EgoLP: Fast and Distributed Community Detection in Billion-node Social Networks

Nazar Buzun, Anton Korshunov, Valeriy Avanesov,
Ilya Filonenko, Ilya Kozlov, Denis Turdakov
Institute for System Programming of
the Federal Agency of Scientific Organizations
Moscow, Russia
Email: {nazar, korshunov, avanesov, filonenko,
kozlov-ilya, turdakov}@ispras.ru

Hangkyu Kim
Samsung Electronics Co. Ltd.
Suwon, South Korea
Email: hangkyu.kim@samsung.com

***Abstract*—Community structure is one of the most important and characteristic features of social networks. Numerous methods for discovering implicit user communities from a social graph of users have been proposed in recent years. However, most of them have performance and scalability issues which make them hardly applicable to population-wide analysis of modern social networks (billions of users and growing). In this paper we present `EgoLP` – an efficient and fully distributed method for social community detection. The method is based on propagating community labels through the network with the help of friendship groups of individual users. Experimental evaluation of Apache Spark implementation of the method showed that it outperforms some state-of-the-art methods in terms of a) similarity of extracted communities to the reference ones from synthetic networks; b) precision of user attributes prediction in Facebook based solely on community memberships; c) likelihood of the discovered community structure according to the proposed generative model. At the same time, the method retains near-linear complexity in the number of edges and is thus applicable to social graphs of up to 10^9 users.**

Keywords*—*Community detection, social networks, graph clustering, distributed algorithms.

I. Introduction

Online social networks are known to inherit the natural community structure of their populations [9]. Users of social services tend to unite either explicitly (by means of grouping functionality of network software) or implicitly (by establishing ties based on shared affiliation, role, activity, social circle, interest, function, or some other property).

According to the common identity and common bond theory [3], people join groups based on identity (i.e., interest in the topics discussed) or bond attachment (i.e., social relationships). However, it is very time-consuming for a user to support a lot of consistent groups and categories explicitly. Further, for many people it could be uncomfortable to explicitly declare memberships in some groups (politics/religion/sexuality/games) because of peer pressure. Finally, certain kinds of groups (geographical/educational/job) are based on attribute similarity and could not be fully recovered due to incompleteness of user profiles.

At the same time, uncovering implicit community structure of a social network (learning the list of communities for any given user) would provide an additional input for many analytic applications and services. For example, great challenges rise in tasks like user-item recommendation, social matchmaking, textual content categorisation and filtering. Recent research [23], [22] shows that recommendation systems could benefit from involving multiple types of linked objects. Besides user-item relations, the system should also account for user-community, item-category or item-author connections. The reason is sparse structure of the observed user-item matrix which causes overfitting. So, additional types of relations, including communities structure, play regularization role in such tasks.

It was shown that user communities in online social networks are overlapping: e.g., in LiveJournal a user participates in ~ 3 communities in average [7]. Besides, even in networks with small number of communities per user the distribution of user-community memberships appears to be power-law [19]. Hence, there always exist users with many communities which makes them highly overlapping. However, most of the proposed overlapping community detection methods [1], [2] suffer from high computational complexity and poor scalability which makes them practically inapplicable when the input data scales beyond 10-100 million users[1].

We present `EgoLP` – an efficient and fully distributed method for extracting implicit communities of social network users given inter-user connectivity data. The method first picks each user and identifies her *egomunities* [11]. The latter are cohesive subgroups within *ego-network* – the network of the user's closest neighbours (e.g., friends or followers). All users are then initialized with community labels and start to iteratively exchange labels based on predefined interaction rules (strategies). During the iterations, egomunities are utilized to aggregate labels propagating to each user from its contacts. Finally, communities are post-processed so that poorly connected communities are split into smaller ones.

The main advantages of the proposed method could be summarized as follows:

- a novel approach to employing egomunities for detecting global communities: egomunities help to aggregate and filter messages propagating between the nodes; as a result, the most popular community IDs (labels) within an egomunity get higher chances to be propagated;
- a method for distributed egomunities retrieval from large graphs;
- a method for identifying weakly connected communities and splitting them into denser sub-communities;
- accuracy improvement over some other popular methods for detecting highly overlapping community structure;

[1]For instance, Facebook reports 1.32 billion monthly active users, Twitter reports 271 million monthly active users, etc.

- efficient distributed implementation based on Apache Spark[2] using Pregel computational model [4];
- low computational complexity: $O(m/w)$, where m – number of graph edges, w – computation cluster size;
- near-linear scalability which allows processing graphs of up to 10^9 nodes in reasonable time.

Other contributions of the presented paper include:

- description of the discovered issues of label propagation methods for community detection;
- a novel method for assessing community structure based on a generative model which combines some important properties of social community structure.

The rest of the paper is organized as follows. Section 2 contains related work description. The problem of social community detection is defined in section 3. In section 4, we describe the proposed method step by step. Evaluation results are described and discussed in section 5. Section 6 contains performance evaluation results. We conclude in section 7 with possible future directions.

II. Related work

Below we describe various community detection methods which resemble our method in different aspects.

A. Overlapping community detection

Observing a variety of network community detection methods, one could pick four main classes of algorithms which provide a good accuracy in revealing highly overlapped community structure. These classes are model-based methods [13], [7], local optimization methods [14], [16], and label propagation methods. However, for the first two classes there appear numerous problems when it comes to scalable time-efficient implementation.

The label propagation (LP) provides a perfect combination of properties: high accuracy in both disjoint and overlapping community detection, low computational complexity, ease of implementation in terms of modern computational paradigms for distributed graph processing (Pregel [4], GraphX [28]), and good scalability.

