Community Detection in Who-calls-Whom Social Networks

Ciprian-Octavian Truică[1](✉), Olivera Novović[2], Sanja Brdar[2], and Apostolos N. Papadopoulos[3]

[1] Computer Science and Engineering Department, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Bucharest, Romania
`ciprian.truica@cs.pub.ro`

[2] BioSense Institute, University of Novi Sad, Novi Sad, Serbia
`novovic@biosense.rs`, `brdars@uns.ac.rs`

[3] Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece
`papadopo@csd.auth.gr`

Abstract. Mobile phone service providers collect large volumes of data all over the globe. Taking into account that significant information is recorded in these datasets, there is a great potential for knowledge discovery. Since the processing pipeline contains several important steps, like data preparation, transformation, knowledge discovery, a holistic approach is required in order to avoid costly ETL operations across different heterogeneous systems. In this work, we present a design and implementation of knowledge discovery from CDR mobile phone data, using the Apache Spark distributed engine. We focus on the community detection problem which is extremely challenging and it has many practical applications. We have used Apache Spark with the Louvain community detection algorithm using a cluster of machines, to study the scalability and efficiency of the proposed methodology. The experimental evaluation is based on real-world mobile phone data.

Keywords: Data mining · Big data analytics · Community detection

1 Introduction

Mobile phone service providers collect large amount of data to monitor user interactions. Each time a user is using a mobile device (for sending an SMS or performing a call), a Call Detail Record (CDR) is created in the database of the service provider. Graphs have a central role in the analysis of mobile phone data collected by service providers. Due to their mathematical formalism and the variety of existing graph-based algorithmic techniques, they can be used efficiently and effectively in social networks, to solve specific problems.

C.-O. Truică and O. Novović contributed equally to this work.

Graph mining is a heavily active research direction with numerous applications [1]. One of the core research directions in the area is the discovery of meaningful communities in a large network [11]. In the majority of real-life applications, graphs are extremely sparse usually following power-law degree distribution. However, the original graph may contain groups of vertices, called *communities*, where vertices in the same community are more well-connected than vertices across communities.

The efficiency of community detection algorithms is heavily dependent on the size of the input graph, i.e., the number of vertices and/or the number of edges, and also on its structural complexity. In addition to the main processing task that must be performed, preprocessing is also a significant step that in many cases is computationally intensive. To handle both preprocessing and main processing efficiently, a potential solution is to use multiple resources and apply parallel or distributed computing techniques, aiming at reducing the overall processing time.

In this work, we focus on the analysis of real world CDR data, and more specifically on scalable community detection. CDRs are in general large in volume, and therefore scalable algorithmic techniques are required. In particular, we demonstrate the use of Apache Hadoop [26], Apache YARN [30], Apache Spark [15], and Apache Hive [28] in the mining process as a proof of concept.

In our previous work [20], we had used a conventional DBMS and Python to extract knowledge from raw telecom data. The experiments were very time consuming and we could analyze only a subset of the data in due time. The results that we obtained have motivated us to apply different processing techniques in order to speed up the experimental evaluation and to be able to analyze the complete dataset.

Our approach for analyzing the data is based on Apache Spark. The proposed implementation considers the complete pipeline, from preprocessing the raw data to knowledge discovery. All necessary tasks are executed within Spark and results are stored in Hive. Based on our performance evaluation results, we show that by applying graph sparsification through filtering, communities are discovered more efficiently, since runtime depends heavily on the number of edges in the graph. On the other hand, by comparing the communities generated with and without filtering we observe that communities remain relatively stable in comparison to the ground truth (unfiltered graph). The discovered communities reflect dynamic social interactions that are along with other components of the city, transport and land use, essential for the understanding of such a complex system [10,13].

The rest of the paper is organized as follows. In the next section we describe briefly related work in the area. The proposed methodology is described in detail in Sect. 3. Section 4 presents the implementation of our pipeline. Section 5 offers performance evaluation results using real-world networks. Finally, Sect. 6 concludes the work and presents briefly some interesting future research directions.

2 Related Work

During the last few years, there is a tremendous growth of new applications that are based on the analysis of mobile phone data [6]. Among them there are many applications with significant societal impact such as, urban sensing and planning [5,10], traffic engineering [2,14], predicting energy consumption [8], disaster management [18,22,33], epidemiology [9,17,32], deriving socio-economical indicators [21,27].

To enable development and run of applications and services on such data, current efforts are directed toward providing access to these large scale human behavioral data in a privacy-preserving manner. Recent initiative of Open Algorithms (OPAL) has suggested approach of moving the algorithm to the data [16]. In this model, raw data are never exposed to outside parties, only vetted algorithms run on telecom companies' servers. This poses huge challenge on efficient processing of data, especially when array of parties is interested in extracting information and getting insights from data.

A significant graph mining task with important applications is the discovery of meaningful communities [11]. In many cases, community detection is solved by using graph clustering algorithms. However, a major limitation of graph clustering algorithms is that they are computationally intensive, and therefore their performance deteriorate rapidly, as we increase the size of the data.

