Visualizing Large-Scale RDF Data Using Subsets, Summaries, and Sampling in Oracle

Seema Sundara, Medha Atre[#+], Vladimir Kolovski, Souripriya Das,

Zhe Wu, Eugene Inseok Chong, Jagannathan Srinivasan

Oracle
One Oracle Drive, Nashua, NH 03062, USA
[#]`atrem@cs.rpi.edu`
`firstname.lastname@oracle.com`

***Abstract*—The paper addresses the problem of visualizing large scale RDF data via a *3-S* approach, namely, by using, 1) *Subsets*: to present only relevant data for visualisation; both static and dynamic subsets can be specified, 2) *Summaries*: to capture the essence of RDF data being viewed; summarized data can be expanded on demand thereby allowing users to create hybrid (summary-detail) fisheye views of RDF data, and 3) *Sampling*: to further optimize visualization of large-scale data where a representative sample suffices. The visualization scheme works with both asserted and inferred triples (generated using RDF(S) and OWL semantics). This scheme is implemented in Oracle by developing a plug-in for the Cytoscape graph visualization tool, which uses functions defined in a Oracle PL/SQL package, to provide fast and optimized access to Oracle Semantic Store containing RDF data. Interactive visualization of a synthesized RDF data set (LUBM 1 million triples), two native RDF datasets (Wikipedia 47 million triples and UniProt 700 million triples), and an OWL ontology (eClassOwl with a large class hierarchy including over 25,000 OWL classes, 5,000 properties, and 400,000 class-properties) demonstrates the effectiveness of our visualization scheme.**

I. INTRODUCTION

Resource Description Framework (RDF) is a W3C standard [1] to describe metadata about web resources or resources in general. The canonical (subject, predicate, object) triple format is amenable to storing any kind of data in RDF format. Although this format promotes interoperability across diverse data sources, the RDF encoded data tend to be verbose in nature. As a consequence, RDF datasets are typically large (e.g., Wikipedia 47 million triples, UniProt 700 million triples [24], etc.).

Visualization of such large scale RDF data is the focus of this paper. Visualization allows the user to gain new insights into the data, which may not occur otherwise. The emphasis here is on discovery by *serendipity*, which is, finding valuable or agreeable things not sought for.

The challenge is how to visualize RDF data, especially of a large scale. A majority of approaches [3, 4, 5] present RDF data as a collection of *facets* (categories). Facets are useful in general but they de-emphasize the relationship-centric nature of RDF data. Thus, a graph-based visualization of RDF data would be desirable to display inter-relationships. However, graph-based visualization of RDF data (such as [6, 7, 8]) has two problems:

Cluttering the Display: Although the state-of-the-art visualization tools (such as Cytoscape [10]) can display graphs of even up to 150,000 objects (nodes + edges), a graph of over 500 objects tends to clutter the standard display of million pixels.

Large Response Time: Also, the larger is the graph to be displayed, more is the time needed to generate a suitable layout and render the display.

In this paper, we present a *3-S* approach that addresses the above problems by using:

Subsets: The ability to show only subsets of a graph. The subsets could be either static subsets such as relationships among instances of a class, or dynamic subsets such as a result of a SPARQL query [2]. However, the subsets themselves could be quite large.

Summaries: The ability to replace portions of the graphs by their respective summaries, which can be subsequently expanded incrementally on demand. This, in effect, allows us to present a *fisheye view* [9] of the dataset consisting of a mixture of portions shown in detail along with portions shown in a summarized form. Such a graph, referred to as a *hybrid* (*detail-summary*) *graph,* leads to a compact representation of the data.

Sampling: If a user is interested in a small representative subset of a large instance collection, sampling may be used. We provide an option to visualize a sample of the graph using various *node-based* and *edge-based* sampling methods.

Example 1: Consider RDF data set for a tiny social network (namespace `sn`). If we are interested in a RDF subset consisting of one hop neighborhood of `sn:John` expressed via SPARQL pattern {`sn:John ?p ?o`}, we obtain the graph shown in Figure 1-a. However, the same information could be displayed in a summarized manner by the graph shown in Figure 1-b. Specifically, the three `#type` edges are replaced by a single *aggregate edge* (`#type`) with the count of 3 and so on.

Furthermore, the visualization scheme comes with a set of incremental graph expansion primitives (including *expand property,* and *expand node*) that allows users to selectively

+ Work done as summer intern at Oracle. Currently, pursuing PhD at Dept. of Computer Science, Rensselaer Polytechnic Inst., Troy, NY.

expand portions of the summaries, thereby creating fish eye views of their interest and control what is viewed without the clutter. Specifically, *expand property* would expand the selected aggregate edge to display the corresponding set of (detail) edges. Figure 2-a shows the result of performing *expand property* on the aggregate `#type` edge of Figure 1-b. Similarly, *expand node* would expand the selected node to display the one hop neighborhood summary for that node. Figure 2-b shows the result of performing *expand node* on `#Jill` node of Figure 2-a. In this manner, a user can incrementally create a fisheye view. For the same RDF dataset or its subsets, different fish eye views could be created, which shows different portions of interest in detail.

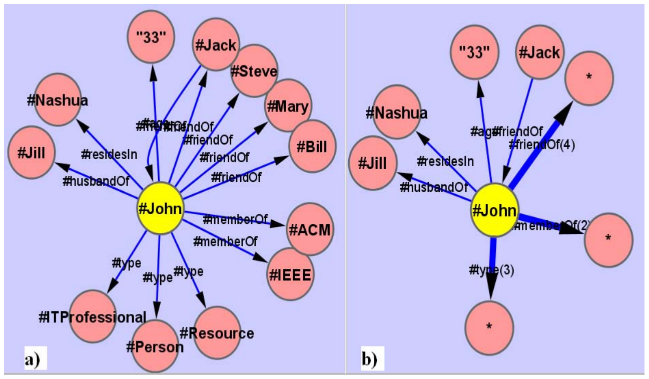


Figure 1. Detailed (1-a) vs. summarized representation (1-b) of a RDF subset

Among the graph expansion primitives, *expand node* adds a small number of triples as it contains summarized information, whereas *expand property* could potentially add a large number of triples to the graph. To avoid cluttered display as a result of this operation, we support *selective expand*, which allows users to specify a SPARQL graph pattern to restrict the property expansion (for example, instead of expanding all `#friendOf` edges of `#John`, user could indicate expanding a subset of them, such as *those friends of John who reside in Nashua*).

The incremental graph expansion operations on a hybrid graph can be viewed as an implicit union of the current displayed graph and the graph returned by the specific expansion primitive. Since the hybrid graph contains aggregate edges with count information, the union operation has to adjust the count appropriately. In general, two independently created fisheye views can be merged using the three graph primitives, namely, *union*, *intersection*, and *difference*. The semantics of these operations on hybrid graphs are covered in Section III.C. These operations will potentially allow collaboration and reuse of fish-eye views among users.

We also support *undo/redo* of incremental graph expansion primitives thereby providing the flexibility of moving back and forth between different fisheye views capturing different levels of detail. The undo/redo operation provided is *physical* in nature, supported by storing the graph prior/post to the operation. One could also support *logical* undo/redo operations by supporting inverse of graph expansion primitives. However, their use in general is somewhat limited as discussed in Section III.

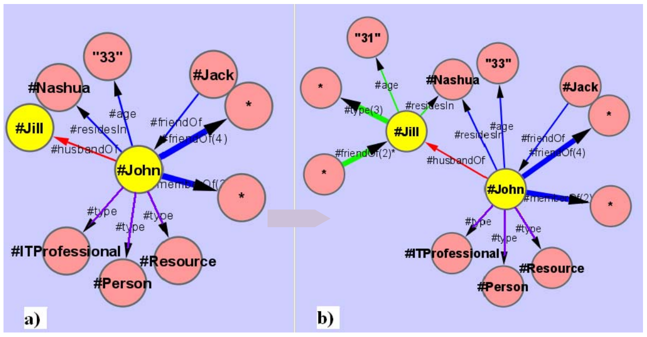


Figure 2. Expand property (2-a) and expand node operation (2-b) on a summarized RDF subset

Lastly, we propose storing the hybrid (summary-detail) fisheye views in RDF store for subsequent usage. For this purpose, we define a RDF based encoding scheme to encode a hybrid graph as a RDF graph. The advantage is that the hybrid graph can be directly visualized from this encoded representation. Furthermore, user can perform queries over such hybrid graphs using the standard SPARQL query language.

