Visualization for Streaming Telecommunications Networks

Rui Sarmento[1,2], Mário Cordeiro[1], and João Gama[1,2]

[1] LIAAD-INESC TEC, University of Porto
[2] Faculty of Economics, University Porto

Abstract. Regular services in telecommunications produce massive volumes of relational data. In this work the data produced in telecommunications is seen as a streaming network, where clients are the nodes and phone calls are the edges. Visualization techniques are required for exploratory data analysis and event detection. In social network visualization and analysis the goal is to get more information from the data taking into account actors at the individual level. Previous methods relied on aggregating communities, k-Core decompositions and matrix feature representations to visualize and analyse the massive network data. Our contribution is a group visualization and analysis technique of influential actors in the network by sampling the full network with a *top-k* representation of the network data stream.

Keywords: Large Scale Social Networks Sampling; Data Streams; Telecommunication Networks.

1 Motivation

The analysis of social networks that emerge from a set of phone calls using regular services from a telecommunication service provider is a demanding problem. In these networks, a node represents a user and a phone call is represented by an edge between two nodes. Common networks in telecommunication services have millions of nodes and billions of edges where data from started phone calls flows at high speed. This is the reason why these networks are complex and difficult to analyse. Sampling from large network is known to be a hard problem to solve with typical hardware or software. State-of-the-art software and hardware reveal some limitations to deal with networks with more than a few thousands nodes and edges. Computational memory and power are the main constraints to perform the visualization of large networks. Even if the software is capable of representing a network of millions of nodes on the screen, the user may struggle to gather some valuable information from the visual outcome. In this work we propose processing the data as a stream of networked data with Landmark, Sliding Windows and *top-k* algorithm applications to enhance the network visualization enabling knowledge acquisition from the output. The main goal is to sample the data stream by highlighting the *top-k* nodes, providing a clear insight about the most active nodes in the network. We also present a case study of our methods applied to Telecommunication network data with several millions of nodes and edges. Results were obtained with a common commodity machine. In the following section we present an overview of previous work on the subject of social network visualization and summarization, focusing on the *top-k* algorithms. Section 3 describes the system architecture that was developed to simulate social network streaming and generate visualization. Section 4 has the details of the data case study concerning data description, algorithm developments and experiments. Finally, in Section 5, we underline major contributions and results and propose future directions.

2 Related Work

Two common strategies for sampling are *random sampling* and *snowball sampling*. In *snowball sampling* a starting node is selected. The network is built from that node, starting on its 1st order connections, moving to the 2nd order connections, 3rd order connection, and so on, until the network reaches the right size for analysis. This approach is easy to implement, but has some pitfalls. It is biased toward the part of the network sampled, and may miss other features. Nevertheless, it is one of the most common sampling approaches. The *random sampling*, randomly selects a certain percentage of nodes and keeps all edges between them. In an alternative approach, it randomly selects a certain percentage of edges and keeps all nodes that are mentioned. The main problem with this method is that edge sampling is biased towards high degree nodes, while node sampling might lose some structural features of the network.

2.1 Visualization

The definition of large-scale networks regarding number of nodes or edges diverges. Publications may consider a large-scale network ranging from dozens of thousands of nodes to millions of nodes and billions of edges. The main goal of any graph visualization technique is to be visually understandable. It is also desirable that the information is represented in a clear and objective way to convey knowledge to the viewer. To achieve this goal two types of graph representation, node-link and matrix graph representations [16] may be used. Visualization readability is highly related with the network size (number of nodes) and density (average number of edges per node). It is known that node-link representation has low performance with dense networks and requires aggregation methods reducing density to increase visual comprehensibility of the output. Matrix representation is usually combined with hierarchical aggregation [1]. Hierarchical clustering implies grouping the nodes but not their ordering. The main goal of this representation type is to have a fast clustering algorithm and meaningful clusters. Matrix representation methods may also rely on the reordering of rows and columns in the representation matrix instead of just clustering the nodes [12]. This type of ordered matrix representation might enhance the structure visualization because the data is more than simply clustered. The main drawback of this solution is that it is unfeasible for networks of millions of nodes that need a large amount of computations for reordering the matrix. More recently, Elmqvist et al. [7] introduced fast reordering mechanism, data aggregations and GPU-accelerated rendering to deliver higher scalability solutions. Other solutions rely on controlling the visual density of the graph view and restricting the clustering overlap probability to low levels [20]. Moreover a new probability based network metric was introduced by Ham et al. [10] to identify potentially interesting or anomalous patterns in the networks.

