HighOnGraph: A graph visualization and Query Tool

Charupriya Sharma [*]
IIIT-Delhi
New Delhi, India
charupriya11037@iiitd.ac.in

Prateek Mehra
IIIT-Delhi
New Delhi, India
prateek11078@iiitd.ac.in

Utkarsh Gupta
IIIT-Delhi
New Delhi, India
utkarsh11117@iiitd.ac.in

Srikanta Bedathur
IBM India Research Lab
New Delhi, India
bedathur@iiitd.ac.in

Rajiv Raman
IIIT-Delhi
New Delhi, India
rajiv@iiitd.ac.in

ABSTRACT

HighOnGraph(HOG) is a graph visualization designed for very large graphs. It uses a recursive hierarchical community detection in combination with a powerful javascript frontend to render a visualization of graph datasets stored in any graph database in real time. It provides features for user-friendly exploration of the graph on a browser window. HOG also provides community based filters and a unique shortest path query feature that can be used to find paths across the hierarchical communities' layout. We present the role of these dynamic features of HOG in analysis of structure and properties of real-world networks. HOG has been tested with graphs of size exceeding 75,000 nodes and 900,000 edges.

Keywords

Graphs, visualization, shortest path heuristic, hierarchical community detection

Categories and Subject Descriptors

E.1 [**Data Structures**]: Graphs and Networks;Trees; G.2.2 [**Mathematics of Computing**]: :Discrete Mathematics; H.5.2 [**Information Systems**]: Interfaces and Presentation

1. INTRODUCTION

A graph can be used to model a variety of real world networks like Biological Networks, Computer Networks, Social Networks etc. These graphs can vary from a few hundreds of nodes to few billion nodes. As an example, a dataset could be a co-authorship graph, like DBLP computer science bibliography, where nodes represent researchers, and edges between two researchers(represented as nodes) A and B from universities U1 and U2 respectively. could represent a paper P that they co-authored. As of 2012, the DBLP collaboration network had over 300,000 nodes and 1,000,000 edges. Authors could have attributes like name, institute affiliation and research interests. Papers could have details about the conference, citations, etc.

Existing visualisations like Gephi [4] simply layout the graph in its entirety on screen. Not only do these layouts take a long time to render, but when complete, are simply impossible to explore unless the graph was very small. JUNG [7] tries to solve this problem via clustering, but the algorithm it uses are slow, and lack a hierarchical nature, so it again fails to minimize data on screen for large datasets.

Therefore, there is a need for a scalable, user friendly graph visualization tool that can manipulate and analyze the structure of large networks. Such a system will allow us to examine, explore and find properties in large graph datasets that we can not easily isolate using a standard graph plot.

The goal of HOG is three-fold:

1. Implementation of a hierarchical community detection algorithm that exploits properties of a graph like it's degree distribution, average shortest path length and clustering coefficient. Community detection increases the performance of the visualization by reducing the data to render on the screen, as well as highlight properties of the graph for a smoother user interaction.

2. Implementation of a shortest path heuristic that exploits the hierarchical community structure of the graph to execute the shortest path query.

3. To ensure that this system scales well for very large datasets. Also, the system should be fast and responsive to the user input and should not make the user wait.

Focusing on these parameters, the HOG project brings a graph visualization that has a layout that makes it easy to explore graph datasets. A visualization window is accompanied by a right aligned query panel, which can be used easily by even an uninitiated audience.

[*]The names are listed in no particular order, all authors have contributed significantly towards building of the system.

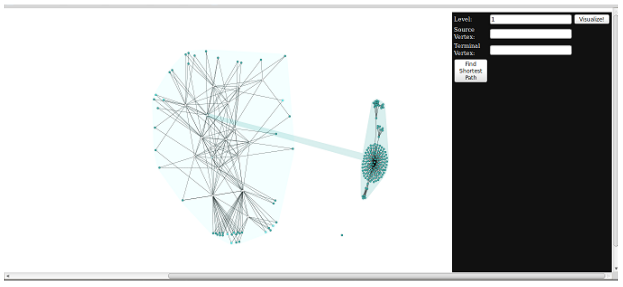
Level: 1

Figure 1: Exploring a Community in HOG

2. SYSTEM DESIGN

The backend of HOG is a graph database. The graph is passed to the community detection algorithm as an edgelist. This algorithm outputs a hierarchical community detection of the graph in the form of a JSON file. The communities that a node is assigned to at each level are also entered as node attributes in the database. The frontend visualizes this JSON file and has a query module for shortest path between both nodes and communities.