An algorithm that has been used for community detection in large networks is the Louvain algorithm, proposed in [7]. This algorithm has many practical applications and it scales well with the size of the data. Moreover, it has been used in several studies related to static or evolving community detection [3].

In this work, we combine the efficiency of the Louvain algorithm with the power of the Apache Spark distributed engine, to demonstrate that we can support the full pipeline of community detection in a efficient manner.

3 Proposed Methodology

In this section, we describe our approach in detail, explaining the algorithms for community detection and filtering as well as the evaluation methods used.

3.1 Graph Mining and Community Detection

Among the different existing graph mining problems, we center our focus on community detection. The graph, in our case, corresponds to user interactions aggregated to Radio Base Station (RBS) level. Therefore, communities correspond to groups of RBSs with strong pair-wise activity within each group. To enable efficient community detection in potentially massive amounts of data, we need to attack the following problems: (i) the algorithmic techniques applied must scale well with respect to the size of the data, which means that the algorithmic complexity should stay below $\mathcal{O}(n^2)$ (where n is the number of graph nodes), and (ii) since we do not know the number of communities in advance, the algorithms used must be flexible enough to be able to infer the number of communities during the course of the algorithm.

To meet the aforementioned requirements, we have chosen to apply a modularity-based algorithm proposed in [7]. The concept of modularity [19] presented by Eq. (1), is used to estimate the quality of the partitions, where A_{ij} is the weight of the edge connecting the i-th and the j-th node of the graph, $\sum_j A_{ij}$ is the sum of the weights of the edges attached to the i-th node, c_i is the community where the i-th node is assigned to, $m = (1/2)\sum_{i,j} A_{ij}$, and $\delta(x, y)$ is zero if nodes x and y are assigned to the same community and 1 otherwise.

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{i,j} - \frac{\sum_j A_{ij} \cdot \sum_i A_{ji}}{2m}\right]\delta(c_i, c_j) \tag{1}$$

Unfortunately, computing communities based on the maximization of the modularity, is an $\mathcal{NP}$-hard problem. To provide an efficient solution, the algorithm proposed in [7] uses an iterative process that involves shrinking the graph, every time modularity converges. In each phase, each node is assigned to a neighboring community that maximizes the modularity of the graph. As long as nodes are moving around communities and modularity grows, we keep on executing this process. When there are no more changes, a shrinking process is executed. Upon shrinking the graph, each community produced during the previous phase is assigned to the same super node of the new graph. Then, the same technique is applied to the new graph. The algorithm terminates when the modularity detected in the new graph is less than the modularity detected in the previous graph. The set of communities that maximize the modularity is returned as an answer. The outline of the technique is depicted in Algorithm 1.

The network we are studying is undirected and weighted, but elimination of some edges by simple thresholding the weight values is not going to reveal the core backbone of the network. Moreover, we needed a method for graph filtering that will consider local properties of the nodes, such as weight over all edges linked to specific node. To meet the aforementioned requirements we have chosen to apply the disparity filter [25]. The disparity filter uses the null model to define significant links, where the null hypothesis states that weighted connections of the observed node are produced by a random assignment from a uniform distribution. The disparity filter proceeds by identifying strong and weak links for each node. The discrimination is done by calculating for each edge the probability α_{ij} that its normalized weight p_{ij} is compatible with the null hypothesis. All the links with $\alpha_{ij} < \alpha$ reject the null hypothesis. The statistically relevant edges will be those whose weights satisfy the Eq. (2).

$$\alpha_{ij} = 1 - (n-1)\int_0^{p_{ij}} (1-x)^{n-2} dx < \alpha \tag{2}$$

We note that smaller values of α_{ij} denote more significant edges. Therefore, filtering is applied by keeping all edges where $\alpha_{ij} \leq \alpha$ and thus removing all edges where $\alpha_{ij} > \alpha$. By changing the alpha threshold value we can filter out the links

Algorithm 1. LOUVAIN $(G(V, E))$

```
   Input: the graph $G$
   Result: the communities of $G$
1  $n \leftarrow |V|$ /* number of graph nodes */
2  $done \leftarrow false$
3  while not done do
4      assign each $u \in V$ to a different community
5      while there is a change do
6          for every node $u \in V$ do
7              $C \leftarrow$ a community that maximizes modularity /* $C$ is a
                 neighboring community or $u$'s community */
8      if newModularity > oldModularity then
9          $G \leftarrow$ shrink graph based on communities /* each community
             becomes a super node in the new graph */
10     else
11         return communities
```

with small significance focusing on more relevant edges. The α_{ij} represent the statistical probability, so it's value is in range [0, 1]. The threshold alpha value is set in regards to the significance level with which we want to apply the filtering. In our experiments, we applied three different filtering levels with probability 95%, 99% and 99.9%, where corresponding α values are 0.05, 0.01 and 0.001 respectively.

3.2 Clustering Evaluation Methods

To evaluate the community detection, we will use classical cluster evaluation methods, i.e. Purity, Entropy, Rand Index and Adjusted Rand Index.

The *Purity* (Definition 1) of a cluster measures the extent to which each cluster contains elements from primarily one class [34].