2.2 *top-k* Itemsets

The problem of finding the most frequent items in a data stream S of size N is mainly how to discover the elements e_i whose relative frequency f_i is higher than a user specified support ϕN, with $0 \leq \phi \leq 1$ [8]. Given the space requirements that exact algorithms addressing this problem would need [3], several algorithms were already proposed to find the top-k frequent elements, being roughly classified into *counter-based* and *sketch-based* [19]. *Counter-based* techniques keep counters for each individual element in the monitored set, which is usually a lot smaller than the entire set of elements. When an element is identified as not currently being monitored, various algorithms take different actions to adapt the monitored set accordingly. *Sketch-based* techniques provide less rigid guarantees, but they do not monitor a subset of elements, providing frequency estimators for the entire set.

Simple *counter-based* algorithms, such as *Sticky Sampling* and *Lossy Counting*, were proposed in [18], which process the stream in compressed size. Yet, they have the disadvantage of keeping a large amount of irrelevant counters. *Frequent* [6] keeps only k counters for monitoring k elements, incrementing each element counter when it is observed, and decrementing all counters when a unmonitored element is observed. Zeroed-counted elements are replaced by new unmonitored element. This strategy is similar to the one applied by *Space-Saving* [19], which gives guarantees for the *top-m* most frequent elements. *Sketch-based* algorithms usually focus on families of hash functions which project the counters into a new space, keeping frequency estimators for all elements. The guarantees are less strict but all elements are monitored. The *CountSketch* algorithm [3] solves the problem with a given success probability, estimating the frequency of the element by finding the median of its representative counters, which implies sorting the counters. Also, *GroupTest* method [5] employs expensive probabilistic calculations to keep the majority elements within a given probability of error. Despite the fact of being generally accurate, its space requirements are large and no information is given about frequencies or ranking. We adopted the *Space-Saving* algorithm described in [19] throughout our *top-k* method because it is a memory efficient application and guarantees most active nodes which is our goal.

3 Streaming Simulation System

This section presents the streaming system to support the visualization tasks. We briefly describe the software components and also present some example messages and protocols used to interconnect these same components.

3.1 Components

The developed system is based primarily on a MySQL database server. With the data conveniently indexed we used R as a language platform to work on and to represent the data that was streaming from the database.

Another requirement is the output availability in remote locations. The best way to do it would be to present the output in a web browser. For this task we chose sigma.js library, a JavaScript library dedicated to graph drawing [14]. It enables the network display on Web pages and may be used to integrate network exploration in rich Web applications.

To connect R output to sigma.js we needed an application running on real-time to make the bridge between the processing language and the browser. For this task we selected node.js that enables the use of web sockets communication. Thus, results may be published in real-time in a web browser. Joyent Inc. describes Node.js as a platform built on Chrome's JavaScript runtime to easily construct fast and scalable network applications [13]. Node.js uses an event-driven, non-blocking I/O model that, according to the authors, makes it lightweight and efficient, suitable for data-intensive real-time applications that run across distributed devices. Fig. 1 represents the system architecture. The network visualization was initially executed with Gephi software, but was abandoned in a early stage of development. Sigma.js was preferred throughout the project.

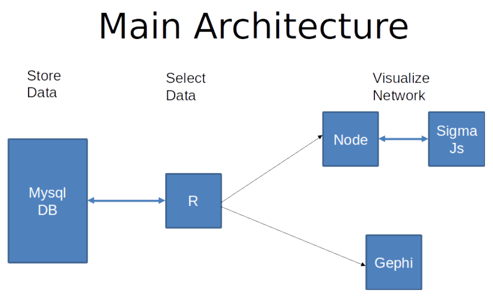


Fig. 1. Streaming System Architecture

In Fig. 1 the generation of messages in this system begins with R Language sending HTTP requests to node.js. The HTTP requests include information about the source and destination node that we wish to output to the final element in the chain, the user browser. After node.js receives the message from

R, it immediately produces a message through the established socket connection with sigma.js (embedded in the html page displayed in the user browser). The number of socket connections established to node.js is equal to the number of connected browsers. If more browsers are connected to the Node.js web server, more socket connections are established. This means that all connected browsers are simultaneously notified via a broadcast websocket message sent by the Node.js event dispatcher.

3.2 Landmark Windows

Algorithm 1 presents the pseudo code of the Landmark Window algorithm. This algorithm provides the representation of all the events that occur in the network starting at a specific timestamp, e.g., 01h48m09s on January 1st, 2012.