2.1 Backend

The graph is stored in Neo4j [3] which is a NOSQL, ACID supported, highly scalable graph database. Neo4j allows support for many graph traversal languages like Gremlin, which is open source . We chose Gremlin to query the database as its use is not restricted to just Neo4j, and we could have the flexibility of switching to another database in the future without major changes to our code. We used Neo4j as our major database, we read our graph from Neo4j using a Gremlin script. Data is transferred to the community detection algorithm by a combination of Gremlin and Java programs.

2.2 Shortest Path

In our layout, we represent the graph as an aggregation of communities. A node in the visualization could actually represent a community, and could be opened up to explore the subgraph of the community. A shortest path query, therefore, could have these community nodes as source and/or targets. In our co-authorship graph example, a user might want to know the shortest path between a linguistics research group in university U1 to an AI department in university U2.

Traditional shortest path algorithms like Floyd Warshall or Dijkstra's algorithm are not useful because we do not know the source and target nodes in term of the dataset, and that there could be many nodes in one of the communities as potential choices. Running these algorithms would compromise the goal of a fast and responsive system.

Our heuristic uses the edges which are between the members of the communities in the original graph. If the community is disconnected, then there is a possibility of giving shortest paths which do not exist by the heuristic. This is because, let us say there are three communities A,B and C. B has two disconnected components B1 and B2. In the original graph, a node in B1 is connected to a node in A and a node in B2 is connected to a node in C. Now, there is a edge between A and B and B and C. If we query the shortest path between A and C, then a possible path could be A -> B -> C, which is not true and hence we needed a community detection algorithm that could not give a disconnected community.

2.3 Community Detection

Several existing implementations commonly used for community detection lacked any recursive nature to community detection[2], often isolating most of a well connected graph as a large community. Such implementations again face the issues with effective visualization. Existing algorithms also had either high time complexity[5], or often yielded communities that were disconnected[6]. It was crucial for our shortest path module (as explained earlier)- that the subgraph associated with each community be connected to avoid false positives.

Our algorithm for community detection isolates communities using how low degree nodes connect to high degree nodes. The algorithm recursively partitions the graph/community subgraph into communities till there is no further partitioning possible. In the universities example, this could mean partitioning the departments, the research groups within each department and then again partitioning a professor and his PhD student under each department. This ensures that communities at a particular level are not very large graphs and are easy to render on the front end.

2.4 Frontend

We developed a web browser based tool that used the force directed layout in D3.js [1], which is a JavaScript library for manipulating data visually. Our frontend visualizes the graph as a set of hierarchical clusters, which could be clicked on to expand into groups of communities at successively lower depths, until the lowest level in the hierarchy (nodes) is reached.

The visualization computes placement for each of the nodes to give least overlapping and crossing nodes and edges respectively. There are other parameters in D3 that let you set the nodes even more effectively by setting things like charge, gravity and friction between nodes to make the visualization smoother and well spread out. The visualization is zoomable and can be panned to see overflowing nodes and links, making it easier to see through overlapping and overcrowding.Clusters are color coded according to hierarchy. Labels appear on mouseover, and they indicate unique identifiers for the nodes and the for links they indicate the source and target node identifiers.

Our visualization uses convex hulls for limiting scattering of nodes in a cluster. Hulls bind a cluster to a highlighted space. This helps in better identification of structure in a cluster. The hulls also highlight the lowest abstraction level being displayed at the moment so that it is easier to identify these nodes. A community node is connected to the subgraph associated with it via a tunnel link to it's hull for easy navigation. An open subgraph can be closed by clicking on its parent community node.

