J Supercomput manuscript No.
(will be inserted by the editor)

Locating Communities on Graphs with variations in Community Sizes

Harris Papadakis · **Costas Panagiotakis** ·
Paraskevi Fragopoulou

Received: September 30, 2011 / Accepted: date

Abstract A fundamental problem in networking and computing is community detection. Various applications like finding web communities, uncovering the structure of social networks, or even analyzing a graph's structure to uncover Internet attacks are just some of the applications for which community detection is important. In this paper, we propose an algorithm that finds the entire community structure of a network, represented by an undirected, unweighted graph, based on local interactions between neighboring nodes and on an unsupervised centralized clustering algorithm. The novelty of the proposed approach is the fact that the algorithm is based on the use of network coordinates computed by a distributed algorithm. Experimental results and comparisons with the Lancichinetti et al. method [14, 15] are presented for a variety of benchmark graphs with known community structure, derived by varying a number of graph parameters. Emphasis is given on benchmark graphs with significant variations in the size of their communities. Further experimental results are presented for two real dataset graphs, namely the Enron, and the Epinions graphs, from SNAP, the Stanford Large Network Dataset Collection. The experimental results demonstrate

This project is implemented through the Operational Program "ARCHIMEDE III: Education and Lifelong Learning" (project P2PCOORD) and is co-financed by the European Union (European Social Fund) and Greek national funds (National Strategic Reference Framework 2007 - 2013).

H. Papadakis
Department of Applied Informatics and Multimedia, Technological Educational Institute of Crete, 71500 Heraklion, Crete, Greece
E-mail: adanar@epp.teicrete.gr

C. Panagiotakis
Department of Commerce and Marketing, Technological Educational Institute of Crete, 72200 Ierapetra, Crete, Greece
E-mail: cpanag@staff.teicrete.gr

P. Fragopoulou
Department of Applied Informatics and Multimedia, Technological Educational Institute of Crete, 71500 Heraklion, Crete, Greece
E-mail: fragopou@ics.forth.gr

the high performance of our algorithm in terms of accuracy to detect communities, and its computational efficiency.

Keywords Community detection, network coordinates, k-means.

1 Introduction

In recent years, the ubiquity of communication networks speeds up the development of Internet applications [26]. Furthermore, social networking [2] has been driving a dramatic evolution due to the increasing use of Web 2.0 elements such as blogs, twitter, Facebook, LinkedIn, wikis, etc. One of the fundamental problems in social networking with a lot of potential applications is to detect effectively the communities that are created by users' interaction.

Several attempts have been made to provide a formal definition for this generally described "community detection" concept in networks. A *strong community* was defined as a group of nodes for which each node of the community has more edges to other nodes of the same community than to nodes outside the community [8]. This is a relatively strict definition, in the sense that it does not allow for overlapping communities and creates a hierarchical community structure since the entire graph can be a community itself. A *weak community*, was later defined as a subgraph in which the sum of all node degrees within the community is larger than the sum of all node degrees toward the rest of the graph [12].

Variations also appear in the method used to identify communities: Most of the algorithms that appear in the literature follow an iterative approach starting by characterizing either the entire network, or each individual node as community, and splitting [9,19] or merging [12], respectively. These methods produce a hierarchy of partitions. There is an entire hierarchy of communities, because communities are nested: small communities compose larger ones, which in turn are put together to form even larger ones. By merging or splitting communities one can build a hierarchical tree of community partitions called *dendrogram*. The modularity criterion defined in [9] is a measure of the quality of a partition, and can be used to identify a single optimal partition, i.e. the one corresponding to the largest modularity value.

Apart from the variations in the definitions and the method employed to identify communities, there are many variation in the final desired solution. As an example, several researcher aim to find the entire hierarchical community dendrogram [9,19] while others wish to identify only the optimal community partition [8]. More recently used approaches aim to identify the community surrounding one or more seed nodes [21]. Some researchers aim at discovering distinct (non-overlapping) communities, while others allow for overlaps [15]. The variations to the problem are non-exhaustive.