Algorithm 1 Landmark Pseudo-Code

```
Input: $start$, $tinc$                         ▷ start timestamp and time increment
Output: $edges$
 1: $R \leftarrow \{\}$                                              ▷ data rows
 2: $E \leftarrow \{\}$                                ▷ edges currently in the graph
 3: $R \leftarrow$ getRowsFromDB ($start$)
 4: $new\_time \leftarrow start$
 5: while ($R <> 0$) do
 6:     for all $edge \in R$ do
 7:         addEdgeToGraph($edge$)
 8:         $E \leftarrow E \bigcup \{edge\}$
 9:     end for
10:     $new\_time \leftarrow new\_time + tinc$
11:     $R \leftarrow$ getRowsFromDB ($new\_time$)
12: end while
13: $edges \leftarrow E$
```

This type of representation is not very useful because it implies a growing number of displayed events on the screen and decrease the comprehensibility of the representation, as this number surpasses some thousands of events. This landmark application is useful in other contexts, for instance, if the network is relatively small and the goal is to check all events in the network. The *top-k* application based on Landmark Window, described in Section 3.4, proved also to be a suitable approach for large network streaming data. It enables the focus on the influential individuals and discard less active nodes in very large networks. The alternative option for Sliding Windows [8] presented in the next subsection would be, in our case, incorrect because there would be a chance to remove less recent graph nodes. Those nodes may be included in the *top-K* list we wish to maintain.

Still, if the goal is to follow the evolution of full network events, the Sliding Windows method, described in the next subsection, is better as it only outputs the events in the current window with the size selected by the user. This option enables the visualization of large evolving networks over time and without compromising data processing performance with large amounts of data.

3.3 Sliding Windows

Dealing with large data streams presents new challenging tasks, for instance, dynamic sample representation of the data. The sliding window Algorithm 2 may be used to address this issue. This sliding window is defined as a data structure with fixed number of registered events [8]. In our case study each event is a call between any particular pair of nodes. As these events have timestamps, the time period between the first call and the last call in the window is easily computed. The input parameters of this algorithm are the start date and time and the maximum number of events/calls that the sliding window can have. The SNA (Social Network Analysis) model used in this application is full network directed because any nodes in the network are represented in the screen, for the particular window of events [11].

Algorithm 2 Sliding Window Pseudo-Code

```
Input: $start$, $wsize$, $tinc$        ▷ start timestamp, window size and time increment
Output: $edges$
 1: $R \leftarrow \{\}$                                    ▷ data rows
 2: $E \leftarrow \{\}$                     ▷ edges currently in the graph
 3: $V \leftarrow \{\}$               ▷ buffer to manage removal of old edges
 4: $R \leftarrow$ getRowsFromDB ($start$)
 5: $new\_time \leftarrow start$
 6: $p \leftarrow \{\}$
 7: while ($R <> 0$) do
 8:     for all $edge \in R$ do
 9:         addEdgeToGraph($edge$)
10:         $E \leftarrow E \bigcup \{edge\}$
11:         $k \leftarrow 1 + (p \bmod wsize)$
12:         $old\_edge \leftarrow V[k]$
13:         removeEdgeFromGraph($old\_edge$)
14:         $E \leftarrow E \setminus \{old\_edge\}$
15:         $V[k] \leftarrow edge$
16:         $p \leftarrow p + 1$
17:     end for
18:     $new\_time \leftarrow new\_time + tinc$
19:     $R \leftarrow$ getRowsFromDB ($new\_time$)
20: end while
21: $edges \leftarrow E$
```

3.4 *top-k* Networks

Algorithm 3 represents our version of the *top-k Space-Saving* algorithm. The *Space-Saving* algorithm is one of the most efficient, among one-pass algorithms, to find the most frequently occurring items in a streaming data. In our case study, we are interested in continuously maintaining the *top-k* most active nodes. Activity can be defined as making a call, receiving a call, or communications pairs of users.