The RDF visualization scheme handles both graphs consisting of asserted triples as well as inferred triples. Generating subsets or summarized graphs over an entailed graph (asserted plus inferred triples) is straightforward. However, creating a sample from the entailed graph poses challenges as sampling the entailed graph can result in an *unsound* entailed graph. Specifically, the sampled graph can contain an inferred triple that cannot be derived from the asserted triples present in the sample. Hence, we employ the strategy of sampling the original graph followed by the inference. This strategy needs further modification as with this approach one could get back an *incomplete* entailed graph. E.g., a schema triple such as (`sn:siblingOf rdf:type owl:TransitiveProperty`) can be sampled out preventing inference of triples based on transitivity. Thus, we retain all schema triples and sample only instance triples. We report the efficacy of sampling methods via experiments in Section V.

Overall, the proposed 3-S scheme provides a mechanism for reducing the clutter in the typically common million-pixel display. Furthermore, we are able to provide *interactive response time* for all operations starting from the initial display of summarized graphs to incremental graph expansion primitives and undo/redo operations. We achieve this by a combination of building a pre-computed set of summaries that reduce the run-time overhead and by optimizing the processing of the common set of SPARQL-like queries against the backend database.

This visualization scheme is implemented as a Java plug-in to Cytoscape, a state-of the art graph visualization tool [10]. For fast and optimal access, a SEM_ANALYSIS PL/SQL package is implemented that handles efficient processing of the common set of SPARQL-like queries against the back end Oracle Database, which stores the RDF/OWL data. Oracle

Database has native support for storing, inferring and querying semantic data [11]. Queries are expressed via SPARQL-like pattern in SQL using Oracle's SEM_MATCH SQL Table function [12].

We experimented with a variety of datasets, including a synthesized LUBM RDF data set, two native RDF datasets (Wikipedia and UniProt), and an eClassOWL ontology [25] that demonstrate the effectiveness of our visualization scheme (see Section V) both in terms of reducing the visual clutter as well as in providing interactive response time.

II. RELATED WORK

The RDF data visualization problem (and in general large graph visualization) has been studied extensively in recent years [3-9, 13, 15-23]. As mentioned in Section I, the tools developed for this purpose can be broadly categorized as non-graph based and graph based approaches.

Non-graph based approaches (such as /facet [3], Haystack [4], mspace [5]) present data mostly in terms of facets, categories or dimensions. In fact, authors in [13] argue that these non-graph based approaches may be more suitable than the traditional approach of representing the RDF data as a big fat graph. Although we recognize the benefit of a non-graph based representation, we argue that the graph structure is suitable for cases where interrelationships are important and need to be visually depicted. Specifically, facets make the relationships implicit whereas graph based representation maintains the relationship explicitly. Also, our contention is that uncluttered and interactive graph-based RDF visualization is possible by using a scheme (such as our 3-S approach) that does not resort to mapping the entire RDF dataset to a single big graph.

In this sense, our approach is closer to graph-based approaches (such as RDF Gravity [6], Welkin [7], IsaViz [8], GViz [23], Ask-GraphView [22]), which visualize RDF data as a *node-link* diagram. *Adjacency matrix* based visualization has also been explored, e.g., Zoomable Adjacency Matrix Explorer (ZAME) [21] can visualize a large dataset comprising of 0.5 million nodes and 6 million edges with interactive response time. Among node-link based systems, Ask-GraphView can visualize large graphs ranging in size up to 16 million triples. The system uses clustering algorithm to automatically generate a hierarchy on the graph, which can be interactively navigated in a top-down manner by expanding individual clusters. Although graphs in the range of millions of edges can be visualized, these systems still fall short as a solution for visualizing large-scale publicly available RDF data. In this regard, the approach adopted in GViz is desirable, which allows customization of the visualization process to meet the user needs.

Similarly, a *fisheye view* approach is promising where the data away from focus (or point of interest) could be either dropped altogether from the view [9] or distorted to occupy very little space [15]. Also, there has been work on using *compound fish eye views* [16] to visualize graphs, which combines aggregations at multiple levels in a single view.

Our approach of building fisheye views is to allow aggregation at two levels (*instance summary* and *instance set summary* graphs as discussed in Section III) to be embedded in a single view, which can be further expanded using the graph expansion primitives. The approach of incrementally expanding from an aggregated view is also used in interactive visualization of large OWL instance sets [17]. This system visualizes an individual and its relationships (properties) and allows property fillers associated with individuals to be expanded on demand. In addition, the cluster associated with a property filler can also be expanded. This visualization scheme always visualizes the entailed graph and provides the capability for a transitive property to expand direct property fillers or all (directly and indirectly related) individuals. In our approach, we provide the capability of visualizing both the asserted RDF graph and the entailed graph (asserted plus inferred triples) and let user choose the appropriate view.

We also employ sampling to handle large-scale visualization. Sampling methods and their goodness in terms of how well they represent the original graph are studied in [18]. Our sampling scheme exploits the semantics of RDF and we study different types of sampling methods (including stratified sampling). Also, we evaluate the sampling methods by taking RDF/OWL inference into account.

Our aggregation technique essentially relies on GROUP BY operations performed against the backend Oracle Database. Summaries have been used extensively in OLAP applications. However, we use summary as a means of achieving compact visualization and in that sense it is similar to the idea of *aggregate markers* discussed in [19].

Apart from the large scale RDF visualization problem, there has been work to visualize heterogeneous data better by automatically deriving spatial and time attributes for resources [20]. This complements our work and could be incorporated into our visualization scheme.

III. RDF GRAPH VISUALIZATION

A. *Terminology and Assumption*

A RDF/OWL based dataset can be viewed as a *RDF/OWL graph* with individual triples of the form (subject predicate object). For example, one could assert (`sn:John sn:friendOf sn:Jack`), where `sn` is the namespace defined for social network dataset. The subjects can be named resources (URIs) or unnamed resources (blank nodes), predicates can be URIs, and objects can be URIs, blank nodes, or literal values. RDF(S) and OWL are more expressive vocabularies layered on top of RDF. For brevity, the term RDF will include both RDF(S) and OWL datasets. Since both schema as well as instance information can be stored as triples in the same dataset, the triples pertaining to schema information (e.g. `sn:age rdfs:range xsd:integer`) will be referred to as *schema triples* and the rest as *instance triples*. Also, the term *graph* would refer to asserted triples only and *entailed graph* would refer to asserted plus inferred triples.

The SPARQL basic graph pattern [2], expressed as a set of triples, would be used to identify a subset of graph. For example, `{?s rdf:type sn:ITProfessional . ?s ?p ?o}` is a conjunction of two conditions: any resource of type `sn:ITProfessional` and all triples pertaining to those resources.

Since RDF has the notion of classes and instances (individuals), predefined subsets such as instances of a class would be referred to as *static subset*, whereas a subset obtained as a result of SPARQL query would be referred to as *dynamic subset*. This distinction is made because visualization of static subsets can be speeded up by pre-computing auxiliary information.

We make *unique name assumption* for summarization purposes, that is, different names (URIs and blank nodes) will be treated as different resources.

B. Basic RDF Visualization Elements

The basic RDF visualization elements are as follows

1) Detail Graphs: The simplest visualization option is to show the complete set or a subset as a RDF graph. This is referred to as a detail graph. For example, Figure 3 shows the complete graph for the sample social network and a subgraph identified by SPARQL graph pattern `{?s sn:friendOf ?o}`.

2) *Summary Graphs:* Next, the summary graph element is introduced to better utilize the visual space. Note that the summarization is done on subsets of the RDF dataset.