A common and distinctive characteristic of methods in this family is the process of exchanging community labels between graph nodes which accumulate income labels and send messages to neighbour nodes to inform about updated labels collection. Most of the popular LP-based methods (including LPA [29], SLPA [5], BMLPA [6], and COPRA [30]) conform to this template and only differ in label initialization and exchanging strategies.

In order to understand the reasons of accuracy decrease in the case of highly overlapping communities [1], we've implemented and evaluated LP(k) – a label propagation algorithm with the simplest strategies of labels exchanging:

- Sender strategy: draw k labels;
- Receiver strategy: store the most frequent k labels.

Note that LP(k) is equivalent to the popular SLPA [5] algorithm if $k = 1$.

The evaluation was made by applying LP(k) to LFR benchmark networks [20] with ground-truth communities and comparing the detected communities against them. The following issues of LP(k) were found and targeted during EgoLP development:

a) Hubs effect: if there are nodes with degree much higher than average degree (>3 times), their labels lead to extra huge communities comparing to reference communities set;
b) The lack of communities at intermediate stages: for a fixed node one of the communities becomes dominating and displace the others. Besides, usually a node has more then one community;
c) Label propagation process could result in poorly connected or even disconnected communities;
d) A vertex adds too many "noise" elements to labels array. Label propagation procedure initialized by reference communities leads to continuous accuracy decrease.

Our experiments indicate that EgoLP helps to reduce the mentioned drawbacks and thus improves accuracy over LP(k), especially for highly overlapping communities. At the same time, we avoid costly computations (like rough cores enumeration in BMLPA), so the resulting complexity remains near-linear (Section IV-E).

B. Egomunity-based methods

Soundarajan et al. [25] introduced *Node Perception* - an algorithm template for merging egomunities into larger global communities. They first detect egomunities within the ego-network of each user, then build a new meta-network of the found egomunities. Finally, the meta-network is clustered in order to obtain the resulting communities. The *DEMON* algorithm by Coscia et al. [17] could be viewed as a specific instance of this template. There is, however, an important difference: egomunities are extracted not from an ego-network of each user, but from a so called *EgoMinusEgo* network (ego-network without an ego node and all its edges). This modification helps to avoid "noise" caused by the presence of a single node connected with the rest of ego-network members. The *EgoClustering* algorithm by Rees et al. [26] is also quite similar except for egomunities are found merely as disjoint components of an ego-network after removing ego with all edges. The later work of Rees et al. [27] provides another way to aggregate an individual's view of her social groups to produce communities.

Unlike the previous approaches, our method employs egomunities as a means to aggregate and filter messages propagating between the nodes. Instead of interacting with each of the neighbours, a node mostly communicates with its egomunities which in a sense shape the messages flow. As a result, the most popular community IDs (labels) within an egomunity get higher chances to be propagated.

C. Scalable methods

Scalability has become an important and desirable feature of community detection methods due to increasing populations of social services. Known implementations include: LPA using Hadoop MapReduce[3], SLPA using MPI [31], the Louvian method using Apache Giraph[4], the propinquity dynamics method using Hadoop MapReduce [32], Scalable Community Detection [33] and many others.

To the best of our knowledge, this work is the first to report an Apache Spark implementation of a community detection

[2]http://spark.apache.org/

[3]http://www.akshaybhat.com/LPMR/

[4]http://sotera.github.io/distributed-louvain-modularity/

method. Firstly, we make use of Bagel (Spark's implementation of Pregel) for thrifty and efficient implementation of the labels exchanging process. Secondly, we benefit from Resilient Distributed Datasets (RDDs) - highly optimized data structures perfectly tailored for iterative algorithms.

III. Problem

From the practical point of view, community is a cluster of social network users created on shared affiliation, role, activity, social circle, interest or function (*functional definition*) [9]. However, there is no commonly accepted *formal definition* of community. Surveys [2], [1] contain numerous definitions which could be argued with counter-examples. In this paper we treat community as a group of social network users and assume that community structure in social graph has a number of characteristic *structural* properties. We outline the most intuitive and important of them below. Our method doesn't optimize any of them explicitly, however, experiments clearly demonstrate that many of them are present in the output communities.

Let's consider a graph $G = (V, E)$, $|V| = n$, $|E| = m$. Communities set (*cover*) is defined as $\mathbf{C} = \{Z_c\}_{c=1}^{K}$, $Z_c \subseteq V$ and $|Z_c| = n_c$. It is expected that a cover has many overlapping elements. The number of entries of the j-th node into different communities is called *membership* and denoted as $m_j \geq 0$. Internal and external edge counts of the node $j \in Z_c$ are denoted as m_{cj} and $k_j - m_{cj}$ respectively, where k_j is j-th node degree.