Algorithm 3 *top-k* Pseudo-Code for outgoing calls inspection

```
Input: $start$, $k\_param$, $tinc$        ▷ start timestamp, k parameter and time increment
Output: $edges$
 1: $R \leftarrow \{\}$                                                    ▷ data rows
 2: $E \leftarrow \{\}$                                 ▷ edges currently in the graph
 3: $R \leftarrow$ getRowsFromDB ($start$)
 4: $new\_time \leftarrow start$
 5: while ($R <> 0$) do
 6:     for all $edge \in R$ do
 7:         $before \leftarrow$ getTopKNodes()
 8:         updateTopNodesList($edge$)                ▷ update node list counters
 9:         $after \leftarrow$ getTopKNodes()
10:         $maintained \leftarrow before \bigcap after$
11:         $removed \leftarrow before \setminus maintained$
12:         for all $node \in after$ do                          ▷ add top-k edges
13:             if $node \subset edge$ then
14:                 addEdgeToGraph($edge$)
15:                 $E \leftarrow E \bigcup \{edge\}$
16:             end if
17:         end for
18:         for all $node \in removed$ do      ▷ remove non top-k nodes and edges
19:             removeNodeFromGraph($node$)
20:             for all $edge \in node$ do
21:                 $E \leftarrow E \setminus \{edge\}$
22:             end for
23:         end for
24:     end for
25:     $new\_time \leftarrow new\_time + tinc$
26:     $R \leftarrow$ getRowsFromDB ($new\_time$)
27: end while
28: $edges \leftarrow E$
```

The input parameters for this setting are the start date and time and also the maximum number of nodes to be represented (the K parameter). This *top-k* application enables the representation of the evolving network of the *top-k* nodes, from the inputted start date and time. The user may also inspect the *top-k* network of the nodes that initiate connections, the nodes that receive connections and the *top-k* representation of the A→B connections.

4 Case Study

This section describes the attributes of our large scale data and provides an overview on the application of our method to real data. Telecommunication networks generate large amount of continuous data from phone users and network equipment. In this case study, we used CDR (call detail records) log files retrieved from equipment in different geographic locations. The network data has roughly 10 million calls per day. This represents an average of 6 million of unique users per day. Each edge represents a call between A and B phone equipments (nodes). The dataset consisted in 135 days of anonymized data. For each edge/call there is a timestamp information with the date and time, with resolution to the second, representing the beginning of the call. The volume of data ranges from 10 up to 280 calls per second usually around mid-night and mid-day time, respectively.

4.1 Data Description

The first processing step was the aggregation of the number of calls from A$\rightarrow$B per day, that returned the distribution of the dataset. This operation was made with a MySql database query by selecting pairs of numbers (caller and receiver) and counting the occurrences of those pairs in the database. The results denote a compressed representation of the original network i.e. without repeated edges. There is evidence the distribution of the aggregated data might have a power law distribution [2] as can be seen in Fig. 2(left). Thus, it might represent few highly frequent calls and many infrequent calls.

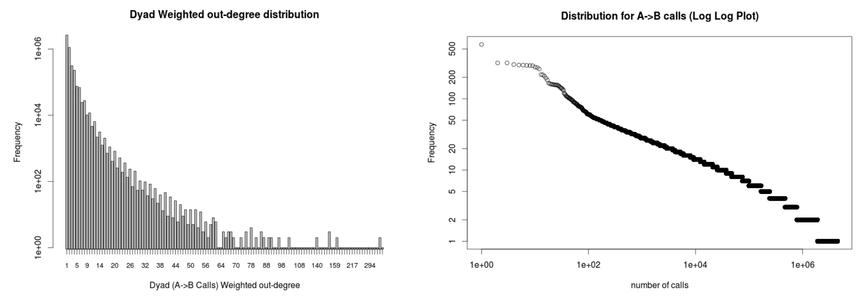


Fig. 2. A$\rightarrow$B Calls Distribution (left) and respective log-log plot (right)

We then generated the log-log representation of the distribution, per day of the aggregated data as seen in Fig. 2(right). This representation is an approximation to a monomial.

For the incoming and outgoing call distributions of the original data, a monomial is also obtained with this representation method. Thus, there is evidence that all distributions follow a power law distribution.

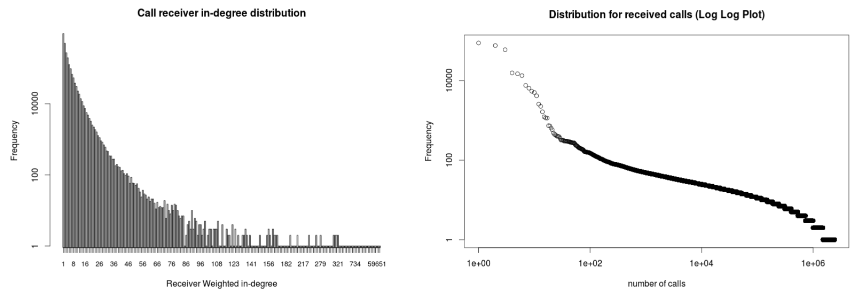


Fig. 3. Distribution of the Received Calls (left) and respective log-log plot (right)

