Social Network Analysis in Streaming Call Graphs

Rui Sarmento[1,2], Márcia Oliveira[1,2], Mário Cordeiro[1], Shazia Tabassum[1], and João Gama[1,2]

[1] LIAAD/INESC TEC, University of Porto
[2] FEP, School of Economics and Management, University of Porto

Abstract. Mobile telecom operators collect and store Call Detail Records (CDRs) in real-time, which detail the communication among subscribers. Call graphs can be induced from these CDRs, where nodes represent subscribers and edges represent the phone calls made. These graphs may easily reach millions of nodes and billions of edges. Besides being large-scale and generated on real-time, the underlying social networks are inherently complex and, thus, difficult to analyze. Conventional data analysis performed by telecom operators is slow, done by request and implies heavy costs in data warehouses. In face of these challenges, real-time streaming analysis becomes an ever increasing need to mobile operators, since it enables them to quickly detect important network events and improve their marketing strategies. Sampling, together with visualization techniques, are required for online exploratory data analysis and event detection in such networks. In this chapter, we report the burgeoning body of research in network sampling, streaming analysis and streaming visualization of social networks and the solutions proposed so far.

Keywords: Call Graphs; Network Sampling; Network Visualization; Streaming Networks.

1 Introduction

Technological advances in computer processing power, disk storage and databases have enabled the collection of a wealth of data on social interactions by mobile telecom operators. The communication among individuals through mobile devices has been used as a proxy of their social relationships and is captured in the form of Call Detail Records (CDRs). CDRs provide detailed information regarding each phone call made between two individuals, such as time, call duration, source number and destination number. These data implicitly define a call graph, where nodes represent the individuals (or mobile operator subscribers or phone users) and there is an edge between two individuals if they called each other. This graph is typically weighted and directed. Weighted because the edges of the graph are assigned a weight indicating the frequency of phone calls, and directed because it includes information about who initiated the phone call.

Since call graphs model the communication among individuals, these represent *social networks* and its analysis falls within the scope of Social Network Analysis (SNA). SNA is an interdisciplinary methodology concerned with the discovery of patterns in the structure of social networks. The focus of SNA is on the link structure of the network rather than on the attributes of the nodes. Important SNA tasks include topological analysis, centrality analysis and community detection. Topological analysis aims at discovering structural properties that characterize the overall topology of the network (e.g., degree distribution, average path length, effective diameter, clustering coefficient, connected components), whereas centrality analysis is focused on finding the key nodes in the network based on the position they occupy in the network structure. The importance of nodes is typically measured in terms of their centrality in the network, which can be quantified by, for instance, the degree, closeness and betweenness centralities. On a different level, the goal of community detection is to discover implicit communities comprised in the network. These communities are groups of nodes that interact more often with each other than with other nodes in the networks and share stronger ties among them.

The analysis of the social networks underlying call graphs, using the SNA methodology, can deliver valuable business insights to mobile telecom operators that can support relevant business tasks. Pinheiro [34] addresses these tasks by combining areas such as social network analysis, analytical studies and marketing expertise to propose improvements in customer service for telecommunications networks. Kayastha et al. [23] reveal another work quite relevant, presenting a compilation of applications, architecture and issues associated with protocols designed for social networking obtained with mobile communications. Examples of applications enabled by SNA methodology include churn prediction, identification of influencers and most active callers, fraud detection and design of targeted marketing campaigns to increase subscribers' loyalty.

However, the network data collected by mobile network operators has specific features that introduces complexity in the design of new methods. CDRs are being continuously generated by the communication activity among subscribers. In addition to the large volume, this data arrives at high rates. Thus, the developed methods should be able to cope with data speed and volume and operate under the one-pass constraint of data streams. Besides scalability and computational efficiency issues, it is also desirable that the outputs of the developed algorithms are comprehensible, in order to foster their real-world deployment. Resorting to appropriate visualization techniques eases the understanding of patterns in the data, especially for non-experts. Since visualization plays an important role in the presentation of results and proves useful in supporting business decisions by the operators' managers, methods for streaming network analysis should be coupled with visualization techniques. Hitherto, few research work has been done to tackle these challenges.

The analysis of data collected by mobile telecom operators is usually performed offline and heavily relies on batch processing. Business Intelligence techniques and tools are typically used to transform these large volumes of raw data

into useful business information, mostly by means of querying and reporting. This information serves several purposes, such as the identification of trends and the extraction of patterns from both users and equipment events. The useful knowledge obtained from the data analysis process is then used to support a wide range of business decisions. Nevertheless, the prevailing *modus operandi* for analysing data by the telecommunication providers is slow, done by request and requires high costs in data warehouses. These characteristics, coupled with an increasing need to quickly react to events (or even anticipate them), avoid customer churn and improve customer service, places real-time streaming analysis in the forefront of the analytics solutions of telecom operators. The development and application of streaming methods, specifically tailored to network mining, grants telecom operators the ability to adapt to changes in the evolution of the network and detect key events in an efficient and timely manner. This ability to react and quickly adapt to real-time events brings benefits to the operators by means of increased revenue and cost reduction. In short, a streaming solution would provide the operators with the means to operate with little or even no latency, therefore being able to automatically respond to events, in a shorter time.

In this chapter, we generically cover the solutions proposed so far on sampling, visualization, community detection and centrality measures for streaming social networks.

The chapter starts with Section 2, where we describe general structural properties of call graphs. In Section 3, we address the sampling process for both static and streaming social networks. Section 4 is devoted to an introduction, followed by a critical discussion, of windowing data models to capture different kind of network events in a streaming environment. Here, we also present network centrality measures that were developed for the analysis of dynamic and large-scale social networks. Finally, in Section 7 we summarise this chapter and discuss open challenges.

2 Properties of Telecommunication Networks

All the experiments and empirical results presented in this chapter are supported by a case study conducted with real-world data collected, and made available, by a mobile telecom operator. The characteristics of the network data will be presented in this section.