We target the following structural properties of community structure in social graphs:

a) *Separability* captures the intuition that a good community is well-separated from the rest of the network. It could be treated as internal/external community edges ratio [9]. In some cases, however, when a node belongs to many different communities, the external edges count may exceed internal edges count. In such situation one may characterize strong (statistically significant) connection of node j and community Z_c by probability

$$p_{jc} = \mathbb{P}(\mu_{cj} \geq m_{cj}),$$

where μ_{cj} is internal edges count in some random graph model with remained node degrees or their expectations. Lower p_{jc} value means better connection.

b) *Density* means that probability of two nodes being connected increases with the number of common communities they belong to [7], [13]. The connection between i and j is usually modelled using A_{ij} – independent random variables:

$$A_{ij} \in \text{Bernoulli}(1 - \mathbb{E}(\lambda_{ij})),$$

$$\lambda_{ij} = \sum_c (z_{ic} z_{jc} \lambda_c + \varepsilon g(z_{ic}, z_{jc}, k_i, k_j, k_{ijc})),$$

where z_{ic} indicates that i is in community Z_c, k_{ijc} – common friends count inside Z_c, λ_c – model parameter associated with internal community edge probability, ε – a small parameter, A – graph adjacency matrix.

c) *Cohesiveness* characterizes the internal structure of the community. It should be relatively hard to split a community into sub-communities.

d) *Cohesion*: if an edge (i, j) is located inside community Z_c then most of the common neighbours of i and j also belong to Z_c [11];

e) Low-degree nodes tend to be part of very few communities, while high-degree nodes tend to be members of multiple groups [9];

f) Number of edges in the community increases superlinearly with the community size [9];

g) User membership has a power-law distribution [9];

h) Community size has a power-law distribution [20].

A. *Property of egomunities*

There is also another property of community structure which we hypothesized and built our method upon: the majority of nodes inside an egomunity also share one or more common global communities.

Unfortunately, we were unable to confirm this property using real-world data. However, experiments with algorithmically discovered egomunities (Table I) provided some evidence supporting our assumption. In the table, NMI_{avg} corresponds to the averaged NMI of user's egomunities and parts of global communities within user's ego-network, Fr_{avg} means average fraction of egomunity covered by any community (on condition coverage is > 0.5 or 0), and P is probability of an egomunity be entirely covered by any community. See section V for the explanation of testing methodology.

TABLE I. Egomunities and global communities correlation in LFR graphs depending on mean node membership O_m.

O_m	2	3	4	5	6
NMI_{avg}	0.78	0.50	0.57	0.32	0.41
Fr_{avg}	0.93	0.68	0.72	0.39	0.53
P	0.50	0.14	0.16	0.05	0.10

B. *Community structure likelihood*

One of the best ways to formalize community detection task is to define a probabilistic graph model. Most of the noted properties of community structure could be explained using the model described below.

Using the abovementioned model for density measuring and applying Poisson approximation we get the following distribution for A_{ij}:

$$A_{ij} = \sum_c A_{ijc} \in \text{Po}\left(\sum_c (\lambda_{1c} + \lambda_{2c} k_{ijc})\right),$$

$$A_{ijc} \in \text{Po}(\lambda_{1c} + \lambda_{2c} k_{ijc}), \quad i, j \in c,$$

$$\lambda_{1c} \in N\left(\frac{\alpha_1}{n_c^{\gamma}}, \sigma^2\right), \quad \lambda_{2c} \in N\left(\frac{\alpha_2}{n_c^{\gamma}}, \sigma^2\right),$$

where $\mathbb{P}(A_{ijc} = 1)$ measures edge (i, j) probability inside community c, k_{ijc} – common friends count of i and j inside community c normalised by average degrees of i and j, Po – Poisson distribution, N – normal distribution, A – adjacency matrix (binary or weighted), n_c – nodes count inside c. This model assumes existence of an ε-community including all graph nodes to account for edges between communities. Described edges model is quite similar to MOSES model [13] and could be referred to as a block model.

Since the distribution parameter $\lambda_{1c}+\lambda_{2c}k_{ijc}$ (related to the edge probability) depends on common friends count, the model also accounts for the cohesion property. Superlinear growth of internal community edges depending on its size is expressed in distributions of λ_{1c}, λ_{2c}. The regularized likelihood for the whole model could be defined as follows:

$$L_G(\mathbf{C}, \alpha_1, \alpha_2, \sigma, \gamma, \beta_1, \beta_2) =$$
$$= p(Z) \prod_{i<j} p(A_{ij}) \prod_{c \in \mathbf{C}} p(\lambda_{1c}) p(\lambda_{2c}) p(n_c) \prod_{v \in V} p(m_v),$$
$$p(Z) = |\mathbf{C}|! \prod_{c \in \mathbf{C}} \left(\frac{n_c}{n}\right)^{n_c} \left(1 - \frac{n_c}{n}\right)^{n-n_c},$$

where Z – user-community membership matrix (the apriori knowledge), m_v – number of communities node v belongs to $\in$ power-law distribution with β_1 parameter, n_c – community size $\in$ power-law distribution with β_2 parameter, that corresponds to the properties of n_c and m_v from Section III. The equation for $P(Z)$ comes from a previously proposed model [19] where nodes and communities are connected using a bigraph and (c, v) connection probability is proportional to $n_c m_v$.

The proposed model could be used for both detecting unknown community structure and evaluating known community structure. The goal of a community detection system is to find configuration $(\mathbf{C}, \alpha_1, \alpha_2, \sigma, \gamma, \beta_1, \beta_2)$ that maximises L_G. At the same time, the goal of a quality measurement system is to find configuration $(\alpha_1, \alpha_2, \sigma, \gamma, \ \beta_1, \beta_2)$ that maximises L_G given C. High likelihood values thus correspond to sets of communities that have a combination of the described properties. We employ the proposed model to estimate the likelihood values of different covers to understand how well these properties are expressed in the communities found by the algorithms.

Involving egomunities property (Section III-A) allows to define an addition to initial likelihood function:

$$L_{\text{ego}}(\mathbf{C}, \lambda_3) = \prod_{v \in V} \prod_{e \in \mathcal{E}(v)} \prod_{j \in e} p(j \in Z_e),$$

where $p(j \in Z_e) \in \text{Be}(\lambda_3)$ is probability that j-th node from egomunity e is inside Z_e community which best-covers e, $\mathcal{E}(v)$ – egomunities set of the v-th node.