Definition 1 (Purity). *Given a set S of size n and a set of cluster C of size k, then, for a cluster $c_i \in C$ of size k_i, the purity is $p(c_i) = \frac{\max_i(n_j^i)}{k_i}$, where n_j^i is the number of elements of the j-th class assigned to the i-th cluster. The overall purity is defined as $P(C) = \sum_{i=1}^{k} \frac{k_i}{n} \cdot p(c_i)$.*

Entropy (Definition 2) is an evaluation method that assumes that all elements of a set have the same probability of being picked and, by choosing an element at random, the probability of this element to be in a cluster can be computed [31].

Definition 2 (Entropy). *Given a set S of size n and a set of clusters C of size k, then, by assuming that all elements in S have the same probability of being picked, the probability of an element $s \in S$ chosen at random to belong to* *cluster $c_i \in C$ of size k_i is $p(c_i) = \frac{k_i}{n}$. Then, the overall entropy associated with C is $H(C) = -\sum_{i=1}^{k} p(c_i) \cdot \log_2(p(c_i))$.*

The *Rand Index (RI)* (Definition 3) is a measure used to determine the similarity between two data clusterings [23].

Definition 3 (Rand Index). *Given a set $S = \{s_1, s_2, ..., s_n\}$ of n elements and two groupings $X = \{X_1, X_2, ..., X_r\}$ and $Y = \{Y_1, Y_2, ..., Y_t\}$ then the Rand Index is $RI = \frac{a+b}{\binom{n}{2}}$, where $a = |S'|$ with $S' = \{(s_i, s_j) | s_i, s_j \in X_k, s_i, s_j \in Y_l\}$ and $b = |S''|$ with $S'' = \{(s_i, s_j) | s_i \in X_{k_1}, s_j \in X_{k_2}, s_i \in Y_{l_1}, s_j \in Y_{l_2}\}$ for some $1 \le i, j \le n, i \ne j, 1 \le k, k_1, k_2 \le r, k_1 \ne k_2, 1 \le l, l_1, l_2 \le t, l_1 \ne l_2$.*

Adjusted Rand Index (ARI) (Definition 4) is a cluster evaluation method that calculates the fraction of correctly classified (respectively misclassified) elements to all elements by assuming a generalized hypergeometric distribution as null hypothesis [31]. ARI is the normalized difference of the Rand Index and its expected value under the null hypothesis [29]. ARI uses the contingency table.

Definition 4 (Adjusted Rand Index). *Given a set S of n elements and two groupings $X = \{X_1, X_2, ..., X_r\}$ and $Y = \{Y_1, Y_2, ..., Y_s\}$, the overlap between X and Y, can be summarized in a contingency table $[n_{ij}]$, where $n_{ij} = |X_i \cap Y_j|$, $a_i = \sum_{j=1}^{s} n_{ij}$ and $b_j = \sum_{i=1}^{r} n_{ij}$. Using the contingency table, the Adjusted Rand Index is defined in Eq. (3).*

$$ARI = \frac{\sum_{i,j} \binom{n_{ij}}{2} - \frac{\sum_i \binom{a_i}{2} \cdot \sum_j \binom{b_j}{2}}{\binom{n}{2}}}{\frac{1}{2}(\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}) - \frac{\sum_i \binom{a_i}{2} \cdot \sum_j \binom{b_j}{2}}{\binom{n}{2}}} \tag{3}$$

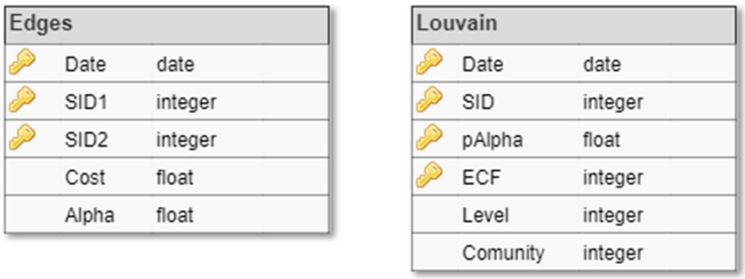
4 Implementation Methodology

4.1 Dataset

Telecommunication interaction between mobile phone users is managed by Radio Base Stations (RBS) that are assigned by the operator. Every RBS has a *unique id*, a *location* and a *coverage map* that provides approximate user's geographical location. CDRs contain the time of the interaction and the RBS which handled it. In available data collection CDRs are spatially aggregated on the grid containing 10,000 cells and temporally aggregated on time slots of ten minutes. Evidently, the final network is composed of 10,000 nodes. However, if we change the granularity and the level of the detail as well as the geographical area that we are interested in, the number of nodes can grow easily to millions.

Community detection is done using the LOUVAIN algorithm. Filtering is used to minimize the graph and keep only the important nodes without losing the communities. In our case, the number of nodes is 10,000. The value of the parameter α_{ij} (Eq. (2)) defines the level of filtering. By changing the filtering level we obtain graphs with fewer edges. We are using these graphs to demonstrate the performance of community detection as we grow the number of edges in the input graph.