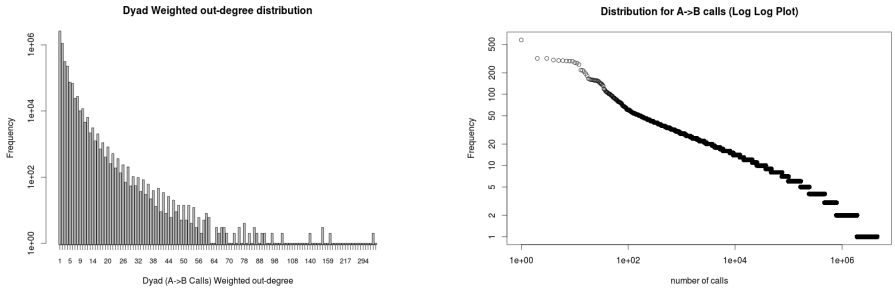
The communication among mobile users generates huge amounts of data that arrives at high rates. To conduct the case study we had access to 135 days of anonymized CDRs retrieved from equipment distributed geographically. These CDRs implicitly define a directed weighted call graph, which is stored in the form of a sparse weighted edge list. This call graph depicts the communication among the operator's subscribers, by modelling the subscribers as nodes, and the phone calls made among them as edges. These edges are weighted by the number of phone calls made. This call graph has, on average, 10 million phone calls per day made by approximately 6 million subscribers. Each edge represents

a private phone call between source number A and destination number B. For each edge/call, there is information regarding the date and time (seconds) when the call was initiated, as well as its duration. The volume of data speed ranges from 10 up to 280 calls per second, usually around mid-night and mid-day time, respectively.

To study the distribution of the available data, we aggregate the data in two different ways:

1. **Dyad weighted out-degree distribution:** Count the number of calls, per day, from source number A to destination number B (A→B);
2. **Out-degree distribution:** Count the number of calls, per day, made by each subscriber.

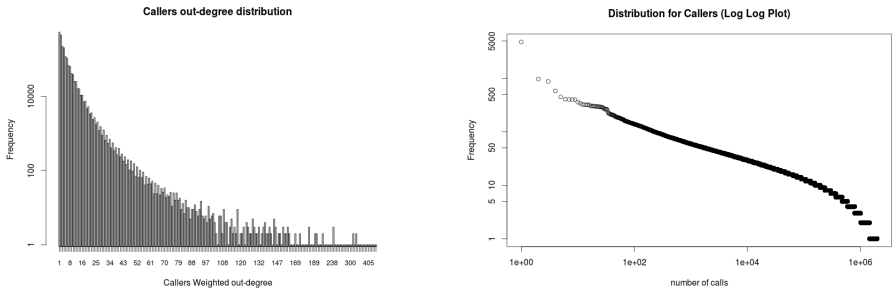
After the previous operation we observed the distribution of the aggregated data and there is some evidence that the tail of these distributions follows a power-law [3], as can be ascertained in Figure 1(a) and Figure 2(a). The analysis of these figures suggest that, for a one-day period, it is expected a high amount of single phone calls between some phone numbers A→B, and a low amount of many phone calls between a few phone numbers A→B. Therefore, it is expected a low amount of highly active subscribers and a large amount of low activity subscribers. We also plotted the distribution of the daily aggregated data in a log-log scale (see Figures Figure 1(b) and Figure 2(b)). These plots show a monomial approximation that suggests that this data is derived from a power-law distributions.



(a) (b)

Fig. 1: (a) Distribution of the A→B phone calls and (b) the corresponding log-log plot.

The power-law hypothesis was tested by following the guidelines described in [9] and using the *poweRlaw* R package. Figure 3 illustrates the hypothesis test for the 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 Figure 3) and α



(a) (b)

Fig. 2: (a) Out-degree distribution of the subscribers and (b) the corresponding log-log plot.

>1. The dashed-lines give approximate 95% confidence intervals. The observed *p-value* when testing the null hypothesis H_0 that the original data is generated from a power-law distribution is 0.1. Since we set the significance level to be 0.05, H_0 cannot be rejected because the *p-value* is higher than 0.05. Given that the tail of the distribution of the phone calls follows a power-law, we can expect that the use of sampling techniques to extract the most active subscribers is feasible and desirable.

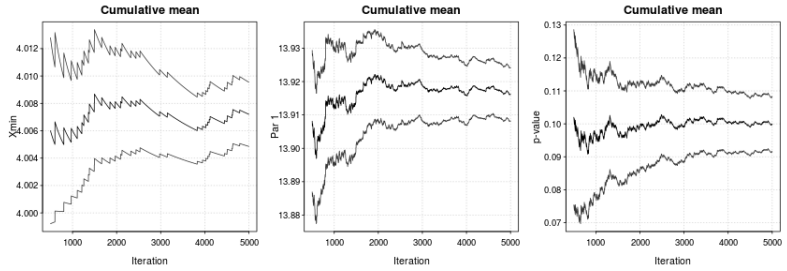


Fig. 3: Original Network - Hypothesis test for the Caller power-law distribution

3 Sampling

The analysis of large streaming networks poses processing or memory issues when using conventional hardware or software. Even if the available computational resources are able to perform the analysis of a network comprised of millions of nodes, it is difficult for the analyst to gather valuable knowledge from the outcome. Thus, in this section, we introduce several sampling methods, as well as

effective visualizations, of large streaming networks in order to address the above-mentioned problems. More specifically, we introduce the *top-K* sampling method that focuses on extracting a sample of the most active nodes in the network. This approach is suitable for networks exhibiting a power-law behaviour and is able to preserve the same distribution of the original network and the global community structure. Besides, it is highly efficient, either for visualization or analysis purposes, because it relies on the *Space-Saving algorithm*.

3.1 Sampling Large Static Networks

Large-scale network sampling has recently become a hot topic in network analysis and only a few methods have been proposed so far. The most common approaches for static network sampling are the *random sampling* and the *snowball sampling*.

In the *snowball sampling* [15], a starting node is chosen. Then, the connections in the 1st, 2nd, to n, neighborhood-order of this starting node are extracted until the network reaches the desirable size. This approach, while easy to implement, has known problems: it is biased towards the region of the network to where the starting node belongs to, and potentially misses important network properties. Yet, it is one of the most common sampling approaches.

On the other hand, *random sampling* [16], randomly selects a user-defined percentage of nodes and keeps all the corresponding edges. Alternatively, the sampling can be performed on the edges, by randomly selecting a user-defined percentage of edges and keeping the corresponding nodes. The main problem with this approach is that random edge sampling is biased towards high-degree nodes, whereas the random node sampling may be unable to generate a representative sample, since the structural properties of the sample may not reflect the ones observed on the original network. Despite these drawbacks, random sampling is easy to understand and implement.

The task, therefore, must be to generate a sample in such a way that the sampled network is representative of the original one in terms of structural properties. A primary question is related with the definition of *representative sample*. Existing work considers measures such as similarity in degree distributions and clustering coefficients [20, 28].

