Visualization of large schemaless RDF data

Jiří Dokulil
Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic
jiri.dokulil@mff.cuni.cz

Jana Katreniaková
Faculty of Mathematics, Physics and Informatics,
Comenius University, Bratislava, Slovakia
katreniakova@dcs.fmph.uniba.sk

Abstract

Since many XML documents do not contain any schema definition, we expected that there will be also RDF documents without RDF schema or ontology. Then the data can only be viewed as a general labeled directed graph and the idea to present the data to the user by drawing the graph seems natural. Because the data can be extremely large, it is impossible to display the whole graph at one time. Only a suitable start node is displayed and the rest of the graph can be explored by incremental navigation. To conserve space and show possible directions of further navigation to the user we have come up with a technique called node merging. By combining suitable graph drawing and navigation techniques we get a tool that can give the user good idea about structure and content of the data.

1. Introduction

Experience from the world of XML data shows that although there are many advanced ways to specify valid structure of a XML document (DTD, XML schema, …) real XML documents often do not contain this definition (the actual numbers range from only 7.4% up to as much as 52% depending on how the data samples are collected) or do not adhere to it [7]. The same could happen with RDF data. So a tool that can explore the data without any prior knowledge about their structure may be needed. Although only the future can tell if these assumptions are right or wrong, we believe the tool will be useful anyway. Since RDF data is basically a directed labeled graph it is natural to present the data in a visual manner. Combined with appropriate navigation technique this tool can give the user a good idea about content of the data, probably even better than if he or she just explored the ontology.

As we have already discussed in [3], working with RDF data brings up several issues. Most important of them is size of the data. The data can be huge (millions of nodes and edges) and contain nodes with extremely high degree (thousands or even hundreds of thousands). This not only limits the possibilities of drawing the graph but also the acceptable complexity (both time and space complexities) of the drawing algorithm.

In the creation of the visualization and navigation algorithm we leveraged the experience with the DataPile [1] and its use as a backbone for an enterprise data integration system. During the DataPile development we created a visualization tool called Tykadlo and this tool served as an inspiration for our ‘node merging’ technique. At the same time we have obtained real-life RDF data that gave us good idea about problems we will face in RDF visualization.

The visualization tool uses the Semantic web infrastructure that is currently being developed at the Faculty of Mathematics and Physics of the Charles University in Prague [11].

2. Data visualization

The visualization of the data is very useful, since the visual perception is the most important one for cognition of information. There are many ways to visualize the same data. In the area of graph drawing, several measurable criteria of ‘a good drawing’ have been introduced (area of the graph, number of edge crossings, number of bends, etc.). From the data visualization point of view the situation is not as simple, because these criteria are not based on experimental data [6]. ‘Good data visualization’ is very subjective and depends on the actual content of the data. Dynamic graph drawing techniques (an approach where the graph is gradually changed by adding or removing vertices or edges) makes preservation of a mental map one of the important aesthetic criteria [9]. The term mental map refers to structural cognitive information a user creates internally by observing the layout of the graph. It represents the user’s underlaying understanding of the information. Empirical analysis show, that a display where it is difficult for the user to maintain a consistent mental map can be misleading. How-
ever, this contradicts measurable criteria for 'good' drawing of the graph so a suitable compromise has to be found.

When visualizing schemaless RDF data we work with a labeled directed graph. The size of the graph causes several problems. First of all, just storing the data required for drawing of the graph in the memory can be a problem. Time required to draw the graph can be another issue especially if we want to modify or navigate through the graph. Last but not least, number of displayed objects can get so high that it can no longer be properly displayed. For instance, user may no longer be able to distinguish edges and vertices or read the content of the vertices.

According to [5], common approaches to visualization of large graphs include geometric zoom, semantic zoom and incremental exploration.

