

RDF Data Exploration and Visualization

Leonidas Deligiannidis
LSDIS Lab and Computer Science
The University of Georgia
Athens, GA 30602, USA
(706) 542-2911
ldeligia@cs.uga.edu

Krys J. Kochut
LSDIS Lab and Computer Science
The University of Georgia
Athens, GA 30602, USA
(706) 542-2911
kochut@cs.uga.edu

Amit P. Sheth
Kno.e.sis Center
Wright State University
Dayton, OH 45435, USA
(937) 239-0625
amit.sheth@wright.edu

ABSTRACT

We present Paged Graph Visualization (PGV), a new semi-autonomous tool for RDF data exploration and visualization. PGV consists of two main components: a) the “PGV explorer” and b) the “RDF pager” module utilizing BRAHMS, our high performance main-memory RDF storage system. Unlike existing graph visualization techniques which attempt to display the entire graph and then filter out irrelevant data, PGV begins with a small graph and provides the tools to incrementally explore and visualize relevant data of very large RDF ontologies. We implemented several techniques to visualize and explore *hot spots* in the graph, i.e. nodes with large numbers of immediate neighbors. In response to the user-controlled, semantics-driven direction of the exploration, the PGV explorer obtains the necessary sub-graphs from the RDF pager and enables their incremental visualization leaving the previously laid out sub-graphs intact. We outline the problem of visualizing large RDF data sets, discuss our interface and its implementation, and through a controlled experiment we show the benefits of PGV.

Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: Search process, Selection process.

General Terms

Algorithms, Management, Measurement, Performance, Design, Experimentation, Human Factors, Verification.

Keywords

Incremental Data Exploration, Ontology Visualization

1 INTRODUCTION

The Semantic Web’s vision is to provide human-readable content that consists of machine-understandable semantics and better machine interoperability. As humans possess great pattern recognition and analytical skills, a great interface to understand information provided by the Semantic Web is via exploration and visualization.

The primary objectives of visualization are to present, transform,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIMS’07, November 9, 2007, Lisbon, Portugal.

Copyright 2007 ACM 978-1-59593-831-2/07/0011...\$5.00.

and convert data into a visual representation, so that, humans can analyze and query the data efficiently. To increase the effectiveness of a visualization tool, the tool should allow users to dynamically explore the visual representation of the data so that they can comprehend the data faster and easier. Humans can process and analyze visual representations of data remarkably faster and more effectively than doing so by reading the numerical or textual representation of the same data [44]. As more and more data is presented to the user, the visual real-estate (e.g. a computer monitor) can become cluttered. One technique to avoid this problem is to display an overall representation of the data. However, if someone is interested in the detailed data, such as the labels of resources and relationships in a path in an RDF graph, the overall representation of the data – referred to as the overview, is not adequate. Even if the real-estate was infinite, laying out a large graph is computationally expensive and takes a considerable amount of time to produce. Besides, the computational power needed to produce readable graphs, visualizing a vast amount of information at once may be undesirable for at least two reasons: (a) too much information may clutter the display that leads to an unorganized and disorienting visual environment, and b) the tool may become so slow that human-computer interactivity may be lost. Thus, PGV begins from a point of interest and then incrementally explores and visualizes more and more data.

PGV consists of two main subsystems. The first subsystem, the PGV explorer is a Graphical User Interface (GUI). The second subsystem is the RDF pager module utilizing a high performance main memory RDF storage system called BRAHMS [28]. The two subsystems communicate with each other utilizing the HTTP 1.1 [18] protocol. Even though we describe both subsystems in this paper, we will primarily highlight the PGV explorer. The PGV explorer uses an incremental algorithm to layout the explored sub-graphs. Initially, we employed the tools from GraphViz [7] [19] [15] for the graph layouts. These tools have not been designed for incremental layouts, as they tend to re-adjust the final layouts to minimize edge crossings, enforce symmetry, etc. This behavior is useful for general purpose graph layouts but it is not suitable for PGV, as it will be discussed later. Instead, PGV produces simple radial layouts at each exploration step. Each layout is combined with the previous layout which effectively creates the incremental layout capability. Because the graphs produced by the exploration of a node are simple radial graphs, we implemented our own radial layout algorithm which dramatically improved the time to layout a graph.

Our research resulted in the following three contributions. First, we designed and implemented a tool that efficiently uses the available computer resources to load sub-graphs of RDF data sets that are of interest to the user during an incremental graph exploration. Second, we showed that rendering important-to-the-user

information utilizes the display real estate better, without cluttering the display with information that is most likely irrelevant to the user. Even if the information is relevant to the user, displaying too much information produces unreadable graphs, consisting of many nodes and edges spanning across tens or even hundreds of virtual displays. PGV not only produces graphs that are easy to comprehend, but that are visually pleasing, as well. Third, our experimental results and our survey questions show the benefits of PGV.

2 BACKGROUND

Semantic information represented in RDF is normally large in size, highly interconnected, and does not follow a fixed schema [33]. Graph Visualization does not scale well to large datasets [9], mainly because of the space needed to visually represent the data on a display. Thus, an exploratory technique is necessary [35] to navigate in the dataset.

Issues pertaining to semantics have been addressed in many fields such as linguistics, knowledge representation, artificial intelligence, information systems and database management. Semantic Analytics involves the application of techniques that support and exploit the semantics of information (as opposed to just syntax and structure/schematic issues [2] and statistical patterns) to enhance the existing information systems [32].