Leskovec et al. [28] introduce a great variety of graph sampling algorithms. They conclude that methods combining random node selection and some vicinity exploration generate the best network samples. They show that a 15% sample is usually enough to match the properties of the original graph and that no list of network properties serving as basis for sampling evaluation will ever be perfect.

3.2 Sampling Large Streaming Networks

Papagelis et al. [33] introduced sampling-based algorithms that quickly obtains a near-uniform random sample of nodes in its neighborhood, given a selected node in the social network. The authors also introduce and analyze variants of these basic sampling schemes, aiming the minimization of the total number of nodes in the visited network, by exploring correlations across samples.

Several approaches have been proposed to gather information from streaming graphs. Typical SNA problems, such as triangle counting, centrality analysis and community detection, have already been implemented in streaming settings. We will delve deeper on these topics further in the chapter.

Network sampling of streaming graphs is still a promising area for future research since, to the best of our knowledge, only few stream-based sampling methods were proposed so far. Ahmed et al. [1] present a novel approach to graph streaming sampling. According to the authors, there was no previous contribution in this topic. The authors propose a novel sampling algorithm, dubbed PIES, based on edge sampling and partial induction by selecting the edges that connect sampled nodes.

3.3 Top-K Sampling with Top-K itemsets

Researchers have been trying to achieve efficient ways of analyzing data streams and performing graph summarization. The exact solution implies the knowledge of the frequency of all nodes and edges, which might be impossible to obtain in large-scale networks.

The problem of finding the most frequent items in a data stream S of size N is basically how to discover the elements e_i whose relative frequency f_i is higher than a user-defined support ϕN, with $0 \leq \phi \leq 1$ [14]. Given the space requirements that exact algorithms addressing this problem would need [8], several algorithms were already proposed to find the top-k frequent elements, being roughly classified into *counter-based* and *sketch-based* [30]. *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 that process the stream in compressed size, such as *Sticky Sampling* and *Lossy Counting*, were proposed by Manku et al. in [29]. Yet, these have the disadvantage of keeping a large amount of irrelevant counters. *Frequent* [11], by Demaine et al., keeps only k counters for monitoring k elements, incrementing each element counter when it is observed, and decrementing all counters when an unmonitored element is observed. Zeroed-counted elements are replaced by new unmonitored element. This strategy is similar to the one applied by the *Space-Saving algorithm*, proposed by Metwally et al.[30], which give 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 [8], by Charikar et al., 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 [10] proposed

by Cormode et al., 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.

Algorithm 1 represents the proposed *top-K* Method application using the *Space-Saving algorithm*.

This type of application is based on a landmark window model [14], which implies a growing number of inspected events in the accumulating time window. This landmark application is useful also in other contexts, e.g., when the network is relatively small and the user wants to check all events in it.

Experiments using the landmark window model showed that this model suffers from the problems we would like to avoid, such as exceeding memory limits. This happens when the number of nodes and edges exceeds dozens of thousands of nodes. The *top-K* algorithm, based on a landmark window model, is an efficient approach for large-scale data. It focus on the most active nodes and discards the least active ones, which are the most frequent according to the power-law distribution. The alternative option to the landmark window model, i.e., the sliding window model [14], would not be appropriate for the *top-K* approach, since it may remove less recent nodes. Those nodes may yet be included in the *top-K* list we want to maintain.

In our scenario, the *top-K* representation of data streams implies knowing the K elements of the simulated data stream from the database. Network nodes that have higher frequency of outgoing connections, incoming connections, or even specific connections between any node A and B, may be included in the graph, as well as their connections.

For this application, the user can insert as input a start date and hour and also the maximum number of *top-K* nodes to be represented (the K parameter), along with their connections.

With the inserted start date and hour, the *top-K* application is expected to return the evolving network of the *top-K* nodes. Functions *getTopKNodes* and *updateTopNodesList* in **Algorithm 1** implement the *Space-Saving algorithm*. As the network evolves over time, new *top-K* nodes are added to the graph. Nodes that exit the *top-K* list of numbers are removed from the *top-K* list and, thus, removed from the graph along with their connections.

Fig. 4 represents the network induced by the top-100 subscribers with the highest number of phone calls, since the midnight of the first day of July 2012, until 00h44m33s. The algorithm shows the 100 most active phone numbers in that period. Fig. 5 depicts a similar network but after running the layout algorithm. This time, the output considers results until 01h09m45s.

4 Window-Based Visualization[3]

Resorting to time window models is an useful strategy to limit the amount of data available for analysis, since it is based on setting a fixed point in time (the

[3]This section is based on the work published in [37].

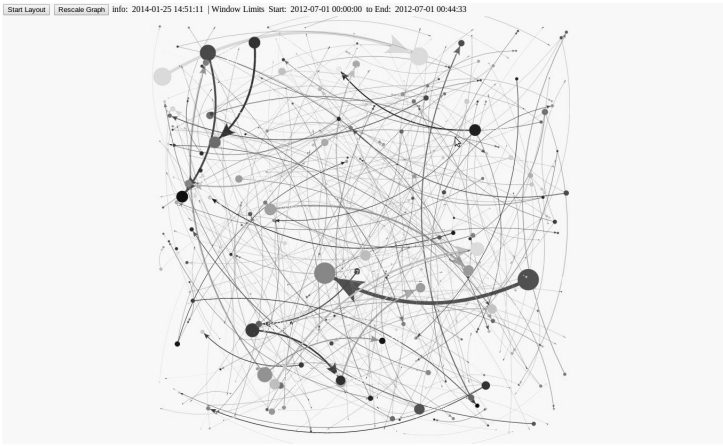


Fig. 4: Network induced by the top-100 subscribers with the highest number of phone calls and corresponding direct connections. This network was generated without running the layout algorithm.

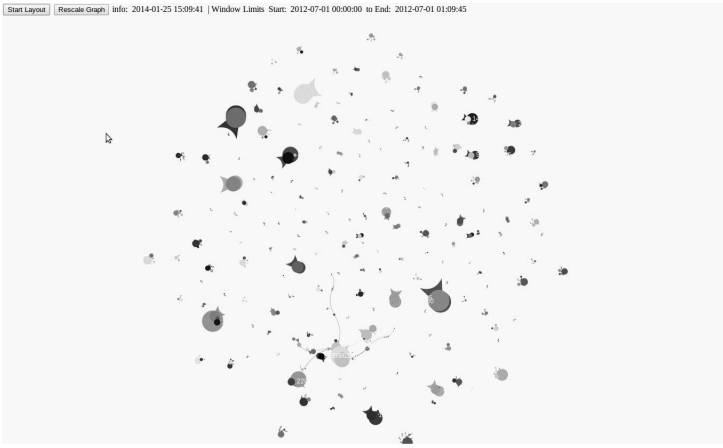


Fig. 5: Network induced by the top-100 subscribers with the highest number of phone calls and corresponding direct connections. This network was generated after running the layout algorithm.