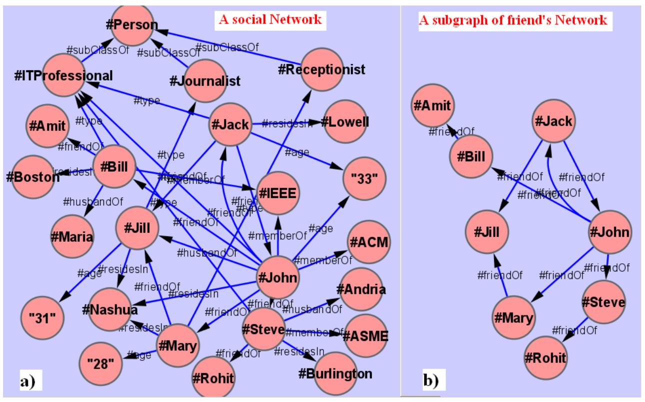


Figure 3. Detail graph for a sample social network (3a) and a subgraph identified by the “friendof” SPARQL pattern (3b)

Specifically, three types of summary graphs are employed in RDF visualization:

- *Instance set summary*: The property counts for instance set corresponding to a class are summarized capturing *one hop neighborhood*. For example, the neighborhood for `sn:ITProfessional` class instances in sample social network example identified by UNION of the results of SPARQL graph patterns `{?s rdf:type sn:ITProfessional . ?s ?p ?o}` and `{?s rdf:type sn:ITProfessional . ?x ?p ?s}` is represented in summarized form in Figure 4-a.
- *Instance summary*: Similarly, the property counts for one hop neighborhood of an instance can be summarized. E.g., neighborhood summary for instance `sn:Jack`, identified by UNION of the results of the SPARQL graph patterns `{sn:Jack ?p ?o}` and `{?s ?p sn:Jack}`, is represented in summarized form in Figure 4-b. Such a graph typically contains *aggregate edges* (property count>1), and *detail edges* (property count=1). Also, for detail edges, the target node is shown by default. In addition, we show *absent edges*, that is, those properties, which are specified for at least one other instance of that class but not specified for the currently selected instance.
- *Node summary:* This generalization of instance summary graph allows visualization of summary for any node. The only difference is that such a graph does not contain any absent edges.

The instance set summary graph always results in compact visualization since number of properties for a class is typically small. However, instance (node) summary graph may not necessarily compact the graph for cases when the instance (node) has *all* single valued properties. However, the large number of neighbors for an instance (node) typically occurs due to the occurrence of multi-valued properties (such as `#friendOf` property in Figure 4-b), which is aggregated into a single edge in our visualization scheme.

A model entailed with OWL semantics could result in generation of equivalences inferred via `owl:sameAs`, `owl:equivalentClass`, and `owl:equivalentProperty`. Currently, our visualization scheme displays inferred triples verbatim. However, in future we plan to provide an option to collapse such equivalences, if so desired, and support summarization based on collapsed representation of the graph.

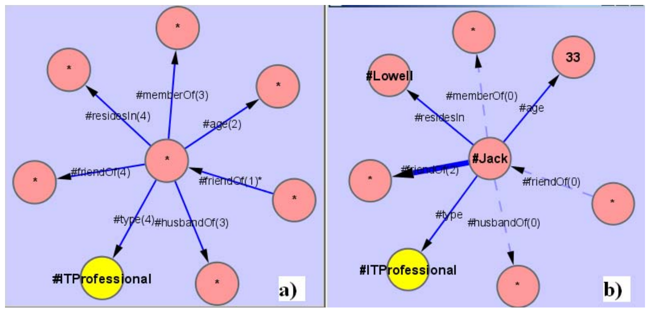


Figure 4. Instance Set and Instance Summary Graph

3) *Hybrid graphs:* A summary graph in general is a hybrid graph in that it contains both aggregate and detailed edges. Such a graph can be further expanded using graph expansion primitives. Expansion of a hybrid graph can be viewed as a union of the original graph with the result of the graph expansion primitive. Such an operation would typically require re-adjusting the aggregate counts as some of the aggregated edges may now be shown as a detailed edge. In general, two hybrid graphs can be merged using union, intersection, and difference operations. However, the presence of aggregate edges requires working out the semantics of these operations, which are covered in Section III.C.

C. Semantics of Hybrid Graph Operations

The semantics need to be defined when the same node occurs in the two hybrid graphs being merged and the node been expanded (i.e., has summarized neighborhood) in both the graphs. Since summarized neighborhood of such a node in a hybrid RDF graph represents a subset of RDF data with only additional transformation being folding of some of its detailed edges into aggregate edges, the neighborhood of a node in two graphs can differ from one another only in the extent of detail (the actual triples). Thus, sum of the aggregated edge's value count and the number of detailed triples for the same expanded node across the two hybrid graphs is same (*neighborhood summary invariant*).

A few notations are introduced to help in specifying the semantics of union, intersection, and difference operations on a hybrid graph. For a given node `N` and a given property `P` in one direction (either outgoing or incoming), assuming outgoing direction without loss of generality in hybrid graph `G`:

- $\mathtt{EDGES_{all}(G,N,P)}$ is the set of all outgoing `P`-edges from node `N`. It corresponds to the actual RDF triples stored in the database.
- $\mathtt{EDGES_{detail}(G,N,P)}$ is the set of detailed outgoing `P`-edges from `N`. It can be represented as a set like {a,b,c} where a,b,c are nodes reachable from the node N with one-hop `P`-edge.
- $\mathtt{EDGES_{agg}(G,N,P)}$ is the aggregate edge `P` at node `N`. It can be represented as `c`, where `c` is a value count.
- $\mathtt{EDGES_{aggSet}(G,N,P)}$ is the set of outgoing `P`-edges from `N` that are in $\mathtt{EDGES_{all}(G,N,P)}$, but not in $\mathtt{EDGES_{detail}(G,N,P)}$. It is the expanded version of `P`-edges in the set $\mathtt{EDGES_{agg}(G,N,P)}$.
- `aggCount(G,N,P)` is the value count of `P`-edge in the aggregated form.

By the definitions above, `EDGES(G,N,P)`, the summarized representation for node `N` for a property `P` can be represented by $<\mathtt{EDGES_{agg}(G,N,P)},\ \mathtt{EDGES_{detail}(G,N,P)}>$.
If a node `N` in hybrid graph `G` has been expanded to show its one-hop neighborhood, then the following must hold true:

- $\mathtt{EDGES_{detail}(G,N,P)\ UNION\ EDGES_{aggSet}(G,N,P)\ =\ EDGES_{all}(G,N,P)}$
- $\mathtt{EDGES_{detail}(G,N,P)\ INTERSECT\ EDGES_{aggSet}(G,N,P)=\ \{\}}$
- $\mathtt{aggCount(G,N,P)\ =\ |EDGES_{aggSet}(G,N,P)|}$

If the aggregate edge for `P` gets expanded based on some criteria (e.g., "friends located in Nashua"), additional detailed edges show up in the hybrid graph resulting in corresponding reduction in the value count associated with the aggregate edge for `P` to maintain the invariants shown above.

UNION: Let union of graphs `G1` and `G2` produce the resulting graph `R`. For any common node `N` that has been expanded in both graphs and a property `P`:

- $\mathtt{EDGES_{detail}(R,N,P)=\ EDGES_{detail}(G1,N,P)\ U\ EDGES_{detail}(G2,N,P)}$
- $\mathtt{aggCount(R,N,P)=\ aggCount(G1,N,P)\ -\ |EDGES_{detail}(G2,N,P)\ MINUS\ EDGES_{detail}(G1,N,P)|}$

Example (Figure 5): If `EDGES(G1,N,P)= <20,{a,b}>` and `EDGES(G2,N,P) = <19,{b,c,d}>` then

`EDGES(G1,N,P) UNION EDGES(G2,N,P)`
`= <18,{a,b,c,d}>.`

INTERSECTION: The resulting hybrid graph `R` computed as merge via intersection of graphs `G1` and `G2` has the following properties for any node `N` expanded in both `G1` and `G2` and a property `P`:

- $\mathtt{EDGES_{detail}(R,N,P)\ =\ EDGES_{detail}(G1,N,P)\ \cap\ EDGES_{detail}(G2,N,P)}$
- $\mathtt{aggCount(R,N,P)=\ aggCount(G1,N,P)\ +\ |EDGES_{detail}(G1,N,P)\ MINUS\ EDGES_{detail}(G2,N,P)|}$.

