Visualization of Large Dynamic Networks

Name: Yibo Yao (11252107)

Advisor: Dr. Larry Holder

School of Electrical Engineering and Computer Science

Washington State University, Pullman, WA 99164

PART I. Abstract

There is a growing number of graph-structured datasets, *e.g.*, social networks, communication networks and biological networks, available for scientific research. Most of them show a dynamic behavior with continuous addition/deletion/modification of nodes and edges. In this directed study project, we aim to build a general framework for showing how to visualize large dynamic networks using Gephi, a powerful graph visualization platform.

PART II. Introduction

Graphs are generally used to represent complex interactions among entities in a broad range of real-world applications. Lots of efforts have been made to discover interesting subgraph patterns from the graph-structured datasets during the last decade. However, with a large amount of dynamic networks emerging in various domains, researchers will need an efficient way to display the rapidly streaming nodes/edges of dynamic graphs in a live fashion. Due to the time-evolving nature and the large size of these dynamic graphs, the traditional static-snapshot approach of large graphs becomes unfeasible in visualizing their dynamics.

In this project, we develop a general framework using Gephi's Graph Streaming API to visualize a large dynamic network which is presented in a rapid stream of edges. We consider two popular dynamic networks: paper citation network and Twitter's retweet network, and feed their edge streams into Gephi's visualization pool through a low-level programming communication with Gephi's Graph Streaming API. Also, we make those dynamic graphs capable of displaying a few prominent nodes which may be considered as important components, *i.e.*, nodes with in-degrees/out-degrees beyond certain threshold values.

This project may help researchers visualize the implicit structures of most popular time-evolving networks and facilitate the development of algorithms for finding interesting subgraph patterns in them.

The rest of the report is organized as follows. Section III gives a brief introduction of the basic utilities including the software, APIs and data format. Then section IV presents how to construct dynamic networks from two large graph-structured datasets. Section V describes the class modules that we have designed. And finally some conclusions are drawn in Section VI.

PART III. Data Format, Software and APIs

In this section, we will briefly describe the data format for information interchange, the software and the APIs we have used in this project.

1. JSON

JSON (JavaScript Object Notation) is an alternative to XML and a standard format for data interchange [1]. Due to its language-independency, it has been used to transmit data consisting of attribute/value pairs between web servers and their applications [2]. Furthermore, JSON can be parsed and generated by a variety of modern programming languages.

JSON has two universal structures: object and array. An ***object*** is an unordered collection of attribute/value pairs, whereas an ***array*** is an ordered list of values. An object is embraced with '{' and '}', with each attribute followed by ':' and each attribute/value pair separated by ','. An array begins with '[' and ends with ']' and contains a set of values which are separated by ','. A ***value*** can be any of the following basic types: *number*, *string*, *boolean*, *array*, *object* or *null*. Figure 1 gives a general JSON representation of a user on social networks.

2. Gephi

Gephi [3] is an open-source software for visualizing and analyzing all kinds of networks. It is able to explore networks up to 50,000 nodes and 1,000,000 edges, and provides efficient layout algorithms for displaying graphs and rich tools for manipulating them.

Furthermore, Gephi allows users to develop plugins, which may make it more powerful in analyzing most of the networks in real world.

```
{
        “firstName”: “Jack”,
        “lastName”: “Kim”,
        “address”: {
                “street”: “Spokane Street”,
                “city”: “Pullman”,
                “state”: “WA”,
                “zipcode”: 99164
        },
        ……
}
```

Figure 1. A simple JSON example

The purpose of Gephi’s Graph Streaming API [4] is to display complex systems in real-time by connecting Gephi with external data sources. It gives a way to add, delete and modify nodes or edges in non-static graphs and makes those graphs evolve in a real-time fashion.

The main streaming format for data exchange supported by Gephi is JSON. JSON’s compactness makes all real-time systems possible to parse data as it arrives without waiting for the end of a stream. And currently, Gephi’s Graph Streaming API has implemented six types of JSON events (see Figure 2) with respect to dynamic graphs: 2 types of graph elements (nodes/edges) and 3 types of operations (add/delete/change).