Algorithms that follow a global approach, not only require the entire network in order to function, but they often require the entire network to be manipulated at each iteration [9,19]. However, real world networks, like for example Web graphs, social networks, communication and autonomous system graphs, peer-to-peer systems, blogs, collaborative networks, citation graphs, database relations, etc, count millions of nodes whose manipulation using global algorithms is prohibitive by any means.

Recently, the emerging and demanding need to analyze this type of networks led the research towards the development of local community detection algorithms [21, 14]. These algorithms are based exclusively on local interaction between pairs of neighboring nodes for the identification of the community structure, thus global knowledge of the network is not required.

In this paper, we propose an algorithm to identify the entire community structure of a network based on interactions between neighboring nodes, which is inspired by the Vivaldi synthetic network coordinate algorithm [3] and the spring relaxation metaphor. Our algorithm is based on the idea that by providing our own, appropriate, definition of distance between nodes, we can use Vivaldi's spring relaxation to position the nodes in space in such a way as to reflect community membership, i.e. nodes in the same community will be placed closer in space than nodes of different communities. In the second phase of the algorithm, *K-means* is used to identify the natural communities formed in space. Extensive experimental results on several benchmark graphs with known community structure indicate that our algorithm is highly accurate in identifying community membership of nodes and computationally efficient. The first part of the experimental phase focuses on graphs with communities of the same size, while the second phase deals with graphs that contain communities with significant variations in size. Additional experimental results are provided for two real dataset graphs, the Enron and the Epinions graphs, from SNAP, the Stanford Large Network Dataset Collection [24].

The remaining of the paper is organized as follows: Section 2 reviews some of the most important approaches to community detection found in the literature. Section 3 briefly presents the main idea behind the Vivaldi network coordinate system which constitutes the cornerstone of our community detection algorithm. The proposed community detection algorithm is presented and analyzed in Section 4. Section 5 describes the experimental framework and comparison results with another known algorithm on a number of benchmark and real dataset graphs. Finally, we conclude in Section 6 with some directions for further research.

2 Related Work

Below we review some of the known methods for community detection and give insight on the approach they follow. For the interested reader, two comprehensive and relatively recent surveys covering the latest developments in the field can be found in [14, 23]. Furthermore, a comparative analysis of some known algorithms is presented in [16]. While the first algorithms for the problem used the agglomerative approach trying to derive an optimal community partition by merging or splitting other communities, recent efforts concentrate on the derivation of algorithms based on local interaction between nodes. A community surrounding a seed node is identified by progressively adding nodes and expanding a small community. This is certainly a more flexible approach to the problem that allows adaptation in distributed environments.

One of the most known community finding algorithms was developed by Girvan and Newman [9, 19]. By gradually removing bridge edges belonging to many

shortest paths between nodes, the graph is split and its hierarchical community structure is revealed. The algorithm is computationally intensive because following the removal of an edge, the shortest paths between all pairs of nodes have to be recalculated. However, it reveals the entire hierarchical community dendrogram of the graph. An important element of the algorithm is the *modularity* which is used to assess the quality of the community partition resulting in each iteration, and as a termination criterion. The modularity essentially indicates the extent to which a given community partition is characterized by high number of intra-community edges compared to inter-community ones. In a different approach, the algorithm presented in [12], named *CiBC*, adopts the inverse approach, initially assuming each node as a different community, and iteratively merging closely connected communities. This algorithm is less intensive computationally since it starts by manipulating individual nodes rather than the entire graph.

The authors of [21] introduce a local methodology for community detection, named *Bridge Bounding*. The algorithm can identify an individual community starting its detection from a seed node and progressively expanding around it. Community expansion stops at edges characterized as bridges. An edge is a bridge if its *edge clustering coefficient* exceeds a predefined threshold. The method is local, has low complexity and allows the flexibility to detect individual communities, albeit less accurately. The entire community structure of a network can be uncovered starting the algorithms at various unassigned seed nodes, till all nodes have been assigned to a community.