Example: If `EDGES(G1,N,P) = <20,{a,b}>` and `EDGES(G2,N,P) = <19,{b,c,d}>` then

`EDGES(G1,N,P) INTERSECTION EDGES(G2,N,P) =`
`<21,{b}>.`

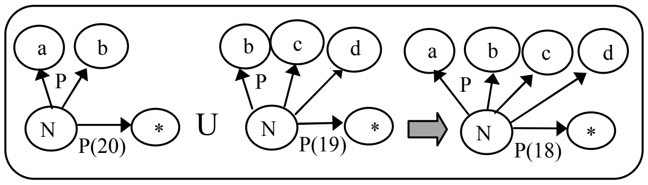


Figure 5. Hybrid Graph Union operation

MINUS: Semantics for the merge operation via difference (or minus) is shown below. The resulting hybrid graph `R` computed as merge via difference of two graphs `G1` and `G2` has the following properties for any node `N` expanded in both `G1` and `G2` and a property `P`:

- $\mathtt{EDGES_{detail}(R,N,P)\ =\ EDGES_{detail}(G1,N,P)\ MINUS\ EDGES_{detail}(G2,N,P)}$
- $\mathtt{aggCount(R,N,P)=\ aggCount(G1,N,P)\ -\ (aggCount(G2,N,P)\ +\ |EDGES_{detail}(G2,N,P)\ MINUS\ EDGES_{detail}(G1,N,P)|)}$

Note that the additional detail in G1 (if any) compared to G2 is reflected in the $\mathtt{EDGES_{detail}(R,N,P)}$ and additional detail in G2 (if any) compared to G1 is being accounted for in computing `aggCount(R,N,P)`.

Example: If `EDGES(G1,N,P)= <20,{a,b}>` and `EDGES(G2,N,P)= <19,{b,c,d}>` then

`EDGES(G1,N,P) MINUS EDGES(G2,N,P)= <-1,{a}>.`

Note that the neighborhood summaries for a node N in two graphs represent the same set of edges is reflected by the fact that the sum of `aggCount(R,N,P)` and $\mathtt{|EDGES_{detail}(R,N,P)|}$ is indeed 0.

For the case when one of the graphs does not have the expanded node, the semantics is straightforward as it may only contain detailed edges. Similarly, if both the graphs have an unexpanded node N, then merge via union, intersection, or difference would follow the traditional graph merge semantics.

D. RDF Graph Views and Operations

The visualization consists of multiple views and operation on individual views (Figure 6) as described below.

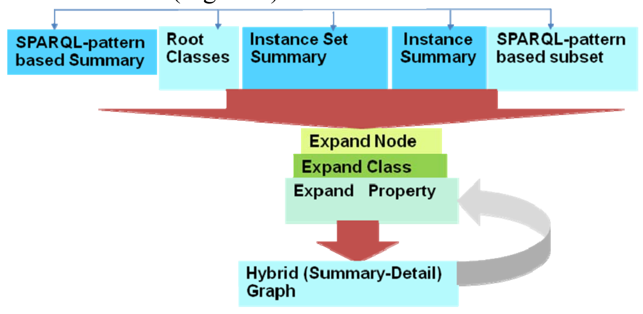


Figure 6. RDF visualization operations overview

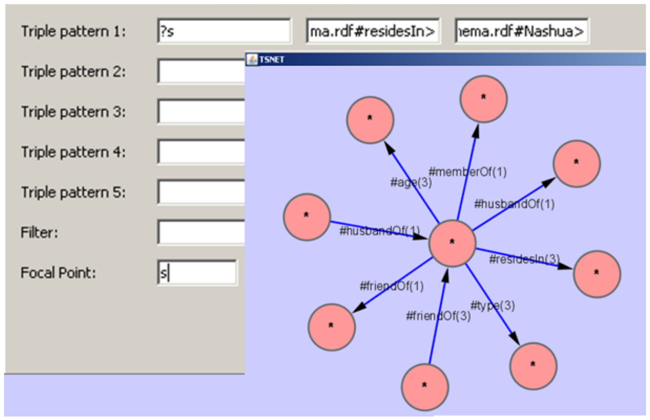
1) Root View: An RDF database could have multiple semantic models, each of which could be expanded to include inferred triples using RDF(S) and OWL. Also, for large datasets, users could be interested in visualizing a sample. Thus, the visualization tool allows a user to specify the semantic model, whether or not the inferred triples should be included in the data set, and the sample size (set to 100% by default) of the data to be viewed. Details of the sampling methods are covered in Section III.F

Even with sampling, the dataset chosen could have millions of triples. Given the size of this dataset and the limitation of visualization tools, it is not feasible to load the entire selected data into the viewer. Hence, as a starting point for viewing/browsing, only root classes (w.r.t. class hierarchy) in the schema of the dataset are selected and displayed. The user can proceed to expand the root nodes to find the area of interest. In addition, user can expand the graph using graph expansion primitives. The resulting graph can be displayed in a new window or included in the current display by turning the `overlay` feature off and on respectively.

2) View Metadata Operations: Certain operations are provided to return information about the metadata of the model. The `get_demographics()` method returns the demographics for the model (such as the number of distinct subjects, predicates, objects, and triples in the model). Another function, `get_schema_demographics()` returns the information about the schema level statistics of the model (such as the total number of subclasses, the maximum fan out of a node in the graph). Also, at any time multiple views could be opened to view the same or a different semantic model. The `get_view_metadata()` function is provided to give information about the selected view, i.e. the model, rule base being used for displaying the entailed graph, the sample size selected, and whether the overlay feature is on/off.

3) *Subsequent Views:* Once a semantic model of interest is loaded and its root classes displayed, the user can either proceed with one of the root nodes as a starting point, or obtain the subset of nodes that the user is interested in. In the latter case, the user can specify a SPARQL-like query pattern to identify the triples of interest, which generates the corresponding sub-graph via `get_subset()` method. For example, if the user is interested in only friend's network then the user can specify the query pattern `{?s sn:friendOf ?o}` (Figure 3-b).

Furthermore, user can also visualize summarized information for a subset by additionally specifying a focal point of interest via `get_subset_summary()` method. For example, to obtain summarized information for people residing in Nashua, user can specify query pattern `{?s sn:residesIn sn:Nashua}` with focal point `s` (Figure 7). This results in displaying property counts obtained via the following query pattern `{?s sn:residesIn sn:Nashua . ?s ?p ?o}` summarizing the neighborhood for set of resources identified by the focal point variable.



Triple pattern 1:	?s	ma.rdf#residesIn>	hema.rdf#Nashua>
Triple pattern 2:			
Triple pattern 3:			
Triple pattern 4:			
Triple pattern 5:			
Filter:			
Focal Point:	s		

Figure 7. Summary for a dynamic RDF subset

4) *Node and Property Level Operations:* Once the initial set of nodes and edges are displayed in the visualizer, the model can be explored further via graph expansion primitives. The function `get_instance_set_summary()` gives a summarized representation for instances of a given class. The various properties that the instances of a given class could have are represented in the summary. Both incoming and outgoing property edges are displayed, with a count being associated with each edge that indicates the number of instances having the specific property (Figure 8, bottom-right). Similarly, `get_instance_summary()`, gives the summary information for particular instance of the class (Figure 8, bottom-left). Various incoming and outgoing edges indicate the properties of the chosen instance.

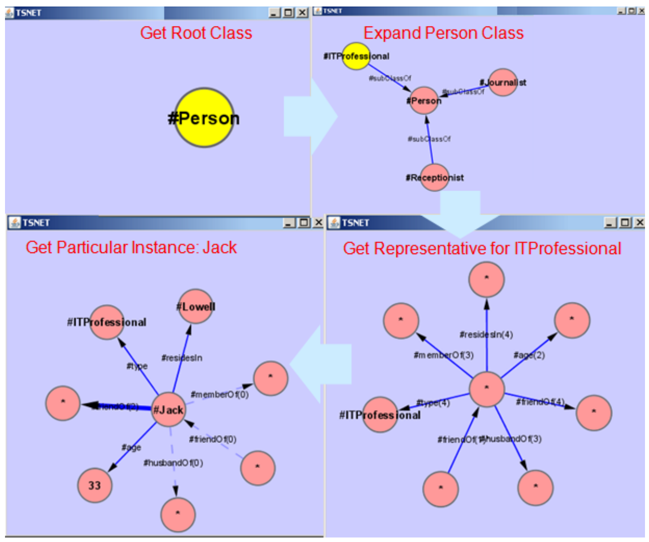


Figure 8. Browsing Social Network RDF dataset

The function `expand_class()` returns all the immediate subclasses of the selected node. The function `expand_node()` returns the node (neighborhood) summary. In order to view the expanded property list for aggregate edges, the `expand_property()` is provided. It expands and shows the individual property edges (Figure 2-a).

5) *Undo/Redo Operations:* As the semantic model is browsed and explored using a series of `expand_class()`, `expand_node()` and `expand_property()` operations, the visualizer supports undo/redo of these operations. This gives the flexibility to explore certain expansion paths and then undo those steps if there is no sub-graph of interest. There is no specified limit to the number of undo/redo operations that can be performed.