In case graph nodes have categorical attributes, the model could be further extended with L_{atr} addition:

$$L_{\text{atr}}(\mathbf{C}, W) = \prod_{v \in V} \prod_{a \in \mathcal{A}(v)} \left(\frac{\sum_c Z_{vc} W_{ca}}{\sum_c Z_{vc}}\right),$$

where $\mathcal{A}(v)$ – attributes of user v, W_{ca} – attribute weight inside community c.

The overall likelihood for an attributed graph with communities and egomunities is

$$L = (L_G + L_{\text{ego}} + L_{\text{atr}})/n.$$

IV. Method

EgoLP has four main stages: pre-processing, egomunities retrieval, label propagation, communities post-processing. We provide the algorithm description with some remarks on our Apache Spark implementation for each of these stages. EgoLP has a number of parameters which are listed in Table II together with their optimal values according to the experiments (see section V for details).

Our design choices are largely motivated by the results of experimental evaluation of the simplest label propagation algorithm (see section II-A for details). We tried to eliminate the discovered drawbacks, improve the resulting accuracy, and at the same time not to hurt scalability or increase total computational complexity significantly.

TABLE II. **Optimal EgoLP parameters** according to the experiments using LFR benchmarks with $O_n = 0.5$ and various O_m.

O_m	**3**	**6**
Egomunities retrieval		
# iterations	15	14
min egommunity size	5	3
min egomunities intersection for union	0.8	0.6
# labels sent/received	3	5
threshold	0.06	0.08
Label propagation		
# iterations	19	25
threshold	0.1	0.06
max membership	20	10
# labels sent	10	5
# labels received	3	1
max combiner map size	40	20
Communities post-processing		
# iterations	15	15
# labels sent/received	1	3
threshold	0.15	0.15

A. Pre-processing

A certain portion of top-degree nodes (*hubs*) are removed from the graph at the very beginning. Hubs are attached to communities at the end of the label propagation phase. This heuristic simplifies big graphs processing (network load, CPU balancing, egomunities finding complexity) and reduces “hubs effect” (Section II-A).

B. Egomunities retrieval

Here we describe a distributed procedure which collects ego-networks and executes the LP(k) algorithm (section II-A) with the following parameters for egomunity detectuion: Te – iterations count, r – threshold, $minc$ – min egomunity size, $minu$ – min egomunities union relative size.

In Algorithm 1 graph nodes are processed in groups $[V_0, \ldots, V_{I-1}]$. Nodes from group V_i make three main actions. Firstly, node $v \in V_i$ sends list of friends to the neighbours with degree d times higher than degree of v and to the neighbours from V_{i+1}. Then it receives adjacent nodes intersection (responses) from the high degree neighbours and complete lists of friends from the other nodes. Finally, when all ego-edges are collected in node v, it executes local community detection algorithm, deletes egomunities with size $< minc$ and replaces the returned egomunities $\{C_i\}$ by one egomunity with all neighbours if

$$\frac{|\bigcap_i C_i|}{v.degree} < minu.$$

We assume that egomunities with size $< minc$ don’t indicate inclusion into community and small union of egomunities with size $> minc$ may mean that communities structure in the ego-network wasn’t detected properly.

Egonet edges collection is parted into I steps to prevent memory overflow, decrease Spark shuffling complexity (depends superlinearly on Spark RDD partitions count) and network load due to receiving common adjacent nodes from group with the previous index. A simple theoretical estimation shows that the lowest network load l_I is achieved when $I \in [2, 4] : l_I/l_1 \sim 0.75$.

Importantly, the retrieved ego-munities of social network users are valuable *per se*. In particular, they could be employed in any application that relies on social circles in target user's contacts (e.g., as a replacement for manual friend grouping tools in Facebook and Google+).

```
Data: Graph $(V, E)$
Parameters: $I$, $Te$, $r$, $minc$, $minu$, $k$, $d$
Result: Egomunities set $\{\mathcal{E}_{ik}\}$, $\mathcal{E}_{ik} \subset V_i$.friendList
$[V_0, \ldots, V_{I-1}]$ = split $V$ by hash function ($v$ % $I$);
$V_{-2} = V_{-1} = V_I = V_{I+1} = \emptyset$;
for $i$ = -2:$I$ do
    for $v \in V_{i+2}$ do
        for $u$ adj $v$ if u.degree > d * v.degree do
            send $v$.adjnodes to $u$;
        end
    end
    for $v \in V_i$ do
        create egonet(seed = $v$);
        findEgomunities($Te$, $r$, $minc$, $minu$, $k$);
        for $u$ adj $v$ if $u \in V_{i+1}$ do
            send common ($u$, $v$).adjnodes to $u$;
        end
    end
    for $v \in V \setminus V_i$ do
        for $u$ adj $v$ if $u \in V_{i+1}$ do
            send $v$.adjnodes to $u$;
        end
        for income message from $u$ do
            send common ($u$, $v$).adjnodes to $u$;
        end
    end
end

findEgomunities($Te$, $r$, $minc$, $minu$, $k$):
for $i$ = 1:$Te$ do
    make LP($k$) iteration;
    each node send/add $k$ labels;
end
for $v \in$ egonet do
    remove elements from $v$.labels with frequency < $r$;
end
remove egomunities with size $\leq minc$;
if $\frac{|\bigcap_i C_i|}{v.degree} < minu$ then
    egomunities = array(all egonet nodes);
end
```

Algorithm 1: Egomunities retrieval.

