2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology

Structure Discovery in Large Semantic Graphs Using Extant Ontological Scaling and Descriptive Semantics

Sinan al-Saffar, Cliff Joslyn, Alan Chappell
Pacific Northwest National Lab
Seattle, WA 98109, U.S.A.

***Abstract*—As semantic datasets grow to be very large and divergent, there is a need to identify and exploit their inherent semantic structure for discovery and optimization. Towards that end, we present here a novel methodology to identify the semantic structures inherent in an arbitrary semantic graph dataset. We first present the concept of an extant ontology as a statistical description of the semantic relations present amongst the typed entities modeled in the graph. This serves as a model of the underlying semantic structure to aid in discovery and visualization. We then describe a method of ontological scaling in which the ontology is employed as a hierarchical scaling filter to infer different resolution levels at which the graph structures are to be viewed or analyzed. We illustrate these methods on three large and publicly available semantic datasets containing more than one billion edges each.**

***Keywords*-Semantic Web; Visualization; Ontology; Multi-resolution Data Mining;**

I. Introduction

A semantic graph is a graphical model for representing knowledge in patterns of interconnected nodes and edges. This model was first developed for artificial intelligence and machine translation [1], [2]. The semantic web represents an extension to those approaches to model human knowledge at the larger scale of the internet [3]. This has lead to a significant increase in the amount of data generated in semantic graph formats thus graphs containing billions of edges are increasingly common.

Semantic graph datasets are interlinked through the use of agreed upon or mappable node identifiers [4]. Thus one can curate datasets separately, or easily combine them into one larger graph. Furthermore, ontological (typing) meta-data is commonly included along with the underlying data (instance) graph, or partially so, or not at all. Analysis, mining, and visualizing methods have (not surprisingly) not been able to keep up with the size and richness of these large models; graph visualization is already notoriously difficult, scaling that to a billion edges even more so.

Consider being handed a ten billion edge semantic graph. How would we go about summarizing, visualizing, and mining the useful patterns in that model? How can we identify the semantically significant structures in the graph? What kinds of resources and architectures are needed to carry out these tasks? The two related methods we present in this work are an attempt to help answer these questions.

As noted above, ontological information, such as class and relation properties, may or may not be provided in a triple store. If so, it may or may not refer to externally defined ontologies, and then may or may not be consistent with those external ontologies. So to identify what *actual* semantic typing is available in a triple store, we have developed methods to derive an **extant ontology** based on the statistical prevalence of relations between typed entities in the graph. When the extant ontology is used for visualization, a user is able to tell what is the percentage of relations between certain node types.

While the extant ontology is intended first for visualization, it is also a candidate itself for further data mining such as clustering or association rule learning. Semantic graphs are rich models in the sense that typed nodes and labeled edges provide more information that typical network analyses problems. This richness provides further opportunity to identify patterns at different levels of abstraction. This is similar to the optimal resolution problem in data mining. For example, in [5] a wavelet-based multi-resolution decomposition is used to formulate two different sets of texture features for clustering. Each feature set operates at a different image resolution, one can be pixel color and the other can be texture. For each feature set, different distance measurement techniques are designed and experimented for clustering images in a database.

In the case of semantic graph data, what is the appropriate level of abstraction at which a visualization or data mining should take place? Should we "look" at individual "pixels" in an image or the different "textures"? We submit that there is no "optimal" resolution, it all depends on what one wishes to examine, the equivalent of tree patterns or forest patterns.

Some more recent work [6] employs multi-resolution data mining to obtain motifs at different resolutions in a time series allowing the user to navigate along the top K motifs. And our own work in semantic graph motif mining [7] may require a coarse-grained approach in order to scale.

Thus our second method of **ontological scaling** facilitates this by using an ontology to provide a scaling factor that can be dialed up or down. Higher and more abstract views are achieved by applying class and property inferences at different hierarchical levels in the ontology and recomputing the extant ontology with those inferences.

978-0-7695-4513-4 2011
U.S. Government Work Not Protected by U.S. Copyright
DOI 10.1109/WI-IAT.2011.241

IEEE computer society

We present our methodology, and then demonstrate the results of applying it to three different real-world semantic graphs: Uniprot [8], LUBM [9], and BTC [10].

II. Graph Dataset Filtering

A dataset containing semantic information, in RDF [11] triple format for example, can conceptually be viewed as a knowledge base of binary predicates represented as a directed, labeled (typed) graph. We refer to the knowledge base graph represented by the edge list of all the original the triples as G_{kb}. Each triple $\langle s, p, o \rangle$ in G_{kb} represents a directed edge with the label p from the subject node s to the object node o.