The dataset (C) provides real world information regarding the *directional interaction strength* (DIS) based on call exchanged between different areas of the city of Milan [4] and it's publicly available online[1]. The DIS between two areas ($SID1$ and $SID2$) is proportional to the number of calls issued from area $SID1$ to area $SID2$. The temporal values, given as a timestamp, are aggregated in time slots which represent the beginning of the interval. The dataset represents a directed graph. For our experiments, two transformations are applied on the original corpus: (i) the directed graph is transformed into an undirected graph, i.e. the edges (SID_i, SID_j) and (SID_j, SID_i) are going to be represented as a single edge (SID_i, SID_j) with the edge $Cost_{ij} = DIS_{ij} + DIS_{ji}$ for the same timestamp, and (ii) the timestamp is aggregated to a calendar date ($Date$), i.e. for an edge (SID_i, SID_j) for each $Date$ the cost is $Cost = \sum DIS_{ij}$. Using this information we compute for each edge the parameter α_{ij} using Eq. (2). Moreover, an *Edge Cost Factor (ECF)* is used to normalize the values for the edges' $Cost \in [9 \cdot 10^{-13}, 466]$ when applying Louvain. This information is stored in a Hive database installed on top of Hadoop's HDFS and MapReduce. Figure 1 presents the database diagram where the information about edges is stored in the *Edges (E)* table, while the information about communities is stored in the *Louvain (L)* table, where *Level* is the algorithm's iterations.



Edges		
🔑	Date	date
🔑	SID1	integer
🔑	SID2	integer
	Cost	float
	Alpha	float

Louvain		
🔑	Date	date
🔑	SID	integer
🔑	pAlpha	float
🔑	ECF	integer
	Level	integer
	Comunity	integer

Fig. 1. The Hive database schema.

4.2 System Architecture

Apache Spark is a unified distributed engine with a rich and powerful APIs for different programming languages [15], i.e. Scala, Python, Java and R. One of its main characteristics is that (in contrast to Hadoop MapReduce) it exploits main memory as much as possible, being able to persist data across rounds to avoid unnecessary I/O operations. Spark jobs are executed based on a master-slave model in cooperation with the cluster manager YARN.

[1] Dataset http://theodi.fbk.eu/openbigdata/.

The proposed architecture is based on the Apache Spark engine using the Scala programming language together with the Spark Dataframes, HiveContext and GraphX libraries. The information is stored in an Apache Hive database which is installed on top of HDFS. The cluster resource manager used is YARN. Our methodology utilizes the following pipeline (Fig. 2):

1. CDRs are aggregated in such a way that each graph node corresponds to a spatial area. This task has been performed by the mobile operator before releasing the data.
2. The original directed graph is aggregated to obtain an undirected one, as the orientation of edges is ignored in our case.
3. Filtering is applied in order to sparsify the network.
4. Community detection is applied using the LOUVAIN algorithm.
5. Visualization is applied.

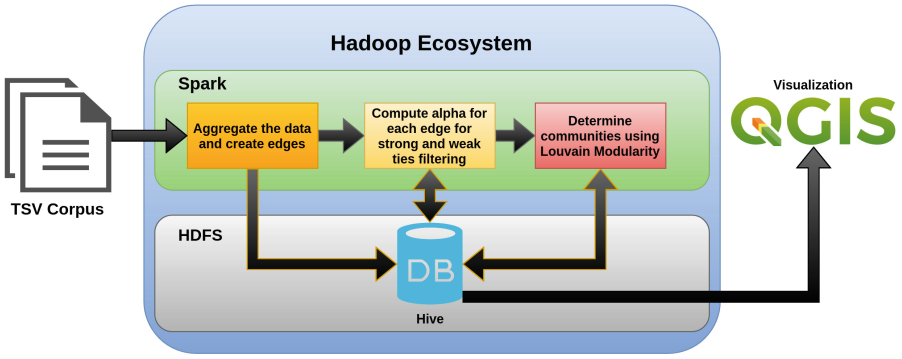

Hadoop Ecosystem
Spark
Aggregate the data and create edges
Compute alpha for each edge for strong and weak ties filtering
Determine communities using Louvain Modularity
Visualization
QGIS
TSV Corpus
HDFS
DB
Hive

Fig. 2. Architecture

4.3 Queries

The Table of Edges. To create the edges (E), the DIS for attributes $DATE$, $SID1$ and $SID2$, which represent a compound primary key that uniquely identify each record, is aggregated. The aggregation constructs an undirected graph using the union between the records where $SID1 \leq SID2$ and $SID1 > SID2$. The resulting graph is stored in the Hive database. The following query, expressed in relational algebra, is used to populate the $Edges$ table:

$$E = \rho_{\frac{Cost}{F_0}}(\pi_{Date,SID1,SID2,F_0}(\gamma_{L_0}(\pi_{Date,SID1,SID2,DIS}(\sigma_{SID1 \leq SID2}(C)) \biguplus \pi_{Date,SID2,SID1,DIS}(\sigma_{SID1 > SID2}(C)))))$$

where C is the corpus and γ_{L_0} is the aggregation operator with $L_0 = (F_0, G_0)$, $F_0 = sum(DIS) = Cost$, the sum is the aggregation function that sums the

DIS for all the pairs $(Date, SID1, SID2)$ and $G_0 = (Date, SID1, SID2)$ is the list of attributes in the GROUP BY clause.