C. Label propagation

Next, label propagation with specific node interaction strategies is employed to uncover global community structure in the input graph:

- Sender strategy: for each adjacent edge a sender node draws ls elements from its labels collection and removes duplicates. So, a node with larger community membership sends more labels in average. Each label could be optionally assigned a weight equal to the edge weight multiplied by community size penalty (*).
- Receiver strategy: incoming labels are distributed between egomunities according to sender indexes. The most frequent lr labels per each egomunity are then appended to labels map (label $\rightarrow$ weight) of the receiver node. The map size is restricted by mx. If the network is weighted, a label frequency is computed as weights sum of edges associated with the label. Otherwise, all labels are treated equivalently.

In directed networks Sender strategy uses only outgoing edges and Receiver strategy uses only ingoing edges.

The proposed egomunity-based Receiver strategy delivers the following advantages compared to other LP-based methods:

- received labels count is estimated more precisely (partially solves lack of communities and noise labels problem);
- the most frequent labels are calculated independently in different egomunities that allows to reveal communities with different connection strength (partially solves dominating community problem).

For distributed LP process Algorithm 2 is executed. Here we also briefly describe some details of Spark workers[5] communication.

```
Data: Graph $(V, E)$, egomunities
Parameters: $T$, $ls$, $lx$, $lr$, $mx$, $T_2$, $r$
Result: Communities set $\{C_i\}$, $C_i \subset V$
for $i$ = 1:$T$ do
    for $v \in V$ do
        for $u$ adj $v$ do
            assign weights to labels (eq. *);
            draw and send $ls$ labels to $u$;
        end
    end
    for $w \in Workers$ do
        for $v_w \in w$ do
            push labels into map (label $\rightarrow$ weight);
            trim map to size $lx$;
        end
    end
    for $v \in V$ do
        for $w \in Workers$ do
            unite combined maps (label $\rightarrow$ weight);
        end
        $v$.labels += receiverStrategy(unitedMap, $lr$);
        trim $v$.labels map to size $mx$;
    end
end
add hubs to $V$;
for $i$ = 1:$T_2$ do
    send/receive iteration without egomunities (single
    egomunity is neighbours set);
end
for $v \in V$ do
    $v$.labels.group.filter$\{(l, freq) \Rightarrow freq > r\}$;
end
```

Algorithm 2: Label propagation.

Since community size distribution is power-law, labels corresponding to larger communities should have lower priority. For this purpose label weights are multiplied by penalty factor

$$\alpha k \max(k, n_c)^{-\alpha-1}, \qquad (*)$$

where n_c is community size, parameter k has optimal value in range $[30, 40]$, parameter α is optimal in range $[-0.1, 0.1]$.

[5]Worker is a Java process responsible for computation on a node of Spark cluster.

```
**Data**: Graph $(V, E)$, communities set $\{C_i\}$, $C_i \subset V$
**Parameters:** $cx$, $\lambda_x$, $k$, $T$, $r$, $minc$
**Result**: Communities set $\{S_i\}$, $\forall i\ \exists j : S_i \subset C_j$
Remove communities with size $> cx$;
for $C_i \in C$ do
    if 2-nd $min|eigenvalue\ (L)| < \lambda_x$ then
        find subcommunities by LP($k$) ($T$, $r$, $minc$);
        find connected components in subcommunities;
        remove $C_i$;
    end
end
```

Algorithm 3: Local communities post-processing.

D. Communities post-processing

Most of LP-methods don't ensure connectedness and cohesiveness of communities (Section II-A). We apply post-processing procedure to infer possible sub-communities in each community obtained at label propagation step. We exploit the known properties of Laplacian matrix for checking whether a community is a candidate for splitting:

$$L = A - diag(\vec{d}),$$

where A – adjacency matrix, $\vec{d}$ – node degrees.

The second smallest eigenvalue of L allows to estimate how well a community is connected internally: zero means presence of disjoint components while low values ($< \lambda_x$) indicate poor connectivity [34]. EgoLP tries to improve the results by applying LP(k) to such suspicious communities. The whole procedure is presented in Algorithm 3.

Note that the smallest eigenvalues could be obtained approximately by iteratively solving linear equation in $O(m_c)$ time, where m_c is edges count in the c-th community.

After local community sub-partitioning one may remove or keep the initial community or some sub-communities intersected with other global communities. For this optional stage one may use community structure likelihood (Section III-B) as a removing criterion.

E. Complexity

Here we provide algorithm complexity estimation of EgoLP composed of external memory and network operations. Summarizing all EgoLP components we obtain:

$$O\left(\frac{m}{w}\left(k_{\max}^{2-\beta} + T\right) + \frac{m}{nw} * cx * K\right).$$

Main characteristic variables used are n – graph nodes count, m – graph edges count, k_i – degree of node j, $1 < \beta < 2$ – a parameter of power-law distribution of node degrees, K – number of communities selected for splitting, cx – maximum community size, T – number of label propagation iterations, p – Spark RDD partitions count, w – Spark workers count.

The last component is relatively small, $T \in [10, 20]$, $k_{\max}^{2-\beta} \in [20, 50]$ due to hubs removing. Hence, the overall complexity of EgoLP could be approximated with $O(m/w)$.