Semantic graph databases typically contain a large amount of supporting, ancillary, redundant, or simply extraneous information over and above the core interesting patterns intended to express relations between the modeled entities being represented. This "bloat" is both noise distracting from core patterns in visualization and discovery and also a computational burden (especially in graphs which are already on the order of billions of edges). So before building the extant ontology we need to identify and remove edges within G_{kb} that are not semantically significant. We do this in a number of stages:

1) **owl:sameAs Cliques:** Nodes in semantic graphs are intended to be unique, as reflected in their Uniform Resource Identifiers (URIs). The `owl:sameAs` predicate is used to identify distinct nodes which are considered to be equivalent within the semantic model. Since `owl:sameAs` is transitive, reflexive, and symmetric, groups of nodes connected by `owl:sameAs` form complete, disjoint subgraphs (cliques) which should be contracted into single meta-nodes.
2) **Ontological Information:** As noted above, many triples in a semantic graph database carry meta-information or typing information in the ontology. So we cast the semantic graph as $G_{kb} = G_{ins} \cup G_{ont}$, where G_{ont} constitutes the ontological sub-graph of G_{kb} consisting of meta-data assertions, and G_{ins} its "instance subgraph" consisting of data assertions.
   In our case, Uniprot and LUBM are provided with G_{ont} and G_{ins} already seperated; while BTC is much more noisy, and contains ontologies and instances in the same dataset. The procedure of separating out the ontologies from the instance graphs may not be trivial within an RDF dataset and there could be more than one ontology accompanying the G_{ins}.
3) **Reification Definitions:** Reification is a mechanism to represent additional information (meta-data) about a triple through materialization of additional nodes and links in G_{ins}. A common use of reification is the representation of provenance, as shown in Figure 1. Consider the triple $\langle$n781, uniprot:has_sequence, n329$\rangle$, indicating that protein n781 has sequence n329. We wish to record that this triple was created by the person represented by node n921.

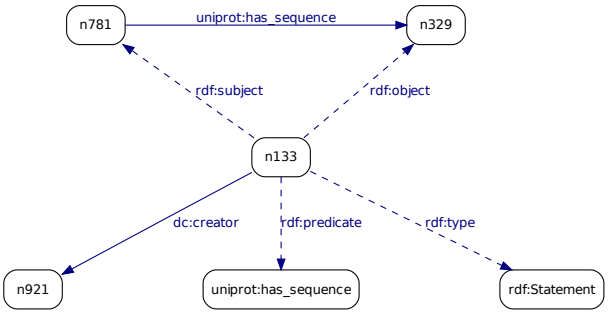

Figure 1. An edge requires four extra edges to define reification

These two information-carrying statements are represented by the solid arrows in Figure 1, but additionally it is required to create a new node n133 to indicate the first triple *itself* as a statement. The dashed arrows represent the statements which define the reification. Hence in order to add the one goal statement, reification instead resulted in the addition of a total of five statements, which are semantically redundant in the context of the model. It is not uncommon for reification to contribute up to 25% of the size of the graph (see table I).
Modern triple stores (e.g. Sesame [12]) provide named graph or context mechanisms that enable more compact representations of reification. When exploiting such mechanisms, pure triple representations would automatically exclude the reification statements. Other approaches such as a hybrid triple stores [13] store the meta-data in a conventional relational database model which is best suited for accommodating such uniform structures. Similarly, this approach automatically removes reification from the triple-based information.
4) **Literal Edges:** The instance graph G_{ins} is comprised of nodes which are either URIs, blanks, or literals. URI and blank nodes are nodes with unique global or local identifiers respectively. They constitute the core of the semantic graph by representing the entities they model, and have types with unique properties on which inference can occur. Literal nodes are string or integer values, not unique objects on which inference can be performed (they are terminals in the graph). We have found that removing literal nodes and terminal edges leading to them can eliminate up to 60% of the edges in the dataset (see table II).

III. The Extant Ontology

Assume an instance graph, G_{ins} which has been filtered as described in the previous section. We then compute a new graph, G_{ext} as an **extant ontology** of G_{ins}. Both graphs are

labeled and directed, but G_{ext} is additionally weighted with quantities (integers or relative frequencies).

Listing 1 shows the SPARQL [14] code for generating a simple extant ontology showing statistical frequencies (integer counts) on edges. As illustrated in figure 2, the algorithm iterates over all nodes s in triples $\langle s, \text{rdf:type}, C \rangle$, thus representing s as being in class C, and creating a single node for C in the extant ontology to represent that class. Whenever an edge $\langle s, p, o \rangle$ with label p is encountered in the instance graph between two nodes of types C_s and C_o, a corresponding edge with the same label and direction is created in the extant ontology between the nodes representing those classes. A number reflecting the frequency of these specific edges between those two specific types is updated and appended to the edge label in the extant ontology. Several nodes in the dataset may have more than one type in which case they contribute to more than one edge count and node label in the extant ontology.