6) *Collapse Node/Edge:* As the graph is explored via a series of `expand_node()`, `expand_class()` and `expand_property()`, the number of triples could grow to a point where they start to clutter the visual display. It is possible to use physical undo/redo to revert back to a previous state (with collapsed nodes and edges) and then selectively expand only the sub graph of interest. However, we can also support logical undo/redo to collapse the expanded nodes, classes and properties, to reduce the size of the graph. The challenge here is to collapse the neighborhood sub-graph pertaining to selected node/property edge. We plan to adopt a conservative approach of collapsing the neighborhood sub-graph nodes only if the nodes are the terminating nodes in the sub graph. Similarly, multiple similar property edges are collapsed into an aggregate edge only if their source/destination node is a terminating node.

E. *Handling Operations Returning Large Results*

The operations that return summarized information can be displayed compactly in a view as the number of distinct properties in a RDF dataset is typically very small (<100). However, the `expand_class()`, `expand_property()`, and `get_subset()` operations can potentially return large number of triples, thereby mitigating the effects of the summaries created earlier. Also, often times, when a class or property is expanded, all the resulting edges are not of interest. For example, a *professor* node in the graph could have a property edge *publications* with an aggregate count of 107. However, user may be interested only in the publications published in SIGMOD (say there are only 7 of them). So, `selective_expand_property()` is introduced, which expands the property while taking a SPARQL graph pattern into account to restrict the expansion (example, e*xpand publication edges belong to SIGMOD*) . The result would be a detailed list of publications in SIGMOD (all 7 of them), along with the aggregate edge adjusted to 100.

In cases when the above selective expansion technique cannot be used and the number of edges resulting from property expansion are too large to be displayed on the graph without clutter, only the first K property edges are displayed. This restriction on the number of edges displayed can be set in the result returned by the database and/or the number of triples displayed by the visualizer. The remaining edges can be displayed via iteration over the results K edges at a time.

F. *Use of Sampling*

The visualizer also supports visualizing sampled RDF data. Once the sampling is done by the methods described below, all the operations described earlier (including subsets, summaries, graph expansion primitives) are available for visualizing the sampled graph.

Although sampling large graphs has been studied, our focus in this paper is take RDF and OWL semantics into account while sampling semantic graphs. Specifically, the impact of inference has to be considered when sampling an RDF graph.

Consider the case where inference is performed first, and then the asserted and inferred graphs are sampled. In this case, some of the sampled inferred triples might become unsound, i.e., they cannot be derived from the triples in the sampled graph. Thus, we employ a strategy of sampling the asserted graph first, and then performing inference on the sample.

Our sampling methods are targeted for datasets that have arbitrary large instance data and comparatively small schemas. Because of this, and in order to preserve as many inferences as possible, we only sample the instance data.

We support the two most common sampling methods for large graphs: random edge sampling (ES) and random node sampling (NS). For NS, we select a uniformly random subset of nodes and then add all outgoing edges for each sampled node. Our decision to add only outgoing, and not all incident edges, was motivated by the fact that RDF graphs are subject-centric. In case of ES, we select a uniformly random subset of edges from the original graph.

To incorporate semantics of RDF/OWL into our sampling method, we also support *stratified* sampling. Stratified node sampling (NSS) is done by enumerating all classes in the graph and then uniformly sampling the instances of each class separately. Analogously, stratified edge sampling is done by treating each property in the graph as a stratum and sampling it independently.

Note that all of the above sampling methods can be easily implemented on top of Oracle by leveraging its support for row sampling (using the SAMPLE clause). There are numerous other methods that we could have implemented; N(S)S and E(S)S were chosen because they are amenable to an efficient implementation on top of a database. Empirical results on the effectiveness of the above sampling methods are shown in Section V.

G. *Encoding Hybrid Graphs Using RDF*

Hybrid graphs arrived at by visualization operations can also be stored in RDF database for subsequent usage. This requires encoding the hybrid graph information using RDF. Specifically, by using blank nodes (named distinctly from blank nodes occurring in the RDF data) and additional properties defined in a summary graph `sg` namespace, hybrid graphs can be easily encoded in RDF.

For example, `sg:aggStatement` (to refer to blank node denoting the aggregated edge), `sg:property` (to refer to property), `sg:propOccurrenceCount` (to refer to property count), and `sg:propTarget` (to refer to property target) are defined.

Figure 9 shows an encoded representation for a portion of an instance summary graph. The selected subject `sn:Jack` is shown as the central node. Summary graph encoding property edges and encoding object nodes are present for each property having a property count not equal to 1. Instances having a property count of 1 are encoded directly (e.g., `sn:residesIn`), without the use of summary graph encoding property edges and object nodes.

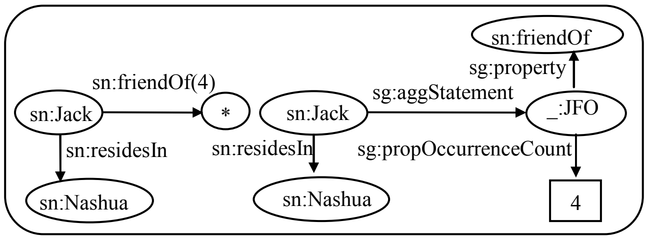


Figure 9. Encoding Hybrid Graph using RDF

Similarly, node summary graph and instance set summary graphs can also be encoded using RDF.

The primary benefit of storing hybrid graphs as RDF is that they can be directly retrieved and visualized. In addition, they can be queried using SPARQL like any other RDF data.

IV. DESIGN AND IMPLEMENTATION

Although, the design and implementation is discussed in the context of Oracle, the 3-S approach can be implemented in any RDF store, which supports SPARQL query language, extended with the ability to perform aggregate queries. Note that the aggregate queries are not part of SPARQL 1.0, but are in the upcoming version of SPARQL.

A. Oracle Semantic Store Overview

Oracle Semantic Store [11] is part of Oracle Database that supports storing, loading, and DML access to RDF/OWL data and ontologies. It allows users to create one or more semantic models to store a RDF dataset or OWL ontology. The built-in native inference engine allows inference on semantic models using OWL, RDF(S), and user-defined rules. The semantic model (and/or *entailed semantic model*, that is, model data plus inferred data) can be queried using SPARQL-like graph patterns embedded in a SQL Table function. The visualizer allows visualizing semantic models (and/or entailed semantic model) stored in Oracle semantic store.

B. Cytoscape Overview

Cytoscape [10] is an open-source, general-purpose software environment that is capable of sophisticated large scale network model analysis and visualization. It provides basic functionality for data integration and visualization. It supports a variety of automated network layout algorithms including hierarchical layout, spring-embedded layout, etc. An attribute-to-visual mapping allows data attributes to control the visual appearance of nodes and edges. A variety of node and edge properties such as color, shape, size and thickness are supported. Furthermore, in a large graph, a subset can be selected. Nodes and edges can be selected based on criteria such as, selection by name, attribute basis, etc.

Cytoscape also allows for external modules to be plugged in that can extend the core and implement new algorithms, additional network analyses and semantics.

C. Cytoscape Plugin Class: Support for RDF Graph Visualization and Operations

Oracle RDF plugin class was developed in Java, as an extension of the `CytoscapePlugin` abstract class. The RDF graph visualization operations (discussed in Section III) are implemented as action items under the RDF Visualizer plugin class, which can be accessed under the Plugins Menu for Cytoscape (Figure 10).

The plugin component gathers information about the requested operation and the selected node/edge or sub graph, and issues a relevant query to the database via the SEM_ANALYSIS package (discussed in the next section).

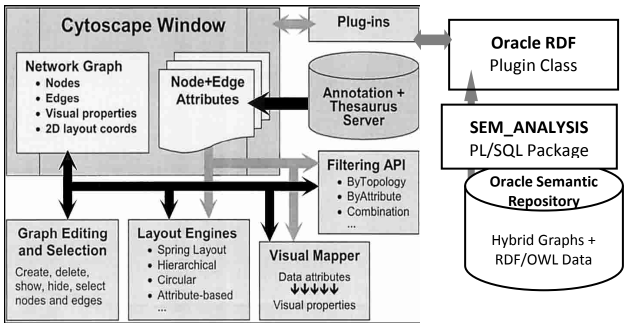


Figure 10. Oracle RDF Visualizer components (the Cytoscape core architecture shown above is from [10])

The results returned from the database are presented to the Cytoscape for display. A triple format is used to exchange information between the plugin and Cytoscape. Each (subject, predicate, object) triple maps to an edge, which optionally includes count displayed as the edge label.