F. Distributed implementation

The core part of EgoLP implementation is based on Pregel computation paradigm [4]. Many optimization algorithms using Pregel message passing interface suffer from synchronous labels or node state updates: a node after collecting information about neighbour nodes states could make wrong decision without taking into account neighbours strategies of states updating. One way to overcome this problem is to use damping factor λ for messages or states. Each state (message) is set to λ times its value from the previous iteration plus 1 - λ times its updated value. In EgoLP we add new labels to node labels array formed at previous iteration. So, nodes don't change their states significantly after a single iteration.

Another remark related to distributed Pregel algorithms is *messages combining*. If the Receiver strategy is an aggregate function, then the messages can be partially processed at each sender worker and final result will be reduced at target worker. This reduces network load and makes computation more balanced.

As for Spark environment configuration, the following variables are important:

- Parallelism: since distributed array is computed in partitions, their count should be set high to prevent memory lack. On the other hand, higher partitions count increases shuffling time superlinearly;
- Serializer buffer size should be set optimally: high value leads to less free memory and more work for garbage collector;
- Worker timeout and open files count should be set excessively high.

V. Experiments

We provide and discuss accuracy evaluation results of EgoLP with synthetic and real graphs using different measures. Further, we compare with other popular community detection methods.

A. Ground-truth communities

Conventionally to measure accuracy of any community detection method one makes use of graphs with *reference* cover (ground-truth communities). An algorithm returns a set of communities, whereupon its similarity with the reference partition is calculated. To estimate the affinity of two covers (X, Y), NMI measure is typically applied [20]:

$$NMI(X, Y) = 1 - \frac{1}{2}[H(X|Y)_{\text{norm}} + H(Y|X)_{\text{norm}}]$$

For each community variable X_k we are looking for the closest Y_k in sense of information lack $H(X_k|Y_j) \to \min_j$, where X_k is random variable corresponding to the occurrence probability of a vertex in community k, $H(X_k|Y_j)$ is conditional enthropy of X_k given Y_j. H_{norm} is calculated as normalization of $H(X_k|Y_k)$ on the amount of all information about X_k and averaging over all communities in X.

NMI range is $[0; 1]$, minimum value means totally different covers, maximum value means equal covers.

B. Synthetic networks

We compare EgoLP method with GCE, OSLOM, SLPA and MOSES which have the highest NMI values in most of the cases according to paper [2]. Figure 1 gives the summary of results for LFR [20] and CKB [19] benchmark networks.

Although OSLOM and MOSES produce rather good results they can't process large graphs (> 1M nodes), because processing neighbours area of each community and pairwise communities comparison requires a lot of time.

Fig. 1. Accuracy evaluation of EgoLP and other methods using synthetic graphs. **Left:** LFR benchmark with $O_n = 0.5N$. **Center:** LFR benchmark with $O_n = 0.8N$. **Right:** CKB benchmark with different values of β_1 parameter (relates to community membership power-law distribution).

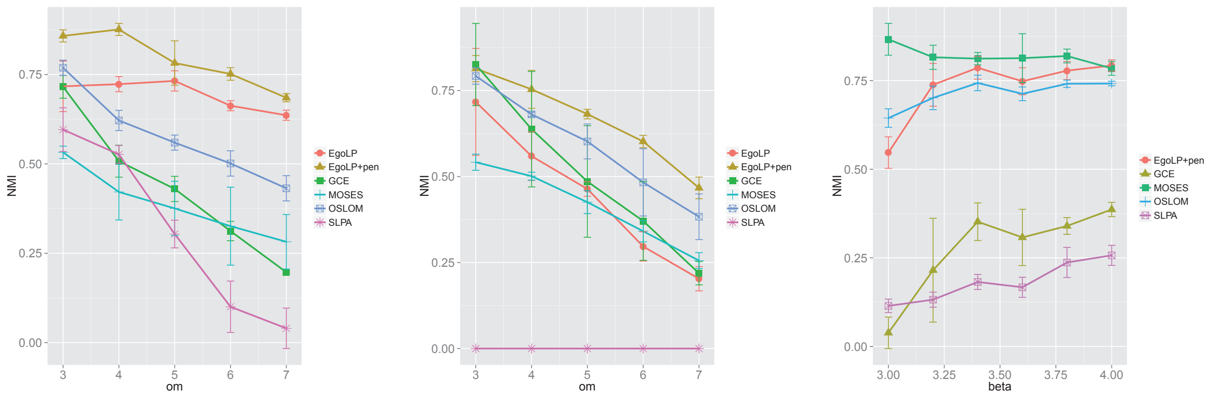
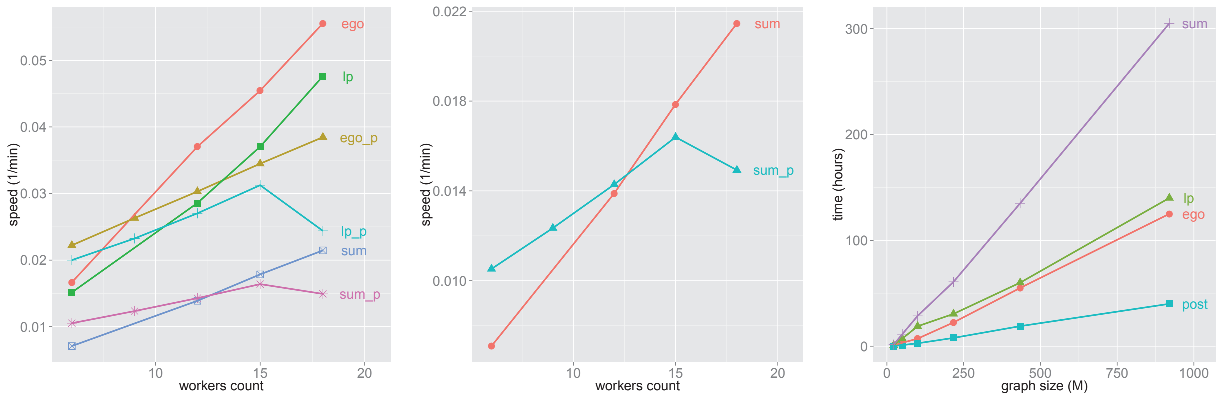