As industry and academia are focusing on information retrieval over semantic metadata, it is increasingly possible to analyze such metadata to discover interesting and useful relationships. This is why visualization is becoming an increasingly important component in Semantic Web tools. Particularly, visualization has been used in tools that support the development of ontologies, such as semi-automatic ontology extraction tools (OntoLift [31], Text-to-Onto [29]) and ontology editors (Protégé [30], OntoLift). Such tools employ schema visualization techniques that primarily focus on the structure of the ontology, including its concepts and their relationships.

However, sometimes the information about the population of the ontology (entities, instance data) is often more important than the structure of the ontology that is used to describe these instances. The Cluster Map [3] technique focuses on visualizing instances and their classifications according to the concepts of the ontology.

Some ontologies, such as WordNet [13] or SWETO [4], cannot be easily visualized as graphs, as they consist of large numbers of nodes and edges. There are many large bioinformatics ontologies such as the Gene ontology [12], GlycO [5] and ProPreo [38], which have several hundred classes at the schema level, and the latter two have about half a million to over six million instances, respectively. Recently, the UniProt database [34], a very large protein database, has been converted to RDF representation (<http://dev.isb-sib.ch/projects/uniprot-rdf/>).

Dynagraph [14] is a C++ platform-neutral library based on GraphViz [19], which can be used to incrementally lay out sub-graphs. One side effect of this incremental layout algorithm is that it rearranges the previously laid out nodes and edges, which could be a desirable for a general purpose tool. Similarly, uDraw [16] provides an environment to lay out graphs and also to manipulate their placement. However, the internal constraints of uDraw enforce the level of the nodes and edge crossings to be minimized leading to, sometimes, undesirable effects. For example, the user may need to place a Node at the upper left corner of the canvas.

uDraw would then automatically move the Node to wherever it thinks it is best. A similar tool with the same behavior is TouchGraph [45]. TouchGraph is a tool relying on a spring-embedding algorithm to implement customizable layouts. It is a useful tool but also deemed disruptive to users because it keeps re-adjusting the graph to a layout it determines to be best.

Jambalaya [41] is a tool for visualizing schemas and instances of ontologies. It allows users to browse the ontology detail at several levels by zooming in and out using animation to change the level of detail. The primary disadvantage of Jambalaya is that the users can get lost deep in a detailed view and as a result lose the “overview” of the visualization. The out of focus space (space containing information secondary to the user) should take less attention not to overwhelm the user but rather aid his/her focus on the point of interest by leaving the remainder space in lower level of detail [39] [37] [36].

Visualizing large ontologies has created challenges in the research on visualization. Viewing the whole graph at once is not always recommended, since the limiting factor is the visual “real estate” (the computer display).

InfoSky [23] is a system implementing a real-world metaphor, the telescope. InfoSky allows exploration of large hierarchical structured knowledge spaces, more specifically collection of documents, web pages, etc. Using the telescope mechanism, InfoSky employs a planar graphical representation of the collection; the magnification level can be adjusted to access different levels of detail.

CropCircles [43] is a tool that enhances user’s ability to view class structures and inheritance hierarchies at a glance. In CropCircles circles represent nodes in a tree. Every child circle is nested inside its parent circle and a circle’s diameter is proportional to the size of the sub-tree rooted at that node.

Faceted browsing [20] is an exploration technique for large datasets. An RDF based faceted browsing implementation is shown in [8]. Faceted interfaces are better than keyword search queries and search criteria can be altered at each step of the exploration. One of the main problems with faceted browsers is the increased number of choices presented to the user at each step of the exploration.

Many [25] [24] [17] ontology based visualization tools are based on the Self-Organizing Map (SOM) [26] technique, at least partially. WEBCOM [25] is a tool that utilizes SOM to visualize document clusters where semantically similar documents are placed in a cluster. The order of document-clustering helps in finding related documents. ET-Map [17] is also used to visualize a large number of documents and websites and uses a variation of SOM.

Spectable [11] [6] visualizes ontologies as taxonomic clusters. These clusters represent groups on instances of individual classes or multiple classes. Spectable displays class hierarchical relations while it hides any relations at the instance level. One of the main advantages of Spectable is to present a large number of instances, but with a simple ontology, to the users. Each instance is placed into a cluster based on its class membership and instances that are members of multiple classes are placed in overlapping clusters. This visualization provides a clear and intuitive depiction of the relationships between instances and classes. In [24], a holistic “imaging” is produced of the ontology which contains a semantic layout of the ontology classes where instances and their relations

are also depicted. However, this approach is not suitable to visualize instance overlap like in Spectable.

Research in visualizing multimedia enriched environments has demonstrated the need to utilize the third dimension. Examples include the visualization of maps [21] [22], tracking of activities, events, documents, etc [10], and multimedia enriched environments [27].

3 PGV EXPLORER

PGV's strength is realized in cases where a user wants to simply explore his data without knowing the exact information and graph patterns he is looking for. On the contrary, when a user knows that an existing path or relationship does exist in an RDF graph and wants to visualize this path(s), a visualization tool might be more appropriate. However, a user may simply want to explore the data in hope of finding relationships between resources and entities and gain new knowledge in a different way, instead of relying on a ranking algorithm to tell him what is important and relevant. In PGV, a user can dynamically, at every exploration step, change the exploration criteria and investigate different paths in an RDF graph representation.

The PGV explorer consists of two subsystems. The first subsystem is the *Starter* and the second subsystem is the *Visualizer*. The Starter is used to find and isolate the starting node/resource where the exploration will begin. After the user finds a node of interest, the Explorer is used to incrementally explore an RDF graph. The Explorer is the main subsystem used by a user to interact with PGV.