Since the Cytoscape user can open multiple connections to the database, or multiple views could be open for the same connection, the plugin also maintains metadata information for each open view. Current triple list (displayed in the view window), model, rulebase, and overlay choice are stored in the view metadata. Additionally, the before and after image (of the subgraph) for a given operation, is also stored in the metadata for undo/redo handling.

D. SEM_ANALYSIS Package: Support for Optimized Access to Semantic Data

The PL/SQL SEM_ANALYSIS package is the interface between the plug-in class and the Oracle database. Although currently used in conjunction with Cytoscape, it can be used with other visual plugins or as a standalone package.

The methods in the SEM_ANALYSIS package can be classified as : 1) *demographics gathering methods*: to obtain instance and schema level demographic information for the model; 2) *instance information gathering methods*: to obtain the one-hop connectivity graph for a node, get a sub graph satisfying the specified triple pattern, get property value triples for an instance via the specified property, get the properties of an instance that have an occurrence count of one or more than one; 3) *schema information methods*: to get all the root classes , or subclasses for a class in a model; 4*) utility methods*: to obtain a list of the models and rulebases in the database, get an instance list satisfying a specified property, methods to generate summaries and samples.

V. INTERACTIVE VISUALIZATION

A. *Performance Characteristics*

The first set of experiments were conducted on Oracle 11g Enterprise Edition using a single CPU desktop (Intel P4 3.0Ghz with Hyper-Threading), 2GB RAM, and 80GB hard disk. Three datasets (LUBM 1 million, the UniProt 5 million, and Wikipedia 47 Million datasets) were loaded and additional triples were inferred based on RDF(S)/OWL semantics. Summaries were created for the data sets. We measured the summary creation time for the original model as well as with inferred triples. Table I shows the summary creation time, which as expected grows with the size of the dataset. Also, with inference, the summary creation takes longer as the summary is created on a larger dataset.

TABLE I. SUMMARY CREATION TIME

	LUBM 1 million	**UniProt 5 million**	**Wikipedia 47 Million**
Original model	1 min 18s	2 min 59s	14 min 49s
Original model+Inf.	1 min 19s	3 min 55s	21 min 24s

Next, we examined response time for the three key operations (obtaining instance set summary, instance summary, and node summary) both with and without pre-computed summaries. Acceptable *interactive response time* is taken to be a response within several seconds. Table II shows the performance numbers for a query to *get the instance set summary* for the 3 datasets.

TABLE II. GET INSTANCE SET SUMMARY QUERY RESULTS (IN SEC)

	LUBM 1M	**UniProt 5M**	**Wiki 47M**
Without Summary	0.50	2.80	NM*
With Summary	0.01	0 .01	0.01

For queries executed without summary, with the increase in size of the dataset, the instance set summary query response time increases (0.5s for 1 million to 2.8s for 5 million dataset). However, with pre-computed summaries, even for 47 million dataset, we get the results in 0.01s, thereby validating the usefulness of summaries especially for large datasets.

Table III shows the performance numbers for a query to *get the node summary* and *instance summary* for the 3 datasets. Here again pre-computed summaries provide orders of magnitude improvement in performance for instance summary.

UniProt 700 Million Dataset: We verified the performance (in the presence of pre-computed summaries) of obtaining instance set summary, instance summary, and node summary against UniProt 700 million dataset on a single CPU desktop machine P4 3.0 GHz processor with 4GB memory, 7200 rpm SATA 500 GB hard disk. The summary creation time was 3 hr 57 minutes. The response time for all operations was less than *.01 second*, demonstrating the scalability of our visualization scheme.

TABLE III. NODE AND INSTANCE SUMMARY QUERY TIME (IN SEC)

	Node Summary			**Instance Summary**		
	LUBM 1M	**UniProt 5M**	**Wiki 47M**	**LUBM 1M**	**UniProt 5M**	**Wiki 47M**
Without Summary	0.01	0.01	0.01	2.04	3.39	NM*
With Summary	0.01	0.01	0.01	0.01	0.01	0.01

We also investigated the use of different sampling methods to create a sample. Table IV shows the sample creation time for each of the methods using a sample size of 15%. Although sample creation is an offline operation, the creation times are small (less than 3 minutes for sample creation from a 1 million dataset). Also, the stratified sampling methods (ESS and NSS) do not incur any significant overhead.

TABLE IV. LUBM DATASET SAMPLE CREATION TIME

	ES	**ESS**	**NS**	**NSS**
Creation Time	2min30s	2min28s	2min26s	2min24s
No of triples	191818	191921	192765	208667

B. *Visualization of LUBM, Wikipedia, UniProt Datasets, and eClassOWL Ontology*

1) Visualization of LUBM Dataset: For LUBM dataset we issue a `get_subset()` query to get a subset and perform a node expansion. Next, we issue the same query on the entailed model. The results are shown in Figure 11. The visualization gives a clear picture of the impact of inference through the presence of new properties and/or increased count for existing properties.

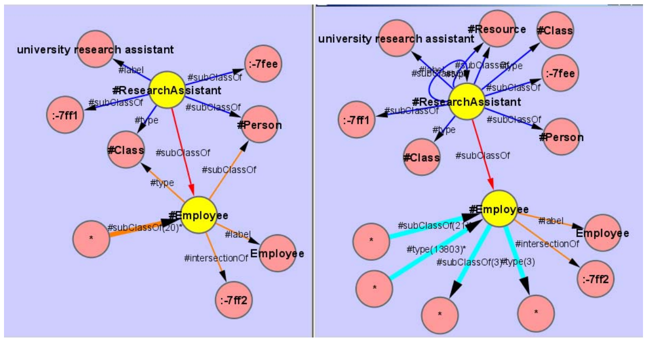


Figure 11. A subset over LUBM vs. entailed LUBM model

2) Visualization of UniProt DataSet: Assume that the user wants to get information about a given protein (P64484). The user can issue a `get_subset()` query qualified with the protein name resulting in the graph shown in Figure 12 (a).The user might be interested in knowing how this particular protein compares with other instances of protein, i.e., does it have all the representative properties of its class? etc. The user can then issue a `get_instance_summary()` query on the protein class and get it's summary information (Figure 12(b)).

* Not measured since the query time without summary is orders of magnitude more than with summary.

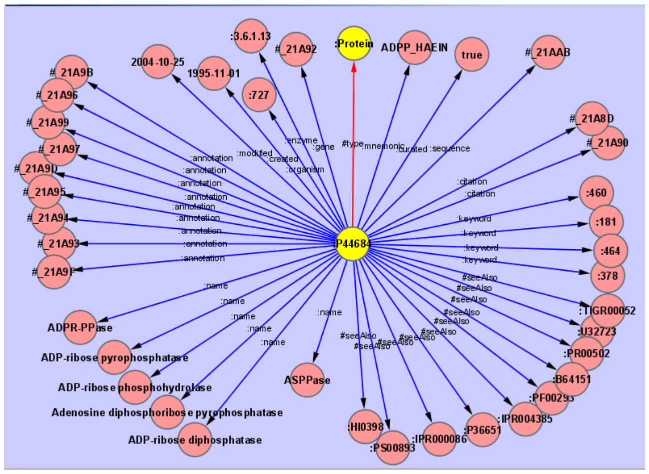


Figure 12(a). One hop neighborhood for protein P44684

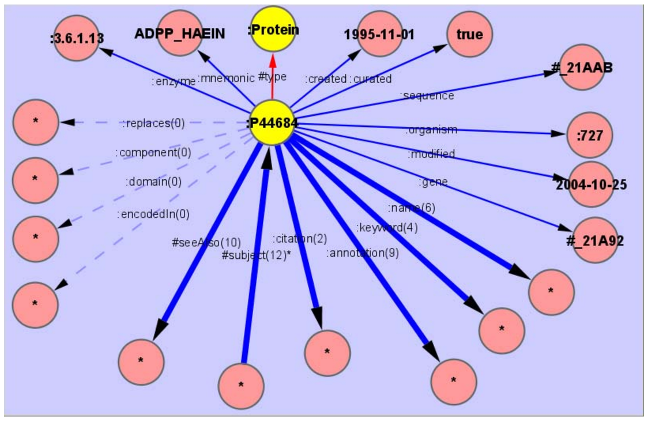


Figure 12 (b). Instance Summary graph for protein P44684

From Figure 12(b), the user knows for example, that `encodedIn` property is absent for this protein, or that the gene associated with this protein is `#_21A92`. User can be interested in further exploration of the gene and after a series of (node and property) expansions can arrive at the graph in Fig. 12 (c).