Fig. 2. Performance evaluation of EgoLP using synthetic graphs. **Left and center:** EgoLP speed (1/time) depending on Spark workers count: 'ego' – egomunities retrieval, 'lp' – label propagation, 'sum' – total speed, '[ego|lp|sum]_p' – fixed number (50) of partitions per worker. **Right:** EgoLP running time depending on graph size: 'ego' – egomunities retrieval, 'lp' – label propagation, 'post' – communities post-processing, 'sum' – total time.



C. Community structure likelihood

If a reference set of communities is not available, it is also possible to estimate quality of found communities with a likelihood function of a graph probabilistic model introduced in Section III-B.

For experiment with real social graphs we took *Facebook100* – a collection of networks of one hundred American colleges and universities [21]. All the networks are undirected, binary and attributed (student/faculty status flag, gender, major, second major/minor, dorm/house, year, high school). Likelihood measurements on *Facebook100* graphs (see Table III) characterize the score of EgoLP as close to the best. The highest average likelihood was demonstrated by the MOSES algorithm which in turn has low quality on LFR graphs and already mentioned problems with scalable implementation (Section II-A).

D. Attributes prediction

We've also evaluated the found communities by their usefulness in attributes classification task [24]. Input data comes from the Facebook100 dataset (section V-C). We've applied different algorithms to find communities, employed community labels as classification features for training and measured quality of a classifier which predicts year of graduation and dorm. In this application MOSES has the best score and EgoLP is at the second position.

TABLE III. Likelihood values of community structures extracted from Facebook100 dataset. The best values of are shown in bold, the second best values are shown in italics.

dataset	EgoLP	SLPA	GCE	OSLOM	MOSES
Caltech	*-113*	-120	-119	-116	**-106**
Cal65	*-206*	-207	-225	-217	**-177**
Lehigh	*-233*	*-233*	-259	-257	**-207**
Princeton	*-273*	-315	-311	-318	**-248**
UChicago	*-193*	-211	-221	-214	**-174**
Wellesley	*-194*	-207	-218	-209	**-176**

TABLE IV. Precision of predicting dorm and graduation year of colledge students from Facebook100 dataset. The best values are shown in bold, the second best values are shown in italics.

dataset/attribute	EgoLP	GCE	OSLOM	MOSES	SLPA
UChicago/year	*0.64*	0.56	0.62	**0.72**	0.56
UChicago/dorm	*0.57*	0.54	0.55	**0.66**	0.41
Caltech/year	*0.51*	0.44	0.42	**0.70**	0.38
Caltech/dorm	0.79	**0.85**	*0.83*	0.78	0.70
Wellesley/year	0.71	*0.75*	*0.75*	**0.86**	*0.75*
Princeton/year	*0.82*	0.77	0.78	**0.88**	0.68
Lehigh/year	*0.74*	0.71	0.71	**0.87**	0.70
Cal65/year	**0.71**	0.60	0.63	*0.70*	0.68
average	*0.69*	0.65	0.66	**0.77**	0.61

VI. Performance

For performance evaluation, we generated a set of random social-like graphs with 22, 50, 100, 217, 434 and 920 million nodes and average degree of 100. The experiment was done on a cluster with 18 workers, 100 cores total, 24 Gb RAM and $4{\times}1$ Tb hard disks on each worker. During each run, 10 iterations of label propagation were performed. Figure 2 (right) demonstrates the results.

For scalability measurements a graph with 25 million nodes and 100 average degree was considered. Experiment was done on cluster with 18, 15, 12, 9 and 6 workers (12 cores, 24 Gb RAM, $4{\times}1$ Tb hard disks on each worker). Figure 2 (left and center) illustrates results of the experiment for different stages of EgoLP. Average scalability plot coefficient $\triangle speed/\triangle workers$ is about 0.0012 for all stages line and k-times ($k = 2, 2.5, 3$ in figure 2) workers increment brings about k-times speed up. In some cases one have to keep a fixed Spark partitions per worker (for example, if workers are added synchronously as the graph size increases). Experiments with this scenario demonstrate worse scalability compared to setups with a fixed sum of partitions in all workers.

VII. Conclusion

We presented `EgoLP` – a linear time, fully distributed method for extracting implicit communities of social network users given inter-user connectivity data. It appears to be a unique representative of the label propagation family capable to detect significantly overlapped communities. Resulting covers found by our method satisfy most of the required properties of community structure. Comparison with state-of-the-art algorithms shows relatively high accuracy in experiments with synthetic graphs and real social networks. Scalability experiment demonstrates an effective speed up with workers incrementation.

Possible future directions include extending the method for attributed networks and estimating hierarchy of communities.

References

[1] J. Xie, S. Kelley, B. K. Szymanski. Overlapping community detection in networks: the state of the art and comparative study. CoRR abs/1110.5813. 2011.

[2] N. Buzun, A. Korshunov. Innovative methods and measures in overlapping community detection. Proceedings of International Workshop on Experimental Economics in Machine Learning 2012, KU-Leuven, pp. 20-32. 2012.