Since neither RDF schema nor ontology is available, semantic zoom can not be used. For this reason we chose the incremental exploration and visualization technique. Another advantage of this approach is the fact that it does not work with the whole graph which makes it less time and memory consuming. We give the user the possibility to explore the neighborhood of the displayed subgraph by extending the displayed part of the graph by one (or more) nodes. This way we create a navigation tree for the data. This tree stores the nodes and edges that are currently displayed to the user along with the nontree edges between displayed nodes.

2.1. Layout algorithm

Currently displayed graph is determined by displayed vertices and the navigation tree, that specifies the path from every node to the root.

One of the popular ways to display large graphs and especially trees is radial layout that places vertices on concentric circles with centers placed on some significant vertex (see Fig. 1(a)). Since we display vertices as rectangles that can not be rotated (they contain textual information) we need a layout of vertices where height in pixels (difference of y-coordinates) of an area corresponding to an angle of influence does not depend on the location of this area (this height is called vertical range in the following text). We modified the radial layout and created so called square layout (see Fig. 1(b) and [3]). It turned out that placing vertices into all four quadrants does not make better use of available space. But when nontree edges are added to the graph, this method generates less readable drawing of the graph with more crossing of the edges. For this reason, we decided to use only the first quadrant to place the vertices (see Fig. 1(c)).

First, we focus on drawing of the tree. Vertex \( v \) of the tree is represented by rectangle \( \Gamma(v) \). The height of the rectangle is required to be lower or equal to the width of the rectangle. This limitation is acceptable since labels of the vertices and their content consists mostly of URIs that trend to be long. On the other hand, we can easily change the height of the vertex by displaying only some of the lines that can not be rotated (they contain textual information). The corner of the rectangle \( \Gamma(v) \) that lies on the layer is denoted \( \gamma_0(v) \) in the following text, while the opposite corner is denoted \( \gamma_1(v) \).

Let \( T = (V, E) \) be a tree rooted in the vertex \( r_T \). By \( L(h) \) we denote the vertices on the level \( h \) of the tree \( (L(0) = \{r_T\}) \). We assign the angle of influence \( (\alpha_1(v),\alpha_2(v)) \) to the vertex \( v \), where we fit the whole subtree rooted in \( v \). Having \( \alpha_1(v),\alpha_2(v) \in (0,90) \) and \( r \) be the radius of the depicted layer, then the vertical range is

\[
D = r \left( \frac{\sin \alpha_1(v)}{\sin \alpha_1(v) + \cos \alpha_1(v)} - \frac{\sin \alpha_2(v)}{\sin \alpha_2(v) + \cos \alpha_2(v)} \right)
\]

We fit the successors of \( v \) into this vertical range. Let \( v_1 \ldots v_k \) be the children of the vertex \( v \) and let \( v_i \) have a size of \( H(v_i) \times W(v_i) \). If the minimal distance between vertices is \( \delta \), then the minimal required vertical space for the children of \( v \) is \( \sum_{i=1}^{k}(H(v_i) + \delta) \). Hence the inequality \( D > \sum_{i=1}^{k}(H(v_i) + \delta) \) should hold.

![Figure 1. Layouts used to draw large graphs](image-url)
Vertices coordinates The layout algorithm first displays the root on the coordinate origin (i.e. \( r(0) = 0 \)). For each depth \( h \) of the tree (beginning with \( h = 1 \)) the algorithm works as follows (see also Fig. 2):

Let \( r_{\text{cont}} \) be such radius, that triangle \([r_{\text{cont}}, 0], [0, 0], [0, r_{\text{cont}}] \) completely contains all vertices in layers \( L(1) \ldots L(h - 1) \).

For each vertex \( v \in L(h - 1) \) the angle of influence has already been computed. Let \( v_1 \ldots v_k \) be the children of \( v \) and \( H(v_1) \ldots H(v_k) \) their heights. From the inequality \( D > \sum_{i=1}^{k} (H(v_i) + \delta) \) we compute the minimal required radius \( r_{\text{min}} \) for the children of \( v \).

Let \( r \) be the maximum of the minimal required radii and the radius \( r_{\text{cont}} \). The vertices from \( L(h) \) will be placed on layer with radius \( r \).