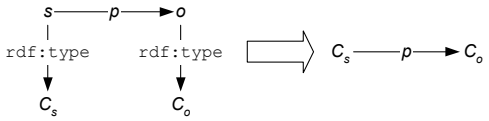

Figure 2. The mapping of triples in the instance graph to the extant

```
SELECT ?class1 ?p ?class2 (COUNT(?p)) as ?count
 from <BTC> {
        ?s ?p ?o.
        ?s a ?class1.
        ?o a ?class2.
}group by ?class1 ?class2 ?p order by desc(?count);
```

Listing 1. SPARQL code to generate a simple extant ontology

Figure 11 shows an example of a portion of the extant ontology we created from the 2010 ISWC bilion triples challenge, where edges are labeled with counts of that particular connection occuring between the two entities in the actual data. Relative frequencies i.e proportion of the overal number of predicates in the graph can be used as well.

IV. Ontological Scaling

Ontological scaling employs an ontology as a selector "knob" to decide on the desired filtering and scale (resolution) to visualize and analyze the data. Using a scaling ontology, subclasses can be replaced by immediate or higher-level super classes which results in an adjustment in the relevant edge types and counts connecting the nodes in the extant ontology. We illustrate the use of ontological scaling with example subgraphs. Consider the subgraph of the Uniprot extant ontology in figure 3, and that additionally we have a scaling ontology with the class hierarchy shown in figure 4, and relation hierarchy shown in figure 7. In this case we used the standard Uniprot ontology [8].

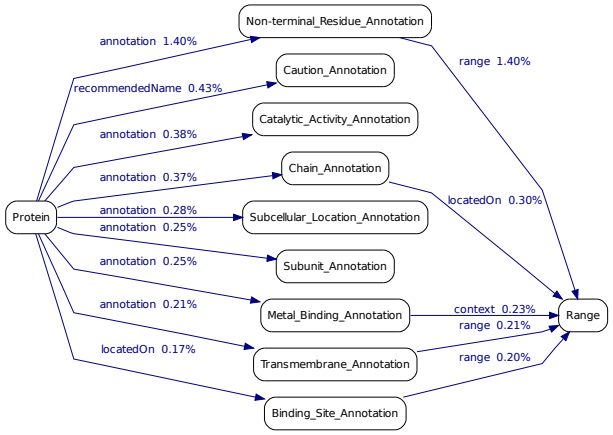

Figure 3. Example subgraph before inferring class subsumption

Applying subsumptive inference on instances in the class hierarchy and recomputing the extant ontology yields the graph in figure 5. Note the reduced number of edges after collapsing all the different subclasses of annotation into one super class, the adjustments of predicate percentages and the preservation of the semantics.

Similarily, we can summarize the extant ontology even further and create a higher abstraction view by applying property inferences such as subsumption and other inference such as symmetry. Consider another subgraph of the Uniprot extant ontology shown in figure 6 as part of the extant ontology and we wish to clear this up a little bit more to understand the relations better. Inference guided by the

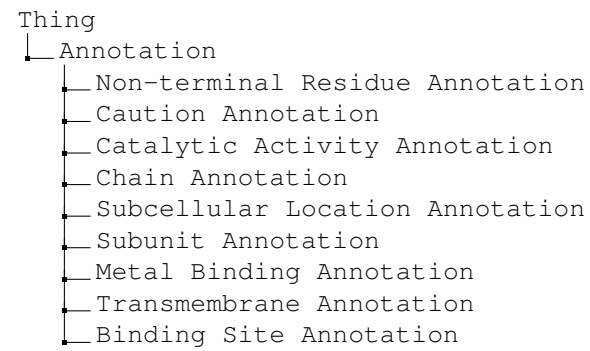

Figure 4. Class hierarchy used for ontological scaling

Figure 5. Subgraph of fig 3 after inferring class subsumption

scaling ontology replaces the edges alternativeName, submittedName, and recommendedName with an edges labeled with their super property: name. The percentage is adjusted as the sum of all the sub properties properties.

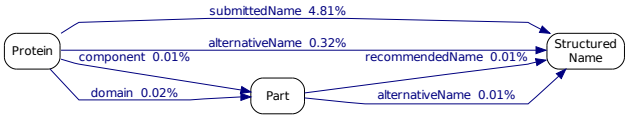

Figure 6. Example subgraph before inferring property subsumption

Ontological scaling can be conducted either uniformly to all terminals of the scaling ontology regardless of their depth so in each iteration the data view gets coarsened for all entities in the scaling ontology terminals, or it can be applied level by level, at depth N, $N-1$, and so on until the root node in which case the finer entities in the extant ontology get closer to their finer counterparts before uniform scaling hits the shorter branches of the tree in the scaling ontology. The choice between these two modes depend on what you are looking for. In the experiment we present here we opted for rolling inference in level by level. In the previous example we assumed that ontologies are represented as class hierarchies which are trees. Multiple inheritance may require additional treatment such as through the creation of extant nodes that represent compound types.