[3] D. A. Prentice, D. T. Miller, and J. R. Lightdale. Asymmetries in attachments to groups and to their members: Distinguishing between common-identity and common-bond groups. Personality and Social Psychology Bulletin, 20(5):484–493, 1994.

[4] G. Malewicz, M. H. Austern, A. J.C Bik, J. C. Dehnert, I. Horn, N. Leiser, G. Czajkowski. Pregel: a system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 135-146. 2010.

[5] J. Xie, B. K. Szymanski, X. Liu. SLPA: Uncovering Overlapping Communities in Social Networks via A Speaker-listener Interaction Dynamic Process. CoRR abs/1109.5720. 2011.

[6] S. Gregory, Z. Wu, Y. Lin, H. Wan, S. Tian. Balanced Multi-Label Propagation for Overlapping Community Detection in Social Networks. Springer Vol. 27. 2012.

[7] J. Yang, J. Leskovec. Community-Affiliation Graph Model for Overlapping Network Community Detection. Data Mining (ICDM), 2012 IEEE. 2012.

[8] J. Yang, J. McAuley, J. Leskovec. Community Detection in Networks with Node Attributes. ICDM'13. 2013.

[9] J. Yang and J. Leskovec. Defining and Evaluating Network Communities based on Ground-truth. ICDM, 2012.

[10] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, B. Bhattacharjee. Measurement and Analysis of Online Social Networks. 5-th ACM/USENIX Internet Measurement Conference. 2007.

[11] A. Friggeri, G. Chelius, E. Fleury. Triangles to Capture Social Cohesion. CoRR, abs/1107.3231. 2011.

[12] M. Mørup, MN. Schmidt. Bayesian community detection. Neural Comput. 2012 Sep;24(9):2434-56. 2012.

[13] A. McDaid, N. Hurley. Detecting highly overlapping communities with Model-based Overlapping Seed Expansion. ASONAM. 2010.

[14] C. Lee, F. Reid, A. McDaid, N. Hurley. Detecting highly overlapping community structure by greedy clique expansion. arXiv:1002.1827. 2010.

[15] A. Lancichinetti, F. Radicchi, J.J. Ramasco. Statistical significance of communities in networks. Phys. Rev. E 81, 046110. 2010.

[16] A. Lancichinetti, F. Radicchi, J.J. Ramasco, S. Fortunato. Finding statistically significant communities in networks. PLoS ONE 6. 2011.

[17] M. Coscia, G. Rossetti, F. Giannotti, D. Pedreschi. DEMON: a Local-First Discovery Method for Overlapping Communities. CoRR, abs/1206.0629, 2012.

[18] A. Lancichinetti, S. Fortunato. Consensus clustering in complex networks. Scientific Reports 2, 336. 2012.

[19] K. Chykhradze, A. Korshunov, N. Buzun, R. Pastukhov, N. Kuzyurin, D. Turdakov, H. Kim. Distributed generation of billion-node social graphs with overlapping community structure. In proceedings of Complex Networks V. Studies in Computational Intelligence Volume 549, 2014, pp 199-208.

[20] A. Lancichinetti, S. Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Phys. Rev. 2009.

[21] A. L. Traud, P. J. Mucha, M. A. Porter. The Social Structure of Facebook Networks. arXiv:1102.2166v1 [cs.SI]. 2005.

[22] X. Wang, J. Sun, Z. Chen, C. Zhai. Latent semantic analysis for multiple-type interrelated data objects. In Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '06). ACM, New York, NY, USA, 236-243. 2006.

[23] D. H. Stern, R. Herbrich, T. Graepel. Matchbox: large scale online bayesian recommendations. In Proceedings of the 18th international conference on World wide web (WWW '09). ACM, New York, NY, USA, 111-120. 2009.

[24] C. Lee, P. Cunningham. Benchmarking community detection methods on social media data. CoRR, abs/1302.0739. 2013.

[25] S. Soundarajan, J.E. Hopcroft. Use of Local Group Information to Identify Communities in Networks. ACM Transactions on Knowledge Discovery from Data (TKDD). 2014

[26] B. S. Rees, K. B. Gallagher. EgoClustering: Overlapping. Community. Detection. via. Merged. Friendship-Groups. - The Influence of Technology on Social Network Analysis and Mining. Lecture Notes in Social Networks, 2013.

[27] B. S. Rees, K. B. Gallagher. Detecting Overlapping Communities in Complex Networks Using Swarm Intelligence for Multi-threaded Label Propagation. Complex Networks. Studies in Computational Intelligence Volume 424, 2013, pp 111-119.

[28] R. Xin, J. Gonzalez, M. Franklin, I. Stoica. GraphX: A Resilient Distributed Graph System on Spark. GRADES (SIGMOD workshop), 2013

[29] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. Physical Review E, vol. 76, no. 3, p. 036106, 2007.

[30] S. Gregory. Finding overlapping communities in networks by label propagation. New Journal of Physics, vol. 12, no. 10, p. 103018, 2010.

[31] K. Kuzmin, S. Y. Shah, B. K. Szymanski. Parallel Overlapping Community Detection with SLPA. SocialCom, 2013

[32] Y. Zhang, J. Wang, Y. Wang, L. Zhou. Parallel Community Detection on Large Networks with Propinquity Dynamics. KDD '09

[33] A. Prat-Perez, D. Dominguez-Sal, J.-L. Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. Proceedings of the 23rd international conference on World wide web (WWW'2014).

[34] M. E. J. Newman. Spectral methods for network community detection and graph partitioning. Phys. Rev. E 88, 042822 (2013)