Radius \( r \) of the square and the angle of influence of vertex \( v \) determine the vertical range \( D \) for the subtree rooted in \( v \).

The distance \( \delta \) between children of \( v \) has to be recomputed from the inequality \( D > \sum_{i=1}^{k} (H(v_i) + \delta) \).

Now, having global parameter \( r \) -radius of the layer and for each vertex \( v \in L(h - 1) \) the parameter \( \delta(v) \). We can compute the display coordinates of children of \( v \) and their angles of influence. More formally, for each \( v_i \) with height \( H(v_i) \) we determine the angle of influence of \( v_i \) and the coordinates of \( \gamma_0(v_i) \). To compute the coordinates we only have to compute angle \( \beta_0(v_i) \) which is the angle of the line connecting the origin of the coordinates and \( \gamma_0(v_i) \). Together with the radius of the layer the vertex is positioned on, the coordinates can be computed easily.

```
LAYOUT ALGORITHM(T)
1 \( \gamma_0(r_T) \leftarrow [0, 0] \)
2 (Place the root vertex \( r_T \) to the coordinates origin)
3 \textbf{for each } \( h \) in \{1,2,\ldots\} \textbf{ do}
4 \textbf{ for each } \( v \) in \( L(h - 1) \)
5 \hspace{1em} \textbf{ do COUNT}(r_{\text{min}}(v))
6 \hspace{2em} \( r \leftarrow \max\{r_{\text{cont}}, \max\{r_{\text{min}}(v) \mid v \in L(h - 1)\} \} \)
7 \textbf{ for each } \( v \) in \( L(h - 1) \)
8 \hspace{1em} \textbf{ do COUNT}(\delta(v))
9
10 \textbf{ for each } \( v \) in \( L(h - 1) \)
11 \hspace{1em} \textbf{ do } (\alpha_1(v), \alpha_2(v)) \text{ is the angle of influence of } v
12 \hspace{2em} v_1 \ldots v_k \text{ are children of } v
13 \hspace{2em} H(v_i) \text{ is the height of the vertex } v_i
14 \hspace{2em} \alpha_1(v_i) \leftarrow \alpha_2(v)
15 \hspace{2em} \textbf{ for } i = 1 \text{ to } k
16 \hspace{3em} \textbf{ do } \alpha_2(v_i) \leftarrow \alpha_1(v_{i-1})
17 \hspace{4em} \gamma_{\text{aux}} \leftarrow \frac{r - \sin \alpha_2(v_i)}{\sin \alpha_2(v_i) + \cos \alpha_2(v_i)} \theta_0\beta_0 \gamma_0(v_i)
18 \hspace{5em} \beta(v_i) \leftarrow \arctg \left( -\frac{y_{\text{aux}} - \delta(v)}{x_{\text{aux}} - \gamma_0(v_i)} \right)
19 \hspace{5em} \alpha_1(v_i) \leftarrow \arctg \left( -\frac{y_{\text{aux}} - \delta(v)}{x_{\text{aux}} - \gamma_0(v_i)} \right)
20 \hspace{5em} x(v_i) \leftarrow r - \frac{r}{\gamma_0(\beta(v_i)) + 1} \theta_0\beta_0 \gamma_0(v_i)
21 \hspace{5em} y(v_i) \leftarrow r - x(v_i)
22```

\[ \text{Figure 2. Layout algorithm – explanation} \]

Tree edges Edge \((u, v)\), where \( u \) is predecessor of \( v \), is displayed as a line connecting \( u \) and \( v \). There are several possibilities on drawing the line. These include:

- connecting \( \gamma_1(u) \) and \( \gamma_0(v) \) (see Fig. 3(a)). It can easily be proved that such line cannot intersect with any vertex. Practical experiments have shown, that although this method is theoretically perfect, the results do not look ‘nice’ when viewed by human user.

- connecting \( \gamma_0(u) \) and \( \gamma_0(v) \) (see Fig. 3(b)). When drawn like this, the line can intersect with another vertex, but it is unlikely to happen in practice.

- connecting centres of vertex \( u \) and vertex \( v \) (see Fig. 3(c)). Just like with the previous method, intersections...
Figure 3. Tree edges

can occur, but they are unlikely. Users considered this to be the best looking option.

Nontree edges The nontree edge $e = (u, v)$ can be also represented as a line between $u$ and $v$. However, this line can intersect other edges or vertices. Since we expect the graph to be sparse, the number of such crossings will not be relevant. Moreover, we allow the user to manipulate the view on the graph, if it gets hard to read.

2.2. Navigation

We need to provide the user not only with well-arranged drawing of the graph but also with means to alter the view of graph to match his or her needs. We decided to allow the user to extend the view by adding a neighbor of an already displayed node.