3.1 The Starter subsystem

In order for a user to explore an RDF graph, a starting point must exist; a resource where the exploration can begin. To find this starting point, the PGV Starter subsystem is used. A user selects a resource from the RDF's schema visual representation and then enters a string as it is shown in figure 1. This is an easy way of finding "a" starting point for the exploration and can be used easily by expert and novice users. However, an expert user can construct manually a complex SPARQL query to select a starting point. A novice user, on the other hand, can first select a class from the RDF schema representation. PGV then constructs a simple SPARQL query where the user can manually enter a string to complete the query before sending it to the RDF pager. The returning result is a list of nodes from where the user selects one to become the starting point of the exploration.

Unlike other query languages where the fields and table names of a database must be known to formulate a query, in SPARQL only relationships and class names need to be known. These relationships and class names are full URIs that people cannot easily remember. The user, however, can find all the class names from the schema's visual representation. At initialization, PGV requests the schema from the backend subsystem and visualizes the schema using the "dot" product of GraphViz. We chose the "dot" product of GraphViz for visualizing the schema because:

- it generates graphs that are directional at the overview level. These graphs can be read from left to right. The nodes are laid out in a way where all edges connecting the nodes begin on the left hand side and end on the right hand side. This enables a user to read the schema classes/nodes easily,

- the generated graph is symmetric, wherever symmetry is applicable,
- the final graph contains minimum number of edge crossings.

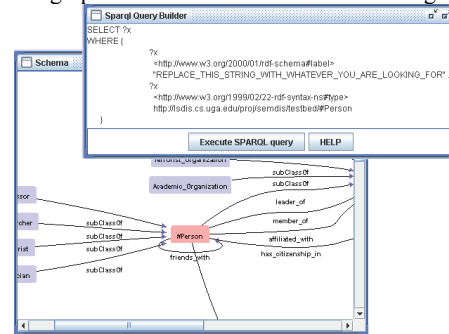


Figure 1. Schema Visualizer window with a highlighted class resource. On top, the Starter window with a SPARQL query.

The user can point and click to select a class to construct a partial SPARQL query as shown in figure 1, where the class "Person" is selected and highlighted. Then, the user can type-in a string in the SPARQL window (the window in the foreground), at an indicated place, to formulate a simple, yet, complete query. The result of the query could include multiple resources that satisfy the given query. The user then needs to select one of the resources to become the starting node for the exploration.

At this point, PGV switches to the Visualizer sub-system where the user is free to start exploring the RDF data-store.

3.2 The Visualizer subsystem

Instead of presenting to the user the entire data set, which most certainly will clutter the display with nodes and edges [40], PGV's Visualizer subsystem begins by displaying an instance that the user selects (from the Starter subsystem) along with all its direct neighbors as shown in figure 2.

The Center node (the instance of interest) is displayed at the center of the graph in green, and its direct neighbors are shown as blue rectangles; literals are shown in white. Explored nodes are displayed in green. The edges connecting the "instance of interest" with its direct neighbors are the relationships extracted from the ontology. Their property name is displayed as a label on them (e.g. has_product). The edges connecting the explored nodes are drawn in red. We use a radial layout to visualize each explored node. This layout seems to be the most appropriate for PGV since we want to display an "instance of interest" along with its direct neighbors around it. The user, however, is free to move each node to any place in the canvas he/she wants. There are widgets in the application window that enable the user to "shrink" and "expand" a group of Nodes; "shrink" means that the distance between the center node and its neighbors becomes smaller, and "expand" means the opposite.

The user can double click on any of the neighbor nodes to explore its neighbors. The previously laid out nodes remain at their original positions. The newly explored Node reveals its neighbors and the Node itself becomes green to indicate that it is explored. The user can freely move individual Nodes (the blue nodes) or group of Nodes (the green nodes along with its direct neighbors); this depends on which node is being dragged.

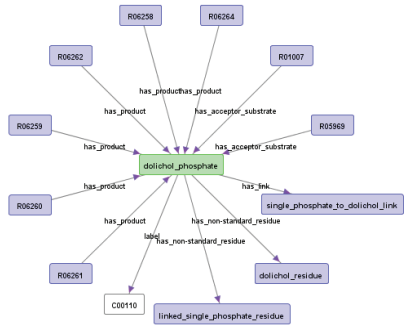


Figure 2. An explored node at the center and its direct neighbors around it.

The edges connecting the explored nodes are highlighted in red. The user can also collapse unrelated nodes to un-clutter the display; collapsed nodes can be expanded by double clicking on them. Nodes in blue can be hidden upon the user’s request. A double mouse click on a green (explored) node collapses or expands the node; hides and shows its unexplored neighbors respectively.

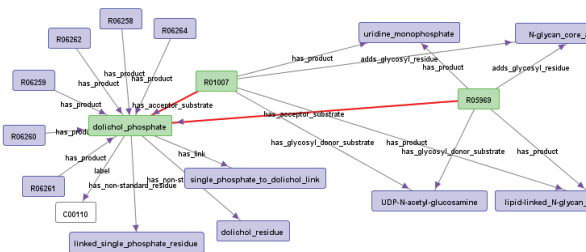


Figure 3. Explored graph before using the Ferris-Wheel technique. Explored nodes are shown in green. Neighboring nodes are shown in blue. Literals are shown in white. Path connecting explored nodes is shown in red.

For some nodes, the number of their immediate neighbors can be very high, perhaps on the order of hundreds. We call such nodes *hot spots*. Because for such nodes the resulting radial layout may be unreadable, the selection of the next node to explore may be very difficult, if not impossible. Simple zooming in is inadequate, as it makes to overall display too large. In order to handle such cases, we use our *Ferris-Wheel* technique. Here, the user selects one “wheel” (the radial display of the neighbors of one node) and zooms-in sufficiently to make the labels legible. At this point only a small fragment of the wheel, say, its top is visible. However, now the user can rotate the wheel (in a Ferris-Wheel-like fashion) to quickly examine the neighbors. While one wheel is being rotated, the rest of the display remains fixed as shown in figures 3 and 4.