The power law hypothesis were tested with the *poweRlaw* R package, that follows applications of power laws hypothesis testing and generation from [4], and the method described in [9]. From now on, we define the caller identifier as the main node for our *top-k* model and we will only provide results and experiments for this situation. Therefore, the weight of each node is related to the number of outgoing calls, i.e. the number of edges representing initiated calls by the intended network node. Fig. 4 illustrates this hypothesis test for power law distribution presenting the mean estimate of parameters x_{min}, α and the *p-value*, being x_{min} the lower bound of the power law distribution. Estimation parameter α is the scaling parameter ("Par 1" in Fig. 4) and $\alpha > 1$. The dashed-lines represent 95% confidence intervals. Testing the null hypothesis H_0 that the original data is generated from a power law distribution the observed *p-value* is 0.1, hence we cannot reject it because the *p-value* is higher than 0.05.

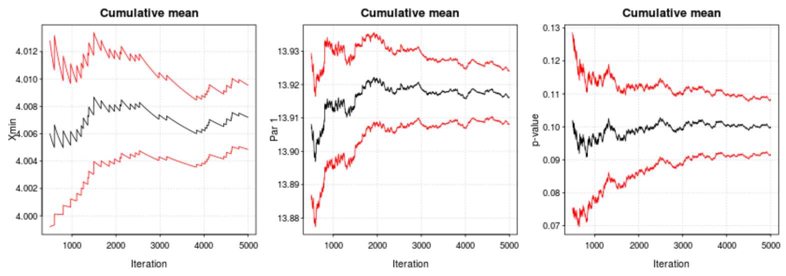
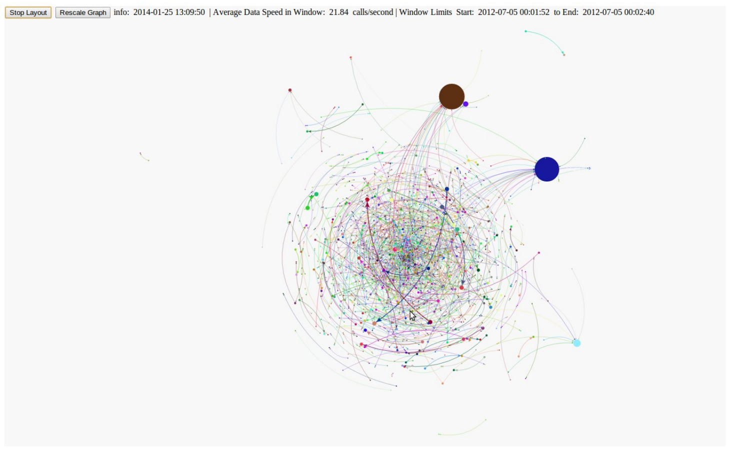


Fig. 4. Original Network - Caller power law Distribution hypothesis Test

The Figures 2 through 4 provide a visualization of an important data attribute, which is the large amount of isolated calls between some pairs of nodes and a low number of repeated calls between them. With the previous results it is acceptable to disregard the isolated calls to improve the quality of visualization and analysis, as will later be described for the *top-k* visualization method.

4.2 Sliding Windows Visualization

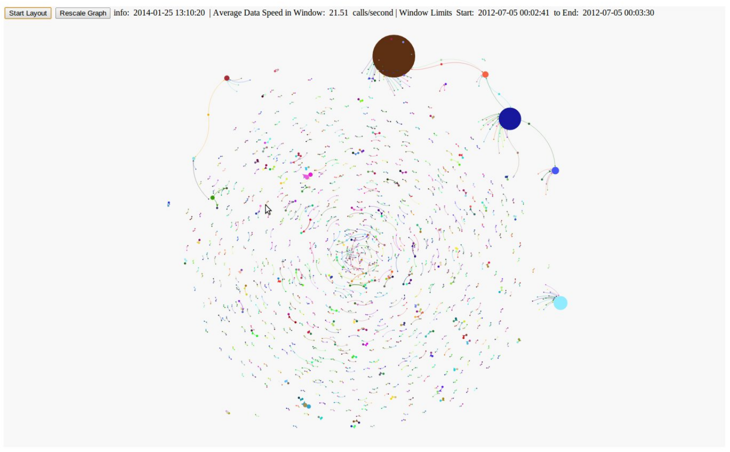


info: 2014-01-25 13:09:50 | Average Data Speed in Window: 21.84 calls/second | Window Limits Start: 2012-07-05 00:01:52 to End: 2012-07-05 00:02:40

Fig. 5. Visualization with Sliding Window approach

Algorithm 2 returns the representation in Fig. 5. It shows a window containing 1000 events/calls for a period of time beginning at 00h01m52s and ending at 00h02m40s. Several users are represented by bigger nodes, meaning more outgoing calls by those particular identifiers. The evolution of the network is represented and it shows that the anonymous brown, dark blue and light blue are the callers with more influence in this window of time.