Node merging We use the technique called node merging to help the user navigate the graph. Vertex does not contain only its label but also list of incoming and outgoing edges. This allows us to present the neighbors of the vertex to the user without using too much space. Important advantage of this approach is the fact that the user picks only the neighbors he or she is interested in and the view is then extended only by these vertices. This way we eliminate problem that a RDF node can have thousands (or even hundreds of thousands) of neighbors. Without node merging we would either have to display all of the neighbors which would hardly create a well-arranged and readable drawing of graph or the algorithm would have to pick only a few of the neighbors to display. If node merging is used and the number of neighbors is small, the neighbors can be displayed directly in the vertex. If the number is higher, the list of neighbors is displayed in a separate window with the option to filter the displayed entries, which allows handling of even nodes with large number of neighbors.

Node merging is also useful for displaying certain special type of nodes. RDF data usually contain nodes representing certain object with outgoing edges representing its properties, e.g. a person together with his or her name, date of birth, etc. Merged node for the person will contain the name and other information directly so the user can see them without expanding the neighbors. Furthermore a lot of drawing space is conserved since the user will probably be interested in these values and would expand all of the neighbors which may mean adding tens of vertices.

Selection of the start node First step of the navigation is picking a suitable start node. In [3] we have considered several strategies. Picking a random node is a simple approach based on an assumption that no matter which node is picked, the user can navigate to an interesting part of the graph with just a few steps. This makes it the preferred method at the moment (see [3] for more options) but we plan to combine it with the analysis of the user behavior in the future. We may observe the way previous users explored the data and pick the starting node according to these observations. For instance by choosing the node that was visited most times.

The selected starting node is displayed as a merged node in the origin of the coordinate system. This way the user is immediately presented with possible directions of further navigation. At this moment, the navigation tree consists solely of this node.

View expansion Let $G = (V, E)$ be the currently displayed graph and $T = (V, E')$ the navigation tree. Then $E' \subseteq E$ holds. The user selected node $v \in V$ with a set of
neighbors \( N(v) \). If \( N(v) \subseteq V \) then all neighbors of \( v \) are already displayed and \( v \) can not be used for further navigation. In the other case, let \( N'(v) = N(v) \setminus V \) be the set of yet undisplayed neighbors of \( v \) and let \( u \in N'(v) \). Then we can extend the displayed graph and navigation tree like this:

- The set of nodes is extended by \( u \), that is \( V \leftarrow V \cup \{ u \} \).
- The set of edges is extended by all edges between \( u \) and the already displayed nodes
  \( (E \leftarrow E \cup \{ (u, w) \mid w \in V \land \exists p : (u, p, w) \in RDF \} \cup \{ (w, u) \mid w \in V \land \exists p : (w, p, u) \in RDF \}) \).
- The edge between \( u \) and \( v \) is added to the navigation tree \( (E' \leftarrow E' \cup \{ (v, u) \}) \).

Drawing of the new view can be done in several different ways. The most simple one is to completely redraw the whole graph. This way we use the least space to draw the new view. Important disadvantage is that it does not preserve the mental map of the user, since the displayed vertices can be moved to completely different places. Furthermore, redrawing the whole graph is not necessary. Usually, we just have to redraw the subtree rooted in the ancestor of the newly added vertex. This means redrawing the angle of influence of the ancestor. This can be done by a simple recursive algorithm that is based on the layout algorithm. Let \( v \) be the ancestor of the newly added vertex \( u \) and \( v_1 \ldots v_k \) the descendants of \( v \).