The user can move the mouse over an unexplored node to reveal its literals, if any, and the full URI of the node using the application’s tooltip, in order to decide if he/she wants to explore that node; tooltip information includes the URI of the Node, its short name, and a list of all literals if any.

The Explorer subsystem runs in two modes: a) “Forward Exploration” (FE) and b) “Forward and Backward Exploration” (FBE). In both modes, when a node is explored, its directly connected nodes are revealed. The difference between the two modes is that FE only displays the explored node’s neighbors that lead to them

while FBE includes the neighbors that lead to the explored node as well, as shown in figures 5 and 6 respectively.

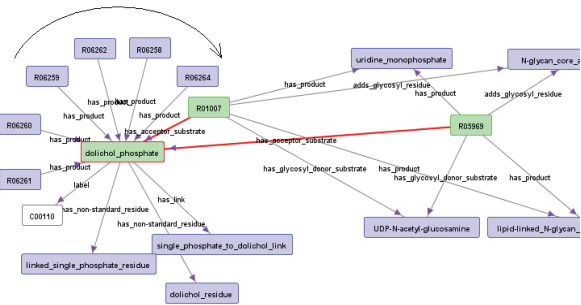


Figure 4. Explored graph after using the Ferris-Wheel technique on the leftmost explored node (clockwise rotation). The rest of the graph stays intact.

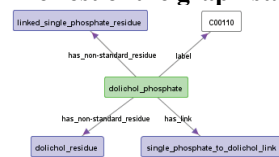


Figure 5. Forward exploration (FE). Node of interest connected to neighbors via outgoing edges only.

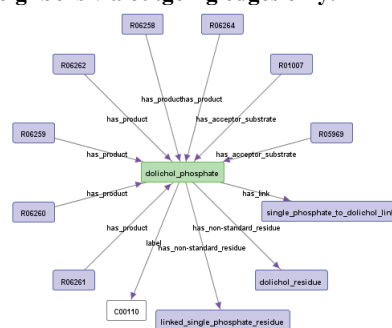


Figure 6. Forward and Backward exploration (FBE). Node of interest connected to neighbors via incoming and outgoing edges.

4 PGV PAGER

The backend of PGV is accessible via a FastCGI interface. It is composed of the RDF Pager module and BRAHMS [28], which is our high performance main-memory RDF data-store. BRAHMS’s high speed query performance makes it an appropriate choice for a real-time RDF exploration system. BRAHMS is designed as a fast main-memory RDF/S storage, capable of storing, accessing and querying large ontologies. It does not use any DB backend and all data is kept in main memory. It is implemented in C++ for high performance and strict memory control (details of BRAHMS’s implementation and performance are shown in [28]).

Instead of loading the entire ontology in memory, PGV sends queries to the RDF Pager module to retrieve information on demand, as shown in figure 7. This happens when the user decides to explore a node by a double mouse click on an unexplored (blue) node. In response, the RDF Pager returns small sub-graphs (in a way similar to paging of search results of a typical Web search engine).

BRAHMS’s main strength is in its speed for simple operations on RDF graphs. It is achieved by its indexing scheme, usage of sim-

ple types as resource identifiers (that speeds-up all comparisons), and a read-only memory model. Results showed that such decisions resulted in high speed for RDF graph computations [28].

Furthermore, the division of resource types offers focused methods for graph search algorithms to concentrate only on given type (of resource or statement) and eliminates the need of filtering unwanted types during search. Consequently, during the search on instance level, the user only accesses instance resources. There is no need to filter out literals or schema classes associated with instances, which makes the search simpler and more effective.

5 ARCHITECTURE

PGV's architecture, shown in figure 7, is separated into two major components: the front-end (PGV explorer) and the back-end RDF Pager powered by BRAHMS. A web server is the link between the PGV explorer and the back-end which utilize the HTTP 1.1 protocol. The web server, upon startup, initializes a FastCGI program that connects to the RDF Pager. Queries from the PGV explorer are sent to the RDF Pager via the FastCGI interface and the resulting resources and sub-graphs are sent back to the PGV explorer, one page (sub-graph) at a time.

The PGV explorer is built using the Java programming language and platform. At every exploration, PGV performs a layout of the result sub-graph, which consists of the node being explored and its direct neighbors. It then lines-up this new sub-graph on top of the existing graph. This produces an incremental layout at each exploration step. In cases where some nodes or edges overlap with other nodes or edges, the user can manipulate the position of the sub-graphs or individual nodes, which yields to the semi-autonomous nature of PGV.

5.1 Layout Algorithm

Graphs in PGV consist of edges and vertices (explored and unexplored). Unexplored vertices are visualized in a radial layout around explored vertices. At every exploration step, a new sub-graph is generated and then overlaid on top of the previous graph; previously laid out vertices and edges are not repositioned.

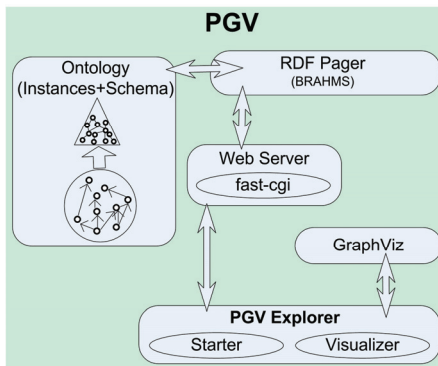


Figure 7. PGV's system architecture.