Strong and Weak Ties. To determine the strong and weak ties for filtering, α_{ij} from Eq. (2) is computed for each unique key $(Date, SID1, SID2)$. Therefore, the integral must be solved in order to use directly the information stored in the database and to improve the time performance: $\alpha = 1-(k-1)\cdot\int_0^{p_{ij}}(1-x)^{k-2}dx = 1-(1-p_{ij})^{k-1}$, where $p_{ij} = \frac{Cost_{ij}}{\sum_{j,i\neq j} Cost_{ij}} = \frac{Cost}{\sum_c} = Alpha$ and $k = N$ the number of nodes.

To compute the sum of costs for each $Date$ and $SID1$ ($\sum_c$), the following query, given in relational algebra, is used:

$$R_1 = \rho_{\frac{\Sigma_c}{F_1}}(\pi_{Date,SID1,F_1}(\gamma_{L_1}(\sigma_{SID1\neq SID2}(E))))$$

where γ_{L_1} is the aggregation operator with $L_1 = (F_1, G_1)$, $F_1 = sum(Cost) = \sum_c$, the sum is the aggregation function that sums the $Cost$ for all the pairs $(Date, SID1)$, and $G_1 = (Date, SID1)$ is the list of attributes in the GROUP BY clause.

The following query is used to compute the number of distinct nodes for each *Date*:

$$R_2 = \rho_{\frac{N}{F_2}}(\pi_{Date,SID1,F_2}(\gamma_{L_2}(E)))$$

where γ_{L_2} is the aggregation operator with $L_2 = (F_2, G_2)$, $F_2 =$ *count(DISTINCT SID1)=N*, the *count* is the aggregation function that counts the distinct number of nodes for a *Date* and $G_2 = (Date)$ is the list of attributes in the GROUP BY clause.

To compute α_{ij}, R_1 must be joined with R_2. The result is stored in the database either as a separate table or by updating the *Edges*'s table column *Alpha*. The query expressed in relational algebra is:

$$A = \pi_{Date,SID1,SID2,Cost,Alpha}(\sigma_{SID1\neq SID2}(E) \bowtie_\theta R_1 \bowtie_{E.Date=R_2.Date} R_2)$$

The θ condition in the first join operator is defined as: $E.Date = R_1.Date \wedge E.SID1 = R_1.SID1$.

Community Detection. To apply the Louvain algorithm and detect communities for a given date (*pDate*) and alpha threshold (*pAlpha*), the following query is used to extract the information from the database:

$$L_M = \pi_{SID1,SID2,Cost}(\sigma_{Date=pDate\wedge Alpha\leq pAlpha}(A))$$

The Louvain algorithm is implemented in the Spark engine using the Scala programming language and the GraphX, Dataframes and HiveContext library. Our implementation extends an existing implementation[2] so that it works with our architecture. In order to apply Louvain, a graph should be given in the input, i.e., using the query L_M. The result of the algorithm is stored in the Hive database in table *Louvain*.

[2] Louvain https://github.com/Sotera/spark-distributed-louvain-modularity.

5 Performance Evaluation

The goal of the evaluation is to demonstrate the performance of community detection with and without filtering, using Apache Spark. Two diverse systems have been used: (*i*) a multi core server machine running CentOS with 16 Intel Xeon E5-2623 CPUs with 4 cores at 2.60 GHz, 126 GB RAM and 1 TB HDD, and (*ii*) a cluster with 6 nodes running Ubuntu 16.04×64, each with 1 Intel Core i7-4790S CPU with 8 cores at 3.20 GHz, 16 GB RAM and 500 GB HDD. The Hadoop ecosystem is running on Ambari and has the following configuration for the 6 nodes: 1 node acts as HDFS NameNode and SecondaryName Node, YARN ResourceManager, and Hive Metastore and 5 nodes, each acting as HDFS DataNodes, YARN NodeManagers, Spark Client, and Hive Client.

For the experiments we fix the number of Spark executors to 16 with one vnode and 3 GB memory each. The same settings have been used in both the Single Machine and Cluster Mode. The complete pipeline has been implemented in Scala, whereas the code is publicly available on GitHub[3].