V. Datasets and Experimental Setup

We examine three publicly available semantic graph datasets with varying properties.

1) BTC2010: This dataset has been crawled from the public Web and contains anything from connected people to connected proteins and semantic Wikipedia entires. The data is available as part of the Billion Triples Challenge [10] held at the International Semantic Web Conference and is comprised of more than 3 billion quads, two fields to specify end nodes and one for the edge label. The fourth field is used to indicate the source of the triple and could be used for provenance (for example). After removing duplicates in the quads we found this dataset to contain 1.43 billion triples after ignoring the quad field and removing duplicates[15].
2) UNIPROT: This is the UniProtKB of the Universal Protein Resource dataset, a comprehensive repository of protein sequence and annotation data. This data has been converted to semantic formant by the UniprotRDF project [8]. The size of this dataset is 2.04 billion triples (graph edges). A few additional small datasets provided are excluded as ancillary.
3) LUBM8K: This is Lehigh University Benchmark and was developed to facilitate the evaluation of Semantic Web repositories in a standard and systematic way. The benchmark is intended to evaluate the performance of those repositories with respect to extensional queries over a large data set that commits to a single realistic ontology. It consists of a university domain ontology, generating customizable and repeatable synthetic data [9]. The 8K version contains information about 8,000 universities and consists of 1.07 billion triples.

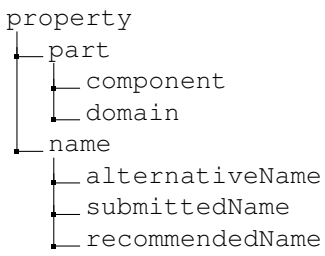

Figure 7. A relation hierarchy.

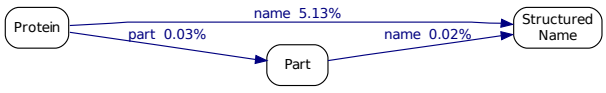

Figure 8. Subgraph of fig 6 after inferring property subsumption

LUBM is the most uniform and schema-compliant of these as it is generated synthetically. BTC is the most *ad hoc* as it is the result of a web crawl and contains many conflicting ontologies and even ontology fragments.

We loaded these datasets in a dedicated triple store running on a high-end server with 48 GBs of memory and two quad-core 2.96 GHz Intel Xeon CPUs. We were able to perform many of the basic experiments and statistics gathering on the data using SPARQL queries though several other scripts and data summarizing methods external to the triple store had to be deployed (for example in performing class and property inferences). More details on dealing with large triple stores can be found in our previous work [7], [15], [16], [17]

VI. Results

A. Initial Filtering

1) Reification: Of the 2 billion edges in the Uniprot graph, about 139K are reified. Since each refied statement requires four edges just to define the reification hook (see figure 1), this means there are $4 \times 138.7\text{M} = 554.9\text{M}$ such wasteful edges in the dataset. BTC2010 contains far fewer refied edges (6.05M) consuming only 24.2M additional edges to implement. LUBM is synthetic and no reification is included by design.

	# Original Edges (M)	# Reification Edges (M)	% Reification
BTC2010	1,400	24	1.7%
UNIPROT	2,040	555	27.2%
LUBM8K	1,070	0	0%

Table I
NUMBER OF EDGES USED TO DEFINE REIFICATION

2) Terminal Nodes and Edges: We removed all edges leading to literals in the three dataset we experimented with and discovered that this significantly reduced the size of the graphs in all cases. In the BTC, the reduction in the number of edges was about 63% while edges leading to terminal literal nodes constitute about 30% in the other dataset.

		Original (M)	No-Literals (M)	% Reduction
BTC2010	Edges	1,430	530	63%
	Nodes	281	221	21%
UNIPROT	Edges	2,040	1,400	31%
	Nodes	461	404	12%
LUBM8K	Edges	1,070	710	34%
	Nodes	263	174	34%

Table II
EFFECT OF REMOVING LITERALS ON THE GRAPH SIZE

B. Coverage of Types and Predicates

To support the use of our approach in creating the extant ontology and refining it further with class and predicate inferences, one needs to understand the spread of the usage of both the types and the predicates in each dataset. For LUBM8K and UNIPROT table III below shows that a few distinct types and predicates exist in the first place thus our approach which is based on typing and inference in both will be comprehensively representative of all these datasets.

	#Distinct Types	#Distinct Predicates
BTC2010	168K	95K
UNIPROT	119	110
LUBM8K	15	17

Table III
TOTAL DISTINCT TYPES AND PREDICATES IN EACH DATASET

Initial examination of the web-crawled BTC data shows that there are around 168K different types and 95K different predicates so one may wonder if so many types and predicates may not be narrowed enough to summarize such a large and diverse dataset. One could understand how representative the top N number of extant ontology edges are of their dataset by summing up their labeled percentages, however, in addition we examined not only the number of types and predicates in BTC but also their cumulative distributions as figures 10 and 9 show.