It is also note worthy the visible connection between the dark blue caller and the brown user being established in the represented window. Fig. 5 also displays the average data speed in the window, i.e. the speed was approximately 22 calls per second. This average data speed is calculated regarding number of events/calls in the window of events and the time period between the events, represented in the visualized window. Throughout other experimental conditions, e.g., with windows around the 12h timestamp, we experienced data speed increases with more calls per second. Considering these data speed changes and after several experiments with window size parameter we concluded that it should not be smaller than 100 events and larger than 1000 events. With the minimum data speed conditions, 100 events represents a window period of around 10 seconds of events. With the maximum data speed and a window of 1000 events, it represents around 5 seconds of calls data. Less than 100 events with this data represents changes in the window, that are too fast to be visually comprehensible, and more than 1000 events represents too much events, decreasing visual interpretability of the final representation.



info: 2014-01-25 13:10:20 | Average Data Speed in Window: 21.51 calls/second | Window Limits Start: 2012-07-05 00:02:41 to End: 2012-07-05 00:03:30

Fig. 6. Visualization with Sliding Window approach (second printscreen, at a later time)

Fig. 6 represents the window between 00h02m41s and 00h03m30s. Progressing from Fig. 5, we can visually check the evolution of the network and conclude that the anonymous brown, dark blue and light blue are the callers with more influence in this window of 1000 events.

4.3 *top-k* Landmark Window

The program started running at midnight of the first day of July 2012. Fig. 7 represents the output of the Top-100 callers with more outgoing calls until 00h44m33s, extracted by Algorithm 3.

Fig. 8 represents the output of the Top-100 anonymous callers with higher number of outgoing calls. The figure displays the screen with the layout algorithm running. Only algorithm results collected until 01h09m45s are represented.

ForceAtlas2 was the selected layout algorithm. This layout algorithm has some good characteristics [15, 17]. These special ForceAtlas2 characteristics are:

- Continuous layout algorithm, that allows the manipulation of the graph while it is being rendered. It is based on the linear-linear model where the attraction and repulsion are proportional to distance between nodes. The convergence of the graph is considered to be very efficient once that features an unique adaptive convergence speed.
- Proposes summarized settings, focused on what impacts the shape of the graph (scaling, gravity...). It is suitable for large graph layout because it features a Barnes Hut optimization (performance drops less with big graphs).

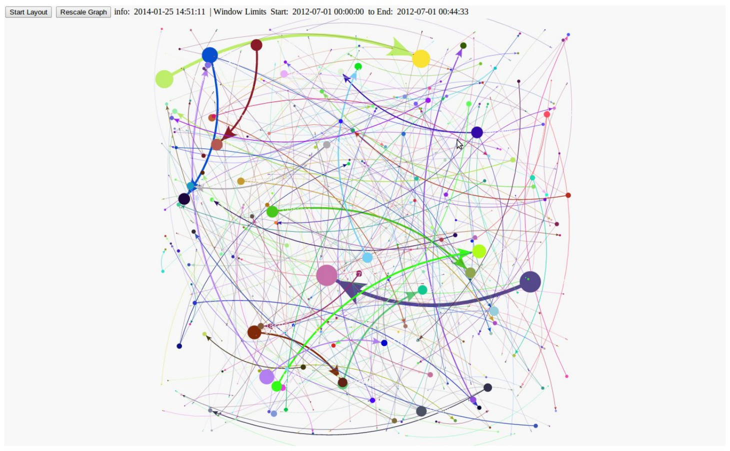

Start Layout
Rescale Graph
info: 2014-01-25 14:51:11 | Window Limits Start: 2012-07-01 00:00:00 to End: 2012-07-01 00:44:33

Fig. 7. Top-100 numbers with more outgoing calls and their connections without running the layout algorithm