The algorithm, shown in figure 8, performs an incremental exploration of a vertex that belongs to the PGV directed graph $G = (V, E)$. V is a set of all vertices and E is a set of all edges (s, d) where $s, d \in V$. Vertex s is the source and vertex d is the destination. Below is the algorithm and is followed by a graphical example illustrating how the algorithm performs an incremental layout.

The parameters to the algorithm are the existing graph G and a pointer to the vertex being explored. In line 1, PGV retrieves from the RDF Pager all neighbor vertices of the vertex being explored to produce a graph G' , shown in figure 9b.

```

EXPLORE_AND_LAYOUT ( $G$ ,  $exploreVertex$ )
1    $G' = (V', E') \leftarrow \text{ExploreVertex}(exploreVertex)$ 
2    $T \leftarrow \emptyset$ 
3   for each edge  $e \in (E' - E)$  do
4      $s \leftarrow \text{sourceVertex}(e)$ 
5      $d \leftarrow \text{destinationVertex}(e)$ 
6     if  $s = exploreVertex$  then
7        $v \leftarrow d$ 
8     else
9        $v \leftarrow s$ 
10    if  $v \notin V$  then
11       $V \leftarrow V \cup \{v\}$ 
12       $T \leftarrow T \cup \{v\}$ 
13    //  $v$  is not an explored vertex since  $e \in (E' - E)$ 
14     $E \leftarrow E \cup \{e\}$ 
15    DO_LAYOUT ( $T$ ,  $exploreVertex$ )

DO_LAYOUT ( $T$ ,  $v$ )
max_w  $\leftarrow \text{MaxLabelWidth}(T)$ 
angle  $\leftarrow 2 * \pi / \text{sizeof}(T)$ 
counter = 0
for each vertex  $u \in T$  do
   $X \leftarrow \sin(\text{angle} * \text{counter}) * \text{max}_w$ 
   $Y \leftarrow \cos(\text{angle} * \text{counter}) * \text{max}_w$ 
   $u[x] \leftarrow v[x] + X$ 
   $u[y] \leftarrow v[y] + Y$ 
  counter  $\leftarrow \text{counter} + 1$ 

```

Figure 8. Incremental layout algorithm.

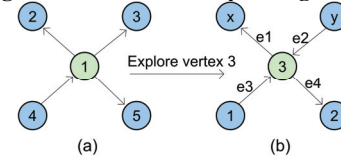


Figure 9. The execution of the **EXPLORE_AND_LAYOUT** algorithm. In (a) the already laid-out graph G is shown. In (b) the graph G' is shown which resulted from starting the exploration of vertex with label 3.

Then in lines 3-14 the algorithm enters a "for" loop where it iterates through all new edges found in G' ; edges found in G that do not exist in G , ($e \in (E' - E)$). These are the edges e_1, e_2 and e_4 ; edge e_3 , which is the edge connecting vertices with labels 1 and 3, exists in G and thus it is omitted. In each iteration, s points to the source and d points to the destination vertex of the current edge (lines 4,5).

Figure 10 shows the partial visual representation of the graph G' and its three edges through which the “for” loop (lines 3-14) iterates.

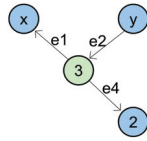


Figure 10.The situation just before the first iteration.

Since all edges include, either as source or destination, the vertex being explored, v points to the vertex that is not the explored vertex (lines 6-9).

The edge examined in the first iteration of the “for” loop is the edge $e1(3-x)$ shown in figure 11. Since the vertex with label “ x ” is a new vertex in G (where $v \notin V$) (line 10), it is inserted in V and in T (lines 11,12); T is a temporary set holding newly created vertices. Additionally, since every edge encountered in the “for” loop is a new edge $e \in (E'-E)$, every edge is inserted in E (line 14).

Similarly, during the second iteration of the “for” loop, the edge $e2(y-3)$ is processed as shown in figure 12. Here, the new vertex that $\notin V$ is vertex labeled “ y ”, which is the source vertex of the edge $e2$ and v points to it (lines 6-9). Since v is a vertex that $\notin V$, it is inserted in V and T (lines 10-12). In line 14, $e2$ is inserted in E . The last edge being processed is the edge $e4(3-2)$ as shown in figure 13. Since the vertex labeled “ 2 ” $\in V$, we simply execute line 14 and we insert the edge $e4$ in E .

At the end of the “for” loop, we call the DO_LAYOUT function and we pass it T , which is a set of all newly created vertices, and a pointer to the vertex being explored. This function performs a radial layout of the newly created vertices around the vertex being explored. It first calculates the radius of the circle where the newly created vertices will be placed. This is done by finding the maximum width of the labels (stored in the max_w variable) of the vertices in T . It then divides the space equally around the explored vertex (stored in the variable $angle$). In the “for” loop, it sets the coordinates of the newly created vertices so that they appear around the explored vertex. This produces the final graph shown in figure 14.

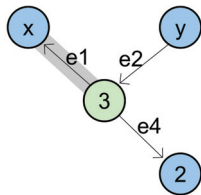


Figure 11. The situation after the first iteration of the loop; edge $e1$ is processed. Shaded edges indicate processed edges. Vertex with label “ x ” is inserted in G .

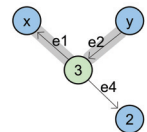


Figure 12. The situation after the second iteration. The $e2$ edge is processed and the vertex “ y ” is inserted in G .

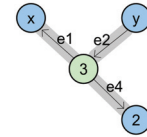


Figure 13. The situation after the “for” loop end. The $e4$ edge is processed. Vertex labeled “ 2 ” is an existing vertex in G .

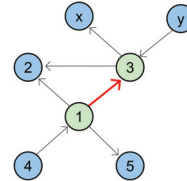


Figure 14. The final graph after exploring vertex with the label “ 3 ”. Explored vertices are connected with red edges as shown here.