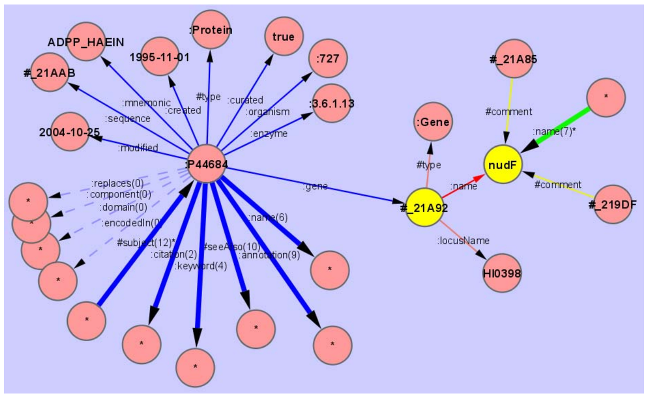


Figure 12 (c) Fisheye view of the protein P44684 neighborhood

3) *Visualization of Wikipedia data:* The 47 Million Wikipedia data set was loaded, which has only two root classes – articles and category. Starting with a subset query on articles about '`VLDB`', the user could follow the link to data_management category. The category can then be further expanded to see the various subjects or classifications of articles. It is interesting to note from the graph (Figure 13) that semantic_web was a subject even back in 2006 (the source of our wikipedia data).

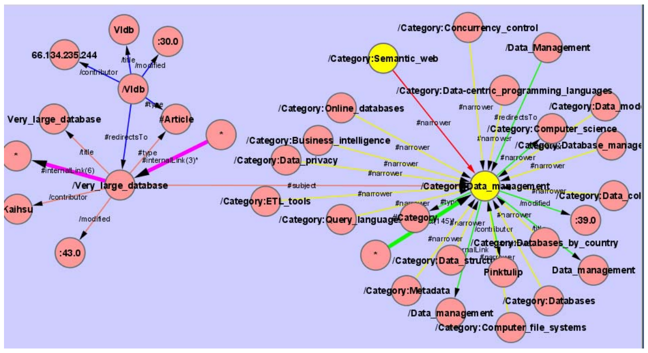


Figure 13. Exploring the Wikipedia data

4) Visualization of a Large eClass Ontology : For the eClass ontology, the number of root classes is very large, as shown in Figure 14, on the left. So, the root classes view displayed when the model is loaded is not very useful for exploration. Also, a `get_subset()` query (retrieving only the first 12 triples) on the eClass exhibits the shallow class hierarchy present in this dataset (Figure 14, on the right). Given the data characteristics, and the fact that the visualizer is currently domain agnostic, it is difficult to navigate and find the areas of interest for the user. So, we plan to support user-defined extensions to the visualizer that would allow the user to define clustering techniques specific to the domain. The classes could then be clustered based on the domain specific criteria and the virtual set of super classes created could be displayed as the root classes. The user could then explore the cluster of interest.

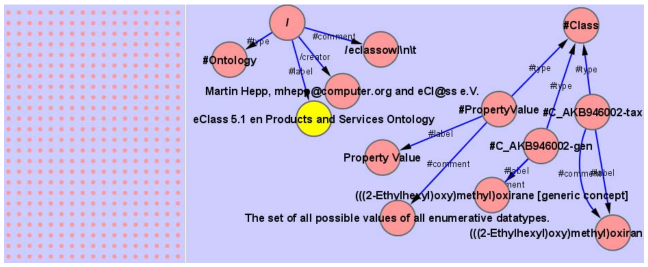


Figure 14. Root classes and a sample subset of eClassOWL

C. Empirical Evaluation of Sampling Methods

This section presents our evaluation of effectiveness of various RDF graph sampling methods in the presence of inference. Our goal was to find which sampling methods produce representative samples of entailed graphs. To the best of our knowledge, ours is the first attempt to evaluate RDF sampling methods in the presence of inference.

1) Evaluation Setup: We used LUBM and UniProt for the sampling experiments. Both LUBM and UniProt allow for large number of inferred triples when running OWL inference. However, the types of inferences that can be performed on these datasets are different: with LUBM, 14 different OWL inference rules can be applied, whereas with UniProt there are only 3 applicable rules (almost all of the UniProt inferences are due to the transitive closure of the `owl:sameAs` equivalence relation).

TABLE V. RDF DATASET (# TRIPLES) STATISTICS

Data	Asserted	Inferred	Growth Factor
LUBM	1273402	621996	0.49
UniProt	4929643	3674421	0.75

Given these two datasets, we compared the original inference graphs with the inferred graphs of the sampled models using the various sampling methods. As described in Section III, the sampling methods were node-sampling (NS and NSS) and edge sampling based (ES and ESS).

Evaluation was done by comparing the sampled and original graphs using the Kolmogorov-Smirnov (KS) test statistic [26], which can measure if two datasets differ significantly. The KS test makes no assumption about the type of data distribution, nor does it take scale into account – only the shapes of the distributions are compared. The KS test statistic is defined as $D = \max_x\{| F_n(x) – F(x)|\}$ where F_n and F are the empirical cumulative distributions of the datasets being compared. For the KS test, we use the following *semantic* criteria:

- *Class membership distribution KS(Class)*: Compare the number of instances for each class in the inferred graph.
- *Property membership distribution KS(Prop)*: Compare counts for each property in the inferred graph.
- *Inference rule entailments KS(Rule)*: Compare the number of inferred triples for reach OWL inference rule.

Since OWL and RDF can be represented as directed labeled graphs, note that the above criteria can be used in conjunction with various *syntactic* criteria (e.g., in-degree distribution) of comparing sampling methods for large graphs.

2) *Evaluation Results:* The results reported below are obtained by averaging the KS-statistic over 5 runs and 8 sample sizes for each sampling method.

LUBM. Results for LUBM are shown in Table VI and Table VII. The KS statistics for class, rule, and property distributions indicate that NS-based methods are more representative than ES with respect to inference. Additionally, the growth factors for NS and NSS are much closer to the growth factor for the original graph (0.49). Upon visual inspection of the property distribution of the inferred graph (see Figure 15), we can see that ES and ESS infer more triples for the rdf:type property (121,000 compared to 71,000 triples for NS and NSS).

TABLE VI. LUBM AND UNIPROT SAMPLING STATISTICS & GROWTH FACTORS

Dataset	Method	KS (Class)	KS (Prop.)	Growth Factor
LUBM	NS	**0.086**	0.101	**0.703**
	NSS	0.103	**0.100**	0.743
	ES	0.192	0.181	1.092
	ESS	0.191	0.182	1.092
UniProt	NS	0.001	**0.247**	**0.113**
	NSS	0.001	0.290	0.082
	ES	0.001	0.325	0.099

The ES-based graph inferred 70% more rdf:type triples than the NS graph because with edge sampling a much larger number of distinct nodes were selected. For LUBM, the number of distinct nodes in the ES graph was 142,000 on average, whereas with NS it was around 93,000. For some OWL rules, having more distinct nodes causes more inferences. For example, consider the "domain" OWL rule:

```
(a p b) (p rdfs:domain c) => (a rdf:type c)
```

With edge sampling, there are more than 143,000 distinct values that *a* could potentially bind to, whereas with NS there are only 93,000 possible bindings.

TABLE VII. KS STATISTIC FOR RULE DISTRIBUTIONS FOR DIFFERENT SAMPLING METHODS

KS(Rule)	NS	NSS	ES	ESS
LUBM	0.152	**0.133**	0.245	0.243
UniProt	0.060	**0.031**	0.082	0.051

Note that the results in Table VII are only for sample size of 15%. For the LUBM dataset, all indicators (KS statistics, growth factor comparison and visual inspection) suggest subject-based node sampling is more representative than edge sampling when inference properties are taken into account. Stratified sampling does not seem to make a difference for both methods.