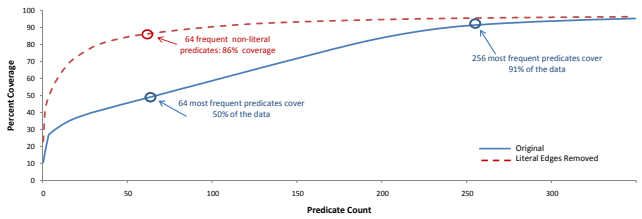

Figure 9. Cumulative Predicate Coverage in BTC

The solid line in figure 9 shows that before removing the terminal edge predicates, the 64 most common predicates cover 50% of the BTC data, while the top 256 predicates cover 91%. The dashed line is after excluding the terminal edges, where now the 64 top non-terminal predicates cover 86% of the data. This is significant in light of the dataset containing more than 95K different predicates.

Figure 10 shows that though there are 168K different classes in BTC, the top 16 most common ones of them cover 80% of the data and 64 classes are sufficient to cover 95% of the dataset.

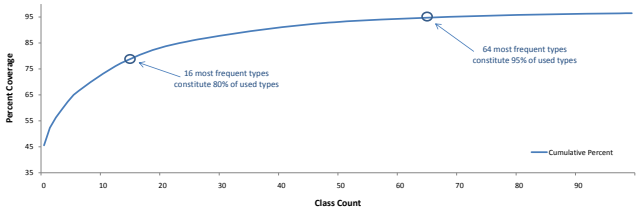

Figure 10. Cumulative Type Coverage in BTC

C. Building Extant Ontologies

We computed the extant ontologies for the three datasets: BTC10, LUBM8K, and UNIPROT. Figure 11 shows the top 30 edge counts in BTC10. There are about 70M triples with the predicate `foaf:known` connecting subject and object of type `foaf:Person`, the highest count. Some Uniprot edges are also present in BTC due web crawling.

The extant ontologies for LUBM8K and UNIPROT are shown in figures 12 and 13. For UNIPROT we truncated a portion of the image so it can be more readable in print. We are making the full .pdf and .dot graph visualization files for these figures available online[1]. The UNIPROT extant is effectively showing, in only 243 edges, the full semantic structure of a 2 billion edge graph.