- an: add node
- dn: delete node
- cn: change node
- ae: add edge
- de: delete edge
- ce: change edge

Figure 2. JSON streaming format supported in Gephi

And each event is wrapped up using a JSON-style:

```
{<event_type>:
        {<object_id>:
                {
                        <attribute_name>: <attribute_value>,
                        <attribute_name>: <attribute_value>,
                        ......
                }
        }
}
```

For example, adding a node with id "A", label "Node1" and size 2 can be wrapped up as follows:

```
{"an":
        {"A":
                {
                        "label": "Node1",
                        "size": 2
                }
        }
}
```

Figure 3. Gephi's graph updating event in JSON-style

In practice, it is recommended to put only one object in one JSON event in order to process the data quickly when it arrives and make it suitable for streaming cases.

The Server module in the Graph Streaming API allows Gephi to work as a master server in a REST architecture [4] and interact with any HTTP client for exchanging data. See [4] for details about how to start Gephi's Master Server and make data interactions between the server and any client in a streaming mode.

3. Twitter Streaming API

Twitter's Streaming API [5] gives real-time access to Twitter's global stream of tweet data. It offers three streaming endpoints: public streams, user streams and site streams, to customize different use cases. In this project, we focus on the public streams endpoint and establish a connection between our application and the endpoint using OAuth authentication method. And then public tweets on Twitter will be delivered lively to our application for further process.

Tweepy [6] is a Python library for connecting the Twitter API and make all API operations accessible in Python environment. One module in Tweepy, named ***StreamListener***, has been designed to facilitate the access to Twitter's Streaming API and provide various programming interfaces to process the raw data retrieved from the public streams. In order to make Tweepy interact properly with Twitter's Streaming API under OAuth mode, one has to apply for *consumer key* and *secret,* and *access token* and *secret* from Twitter's Developers [9].

PART IV. Two Dynamic Networks

In this project, we investigate two popular dynamic network datasets: paper citation network and Twitter's Retweet network.

1. Paper Citation Network

The paper citation dataset considered in this project is Arxiv HEPTH (high energy physics theory), which was originally released in 2003 KDD Cup [7], and was later refined into a XML-based representation [8]. Each paper is denoted by a node with a unique identifier in the graph, and if a paper i cites another paper j, there is a directed edge pointing from node i to node j. The raw dataset downloaded from [8] contains papers submitted in the period from January 1993 to April 2003. And each paper is associated with its submission date to Arxiv. In order to make the citation data flow into Gephi's workspace in a real-time fashion, we first convert the XML-based data into a set of edge streams and then store them in a plain text file.

For each newly submitted paper i, we first gather all the N_i papers from its reference list, and generate N_i lines in the plain text in the following format (suppose the submission date of paper i is d_i):

$$i, j_1, d_i$$

$$i, j_2, d_i$$

$$\ldots\ldots$$

$$i, j_{N_i}, d_i$$

Figure 4. File format of citation dataset

The papers $j_1, j_2, \ldots, j_{N_i}$ are those papers in paper i's reference list. When paper i is submitted into Arxiv on day d_i, N_i directed edges pointing from node i to nodes $j_1, j_2, \ldots, j_{N_i}$ will be generated. It is important to note that the papers to which i points have to be existing in Arxiv. Therefore, each line in the plain text file denotes an edge including the two nodes the edge connects and the creation timestamp of it. In the plain text file, these edges are sorted in an ascending order with respect to their creation timestamps.

2. Twitter's Retweet Network

On Tiwtter, a *retweet* is a re-posting of someone else's tweet to make it shared with the public. Among the huge volume of tweets generated per second on Twitter, a large part of them are retweets. In this project, we focus on the retweets regarding certain given topics (or hashtags) in Twitter's public streams.

By accessing Twitter's Streaming API through Tweepy, we are able to retrieve all retweets with respect to certain specified topics (or hashtags) from the real-time public streams. Each retweet has the author who did the re-posting and the user who wrote the original tweet, which can imply a retweet-relation between the author and the original user. So in the retweet network, the nodes are used to represent the users on Twitter with the usernames being the unique identifiers. And the edges are used to denote retweet-relations between them. If a user i re-posted another user j's tweet, there will be a directed edge pointing from node i to node j in the retweet network. And the timestamp when the retweet happened will also be included as the edge label in the network.

PART V. Class Modules

In this section, we will describe the class modules that we have developed to display the two mentioned dynamic networks in Gephi's visualization pool.

1. A Generic Module: *GephiJsonClient*

We first develop a generic module, namely *GephiJsonClient*, which aims to build a connection between Gephi's Master Server and the external data sources. Each graph updating event, *i.e.*, add/delete/change a node or an edge, will be converted into a corresponding JSON event and then be sent to Gephi's workspace in order for visualization. We have implemented six functions in *GephiJsonClient* to accomplish these tasks (see Figure 5).

