definition in the meta-ontology used above does not change in this analysis. However, the edge definition does. The following is the N3 statements that would be used to create the proper edge set.

```
:Edge1     a gont:edgeDef;
    gont:SPARQL "SELECT DISTINCT ?c ?d
                 WHERE {
                   ?c j.1:influences ?x .
                   ?x j.1:influences ?d .
                   ?c j.1:genre ?y .
                   ?d j.1:genre ?y
                 }";
    gont:edgeName "edge1" .
    gont:edgeType "Directed" .
```

Though the changes to the SPARQL query in the above statements are subtle, their impact is significant. Here, the library is using SPARQL to create a graph representing relations which do not exist explicitly in the data set. The results of the analysis are shown in Figure 2. Due to data being more complete on Wikipedia for American and Science Fiction authors, these writers end up dominating this result set.

VI. Conclusion

Using this library, a user can quickly and dynamically create graphs from semantic data. Changing only statements in a meta-ontology, an individual can pull any number of graphs from the same RDF data set. These graphs can then be used as input to advanced analysis algorithms. Through the use of Jena and JUNG, this software is immediately deployable in a large number of current semantic web projects.

Acknowledgements

This material is based upon work partially supported by the U.S. National Science Foundation (NSF) under Grant Nos. IIS-0621303, IIS-0522672, IIS-0324947, CNS-0323324, NSF IIS-0634875 and by the U.S. Office of Naval Research (ONR) Contract N00014-06-1-0466 and by the U.S. Department of Homeland Security (DHS) through the Center for Dynamic Data Analysis for Homeland Security administered through ONR grant number N00014-07-1-0150 to Rutgers University.

The content of this paper does not necessarily reflect the position or policy of the U.S. Government, no official endorsement should be inferred or implied.

References

[1] http://wikipedia.org.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, pages 722–735, 2007.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web: Scientific american. *Scientific American*, May 2001.

[4] B. McBride. Jena: A semantic web toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002.

[5] J. O'Madahain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey. Analysis and visualization of network data using jung. *Journal of Statistical Software*, VV(II).

[6] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[7] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. W3C Recommendataion, Jan. 2008.

[8] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *16th international World Wide Web conference (WWW 2007)*, New York, NY, USA, 2007. ACM Press.