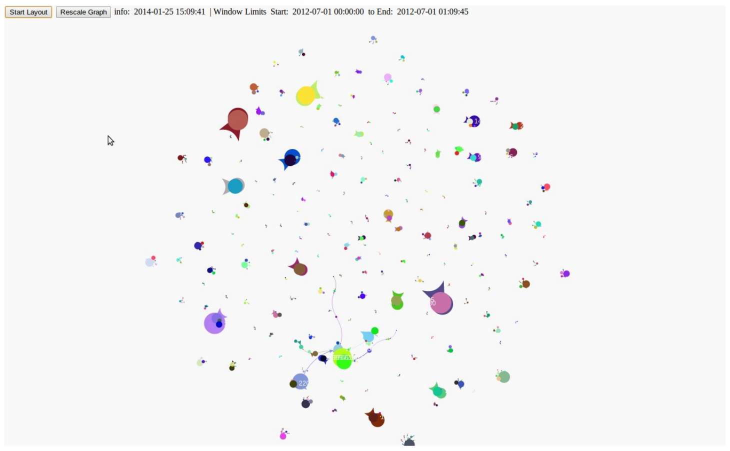

Start Layout
Rescale Graph
info: 2014-01-25 15:09:41 | Window Limits Start: 2012-07-01 00:00:00 to End: 2012-07-01 01:09:45

Fig. 8. Top-100 numbers with more calls and their connections with layout algorithm running

The ForceAtlas2 layout algorithm, although being reported to be slow for more than dozens of thousands nodes, is capable of rendering the layout of the used windows sizes throughout all our experiences. As explained before, it is expected, with our data, that the windows size do not get higher than 1000 events for the Sliding Windows representations and for a K parameter lower than 200 callers in the *top-k* representation. Higher parameters may jeopardize the interpretability of the representation. The layout algorithm also becomes slow.

4.3.1 *top-k* Sampling Attributes Considering that the majority of data includes isolated calls between two nodes our goal is to obtain a sampled version of the data providing the network of most active callers in the network. For that we selected the *Space-Saving* algorithm [19] with different settings and different k parameter i.e. 10000, 50000 and 100000. We obtained these *top-K* respective networks from database querying.

Fig. 9 represents the hypothesis test for power law distribution regarding the *top*-10000 network and the most active caller identifiers. For the *top*-10000 network of the caller phone numbers the observed *p-value* is 0.82. Therefore, we cannot reject the hypothesis H_0 at the 95% confidence level. This result provides evidence the *top-k* sampling method is non-biased regarding the original data distribution.

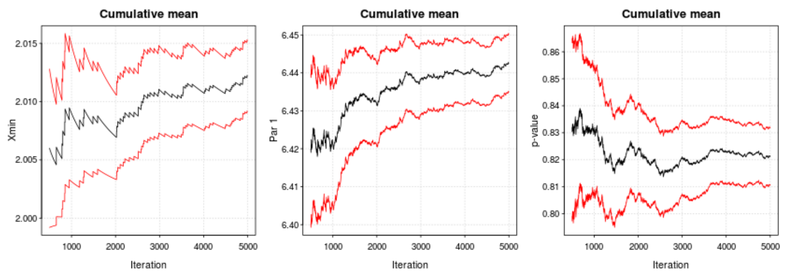


Fig. 9. Top-10000 Network - Caller power law Distribution hypothesis Test

Fig. 10 represents the hypothesis test for power law distribution regarding the *top*-50000 network and for the 50000 most active callers. For the *top*-50000 network of the caller identifiers, the observed *p-value* is 0.16. Thus, we cannot reject the hypothesis H_0 at the 95% confidence level. This result provides even more evidence the *top-k* sampling method maintains the original data distribution.

We also did the hypothesis test for power law distribution for the *top*-100000 network regarding 100000 most active callers. Testing the null hypothesis H_0 that the *top*-100000 network for the caller identifiers is generated from a power law distribution the observed *p-value* is 0 so we cannot accept it because it is lower than 0.05.

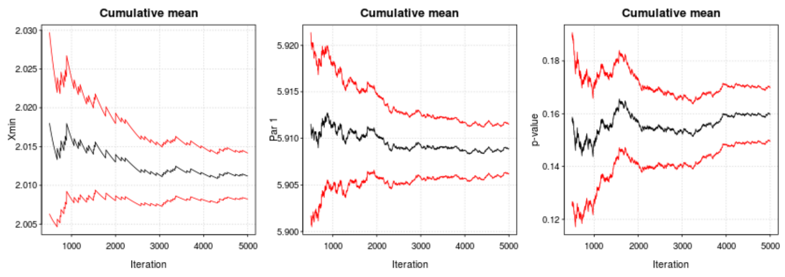


Fig. 10. Top-50000 Network - Caller power law Distribution hypothesis Test

5 Conclusions

This paper presents a new method for Large Scale Telecommunications Networks visualization. With the use of data timestamps we approach the data from a streaming point of view and visualize samples of data in a way that is both understandable to the human analyst and also enables knowledge extraction from the visual output.