The first three functions handle the request regarding a node event, and take that node's unique identifier as one argument. The optional argument is *node_attributes*, a Python dictionary, in which we may specify various attributes of a node, *e.g.*, size, color. The last three functions process an edge event request. The *addEdge()* function accepts an edge's identifier, the source and target nodes of that edge, as the first three arguments. The fourth argument *directed*, which is a boolean value, specifies whether the edge is directed or not. All these three functions will take *edge_attributes* as an optional argument, in which we can assign values to various attributes of an edge.

- addNode (*node_id* [,*node_attributes*])
- deleteNode (*node_id*)
- changeNode (*node_id* [, *node_atttibutes*])
- addEdge (*edge_id*, *source_node*, *target_node*, *directed* [, *edge_attributes*])
- deleteEdge (*edge_id*)
- changeEdge(*edge_id* [, *edge_attributes*])

Figure 5. Functions for graph updating events

These above six event-process functions are responsible for wrapping their events into the corresponding JSON format. Besides those functions, another important function worth mentioning is *send()*, which aims to open a http connection to Gephi's current workspace and send any of the six graph updating events to Gephi's Master Server. Once Gephi receives an event, it will reflect the update in the visualization pool and then make the displayed graphs behave dynamically.

2. *DataLoader* and *CitationStream*

Since the paper citation dataset is stored in a plain text file with a format of edge streams, we have developed a class, named *DataLoader*, to access those edge streams and load them into the main memory. In order to make the edges flow to Gephi's visualization pool in a live format, a member function, *sendData()*, is designed to fetch daily edge streams from the file.

We build another class, *CitationStream*, which actually combines all functionalities of modules *GephiJsonClient* and *DataLoader*. In *CitationStream*, we initialize a *GephiJsonClient* object and a *DataLoader* object. The *GephiJsonClient* object, namely *g*, will set up a connection between the external data source and Gephi's Master Server and make the server ready for receiving the streaming graph updating events. The

DataLoader object, namely *loader*, is responsible for sending data of daily edge streams to our application continuously. The *streamIn()* function serves as an intermediate station which accepts those daily edge streams, and transmit them to Gephi's Master Server by calling graph updating functions defined in *GephiJsonClient*. Figure 6 depicts the major steps of the whole procedure.

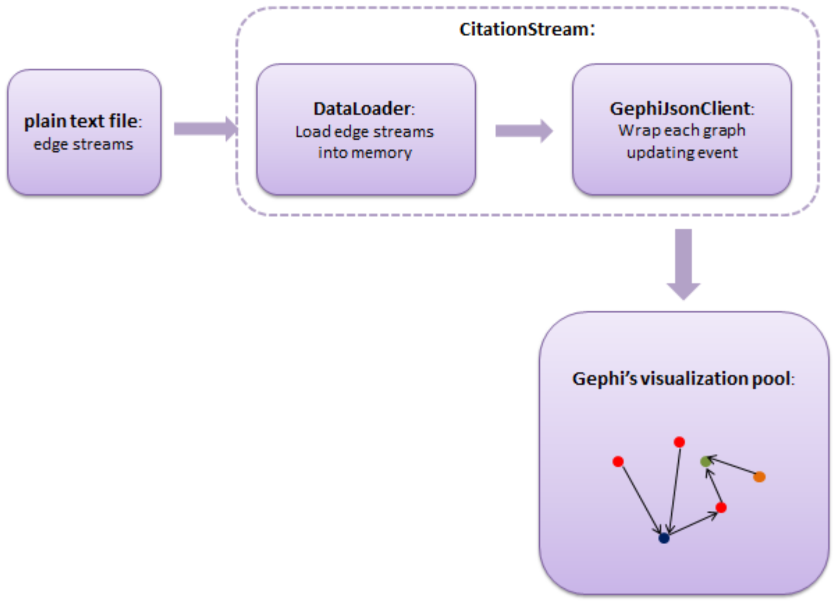
CitationStream:
plain text file: edge streams
DataLoader: Load edge streams into memory
GephiJsonClient: Wrap each graph updating event
Gephi's visualization pool:

Figure 6. Flow chart of visualizing the dynamic citation network

During the transmission, we record the in-degree and out-degree values of each node. The in-degree value corresponds to the number of citations that a paper has received at the current time. And the out-degree value denotes the number of papers that a paper has cited when it is published. A node's in-degree value will potentially change because it tends to get more citations when time evolves. However, a node's out-degree value is a constant since the number of papers in its reference list stays unchanged whenever new edges are streamed in. In Gephi's visualization pool, we highlight the nodes with in-degree or out-degree exceeds certain threshold values, namely *IN_THRESHOLD* and *OUT_THRESHOLD*, by calling *changeNode()* function from *GephiJsonClient* to set different colors and sizes for those nodes.