[1]http://hpc.pnl.gov/people/sinan/wi11paper

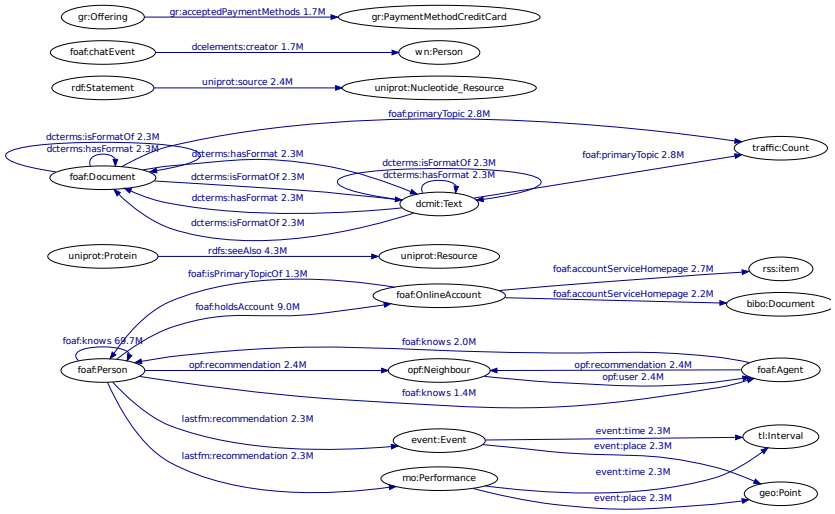

Figure 11. Extant Ontology for the Top 30 Link-node-types in BTC10

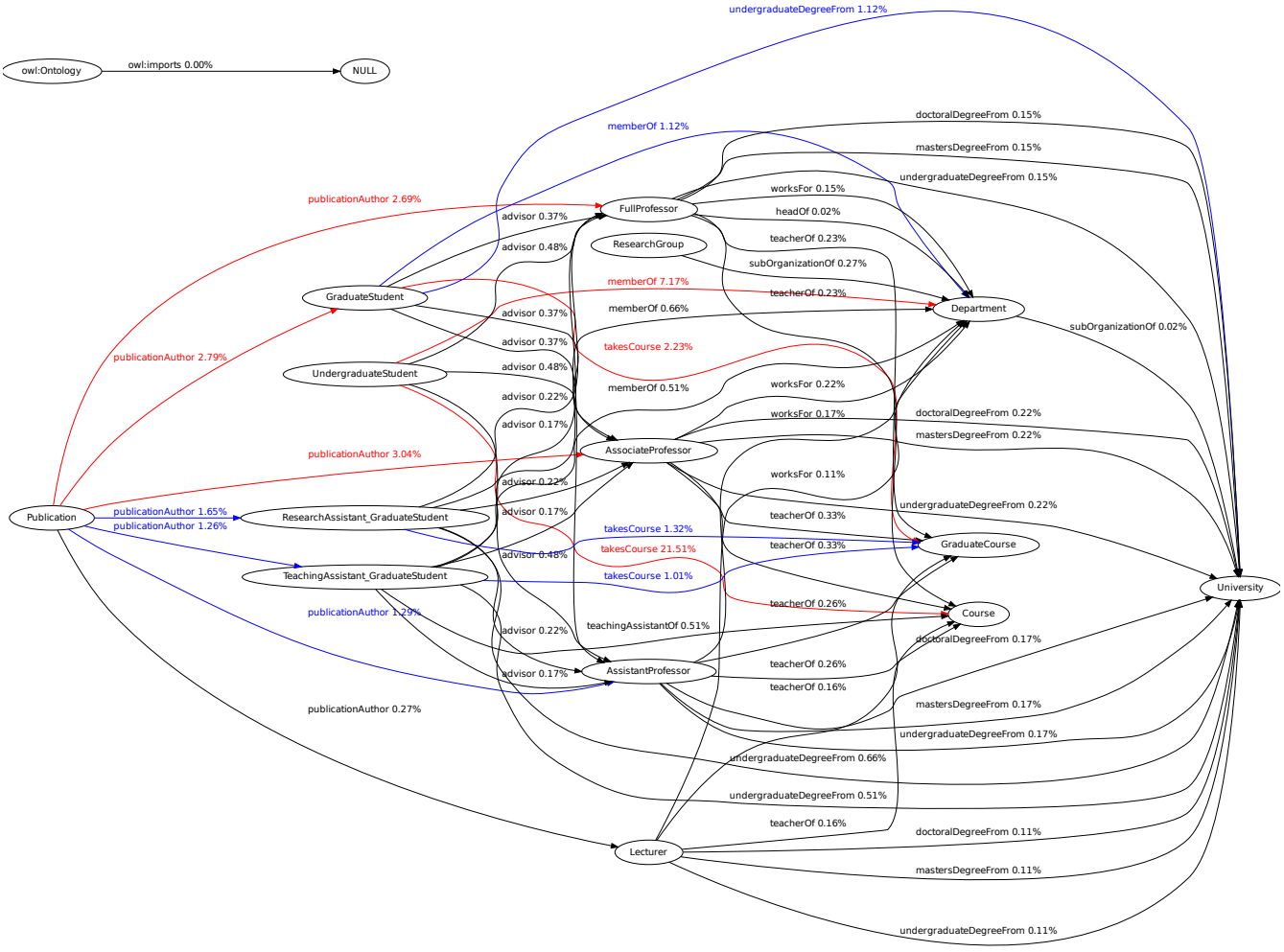

Figure 12. The full extant ontology for LUBM8K.