Landmark Windows experiments proved to suffer from low visual comprehensibility of the network and memory issues with the software. This happens when the number of nodes or edges exceeds some dozens of thousands. With our data this number of nodes represented in the screen typically corresponds to a time period of just a few minutes. Sliding Windows were used as a way to continuously check for the full network events. Sliding Windows enables continuous inspection of the network time evolution. The *top-k* application is a suitable approach to our data that presents a power law distribution. This enables the focus on the influential individuals and discard isolated calls which are the majority of calls in our data. Concerning its computational requirements, our method for evolving networks visualization, especially with Sliding Windows or the *top-k* model may be considered a light method to visualize massive streaming networks. This simulation method enables a data stream visualization close to the node-link level using a common commodity machine. This is a different approach from previous representations mentioned in the related work section. Previous methods use hierarchical aggregation of features, for example node communities.

In future work, we could also perform community detection and display the network with additional information at the node-level. This may be applied to communities, centrality measures for the streaming data.

Future work also includes testing the models with time decay factors that enable the use of the Landmark model, increasing the weight of recent data and disregarding old data. It would also be important for the real-time data update that is displayed. The mentioned methods may be applied to fraud detection or other commercial purposes by visual detection of node related events in the network streaming.

Acknowledgments

This work was supported by Sibila and Smartgrids research projects (NORTE-07-0124-FEDER-000056/59), financed by North Portugal Regional Operational Programme (ON.2 O Novo Norte), under the National Strategic Reference Framework (NSRF), through the Development Fund (ERDF), and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT), and by European Commission through the project MAESTRA (Grant number ICT-2013-612944). The authors also acknowledge the financial support given by the project number 18450 through the "SI I&DT Individual" program by QREN and delivered to WeDo Business Assurance. Finally the authors acknowledge the reviewers for their constructive reviews on this paper.

References

1. James Abello and Frank van Ham. Matrix zoom: A visual interface to semi-external graphs. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '04, pages 183–190, Washington, DC, USA, 2004. IEEE Computer Society.
2. Albert-László Barabási. The origin of bursts and heavy tails in human dynamics. *Nature*, (435):207–211, 2005.
3. Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 693–703, London, UK, UK, 2002. Springer-Verlag.
4. Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.
5. Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. 2003.
6. Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. Frequency estimation of internet packet streams with limited space. In *Algorithms-ESA 2002*, pages 348–360. Springer, 2002.
7. Niklas Elmqvist, Thanh-Nghi Do, Howard Goodell, Nathalie Henry, and Jean-Daniel Fekete. ZAME: Interactive Large-Scale Graph Visualization. In IEEE Press, editor, *IEEE Pacific Visualization Symposium 2008*, pages 215–222, Kyoto, Japon, 2008. IEEE.
8. Joao Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, 1st edition, 2010.
9. Colin S Gillespie. *Fitting heavy tailed distributions: the poweRlaw package*, 2014. R package version 0.20.5.
10. Frank Ham, Hans-Jörg Schulz, and Joan M. Dimicco. Honeycomb: Visual analysis of large scale social networks. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part II*, INTERACT '09, pages 429–442, Berlin, Heidelberg, 2009. Springer-Verlag.
11. Robert A. Hanneman and Mark Riddle. *Introduction to Social Network Methods*. University of California, Riverside, Riverside, CA, USA, 2005.
12. Nathalie Henry and Jean daniel Fekete. Matrixexplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12:677–684, 2006.

13. Joyent Inc. Node js, 2013. [Online; accessed October-2013].
14. A. Jacomy. Sigma js, 2013. [Online; accessed October-2013].
15. M. Jacomy. Forceatlas2, the new version of our home-brew layout., 2013. [Online; accessed 21-Dec-2013].
16. Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, pages 1–5, New York, NY, USA, 2006. ACM.
17. T. Venturini M. Jacomy, S. Heymann and M. Bastian. Forceatlas2, a graph layout algorithm for handy network visualization, 2011. [Online; accessed 29-Dec-2013].
18. G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.
19. Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th International Conference on Database Theory*, ICDT'05, pages 398–412, Berlin, Heidelberg, 2005. Springer-Verlag.
20. Lei Shi, Nan Cao, Shixia Liu, Weihong Qian, Li Tan, Guodong Wang, Jimeng Sun, and Ching-Yung Lin. Himap: Adaptive visualization of large-scale online social networks. In Peter Eades, Thomas Ertl, and Han-Wei Shen, editors, *PacificVis*, pages 41–48. IEEE Computer Society, 2009.