5.1 Runtime Evaluation

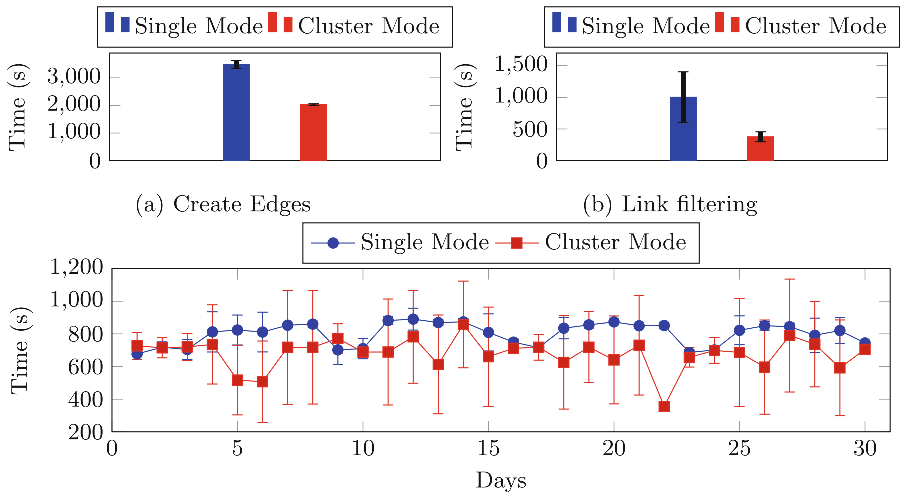
The first experiment measures the performance of edge generation from raw data. The second experiment measures the performance of computing α values (see Eq. (2)). The third experiment is related to the performance of Louvain algorithm over a filtered graph where $\alpha = 0.05$ and $ECF = 1,000,000$.

For each experiment we measure the average runtime over a series of 10 runs. For the first task (creating edges), the cluster environment showed 42% better runtime than the single mode case, whereas the standard deviation is relatively small in both cases. For the task related to computing α values, the cluster showed again better performance, since the computation is 63% faster. In this case, the standard deviation is higher for both environments in comparison to the previous task. Finally, the performance of community detection using Louvain differ for each graph, but in most cases the cluster showed the best performance. In most of the cases the mean values of runtime for Louvain are higher in the case of single machine, whereas the standard deviation is higher in the cluster environment. We hypothesize that these results are a direct consequence of how YARN's ResourceManager schedules the ApplicationManager and NodeManager and Spark's directed acyclic graph (DAG) execution engine optimizes the graph construction for the GraphX library. Figure 3 shows some representative comparative results.

5.2 Community Detection and Evaluation

For community visualization, the QGIS software has been used. We have chosen to present the set of communities generated by using the 8th of November, because the network for this particular day contains the highest number of edges.

[3] GitHub repository https://github.com/cipriantruica/MBD_CommunityDetection.



(a) Create Edges

(b) Link filtering

(c) Louvain community detection

Fig. 3. Runtime results.

Experiments are performed on the cluster for three different threshold values, i.e., $\alpha = \{0.001, 0.01, 0.05\}$, and $ECF = 10^{12}$. We used the results of community detection performed over unfiltered graph as the ground truth result, and compare it with the results when the filtering was applied. After filtering is performed, the Louvain community detection algorithm is executed on Spark. The first level of filtering eliminates more then 50% of edges, while the runtime for Louvain clustering algorithm improves with a factor of 2.46. When $\alpha = 0.01$ almost 70% of edges are eliminated, and the algorithm runtime improves with a factor of 3.7. Filtering with $\alpha = 0.001$ eliminates almost 80% of the edges, the algorithm's runtime improves by a factor of 6.88.

Table 1. Number of nodes and edges after applying different filtering levels, number of communities and runtime community detection.

α	# nodes	# edges	# communities	Time (sec.)
1	10,000	29,099,392	175	2,229.73 ± 340.14
0.05	10,000	12,942,551	174	906.43 ± 279.79
0.01	10,000	9,003,404	176	603.20 ± 174.28
0.001	10,000	6,043,769	186	324.21 ± 127.73

The number of communities changes when filtering is applied. The results for 10 tests are presented in Table 1. The Louvain community detection algorithm converges to the same result for the same input graph. The number of communities is higher for the graphs where the filtering is stricter. That is expected, because the higher level of filtering gets the graph containing the strongest links. Moreover, as the number of nodes stays constant, the removal of edges tends to create more communities. In Fig. 4 we observe the centrality pattern of the clustering for each graph. The structure of communities is denser in the city center area, while in the peripheral parts communities are more spatially spread. That is due to overall higher traffic of people in city center, which reflects to telecom network. We observe also that the number of communities produced from graphs where the filtering with $\alpha = 0.01$ and $\alpha = 0.05$ is applied does not differ much from the number of communities produced from the unfiltered graph. On the other hand, the runtime for filtered and unfiltered graphs differs significantly. Even with the less strict filtering applied, we get a major improvement in processing time, which justifies the use of filtering.