\[
\text{REDRAW}(v) \\
1. \quad \text{if } v \text{ has no children then return} \\
2. \quad \text{COMPUTE}(E, d) \quad // \text{see layout algorithm} \\
3. \quad \alpha_1(v_0) \leftarrow \alpha_2(v) \\
4. \quad \text{for } i = 1 \text{ to } k \\
5. \quad \alpha_2(v_i) \leftarrow \alpha_1(v_{i-1}) \\
6. \quad y_{aux} \leftarrow r \cdot \sin \alpha_2(v_i) + \cos \alpha_2(v_i) \\
7. \quad \beta(v_i) \leftarrow \arctg \left( \frac{y_{aux} - \frac{d(v_i)}{2} - H(v_i)}{r - (y_{aux} \cos H(v_i))} \right) \\
8. \quad \alpha_1(v_i) \leftarrow \arctg \left( \frac{y_{aux} - \beta(v_i) - H(v_i)}{r - (y_{aux} \cos H(v_i))} \right) \\
9. \quad \text{for } i = 1 \text{ to } k \\
10. \quad \text{do REDRAW}(v_i)
\]

We do not have to compute coordinates of vertices outside of the angle of influence that belongs to the parent of the newly added vertex. Their coordinates are stored as the angle \( \beta \) and the radius of the layer they are positioned on. By changing this radius we get new coordinates of the vertices without any need to run the algorithm on them. Of course, this method is not optimal with respect to the total amount of space required to draw the graph, but we decided to use it anyway because it preserves the mental map of the user. As a future improvement, we are considering the idea of using the angle of influence of neighboring vertices if they are not completely used up.

### 3. Existing approaches to RDF visualization

Since RDF data have been around for quite some time, there are already several tools that try to visualize it. Many
of them display the whole graph, which is not suitable for large data because it requires too much resources and the resulting view is not clear. We have selected several typical examples that represent trends of RDF visualization.

- **RDF gravity** [4] works primarily with the whole graph and only offers optional filtering to hide some vertices. Therefore it is most useful to explore ontologies and small data.

- **Node-centric RDF Graph Visualization** [10] is one of the few tools that do not try to display the RDF graph precisely. It always displays tree of node’s ancestors and descendants. If any of them can be reached by more than one path (which would create a nontree edge in our solution) the vertex is displayed multiple times (once for each path) which preserves the tree structure of descendants and ancestors. One disadvantage of this tool is that it only displays two levels of ancestors and descendants and does not try to handle nodes with high degree.

- **Paged Graph Visualization (PGV)** [2] is similar to our tool because it does not try to display the whole graph. It utilizes Ferris-Wheel method to display nodes with high degree. The neighbors are displayed in a circle around the central vertex and rotated to allow clear view of any single neighbor. This approach can be used for nodes with at most hundreds of neighbors. The problem of the Ferris-Wheel approach is that when user navigates the graph in certain directions the view can become hard to read and the mental map of the user is not preserved.

Despite significant diversity of available visualization tools, only a few of them can be used to visualize large data. None of the tools uses technique similar to node merging and all of them run into trouble when the data contain nodes with very high degree although such nodes can commonly be found in the real world data.

### 4. Conclusion

We have developed algorithms for visualization and navigation of large RDF data. Algorithms do not require ontology or schema to work correctly and can handle nodes with very high degree thanks to our node merging technique. There is currently no other tool we are aware of that uses it. Most other tools are suitable only for relatively small data. Our algorithms were created with experience from working with huge real-life RDF data.

In the future we plan to improve method of selecting the starting node, implement better handling of nontree edges and better computation of vertical ranges that would take the size of the subtree into account.

### References