Initially, we utilized the “twopi” product of GraphViz for the radial layouts. However, because we had to convert the data coming in from the RDF pager to “twopi”’s input format, then issuing an `exec()` call to execute “twopi” and then parsing the output of “twopi”, the performance of the exploration step was significantly affected. “twopi” implements a generic radial layout algorithm. Since PGV needs very simple radial layouts to place vertices around a single explored vertex, we managed to significantly improve PGVs performance by implementing in Java the DO_LAYOUT algorithm. This significant improved performance is shown in figure 15. For example, to layout 482 vertices with “twopi” takes about 30 seconds. The same layout utilizing the DO_LAYOUT function takes less than half a second!

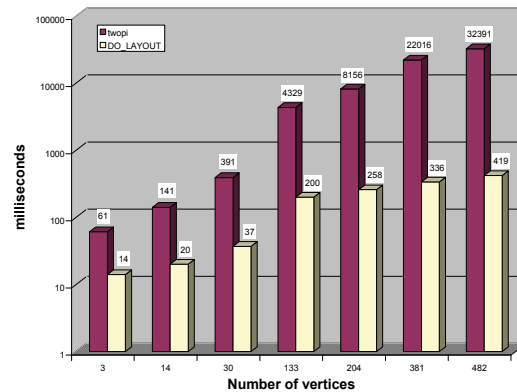


Figure 15. Comparing performance between twopi and DO_LAYOUT().

6 EXPERIMENTAL SETTING

RDF-Gravity [42] is an excellent visualization tool for directed graphs built in RDF and OWL. Via its pre-built filters, a user can choose what information is rendered. Even though Gravity is not an exploration tool, via its filtering mechanisms it resembles an exploration tool. However, RDF-Gravity loads the entire RDF source in memory yielding unreadable displays when the data set is large.

Of course, the whole graph does not need to be loaded into Gravity, but only a small sub-graph (a result of a query). However, this precludes the main purpose of exploration of an unknown ontology, where one would not know what to query. Additionally, “moving forward” in Gravity would require running additional

queries in order to provide a smooth transition in the desired direction of the exploration and provide the needed nodes. Such queries would be very difficult, if not impossible to formulate.

Since our GlycO RDF data set consists of 12322 statements (not including schema statements), Gravity produces unreadable renditions. Thus, we extracted a subset of the data that consisted of only 350 statements and then we loaded the data into Gravity.

7 THE PROCEDURE

Ten subjects volunteered to participate in the experiment. The subjects were undergraduate and graduate students and their age ranged from 18 to 40. Their average age was 25. All ten subjects successfully completed the experiment. One subject was a female.

The subjects had a short 5 minutes training session to familiarize themselves with both of the interfaces. First, the experimenter explained the user interfaces. The subjects were given 10 minutes to practice and become familiar with the user interfaces. The data set used for this practice was different from the one used in the trials.

All subjects used both interfaces. The order of the two was counterbalanced to eliminate differences in fatigue and learning. The order in which the subjects used the PGV and Gravity did not indicate an important effect on the performance – all subjects performed better in PGV. There was a 5 minute break between the two trials.

After the training and the practice session, the subjects were asked to find a path in the graph from the start node (dolichol_phosphate) to the destination node (glyco_peptide_G00009) following paths with edges labeled has_product, has_acceptor_substrate or has_reactant. The path represents a well known N-Glycan Biosynthesis pathway [1], represented in the GlycO ontology. We chose this path as a good example of a long exploration in an RDF graph. It is composed of 30 relationships and intermediate nodes.

The subjects told us when they were ready and we gave them a verbal signal to proceed in their task. The experiment was terminated when the subjects found the path between the two given nodes. In the analysis, we used as the performance measure the elapsed time from beginning of their task until its completion.

At the end of the experiment we asked the subjects to respond to the following statement on a 5-point Likert scale (1. strongly disagree 5. strongly agree) in order to check if the training achieved its goal: “The training session familiarized me with using GRAVITY and PGV.” The responses show that they were sufficiently familiarized with both interfaces: (Mean=4.4, Standard deviation=0.84).

Finally, the subjects filled out a survey containing questions about satisfaction with the interfaces. The survey contained questions adapted from [12]; end-user satisfaction.

8 RESULTS

Our hypothesis was that there is a significant difference in the population means of the different levels of our single factor, which was the “interface type” (PGV and GRAVITY). We used a within-subject experimental design. Our independent variable was the interface type and our dependent variable was performance (time to complete the task; finding the path between the two given nodes).

We tested the hypothesis by comparing the means of the pooled performance scores using one-way analysis of variance (ANOVA) with repeated measures. We compared the results based on the interface type and we found that using PGV, even though the data set was much larger compared to the data used in GRAVITY, users completed their task faster, and this difference in performance is significant ($F[1,9] = 28.267, p < 0.001$); (Gravity: mean=439.9, standard deviation=158.9) and (PGV: mean=150.6, standard deviation=72.06). The data distribution is shown in figure 16 using a box-and-whisker plot.

Finally, we were interested in exploring subject’s satisfaction using RDF-Gravity and PGV. We asked subjects about ease of getting started, interface friendliness, and general satisfaction.

Table 1 shows the questions and subjects’ responses. As the ANOVA results in the last two columns of table 1 indicate, subjects’ satisfaction was statistically different. The subjects found that PGV’s interface is more user friendly, easier to get started with and use it; these are the first three questions in table 1. Overall however, the subjects were satisfied with both interfaces (according to the last question shown in table 1).

