Scalable Data Analysis on Provenance Data



**Jigar Dalwadi**

**01684802**

CIS 602-01

Scalable Data analysis

# INTRODUCTION

The idea of the project is make a dynamic provenance graph and deal with the nodes. We are using dagre js to make graph. Provenance graph can be too large and for that reason it can be very hard to understand. Our goal is to find a solution which helps us to minimize the number of nodes. By minimizing the nodes we are making the graph more easy to understandable.

# DATASET

Here we are using PROV JSON data. Here the provenance data contains entities, activities and agents. The entities are charts,datasets etc. The activities are connected to entities according to their use with the label on its edges. The agents are the nodes which are performing the activities on the different entities.

# IMPLEMENTATION

We create a javascript which takes input from from our json file. I have created functions for nodes and edges. These functions works on key value pair. I have given keys as an input to create nodes and edges. After creating the graph I have added rendering and zoom in zoom out functions. Procedure to make it scalable.

# PROCEDURE

1. After getting the graph I am applying Dijkstra's Algorithm to get the distance from a particular node to it’s connected node.

“”var dij = new dagreD3.graphlib.alg.dijkstra(g,"ex:composition",);””

This will return the distance of all nodes connected to it. After getting these we’ll create one function which will take the variable from dijkstra’s algo.

From the graph we notice some patterns, on which we’ll apply our scalability approaches.

2.We are first calculating the edges coming in the node and also the edges going out from that node

Eg: - var incoming1 = g.inEdges(v); here v is the first node from the dij var

var outEdges1 = g.outEdges(w); here w is the second node from the dij var

3. After getting the edges we ‘ll delete the nodes which we want to combine in one node. After deleting those nodes we’ll create one node which will take the inputs of the deleted nodes. Also it will give the output edges which were there from the deleted nodes.

Eg: - g.setNode(a, {lable: a, shape: "ellipse"});

for(var item in arrayObject1) {

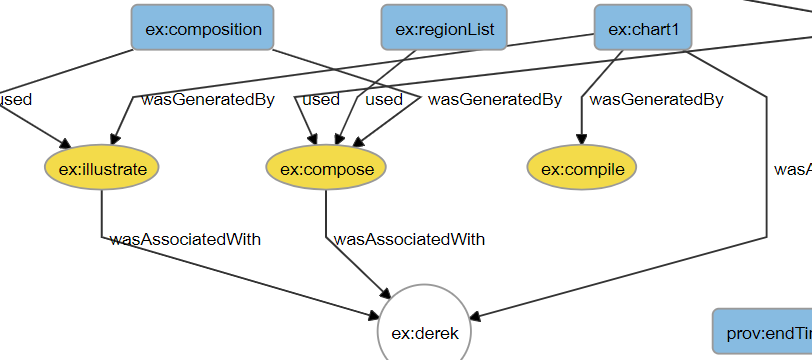
g.setEdge(arrayObject1[item].v,a);

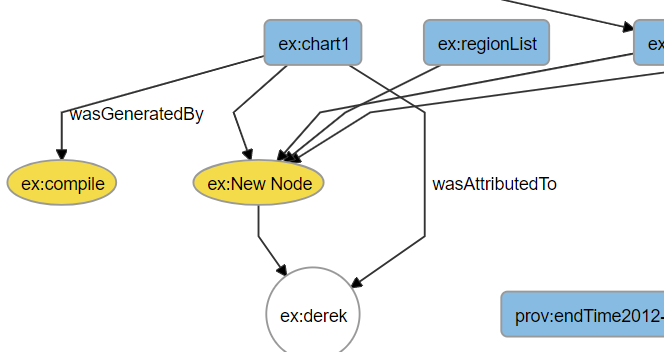
}

4. We’ll create a button which will execute this function which will combine the set of nodes and makes it into one single node. By pressing this button again you will get the original graph back.

# RESULTS

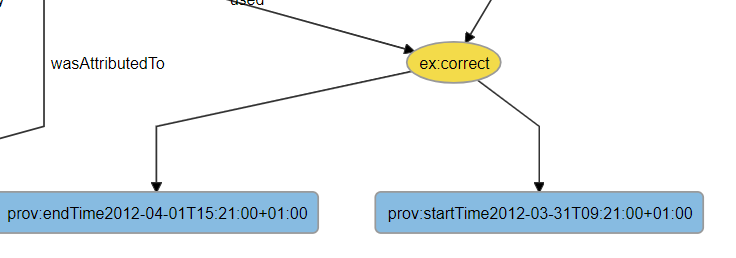
By combining these nodes we are making the graph easy to understand. The basic structure of the nodes was like this



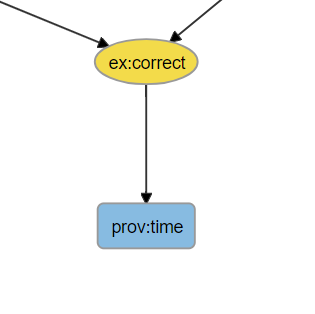
Here you can see that there are two nodes having same inputs from the node ex:composition and these two nodes are giving output at the same node. So after applying the dijkstra’s algorithm we are deleting these nodes and creating one node instead. 

As you can see that the complexity of that part is decreased.

One more thing I have done is combining two leaf nodes which are having same properties and same parents.



These prov:endtime and prov:starttime will become one node as prov:time



# CONCLUSION

From these graph we can say that by combining some nodes we are able to decrease the complexity. If in these small graph we can say the differences that it will be a big help when we’ll apply these in the bigger graph. It will make the analysis way more easier as it will make the graph easy to understand.

# REFERENCES

1. <https://github.com/cpettitt/graphlib/wiki/API-Reference#compound-graphs>
2. For getting data and creating the graph <https://github.com/cpettitt/dagre-d3>
3. <https://www.w3.org/TR/prov-primer/>