Algorithm 1 Top-K algorithm for call graphs

Input: *start*, *k_param*, *tinc* ▷ start timestamp, k parameter and time increment
Output: *edges*

```
1: R ← {}                                                    ▷ data rows
2: E ← {}                                                    ▷ edges currently in the graph
3: R ← getRowsFromDB (start)
4: new_time ← start
5: while (R <> 0) do
6:     for all edge ∈ R do
7:         before ← getTopKNodes(k_param)
8:         updateTopNodesList(edge)                          ▷ update node list counters
9:         after ← getTopKNodes(k_param)
10:        maintained ← before ⋂ after
11:        removed ← before \ maintained
12:        for all node ∈ after do                           ▷ add top-k edges
13:            if node ⊂ edge then
14:                addEdgeToGraph(edge)
15:                E ← E ⋃ {edge}
16:            end if
17:        end for
18:        for all node ∈ removed do                         ▷ remove non top-k nodes and edges
19:            removeNodeFromGraph(node)
20:            for all edge ∈ node do
21:                E ← E \ {edge}
22:            end for
23:        end for
24:    end for
25:    new_time ← new_time + tinc
26:    R ← getRowsFromDB (new_time)
27: end while
28: edges ← E
```

so-called *landmark*) from which the data starts being observed. A disadvantage of this method is that the amount of data inside the window quickly grows to a prohibitive size. Other way of limiting data is by using a fixed sliding window model. These windows are bounded by the number of data points or the number of time units, being both constant.

4.1 Landmark Windows

Algorithm 2, regarding streaming landmark window models, provides the representation of all the events (e.g., edge and node addition or removal) that occur in the network, starting at a specific time stamp, for example, 01h48m09s of 1st of January 2012.

This type of window model is not very useful in a streaming scenario, because it implies a growing number of events outputted on the screen and the comprehensibility lowers as this number reaches, or exceeds, a few thousands of

Algorithm 2 Algorithm based on a landmark window model [37]

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: $R \leftarrow$ 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: $edges \leftarrow E$

events. This landmark application is however useful in other contexts, for example, if the network is relatively small and the analyst is interested in checking all events in the evolution of the network. Nevertheless, if the analyst wants to follow the evolution of a large streaming network, the application described in the next subsection is more appropriate.

4.2 Sliding Windows

For this large data stream, we generate a dynamic sample representation of the data by using a sliding window model. This sliding window is defined as a data structure with fixed number of registered events. Each event is a phone call between pairs of subscribers. Since these events are annotated with time stamps, the time period between the first call and the last call in the window is easily computed. The input parameters of this algorithm are (i) starting date and time, and the (ii) maximum number of events/calls the sliding window can have. The SNA model used in this application is a full weighted directed network, since all the nodes and edges in the network are outputted for a particular instance of the sliding window [19].

An example of the obtained results is provided in Fig. 6. Nodes are sized according to their weighted degree. Thus, larger nodes correspond to more active subscribers, i.e., subscribers associated with a higher number of phone calls (either received or made). This is the representation of a window with 1000 events/calls, for a time period starting at 00h01m52s and ending at 00h02m40s. The evolution of the network is visually and immediately conclusive. There are three nodes with the largest number of connections/phone calls in the network.

From the figure, we can also see the connection established between two of these three largest nodes. Fig. 6 also displays the average data speed in the window (approximately 22 calls per second). This average data speed is computed by counting the number events (i.e., phone calls) inside the window, which

Algorithm 3 Algorithm based on a sliding window model [37]

Input: *start*, *wsize*, *tinc* ▷ start timestamp, window size and time increment
Output: *edges*

$R \leftarrow \{\}$ ▷ data rows
$E \leftarrow \{\}$ ▷ edges currently in the graph
$V \leftarrow \{\}$ ▷ buffer to manage removal of old edges
$R \leftarrow$ getRowsFromDB ($start$)
$new\_time \leftarrow start$
$p \leftarrow \{\}$
while ($R <> 0$) **do**
  for all $edge \in R$ **do**
    addEdgeToGraph($edge$)
    $E \leftarrow E \bigcup \{edge\}$
    $k \leftarrow 1 + (p \bmod wsize)$
    $old\_edge \leftarrow V[k]$
    removeEdgeFromGraph(old_edge)
    $E \leftarrow E \setminus \{old\_edge\}$
    $V[k] \leftarrow edge$
    $p \leftarrow p + 1$
  end for
  $new\_time \leftarrow new\_time + tinc$
  $R \leftarrow$ getRowsFromDB (new_time)
end while
$edges \leftarrow E$

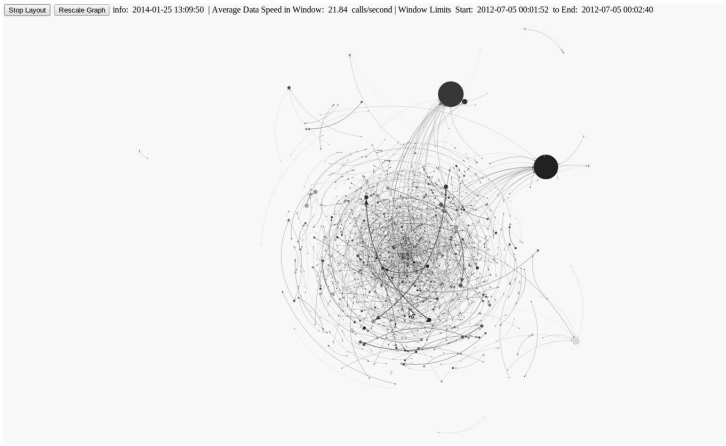


Fig. 6: Visualization of the call graph using a sliding window approach [37]

comprises all the events observed during the time period associated with the window length. Under different experimental conditions, namely when analysing the