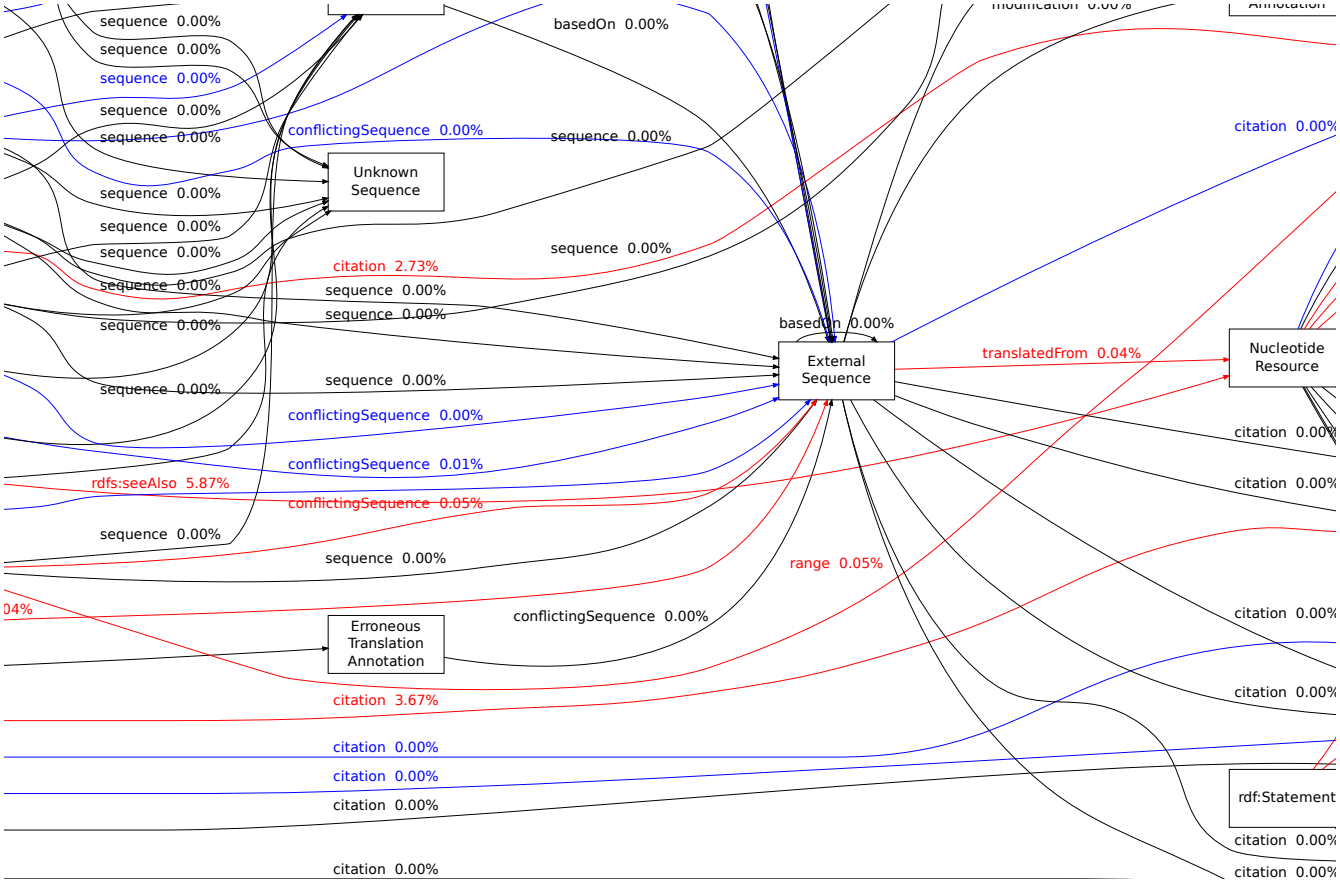

Figure 13. Part of the extant ontology for Uniprot - image truncated to scale for readability. See http://hpc.pnl.gov/people/sinan/wi11paper for full details

D. Ontological Scaling

Figure 14 shows the same extant ontology of figure 13 with ontological scaling applied using the class and relation hierarchies shown in figures 4 and 7, with three iterations for class inferences and one iteration for predicate inference. This is level 1 of the scaling ontology. Level 0 would contains a single node labeled with "Thing" and a single edge connecting that node to itself with the label "property" and a percentage of 100. Examining the ontologically scaled figure of UNIPROT reveals that about 52% of the edges are `rdfs:seeAlso` edges which is essentially equivalent to a regular web link so there is a need to produce dataset with even richer semantics.

VII. Conclusions

We presented two methods for summarizing and visualizing the contents of semantic graphs. We explained conceptually why they are useful and also demonstrated how they can be used to understand and interpret real datasets. The extant ontology represents a typed statistical view of the structure and patterns of connectedness between distinct types in the graph. Inference can be applied to a scaling ontology to decide on the level of detail present to the analyst whether this analyst is a human or another automated procedure. This is essential for optimizing resources in such big graphs and also necessary for visualizing the relevant components and reduce information overload. Furthermore the scaling ontology can be viewed as a controlling mechanism to decide on the level of trade-offs between computation and details of the results. We have shown that even in what may seem as diverse and huge datasets such as the BTC web crawl with hundreds of thousands of classes and predicates, only a very tiny portion of those is necessary to cover the entire dataset. This is good news for implementors of triple stores as they can optimize for these facts. For example implementing classes and terminal edges as node properties significantly reduces the graph size and the computational requirements to search and infer over it. However, this can be less exciting for application developers as the semantic datasets we examined seem to not yet be sufficiently covered with the diverse set of different predicates and classes needed to motivate some of the expected use cases.