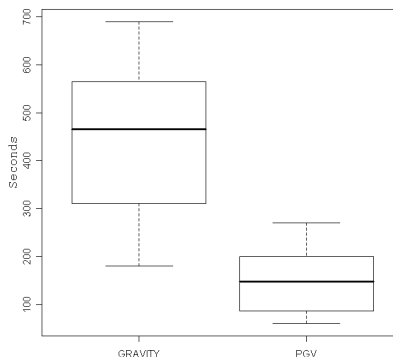


Figure 16. Data distribution using a box-and-whisker plot

Table 1. Survey questions and results.

| Survey Questions | Gravity | | PGV | | F(1,18) | p |
|-------------------------|---------|------|-----|------|---------|--------|
| | M | SD | M | SD | | |
| User friendly | 3.1 | 0.74 | 4.3 | 0.48 | 18.514 | <0.001 |
| Easy to use | 3.0 | 1.15 | 4.5 | 0.53 | 13.966 | <0.01 |
| Easy of getting started | 2.9 | 1.2 | 4.1 | 0.88 | 6.545 | <0.05 |
| I am satisfied with | 3.4 | 0.97 | 4.0 | 0.67 | 2.613 | 0.1234 |

M and SD represent the mean and standard deviation respectively. F and p are from ANOVA analyses that compare the means of the answers in Gravity and PGV. All questions were based on 5-point Likert scale ranging from strongly disagree (1), to strongly agree (5).

9 CONCLUSION

We presented a highly interactive tool, PGV, for exploring and visualizing large RDF ontologies. PGV’s primary goal is to present the information to the user in a simple and intuitive way. It provides an environment where users can select a small set of objects to examine dynamically in real-time, providing better contextual information. Our Ferris-Wheel-like technique is used to explore the so called hot spots in the graph, i.e. nodes with large

numbers (hundreds) of immediate neighbors. The users can select and manipulate objects using a simple point-and-click interface. Through our study and experimental results we found that PGV can be a valuable tool for data exploration and examination.

10 ACKNOWLEDGMENTS

This research was principally supported by "SemDis: Discovering Complex Relationships in Semantic Web" funded by the National Science Foundation (NSF) grant IIS-0325464, and projects funded by ARDA. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, ARDA or UGA.