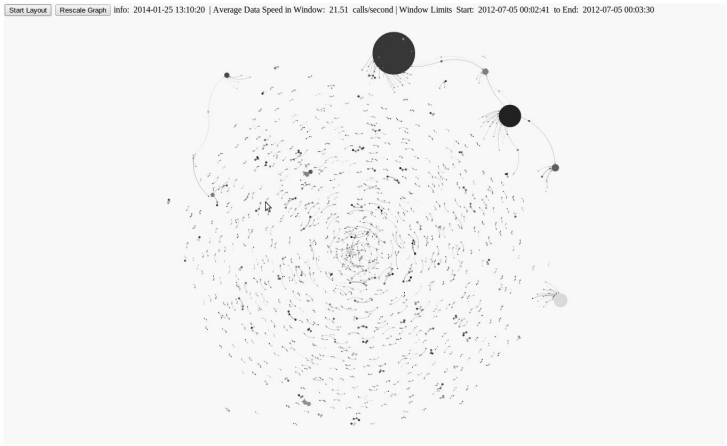


Fig. 7: Visualization of the call graph using a sliding window approach (2nd version) [37]

window obtained for mid-day, the data speed increases, with more phone calls per second. After several experiments with different window sizes, and considering that the data speed changes, we concluded that this speed should not be smaller than approximately 100 events and also not larger than approximately 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 data. Using this data, less than 100 events represents changes in the window that are too fast to be visually comprehensible, and more than 1000 events represents too many events, reducing the visual comprehensibility of the output.

Fig. 7 represents the next window instance, starting at 00h02m41s and ending at 00h03m30s. Considering the previous Fig. 6, we can visually observe the evolution of the network and observe that there is a new smaller node establishing connection to the most active nodes identified before, in this window of 1000 events.

5 Centrality Analysis

One of the most relevant tasks in SNA is the computation and interpretation of centrality measures. Centrality is often used for finding important nodes by analyzing their position in the network. The concept of *importance* is multidimensional. Each relevant dimension, such as reachability, embeddedness, influence, support, ability to span structural holes and control of information flow, is captured by different centrality measures. Examples of classical measures are *degree*, *betweenness*, *closeness* and *eigenvector centrality*. The first three were proposed

by Freeman [13] and the last one by Bonacich [5]. There are two fundamentally different classes of centrality measures: node-level measures (e.g., degree) and network-level measures (e.g., density). The former evaluates the centrality of each node/vertex (or edge/link) in the network, whereas the latter evaluates the centrality of the whole network.

While some of these measures can be straightforwardly computed for streaming graphs, since they only require the update of counters (e.g., degree and density), other measures are computationally expensive because they rely on the computation of the shortest paths across all nodes in the network (e.g., closeness and betweenness). The computation of the all pairs shortest paths is feasible when dealing with small networks of a few tens of thousands of nodes and links, but it quickly becomes prohibitively expensive as the network grows larger. A possible solution to circumvent this problem is to calculate these measures on snapshots of the dynamic network. However, since streaming networks typically change at a fast pace, this solution is frequently not fast enough to provide accurate and up-to-date information, which makes it unfeasible for practical applications. Given this limitation, incremental algorithms for computing shortest-path-based centrality measures were developed. Incremental algorithms keep analytic information without performing all computations from scratch, thus being suitable for dynamically changing networks.

Since betweenness and closeness are among the most widely used centrality measures, are expensive to compute in streaming environments and are essential to identify important subscribers in a call graph (e.g., mobile users with high reachability or users who control the information flow between communities), here we report the advances on algorithms for computing these two measures. Note that these algorithms assume a landmark window model when updating the centralities.

5.1 Betweenness

Betweenness is a classical centrality measure that quantifies the importance of a network element (node or edge) based on the frequency of its occurrence in shortest paths between all possible pairs of nodes in the network. The intuitive idea behind this measure is to identify graph elements that act as bridges, i.e., which connect dense regions of the network and without which the information would not pass from one of these regions to the other. An edge with high betweenness is likely to act as a bridge between dense graph regions and, thus, occurs in many shortest paths. Nodes with high betweenness are usually located at the ends of these edges. These nodes occupy a strategic position in the network, which allows them to control the information transfer between different network regions, either by blocking the information between them or by accessing it before other nodes belonging to their region.

Node (or edge) betweenness is computed as the fraction of shortest paths that go through a given node (or edge) among all shortest paths in the network. This is a global centrality measure since it requires complete information about the network in order to compute all pairs shortest paths. Since it is based on

the computation of shortest paths for the whole network, betweenness is computationally demanding. The best known algorithm for computing this measure in static unweighted graphs was proposed by Brandes [7]. This algorithm runs in $O(nm)$ time, being n the number of nodes/vertices and m the number of links/edges, and has a space complexity of $O(n^2 + nm)$, which is prohibitive for networks with millions of nodes and billions of links. Given this, it is necessary to develop new algorithms that avoid the full recomputation of betweenness every time a new edge or node is added to, or removed from, the network. Incremental algorithms offer a solution to this problem, since they can handle large-scale data and can adapt to incremental changes in evolving networks. This is achieved by performing early pruning and by updating only the regions of the network affected by the changes.

Recently, several solutions were proposed to compute node and/or edge betweenness centrality in streaming networks. Lee et al. [27] introduced QUBE, an incremental algorithm that relies on the decomposition of the graph into biconnected components. The performance of the algorithm is strongly associated with the size of the components, which is usually very large in real-world applications. Consequently, it suffers from scalability and efficiency problems, which are of utmost importance in streaming settings. In the same year, Green et al. [17] proposed an exact algorithm, which extends Brandes's approach [7], to compute node betweenness in unweighted streaming graphs. The idea is to preserve information from prior computations of betweenness values and the needed data structures and update only the values and data structures directly affected by the changes in the network. However, the algorithm has some drawbacks: it only supports one type of change (insertion of new edges), and has the same space complexity of Brandes's static algorithm, which is $O(n^2 + nm)$. Hence, the algorithm is not fully suited to handle large-scale and dynamic networks. Kas et al. [22] propose a slightly better solution. They present an incremental algorithm for dynamic maintenance of node betweenness centrality values in rapidly growing networks, using as a building block the dynamic all pairs shortest path algorithm introduced by Ramalingam and Reps [36]. Similarly to [17], the technique of Kas et al. is also based on keeping in memory information from previous computations but using data structures that are faster to update (e.g., shortest distances and number of shortest paths). Although the computational complexity can be lower than the one obtained by [17], the space complexity is similar, turning it prohibitive for very large graphs. More recently, Kourtellis et al. [26] proposed an incremental and scalable algorithm for online computing of both node and edge betweenness centralities in very large dynamic networks. Besides being adapted to fully dynamic networks, where nodes and links are added and removed over time, these authors propose an algorithmic framework that can be efficiently used for real-world deployment (e.g., for identifying strategic subscribers in call graphs) since the algorithm allows for out-of-core implementation and is tailored for modern parallel stream processing engines.