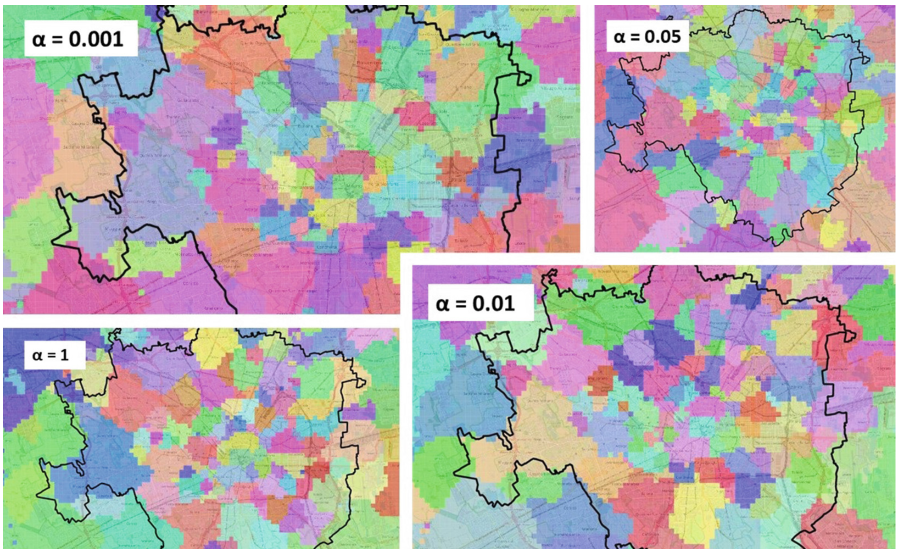


Fig. 4. Communities formed for different filtering thresholds.

Community evaluation is performed using *Purity*, *Entropy*, *Rand Index* and *Adjusted Rand Index* measures. The results are given in Table 2 and they are

Table 2. Clustering evaluation for different threshold values.

Evaluation measure	$\alpha = 0.001$	$\alpha = 0.01$	$\alpha = 0.05$
Purity	0.055	0.052	0.048
Entropy	0.902	0.909	0.912
Rand Index	0.981	0.985	0.987
Adjusted Rand Index	0.662	0.745	0.781

obtained by using Louvain without filtering as the ground truth. *Purity* measures how many elements from the communities determined by Louvain, when filtering is applied belong, to the ground truth communities. This measure is low because the number of communities differ between the ground truth and each different α threshold. The *Entropy* measures the probability of a node chosen at random to belong to a community. The high results of this measure describes how much we can, on the average, reduce the uncertainty about the cluster of a random element when knowing its cluster, the ground truth, in another clustering of the same set of elements. The *Rand Index* provides information on how a new clustering is compared to a correct one, the ground truth, and it is highly dependent on the number of clusters. This measure is high for all the tests because the *Rand Index* converges to 1 as the number of clusters increases which is undesirable for a similarity measure [12]. To address this problem we also computed the *Adjusted Rand Index* which shows a clearer classification for the filtering technique. For the threshold $\alpha = 0.05$, as well as for $\alpha = 0.01$, the number of communities remain stable and closer to the ones detected in the ground truth, although the number of edges decreases significantly (see Table 1).

6 Conclusions

Mobile phone records offer many potentials for knowledge discovery with significant impact. In particular, community detection is a task related to networks and aims at the discovery of groups of nodes that are densely connected. In general, community detection is solved by executing graph clustering algorithms, which is a computationally intensive task, and therefore scalable algorithms should be applied to guarantee efficiency for large networks.

This work focuses on community detection in a distributed environment, based on real-world mobile phone data. The first results has shown that parallelism is an important tool to attack scalability issues, since we can analyze larger graphs by using a cluster of machines. Moreover, we have shown that by applying sparsification through filtering, we may boost performance even further without penalizing the quality of the community detection result.

There are several interesting directions for future work, such as: (i) the modification of the modularity definition to reflect the association between CDR records and spatial information, (ii) the implementation and comparison of different community detection algorithms (e.g., Infomap [24]) and filtering techniques, (iii) the impact of filtering on other community detection algorithms, and (iv) the design of distributed community detection techniques for evolving networks, combining the mobile data information with the spatial location and tracking changes in communities as the network evolves. In addition, we are planning to test the proposed methodology for massive real-world networks, to be able to study scalability more thoroughly.

Acknowledgement. This article is based upon work from COST Action IC1406 High-Performance Modelling and Simulation for Big Data Applications (cHiPSet), supported by COST (European Cooperation in Science and Technology).