11 REFERENCES

- [1] A. Helenius and M. Aebi, *Roles of N-Linked Glycans in the Endoplasmic Reticulum*, *Annual Review of Biochemistry*, 73. 1019-1049, 2004.
- [2] A. Sheth, *Semantic Meta Data For Enterprise Information Integration*, *DM Review*, http://dmreview.com/article_sub.cfm?articleId=6962, Jul. 2003.
- [3] Aduna Cluster Map Library version 2005.1, (*Integration Guide*), <http://aduna.biz/products/technology/clustermap/docs/Aduna-Cluster-Map-2005.1-Integration-Guide.pdf>, 2005.
- [4] B. Aleman Meza, C. Halaschek, A. Sheth, I. B. Arpinar and G. Sanna-pareddy, *SWETO: Large Scale Semantic Web Test-bed*, *International Workshop on Ontology in Action*, Banff, Canada, 2004.
- [5] C. Thomas, A. Sheth and W. York, *Modular Ontology Design Using Canonical Building Blocks in the Biochemistry Domain*, *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS)*. IOS Press., Nov. 2006.
- [6] Christiaan Fluit, Marta Sabou and Frank van Harmelen, *Ontology-based Information Visualisation*, in C. Geroimenka V. Chen, ed., *Visualising the Semantic Web*, Springer Verlag London, 2002.
- [7] Emden R. Gansner and Stephen C. North, *An Open Graph Visualization System and Its Applications to Software Engineering*, *Software - Practice and Experience*, 30 (11) (2000), pp. 1203-1233.
- [8] Eyal Oren, Renaud Delbru and Stefan Decker, *Extending faced navigation for RDF data*, *International Semantic Web Conference (ISWC)(to appear)*, Nov. 2006.
- [9] F. Frasinca, A. Telea and G.-J. Houben, *Adapting graph visualization techniques for the visualization of RDF data*, *Visualizing the Semantic Web*, 2006, pp. 154-171.
- [10] K. M. Fairchild, S. E. Poltrock and G. W. Furnas, *SemNet: Three-dimensional graphic representation of large knowledge bases*, *Cognitive Science and its Application for Human-Computer Interface*, Erlbaum, Hillsdale, NJ., 1988, pp. 201-233.
- [11] Frank van Harmelen, Jeen Broekstra, Christiaan Fluit, Herko ter Horst, Arjohn Kampman, Jos van der Meer and M. Sabou, *Ontology-based Information Visualization*, *Proc. of the workshop on Visualization of the Semantic Web (VSW'01)*, London, 2001.
- [12] Gene Ontology Home, <http://www.geneontology.org/>.
- [13] George A. Miller, *WordNet: A Lexical Database for English*, *Communications of the ACM* 38 (11) (1995), pp. 39-41.
- [14] Home page of Dynagraph, <http://dynagraph.org/>.
- [15] Home page of GraphViz, <http://www.graphviz.org/>.
- [16] Home page of uDraw, <http://www.informatik.uni-bremen.de/uDrawGraph/en/index.html>.
- [17] Hsinchun Chen, Chris Schuffels and R. Orwig, *Internet Categorization and Search: A Self-Organizing Approach*, *Journal of Visual Communication and Image Representation*, 7 (1) (1996), pp. 88-102.
- [18] Hypertext Transfer Protocol -- HTTP/1.1 (RFC2616), <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.
- [19] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North and G. Woodhull, *Graphviz - Open Source Graph Drawing Tools*, *Graph Drawing*, 2001, pp. 483-484.
- [20] K.-P. Yee, K. Swearingen, K. Li and M. Hearst, *Faceted metadata for image search and browsing*, *In Human-Computer Interaction (CHI'03)*, Florida, 2003.
- [21] T. Kapler, R. Harper and W. Wright, *Correlating Events with Tracked Movements in Time and Space: A GeoTime Case Study*, *Intelligence Analysis Conference*, Washington, DC., 2005.
- [22] T. Kapler and W. Wright, *GeoTime Information Visualization*, *Proc. of IEEE InfoVis*, 2004.
- [23] Keith Andrews, Wolfgang Kienreich, Verdran Sabol, Jutta Becker, Georg Droschl, Frank Kappe, Michael Granitzer, Peter Auer and K. Tochtermann, *The InforSky Visual Explorer: Exploiting Hierarchical Structure and Document Similarities*, *Information Visualization*, 1 (3/4) (2002), pp. 166-181.
- [24] KeWei Tu, Miao Xiong, Lei Zhang, HaiPing Zhu, J. Zhang and Y. Yu, *Towards Imaging Large-Scale Ontologies for Quick Understanding and Analysis*, *Proceedings of the Fourth International Semantic Web Conference (ISWC2005)*, LNCS 3729/2005, Galway, Ireland, 2005, pp. 702-715.
- [25] T. Kohonen, *Self-organization of very large document collections: State of the art.*, *Proc. of the 8th International Conference on Artificial Neural Networks (ICANN98)*, 1998, pp. 65-74.
- [26] T. Kohonen, *Self Organizing Maps*, *Springer Series in Information Sciences*, Springer Espoo, Finland, 1994.
- [27] Leonidas Deligiannidis, A. P. Sheth and B. Aleman-Meza, *Semantic Analytics Visualization*, *In Proc. of the IEEE International Conference on Intelligence and Security Informatics (ISI-2006)*, San Diego, CA, USA, 2006.
- [28] M. Janik and K. Kochut, *BRAHMS: A WorkBench RDF Store And High Performance Memory System for Semantic Association Discovery*, *Fourth International Semantic Web Conference ISWC 2005*, Galway, Ireland, 2005.
- [29] A. Maedche and R. Volz, *The Text-To-Onto Ontology Extraction and Maintenance System*, *ICDM-Workshop on Integrating Data Mining and Knowledge Management*, San Jose, California, USA.
- [30] Noy N. F. Sintek, M. Decker, S. Crubezy, M. Fergerson and M. A. R. W. Musen, *Creating Semantic Web Contents with Protege-2000*, *IEEE INTELLIGENT SYSTEMS Bibliographic details*, 16 (2) (2001), pp. 60-71.
- [31] OntoLift Prototype, *WonderWeb: Ontology Infrastructure for the Semantic Web*, <http://wonderweb.semanticweb.org/deliverables/documents/D11.pdf>.
- [32] I. Polikoff and D. Allemang, *Semantic Technology* http://www.topquadrant.com/documents/TQ03_Semantic_Technology_Briefing.PDF, TopQuadrant Technology Briefing, 1 (1) (2003).
- [33] R. Angles and C. Gutierrez, *Querying RDF data from a graph database perspective*, *2nd European Semantic Web Conference (ESWC)*, Greece, 2005, pp. 346-360.
- [34] R. Apweiler, A. Bairoch, C. Wu, W. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. Martin, D. Natale, C. O'Donovan, N. Redaschi and L. Yeh, *UniProt: the Universal Protein knowledgebase*, *Nucleic Acids Res* 32: D115 - D119, Jan 2004.
- [35] R. W. White, B. Kules, S. M. Drucker and M. Schraefel, *Supporting exploratory search*, *Communications of the ACM*, 49 (4) (2006).
- [36] Ramana Rao and Stuart K. Card, *The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information*, *Proc. ACM Conf. Human Factors in Computing Systems*, 1994.
- [37] G. G. Robertson, S. K. Card and J. D. Mackinlay, *Information Visualization Using 3D Interactive Animation*, *Communications of the ACM*, 36 (4) (1993), pp. 57-71.
- [38] S. Sahoo, C. Thomas, A. Sheth, W. York and S. Tartir, *Knowledge Modeling and its application in Life Sciences: A Tale of two ontologies*, *The 15th World Wide Web (WWW, 2006) conference*, Edinburgh, UK, May 2006.
- [39] R. Spence and M. D. Apperley, *Data Base Navigation: An Office Environment for the Professional*, *Behavior and Information Technology*, 1 (1) (1982), pp. 43-54.
- [40] Stephen G. Eick and Graham J. Wills, *Navigating Large Networks with Hierarchies*, *Proc. of IEEE Conf. Visualization*, 1993, pp. 204-210.
- [41] Storey M.A.D., Noy N.F., Musen M.A., Best C., Fergerson R.W. and Ernst N., *Jambalaya: An Interactive Environment for Exploring Ontologies*, *Proc. of the International Conference on Intelligent User Interfaces (IUI)*, 2002.
- [42] Sunil Goyal and Rupert Westenthaler, *RDF Gravity (RDF Graph Visualization Tool)*, *Salzburg Research, Austria* <http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html> (Retrieved on Nov. 20 2006).
- [43] Taowei David Wang and Bijan Parsia, *CropCircles: Topology Sensitive Visualization of OWL Class Hierarchies*, *5th International Semantic Web Conference*, LNCS, Athens, GA, Nov. 5-9, 2006.
- [44] Thomas A. DeFanti, Maxine D. Brown and Bruce H. McCormick, *Visualization: Expanding Scientific and Engineering Research Opportunities*, *IEEE Computer*, 22 (8) (1989), pp. 12 - 25
- [45] TouchGraph LLC home page, <http://www.touchgraph.com/>.