3. SYSTEM FEATURES

Interactive visualization On loading a graph into HOG, it renders a visualization of the top level of communities detected as a set of community nodes, and edges that represent inter-community interactions. The system has features like zoom and pan feature which will allow the user to explore the graph using usual mouse actions. The nodes in

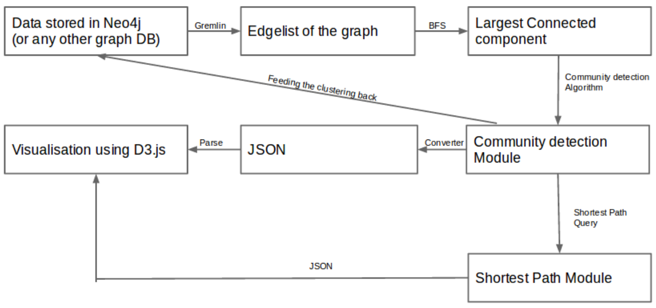
Data stored in Neo4j (or any other graph DB)
Gremlin
Edgelist of the graph
BFS
Largest Connected component
Community detection Algorithm
Feeding the clustering back
Visualisation using D3.js
Parse
JSON
Converter
Community detection Module
Shortest Path Query
JSON
Shortest Path Module

Figure 2: System Pipeline

different hierarchies are bound together in a convex hull, each community being represented by a different colour for simplicity. The colors of nodes grow increasingly darker to indicate smaller size .

Shortest Path Query Method A user can choose either a node in the dataset, or a community node as source and target in a shortest path query by simply entering their labels in the shortest path query panel. The system provides a heuristic for shortest path queries across communities. The nodes and edges in the shortest path returned are highlighted in the visualization. Community nodes participating in the shortest path can be clicked open to see how the path goes through the subgraph associated with that community node. In our university examples, this feature could be potentially be used to query the shortest path across research groups of different universities. The path could be explored further to see which people participate in this path.

Community Level-Based Filters Our algorithm computes community detection in multiple levels. A user may choose to input a level and see how the graph looks at that level of aggregation. This could potentially be used to visualise how the departments of different universities interact, instead of seeing the top level of inter-university interaction

4. DEMO PLAN

The goal of the demo is to illustrate the use of HOG in visualization a large graph via hierarchical community detection and executing shortest path queries in this layout. This demonstration will show the use of HOG in rendering an efficient visualization and querying method of very large networks.
The user will be able to load a graph from any existing graph saved in Neo4j. They will be able to experience the system, its smooth visualization of the output of the community detection algorithm and will be able to query these communities via the shortest path. The users will also be able to choose to filter out nodes of particular level of community detection. We will also demonstrate how our system is able to visualise graphs which the current state of the art systems struggle with.
A demo video is available at `https://www.youtube.com/watch?v=jhtIcNcUw0g&feature=youtu.be`

5. PERFORMANCE MEASURE

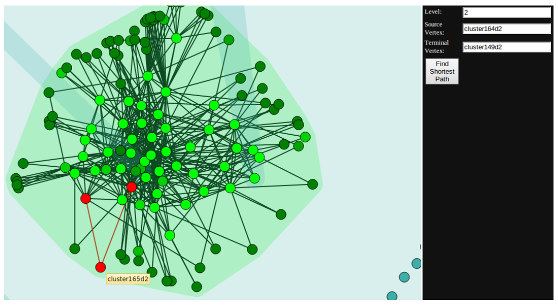

Figure 3: Shortest path between two communities in DBLP

We tested the system using a graph of up to 77360 nodes and 905468 edges on a 4 GB RAM, Intel(R) Core(TM) i5 CPU M 480 @ 2.67GHz processor system. The visualization was rendered on Google Chrome and Mozilla Firefox browsers, and the system rendered smoothly and gave a good user experience.

6. FUTURE WORK

Better running time of the community detection algorithm could be achieved by using more advanced (and efficient) data structures. We would like to modify our community detection algorithm such that the node and edge attributes are taken into account. In coming versions of HOG, we would explore the temporal nature of some graphs and try to visualise how communities change over time. We would also like to improve and extend the shortest path heuristic.

7. REFERENCES

[1] M. Bostock. D3. js-data-driven documents, 2012.
[2] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5), 2006.
[3] N. Developers. Neo4j. *Graph NoSQL Database [online]*, 2012.
[4] N. Gephi. Gephi-the open graph viz platform, 2010.
[5] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.
[6] M. E. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.
[7] J. OâĂŹMadadhain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey. Analysis and visualization of network data using jung. *Journal of Statistical Software*, 10(2):1–35, 2005.