A different approach from the above mentioned was introduced by Kim and Anderson [25]. They presented the time-ordered graph model, which converts a

dynamic network into a static network with directed flows, and propose temporal centrality measures to extract information from the graph. These measures are simple extensions of the classical measures to this specific type of dynamic model. However, this method was devised for dynamic networks which evolve in non-streaming scenarios, thus not being fully suitable to the online analysis of call graphs.

5.2 Closeness

Closeness is another classical SNA measure that quantifies the importance of a node based on its ability to reach other nodes in a network through shortest paths. The higher the closeness, the less the cost for a node to reach the rest of the network. For instance, in a call graph, the subscriber with highest closeness can quickly spread information to other nodes of the network, as long as the nodes belong to the same connected component.

Closeness is computed as the inverse of the sum of the shortest path distances between a node i and the remaining $n-1$ nodes in a network of size n. Similarly to betweenness, a few incremental algorithms were proposed to compute the closeness centrality in large dynamic graphs. Kas et al. [21] proposed an algorithm for the fast computation of closeness in large-scale networks. Their technique supports efficient computation of all pairs shortest paths and is suited to dynamic networks, since it handles addition, removal and modification of nodes and links. This algorithm is similar to the one proposed by the same authors in [22]. A more recent work by Khopkar et al. [24] presents a partially dynamic and incremental closeness algorithm that runs in $O(n^2)$.

6 Community Detection

In a social network, a community represents individuals that form a group distinguishable by its properties or characteristics. In other words, when we say we encountered a community it might be, for example, a group of friends, family, work colleagues or other group of individuals sharing the same role, characteristics or label in the context of a network. Detection of communities on a network has many applications, for example, clients that have the same interests and are geographically close to each other might benefit with the implementation of mirror servers. These servers provide faster services on the World Wide Web. The identification of retail clients with similar interests in products enables the retailer to give better recommendation services and therefore increases the probability of increasing profits and the service quality. In telecommunications and computer networks, the community structure of nodes may help the improvement of the compactness of routing tables, maintaining efficient choice of communication paths. Regarding community structure, several areas consider important if the node is located in the center of a community or if it lies on the boundaries of the community. In the first case, the node might have an important control and stability function within the community. In the second case, the node might

have functions enabling information exchange between communities. The identification of central and peripheral nodes is of high importance, for example, in social and metabolic networks [12]. One of the most efficient methods used for community detection, either in static environments or in dynamic ones, is the Louvain method [4].

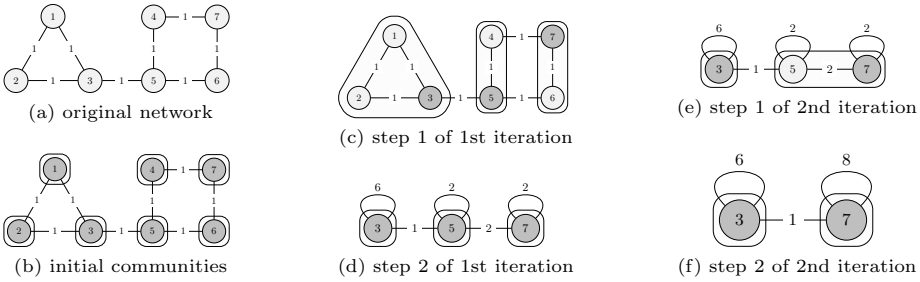


Fig. 8: The original Louvain algorithm steps.

Figure 8 briefly explains how the *Louvain* algorithm works, by illustrating the sequential steps that the algorithm performs for identifying communities in the network. The Louvain method is non-deterministic and performs a greedy optimization to maximize the modularity [31] of all the network partitions. A two-step optimization is performed at each iteration. In step 1, the algorithm seeks for small communities by locally optimizing the modularity. Only local changes of communities are allowed. In step 2, nodes belonging to the same community are aggregated in a single node representing that community in order to build a new aggregated network of communities. Steps are repeated iteratively until no increase of modularity is possible and a hierarchy of communities is generated. Figure 8a represents the initial network; Figure 8b represents initial individual node communities; Figure 8c represents local modularity optimization after the first step; Figure 8d represents the community aggregation results and the new initial communities; Figure 8e and Figure 8f, are the two Louvain steps, where the local modularity optimization and community aggregation for the second iteration are presented; The algorithm stops at the 2nd iteration, once increasing modularity is no longer possible. Despite the fast convergence property of the algorithm, which allows the identification of communities in very large networks, its non-deterministic behaviour and internal network structure make it only suitable for static networks.

Methods for detecting communities in dynamic networks were already proposed in the literature. Shang et al. [39] propose the addition of new edges in a two-step approach by using the Louvain Method in the first step and then applying incremental updating strategies on the detected communities in the second step. Thus, the obtained results with this algorithm are dependent on the community structure at its starting point. Another extension of the Louvain method is the one introduced by Nguyen et al.[32]. The proposed QCA algorithm

for the efficient detection of communities, starts by detecting the initial communities and then adapts itself to the communities changes by doing addition and removals of nodes and edges within and across communities. Bansal et al. [2], propose a fast community detection algorithm that uses community information from previous time steps. In their experiments, these authors achieve execution time improvements of as much as 30% over the static methods and maintaining the quality of the community partitions. More recently, the use of spectral clustering became one of the most used methods for community detection. There are several examples of its use. In [43], Yun et al., introduce community detection with partial information. With their method, only a sample of the nodes is observed. Again, the developed algorithm developed is spectral. They extract the clusters only when it is possible. Thus, the authors address the memory limit problem for community detection from a streaming point of view, and achieve asymptotic reconstruction of the clusters with a memory requirement which is sub-linear with the size of the network. The memory requirement of the algorithm is non-increasing. Bouchachia et al. [6] present an algorithm that performs enhanced spectral incremental clustering. This algorithm does not need to calculate the clustering for the whole network each time new nodes or edges appear in the stream. Several Label propagation algorithms such as, for example, LPA [35, 41], COPRA [18] and SLPA [42] were also proposed. Results shown that they perform well in static networks, however when those algorithms are applied in different snapshots of an evolving network they produce different communities at each run. When tracking communities in dynamic networks the instability of the partitions is undesirable. LabelRankT, a new label propagation technique proposed by [40] was designed to overcome this problem.