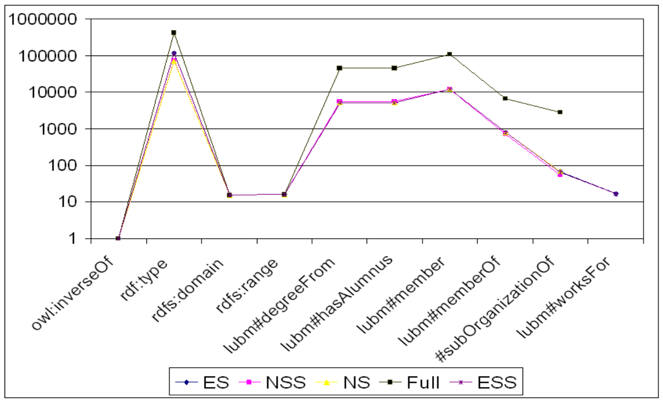


Figure 15. LUBM inference graph property distribution

UniProt: Unlike LUBM, all of the inferences in UniProt are derived from the properties of the `owl:sameAs` equivalence relation. For the 5M dataset, this produces up to 1M inferred `sameAs` triples and 2.5M additional triples using the following "copy" rule: `(b owl:sameAs b1) (a p b) => (a p b1)`.

Evaluation results are shown in Table VI and Table VII. Note that the limited set of applicable inference rules for UniProt makes it harder to evaluate the different sampling methods. For instance, the class membership distribution has only two values for the random variable, the only classes being `Journal_Citation` and `Patent_Citation`. Additionally, there are only 3 rules being fired so the KS statistic for the rule distribution (Table VII) also has insufficient data points. The only relevant statistics are the property distribution comparison (since there are 14 properties; Figure 16) and growth factors.

The property distribution slightly favors the NS-based methods as in the LUBM case. However, having only one relevant criterion for comparison is not enough to draw conclusions on the effectiveness of the sampling methods for UniProt.

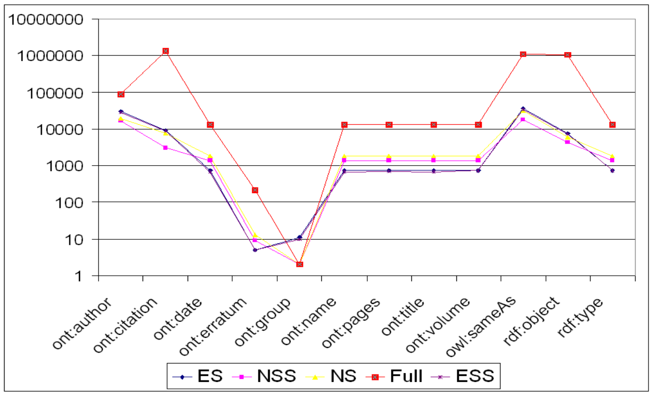


Figure 16. UniProt inference graph property distribution

The growth factors (GF) for UniProt behave quite differently from LUBM. In the case of LUBM, all growth factors for the sampled graphs were larger than the growth factor for the original graph. With UniProt, the sampled graph growth factors are substantially lower. This is due to the quadratic nature of the `owl:sameAs` inferences: for a clique of n nodes connected with the `sameAs` relation, we will infer n^2 sameAs triples, so $GF = n$. Reducing the size of the clique by a factor of *m,* will also reduce the growth by the same factor.

VI. CONCLUSION AND FUTURE WORK

The 3-S approach to RDF visualization allows interactive visualization of very large datasets. The approach of incrementally creating a fisheye view by starting with a summary and expanding it using graph expansion primitives provides control to the user in maximizing the usage of visual display. Selective expand on property further allows user to control the expansion of the graph.

For smaller datasets (up to 5 million triples), user can visualize RDF data without creating the summaries. For larger datasets, pre-creating summaries ensures interactive response time. Our experimentation with various sampling methods demonstrates that even in the presence of inference representative sampling can be created.

The use of state-of-the-art Cytoscape graph visualization tool allowed us to build the system in a very short time. All the presentation aspects, including graph layout and rendering were already provided by Cytoscape, thereby allowing us to focus on the efficient execution of backend queries in Oracle Database.

We expect that the visualizer would allow users to browse very large RDF datasets stored in Oracle. Also, the visualizer can be useful as a teaching tool for RDF/OWL beginners. With the way the RDF graph is presented, users quickly become familiar with the underlying data model.

In future, we plan to augment the visualizer with faceted browsing functionality and explore visualization of resources based on associated or derived temporal and spatial attributes. Also, we will consider incrementally refreshing the generated summaries as the source RDF data is updated.

ACKNOWLEDGMENT

We thank Jay Banerjee for his encouragement and support.

REFERENCES

[1] Resource Description Framework. http://www.w3.org/RDF/
[2] SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/
[3] Hildebrand, M., Ossenbruggen, J., Hardman, L.: /facet: A Browser for Heterogeneous Semantic Web Repositories. International Semantic Web Conference 2006: 272-285.
[4] Quan, D., Huynh, D., Karger, D.: Haystack: A Platform for Authoring End User SemanticWeb Applications"; Proceedings of the 2nd International Semantic Web Conference (2003), 738-753.
[5] schraefel, m.c., Smith, D., Owens, A., Russell, A., Harris, C., Wilson, M.: The evolving mspace platform: leveraging the semantic web on the trail of the memex"; Proceedings of the sixteenth ACM conference on Hypertext and hypermedia, New York (2005), 174-183.
[6] Goyal, S. and Westenthaler, R.: RDF Gravity; Salzburg Research (2004). http://semweb.salzburgresearch.at/apps/rdf-gravity/
[7] IsaViz: A Visual Authoring Tool for RDF (2001-2006). http://www.w3.org/2001/11/IsaViz/
[8] SIMILE: Welkin (2004-2005). http://simile.mit.edu/welkin/
[9] Furnas, G. W., Generalized fisheye views. Human Factors in Computing Systems CHI'86 Conference Proceedings , Boston, April 13-17, 1986, 16-23.
[10] Shannon, P., et al, Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res, Vol. 13, No. 11. (November 2003), pp. 2498-2504.
[11] Oracle Semantic Technologies Center, http://www.oracle.com/technology/tech/semantic_technologies/
[12] Chong, E. I., Das, S., Eadon, G., Srinivasan, J.: An Efficient SQL-based RDF Querying Scheme. VLDB 2005: 1216-1227.
[13] schraefel, m.c., and Karger, D., "The Pathetic Fallacy of RDF", Proceedings of the 3rd International Semantic Web User Interaction Workshop, Athens, Georgia, USA, 2006.
[14] Tu, K., et al: Towards Imaging Large-Scale Ontologies for Quick Understanding and Analysis. International Semantic Web Conference 2005: 702-715.
[15] Sarkar, M., Brown, M. H., Graphical Fisheye Views of Graphs. CHI 1992: 83-91.
[16] Abello, J., Kobourov, S. G., Yusufov, R., Visualizing Large Graphs with Compound-Fisheye Views and Treemaps. Graph Drawing 2004: 431-441.
[17] Noppens, O., Liebig, T., Understanding Large Volumes of Interconnected Individuals by Visual Exploration. ESWC 2007: 799-808.
[18] Leskovec, J., Faloutsos, C.: Sampling from large graphs. KDD 2006: 631-636.
[19] Shneiderman, B., Extreme visualization: squeezing a billion records into a million pixels. SIGMOD Conference 2008: 3-12.
[20] Cammarano, M., et al, Visualization of Heterogeneous Data. IEEE Trans. Vis. Comput. Graph. 13(6): 1200-1207 (2007).
[21] Elmqvist,N., et al, ZAME: Interactive Large-Scale Graph Visualization. PacificVis 2008: 215-222.
[22] Abello,J., Ham, F., Krishnan, N.,ASK-GraphView: A Large Scale Graph Visualization System. IEEE Trans. Vis. Comput. Graph. 12(5): 669-676 (2006).
[23] Frasincar, F., Telea, A., and Houben,G. J. Adapting graph visualization techniques for the visualization of RDF data, Visualizing the Semantic Web, 2006, pp. 154--171.
[24] UniProt. http://www.uniprot.org/
[25] eClassOWL - Fully-fledged Products and Services Ontology in OWL. www.heppnetz.de/projects/eclassowl/
[26] Eadie, W.T. et al (1971). Statistical Methods in Experimental Physics.Amsterdam:North-Holland.