References

1. Aggarwal, C.C., Wang, H.: Managing and Mining Graph Data. Springer, London (2010)
2. Alexander, L., Jiang, S., Murga, M., González, M.C.: Origin-destination trips by purpose and time of day inferred from mobile phone data. Transp. Res. Part C Emerg. Technol. **58**, 240–250 (2015)
3. Aynaud, T., Guillaume, J.: Static community detection algorithms for evolving networks. In: International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, pp. 513–519 (2010)
4. Barlacchi, G., De Nadai, M., Larcher, R., Casella, A., Chitic, C., Torrisi, G., Antonelli, F., Vespignani, A., Pentland, A., Lepri, B.: A multi-source dataset of urban life in the city of Milan and the Province of Trentino. Sci. Data **2** (2015). Article ID 150055
5. Becker, R.A., Caceres, R., Hanson, K., Loh, J.M., Urbanek, S., Varshavsky, A., Volinsky, C.: A tale of one city: using cellular network data for urban planning. IEEE Pervasive Comput. **10**(4), 18–26 (2011)
6. Blondel, V.D., Decuyper, A., Krings, G.: A survey of results on mobile phone datasets analysis. EPJ Data Sci. **4**(1), 10 (2015)
7. Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theor. Exp. **2008**(10) (2008). Article ID P10008
8. Bogomolov, A., Lepri, B., Larcher, R., Antonelli, F., Pianesi, F., Pentland, A.: Energy consumption prediction using people dynamics derived from cellular network data. EPJ Data Sci. **5**(1), 13 (2016)
9. Brdar, S., Gavrić, K., Ćulibrk, D., Crnojević, V.: Unveiling spatial epidemiology of HIV with mobile phone data. Sci. Rep. **6**, article id 19342 (2016)
10. Calabrese, F., Ferrari, L., Blondel, V.D.: Urban sensing using mobile phone network data: a survey of research. ACM Comput. Surv. **47**(2), 25:1–25:20 (2014)
11. Fortunato, S.: Community detection in graphs. Phys. Rep. **483**(3), 75–174 (2010)
12. Fowlkes, E.B., Mallows, C.L.: A method for comparing two hierarchical clusterings. J. Am. Stat. Assoc. **78**(383), 553–569 (1983)
13. Gao, S., Liu, Y., Wang, Y., Ma, X.: Discovering spatial interaction communities from mobile phone data. Trans. GIS **17**(3), 463–481 (2013)
14. Järv, O., Ahas, R., Saluveer, E., Derudder, B., Witlox, F.: Mobile phones in a traffic flow: a geographical perspective to evening rush hour traffic analysis using call detail records. PLoS ONE **7**(11), 1–12 (2012)
15. Karau, H., Konwinski, A., Wendell, P., Zaharia, M.: Learning Spark: Lightning-Fast Big Data Analytics, 1st edn. O'Reilly Media Inc., Sebastopol (2015)
16. Lepri, B., Oliver, N., Letouzé, E., Pentland, A., Vinck, P.: Fair, transparent, and accountable algorithmic decision-making processes. Philos. Technol. 1–17 (2017)
17. Lima, A., De Domenico, M., Pejovic, V., Musolesi, M.: Disease containment strategies based on mobility and information dissemination. Sci. Rep. **5**, article id 10650 (2015)
18. Lu, X., et al.: Detecting climate adaptation with mobile network data in Bangladesh: anomalies in communication, mobility and consumption patterns during cyclone Mahasen. Clim. Change **138**(3–4), 505–519 (2016)
19. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69** (2004). Article ID 026113

20. Novović, O., Brdar, S., Crnojević, V.: Evolving connectivity graphs in mobile phone data. In: NetMob, The Main Conference on the Scientific Analysis of Mobile Phone Datasets, pp. 73–75. Vodafone (2015)
21. Pappalardo, L., Pedreschi, D., Smoreda, Z., Giannotti, F.: Using big data to study the link between human mobility and socio-economic development. In: IEEE International Conference on Big Data (Big Data), pp. 871–878 (2015)
22. Pastor-Escuredo, D., Morales-Guzmán, A., Torres-Fernández, Y., Bauer, J.-M., Wadhwa, A., Castro-Correa, C., Romanoff, L., Lee, J.G., Rutherford, A., Frias-Martinez, V., Oliver, N., Frias-Martinez, E., Luengo-Oroz, M.: Flooding through the lens of mobile phone activity. In: Global Humanitarian Technology Conference (GHTC), pp. 279–286. IEEE, October 2014
23. Rand, W.M.: Objective criteria for the evaluation of clustering methods. J. Am. Stat. Assoc. **66**(336), 846–850 (1971)
24. Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure. Proc. Natl. Acad. Sci. **105**(4), 1118–1123 (2008)
25. Serrano, M.Á., Boguná, M., Vespignani, A.: Extracting the multiscale backbone of complex weighted networks. Proc. Natl. Acad. Sci. **106**(16), 6483–6488 (2009)
26. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: Symposium on Mass Storage Systems and Technologies, pp. 1–10 (2010)
27. Steele, J.E., et al.: Mapping poverty using mobile phone and satellite data. J. R. Soc. Interface **14**(127), article id 20160690 (2017)
28. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. Proc. VLDB Endow. **2**(2), 1626–1629 (2009)
29. Truică, C.-O., Rădulescu, F., Boicea, A.: Comparing different term weighting schemas for topic modeling. In: International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2016), pp. 307–310 (2016)
30. Vavilapalli, V.K., et al.: Apache Hadoop YARN: yet another resource negotiator. In: Annual Symposium on Cloud Computing, pp. 5:1–5:16 (2013)
31. Wagner, S., Wagner, D.: Comparing clusterings: an overview. Technical report (2007)
32. Wesolowski, A., et al.: Impact of human mobility on the emergence of dengue epidemics in Pakistan. Proc. Natl. Acad. Sci. **112**(38), 11887–11892 (2015)
33. Wilson, R., et al.: Rapid and near real-time assessments of population displacement using mobile phone data following disasters: the 2015 Nepal earthquake. PLoS Curr. **8** (2016)
34. Zhao, Y., Karypis, G.: Criterion functions for document clustering: experiments and analysis. Technical report (2002)