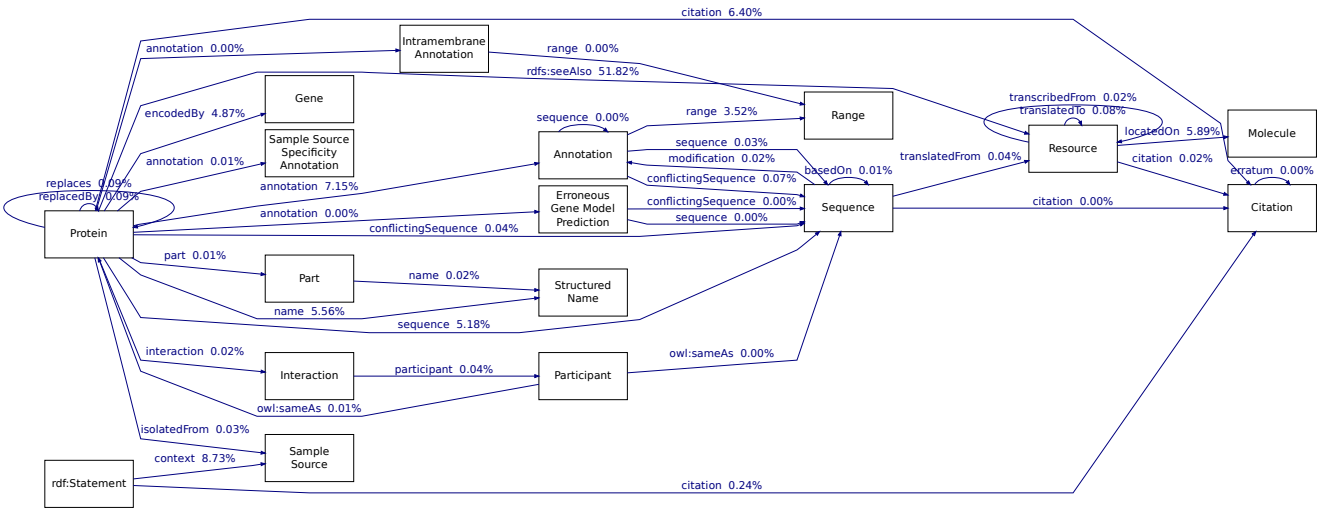

Figure 14. Uniprot extant from fig 13 after ontological scaling using inferences of class and property subsumptions up to level 1

REFERENCES

[1] J. F. Sowa, "Principles of semantic networks." Morgan Kaufmann, 1991.

[2] A. F. Parker-Rhodes, "Mechanical translation program utilizing an interlingual thesaurus," *Report on research*, 1956.

[3] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, no. 5, pp. 34–43, May 2001.

[4] C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, Eds., *Linked Data on the Web (LDOW)*, ser. CEUR Workshop Proceedings, no. 628, Aachen, Apr. 2010. [Online]. Available: http://CEUR-WS.org/Vol-628/

[5] G. Sheikholeslami, A. Zhang, and L. Bian, "A multi-resolution content-based retrieval approach for geographic images," *Geoinformatica*, vol. 3, pp. 109–139, June 1999.

[6] N. Castro and P. Azevedo, "Multiresolution Motif Discovery in Time Series," in *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, 2010, Columbus, Ohio, USA*. SIAM, 2010, pp. 665–676.

[7] C. Joslyn, S. al Saffar, D. Haglin, and L. Holder, "Combinatorial Information Theoretical Measurement of the Semantic Significance of Semantic Graph Motifs," in *Proceedings of the Mining Data Semantic Workshop (MDS 2011), SIGKDD*, 2011.

[8] "Uniprot RDF," Website, http://dev.isb-sib.ch/projects/uniprot-rdf/.

[9] "The Lehigh University Benchmark," Website, http://swat.cse.lehigh.edu/projects/lubm.

[10] "The 2010 ISWC Billion Triples Dataset," Website, http://challenge.semanticweb.org.

[11] "Resource Description Framework (RDF): Concepts and Abstract Syntax," W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[12] J. Broekstra, A. Kampman, and F. V. Harmelen, "Sesame: An architecture for storing and querying rdf data and schema information," in *Semantics for the WWW*. MIT Press, 2001.

[13] A. Chappell, P. Paulson, E. Stephan, and C. Joslyn, "Hybrid triple/relational knowledge store concepts," in *Technical Report PNWD-4230*. Pacific Northwest National Laboratory, Richland, WA, 2010.

[14] "SPARQL Query Language for RDF," W3C Recommendation 15 January 2008, http://www.w3.org/TR/rdf-sparql-query/.

[15] C. Joslyn, B. Adolf, S. al-Saffar, J. Feo, E. Goodman, D. Haglin, Greg Mackey, and D. Mizell, "High performance descriptive semantic analysis of semantic graph databases," *1st Workshop on High-Performance Computing for the Semantic Web at ESWC*.

[16] C. Joslyn, B. Adolf, S. al-Saffar, J. Feo, E. Goodman, D. Haglin, G. Mackey, and D. Mizell, "High performance semantic factoring of giga-scale semantic graph databases," in *Report on the 9th International Semantic Web Challenge PNNL submission*, November 2010.

[17] E. Goodman, E. Jimenez, D. Mizell, S. al-Saffar, B. Adolf, and D. Haglin, "High-performance computing applied to semantic web databases," in *The 8th Extended Semantic Web Conference*, 2011.