6.1 Top-K Communities [4]

Top-K communities are defined as groups of densely connected nodes in *top-K* networks. As the name implies, in this work these communities are detected considering only the *top-K* nodes and their 1st and 2nd neighborhood-order connections. This method samples the original network in a way that preserves its structural properties and the community structure of the original network. We apply *top-K* sampling to obtain the nodes that belong to the *top-K* group. To retrieve their network we query the database so as to collect all connections/edges representing the network with the neighbors of the *top-K* nodes. After generating the sampled networks, the *Louvain method* [4] is applied to find the communities. Figure 9 represents the matching between the community membership obtained for the *top*-10000 network and the community membership of the original network retrieved by the *Louvain method*. This task was performed for an entire day of streaming data. The matching of communities for the two scenarios (sampled network and original network) is performed by retrieving the percentage of matching community members between any *top-k* network community and the original network communities.

[4] This section is based on the work published in [38].

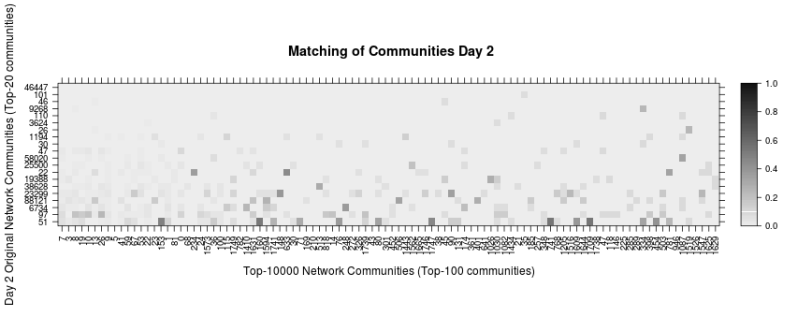


Fig. 9: Matching of community members in the community structure of the sampled network and the community structure of the original network

Further analysis of Figure 9 shows the matching of the 100 largest communities for the sampled network and the 20 largest communities in the original network. The proportion of community member matching varies with a color gradient between 0 (light grey) and 1 (black). There is considerable matching of the *top*-10000 sampling communities and the 20 largest communities of the original network. These highly active callers and the communities they belong to are therefore represented in the *top-K* sampling, as expected.

Experiments were also conducted using other days of the dataset. The results are very similar and consistent throughout full day data comparisons and for the complete dataset of more than 100 days. In all comparisons it is visible that larger original dataset communities are matched by communities retrieved with the proposed *top-K* sampling method.

7 Conclusions and Open Issues

Sampling of large social networks is still in its infancy and there are important open research issues and unsolved problems not yet satisfactorily addressed by the scientific community. Firstly, it is necessary to achieve consensus on the definition of *representative sample*. Based on previous research, a representative sample is a subgraph of the original network that matches its structural properties. However, it is not yet clear which structural properties (e.g., degree distribution, global clustering coefficient, motifs, community structure) should be preserved in the sample. Another matter of concern is the sample size. Despite the promising results of empirical studies [28], a rigorous formal study of the most appropriate sample size according to the size and characteristics of the original network still has to be done.

Regarding visualization, the proposed solutions for streaming networks are still quite simple and leave much space for improvement. A straightforward extension of the presented techniques would be to include additional information by means of visual cues, such as color, size and shape. An idea would be to

integrate the visualization techniques with the incremental algorithms used to compute computationally demanding centrality measures (e.g., closeness and betweenness) and then include this node-level information on the visual output. On the other hand, it is necessary to explore new types of representations, that go beyond the graph model, for visually displaying interesting patterns in streaming networks.

Community detection on large dynamic social networks faces many of the challenges of static community detection, namely in what regards the lack of a consensual definition of *network community* and the evaluation of the network partitions produced by community detection algorithms. A possible way to circumvent these problems would be to support the definition of community on the specific domain and develop evaluation measures specifically tailored for the application. For instance, telecommunication providers might be interested in finding communities defined not only by the connections induced by the mobile communications, but also by business variables (e.g., revenue generated by each user), geographical position and demographic attributes. Depending on the purpose of the community detection task, telecommunication companies could, for instance, perform the evaluation based on the similarity of the users' response to the marketing campaigns targeted to the community they were assigned to. Another important issue, especially for telecommunication providers, is to create models and procedures to characterize and define profiles of communities.

Overall, the major challenge is to devise a system that integrates all the relevant steps involved in the process of extracting useful knowledge from large streaming social networks. This system should be built upon rigorous methods and appropriate algorithms, while being user-friendly, in order to encourage its use by business managers and decision makers. The use of social network analysis applied to streaming telecommunications networks will benefit society. More specifically, the network users, in terms of service quality, largely due to the rapidity of action that allows operators. Service problems, which ultimately may adversely affect the service to the point where the customer will want to leave the operator, will decrease significantly. Otherwise, the problems that will probably arise upon it will necessarily be discussed in more detail in the future, relate to the use of such information for dissemination of network spam, dubious character information or that might involve the loss of the network user's privacy.

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. Márcia Oliveira gratefully acknowledges funding from FCT, through Ph.D. grant SFRH/BD/81339/2011.

References

1. Ahmed, N.K., Neville, J., Kompella, R.: Space-efficient sampling from social activity streams. In: Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications (BigMine 2012), pp. 53–60. ACM (2012)
2. Bansal, S., Bhowmick, S., Paymal, P.: Fast community detection for dynamic complex networks. In: L. da F. Costa, A. Evsukoff, G. Mangioni, R. Menezes (eds.) Complex Networks, *Communications in Computer and Information Science*, vol. 116, pp. 196–207. Springer Berlin Heidelberg (2011)
3. Barabasi, A.L.: The origin of bursts and heavy tails in human dynamics. Nature **435**(7039), 207–211 (2005)
4. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **2008**(10), P10,008 (2008)
5. Bonacich, P.: Power and centrality: A family of measures. American Journal of Sociology **92**(5), 1170–1182 (1987)
6. Bouchachia, A., Prossegger, M.: Incremental spectral clustering. In: M. Sayed-Mouchaweh, E. Lughofer (eds.) Learning in Non-Stationary Environments, pp. 77–99. Springer New York (2012)
7. Brandes, U.: A faster algorithm for betweenness centrality. Journal of Mathematical Sociology **25**(2), 163–177 (2001)
8. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP 2002), pp. 693–703. Springer-Verlag (2002)
9. Clauset, A., Shalizi, C.R., Newman, M.E.: Power-law distributions in empirical data. SIAM review **51**(4), 661–703 (2009)
10. Cormode, G., Muthukrishnan, S.: What's hot and what's not: tracking most frequent items dynamically. ACM Transactions on Database Systems **30**(1), 249–278 (2005)
11. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: R. Möhring, R. Raman (eds.) Algorithms-ESA 2002, *Lecture Notes in Computer Science*, vol. 2461, pp. 348–360. Springer Berlin Heidelberg (2002)
12. Fortunato, S.: Community detection in graphs. Physics Reports **486**(3-5), 75–174 (2010)
13. Freeman, L.C.: Centrality in social networks conceptual clarification. Social Networks **1**(3), 215–239 (1979)
14. Gama, J.: Knowledge Discovery from Data Streams, 1st edn. Chapman & Hall/CRC (2010)
15. Goodman, L.A.: Snowball sampling. The Annals of Mathematical Statistics **32**(1), 148–170 (1961)
16. Granovetter, M.: Network sampling: Some first steps. American Journal of Sociology **81**(6), 1267–1303 (1976)