Additionally, in order to make the visualization pool neat enough, we have restricted the number of nodes to be displayed in the pool to 1,000. A queue structure is used to maintain the nodes which are currently displayed in the pool. When a new node streamed in, if the queue has enough space, that new node will be pushed into the queue. Otherwise, the oldest node which has stayed in the pool for a long time will be popped out from the queue and also be deleted from the pool by calling *deleteNode()*

function from *GephiJsonClient*. And then the new node is pushed into the queue and added into the visualization pool.

A new node flowing into the pool will trigger the creation of new edges, through which that node will connect to its referenced nodes. For those referenced nodes which are currently present in the queue, we simply feed new edges into the pool by calling *addEdge()* functions from *GephiJsonClient*. However, there may be cases that some of those referenced nodes do not exist in the pool because they have been popped out from the queue due to their ages. Then we have to check the in-degree value of each of these old nodes, and bring it back into the queue and make it visible in the pool if its in-degree exceeds IN_THRESHOLD.

3. *RetweetStream*

The *RetweetStream* class is inherited from the *StreamListener* class in Tweepy. The *on_status()* function has been overwritten inside *RetweetStream* and made possible to listen to the public Twitter streams. When a new tweet is published on Twitter, we are able to get access to all meta-information of that tweet and retrieve it for further process if it is a retweet. By parsing that retweet, we will get the author of the retweet as well as the user who has posted the original tweet, and generate a directed edge pointing from the author to the original user with timestamp of the retweet being the edge label. The edge is then fed into Gephi's visualization pool using the same mechanism from what we have done in *CitationStream* module.

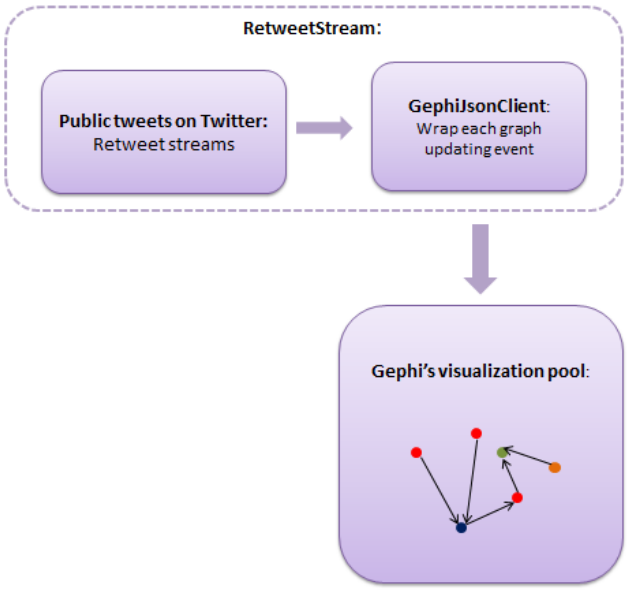
RetweetStream:
Public tweets on Twitter: Retweet streams
GephiJsonClient: Wrap each graph updating event
Gephi's visualization pool:

Figure 7. Flow chart of visualizing the dynamic retweet network

PART VI. Conclusion and Future Directions

In this directed study project, we have developed a general framework to display a large dynamic network in Gephi's visualization pool through a low-level programming communication with Gephi's Graph Streaming API. And we have shown how to feed the edge streams of two popular large networks into the visualization pool and make them behave dynamically.

We hope the presented project will help facilitate the development of graph mining techniques on large dynamic graph-structured data, especially in our future research on supervised learning in dynamic graphs.

REFERENCES

1. http://www.json.org/

2. http://en.wikipedia.org/wiki/JSON

3. https://gephi.org/

4. https://marketplace.gephi.org/plugin/graph-streaming/

5. https://dev.twitter.com/docs/streaming-apis

6. https://github.com/tweepy/tweepy

7. Gehrke, Johannes, Paul Ginsparg, and Jon Kleinberg. "Overview of the 2003 KDD Cup." ACM SIGKDD Explorations Newsletter 5.2 (2003): 149-151.

8. https://kdl.cs.umass.edu/display/public/HEP-Th

9. https://dev.twitter.com/