17. Green, O., McColl, R., Bader, D.A.: A fast algorithm for streaming betweenness centrality. In: Proceedings of the 2012 International Conference on Privacy, Security, Risk and Trust (PASSAT 2012) and 2012 International Conference on Social Computing (SocialCom 2012), pp. 11–20. IEEE Computer Society (2012)
18. Gregory, S.: Finding overlapping communities in networks by label propagation. New Journal of Physics **12**(10), 103,018 (2010)
19. Hanneman, R.A., Riddle, M.: Introduction to Social Network Methods. University of California, Riverside, Riverside, CA, USA (2005). URL http://www.faculty.ucr.edu/ hanneman/nettext/index.html
20. Hubler, C., Kriegel, H.P., Borgwardt, K., Ghahramani, Z.: Metropolis algorithms for representative subgraph sampling. In: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), pp. 283–292. IEEE Computer Society (2008)
21. Kas, M., Carley, K.M., Carley, L.R.: Incremental closeness centrality for dynamically changing social networks. In: Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013), pp. 1250–1258. IEEE Computer Society (2013)
22. Kas, M., Wachs, M., Carley, K.M., Carley, L.R.: Incremental algorithm for updating betweenness centrality in dynamically growing networks. In: Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013), pp. 33–40. IEEE Computer Society (2013)
23. Kayastha, N., Niyato, D., Wang, P., Hossain, E.: Applications, architectures, and protocol design issues for mobile social networks: A survey. Proceedings of the IEEE **99**(12), 2130–2158 (2011)
24. Khopkar, S.S., Nagi, R., Nikolaev, A.G., Bhembre, V.: Efficient algorithms for incremental all pairs shortest paths, closeness and betweenness in social network analysis. Social Network Analysis and Mining **4**(1), 1–20 (2014)
25. Kim, H., Anderson, R.: Temporal node centrality in complex networks. Physical Review E **85**(2), 026,107 (2012)
26. Kourtellis, N., Morales, G.D.F., Bonchi, F.: Scalable online betweenness centrality in evolving graphs. arXiv preprint arXiv:1401.6981 (2014)
27. Lee, M.J., Lee, J., Park, J.Y., Choi, R.H., Chung, C.W.: QUBE: a Quick algorithm for Updating BEtweenness centrality. In: Proceedings of the 21st International Conference on World Wide Web, pp. 351–360. ACM (2012)
28. Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006), pp. 631–636. ACM (2006)
29. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB 2002), pp. 346–357. VLDB Endowment (2002)
30. Metwally, A., Agrawal, D., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: Proceedings of the 10th International Conference on Database Theory (ICDT 2005), pp. 398–412. Springer-Verlag (2005)
31. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69**(2), 026,113 (2004)
32. Nguyen, N.P., Dinh, T.N., Xuan, Y., Thai, M.T.: Adaptive algorithms for detecting community structure in dynamic social networks. In: Proceedings of the 2011 IEEE International Conference on Computer Communications (INFOCOM 2011), pp. 2282–2290. IEEE Computer Society (2011)
33. Papagelis, M., Das, G., Koudas, N.: Sampling online social networks. IEEE Transactions on Knowledge and Data Engineering **25**(3), 662–676 (2013)

34. Pinheiro, C.A.R.: Social network analysis in telecommunications, vol. 37. John Wiley & Sons (2011)
35. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Physical Review E **76**(3), 036,106 (2007)
36. Ramalingam, G., Reps, T.: On the computational complexity of dynamic graph problems. Theoretical Computer Science **158**(1), 233–277 (1996)
37. Sarmento, R., Cordeiro, M., Gama, J.: Visualization for streaming networks. In: Proceedings of the 3rd Workshop on New Frontiers in Mining Complex Patterns (NFMCP 2014), pp. 62–74 (2014)
38. Sarmento, R., Cordeiro, M., Gama, J.: Streaming networks sampling using top-k networks. In: Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS 2015), p. to appear. INSTICC (2015)
39. Shang, J., Liu, L., Xie, F., Chen, Z., Miao, J., Fang, X., Wu, C.: A real-time detecting algorithm for tracking community structure of dynamic networks. In: Proceedings of the 6th SNA-KDD Workshop (SNA-KDD 2012), pp. 1–9. ACM (2012)
40. Xie, J., Chen, M., Szymanski, B.K.: Labelrankt: Incremental community detection in dynamic networks via label propagation. In: Proceedings of the Workshop on Dynamic Networks Management and Mining (DyNetMM 2013), pp. 25–32. ACM (2013)
41. Xie, J., Szymanski, B.K.: Community detection using a neighborhood strength driven label propagation algorithm. In: Proceedings of the IEEE Network Science Workshop (NSW 2011), pp. 188–195. IEEE Computer Society (2011)
42. Xie, J., Szymanski, B.K.: Towards linear time overlapping community detection in social networks. In: P.N. Tan, S. Chawla, C. Ho, J. Bailey (eds.) Advances in Knowledge Discovery and Data Mining, *Lecture Notes in Computer Science*, vol. 7302, pp. 25–36. Springer Berlin Heidelberg (2012)
43. Yun, S.Y., Lelarge, M., Proutiere, A.: Streaming, memory limited algorithms for community detection. In: Advances in Neural Information Processing Systems (NIPS 2014), pp. 3167–3175 (2014)