An exceptionally interesting method for community detection appears in [15]. Although most previous approaches identify distinct (non-overlapping) communities, this algorithm is developed based on the observation that network communities may have overlaps, and, thus, algorithms should allow for the identification of overlapping communities. Based on this principle, a local algorithm is devised developing a community from a starting node and expanding around it based on a *fitness* measure. This fitness function depends on the number of inter- and intra-community edges and a tunable parameter α. Starting at a node, at each iteration, the community is either expanded by a neighboring node that increases the community fitness, or shrinks by omitting a previously included node, if this action results in higher fitness for the resulting community. The algorithm stops when the insertion of any neighboring node would lower the fitness of the community. This algorithm is local, and able to identify individual communities. The entire overlapping and hierarchical structure of complex networks can be found by initiating the algorithm at various unassigned nodes. The algorithm has low complexity and is highly flexible and efficient in detecting communities.

Other community finding methods of interest involve [8] in which the problem is regarded as a maximum flow problem and edges of maximum flow are identified to separate communities from the rest of the graph. In clique percolation [6, 20] a complete subgraph of k nodes (*k-clique*) is rolled over the network through other cliques with $k-1$ common nodes. This way a set of nodes can be reached, which is identified as a community. A method based on voltage drops across networks and the physics kirchhoff equations is presented in [27]. A mathematical Markov stochastic

flow formulation method known as *MCL* is presented [25], and a local community finding method in described in [1].

3 Synthetic Network coordinates

Network coordinate systems were designed to predict latencies between network nodes, without the need for explicit measurements using probe queries. These algorithms assign synthetic coordinates to hosts, so that the distance between two hosts' coordinates provides an accurate latency prediction between them. This technique provides to applications the ability to predict round trip time with less measurement overhead than probing. Vivaldi is a fully decentralized, light-weight, adaptive network coordinate algorithm that predicts Internet latencies with low error. Vivaldi uses the Euclidian coordinate system (in *n-dimensional* space, where *n* is a parameter) and the associated distance function. Conceptually, Vivaldi simulates a network of physical springs, placing imaginary springs between pairs of network hosts.

Let $G = (V, E)$ denote the given graph comprising a set V of nodes together with a set E of edges. Each node $x \in V$ participating in Vivaldi maintains its own coordinates $p(x) \in \Re^n$ (the position of node x that is a point in the *n-dimensional* space). Initially, all node coordinates are set at the origin. Periodically, each node communicates with another node (selected among a small set of nodes known to it). Each time a node communicates with another node, it measures its distance and learns that node's coordinates. Subsequently, the node allows itself to be moved a little by the corresponding imaginary spring connecting them.

When Vivaldi converges, any two nodes' Euclidian distance will match their actual distance, even though those nodes may never had any communication. When node x with coordinates $p(x)$ learns about node y with coordinates $p(y)$ and measured distance $dist$, it updates its coordinates using the following update rule:

$$p(x) = p(x) + \delta \cdot (dist - ||p(x) - p(y)||) \cdot u(p(x) - p(y)) \quad (1)$$

where $||.||$ denotes the Euclidean norm operation and $u(p(x) - p(y))$ denotes the unit vector that has the same direction with $p(x) - p(y)$. Finally, δ can be viewed as the fraction of the way the node is allowed to move towards the perfect position for the current sample, and is updated during the process. The estimation of δ is analytically described in Section 2.5 of [3]. Equation (1) is identical to the individual forces of a spring from a straightforward application of Hook's law. Unlike other centralized network coordinate approaches, in Vivaldi each node only maintains knowledge for a handful of other nodes. Each node computes and continuously adjusts its coordinates based on its measured distance to a handful of other nodes.

4 Local community finding

The proposed local community finding algorithm comprises the following steps:

– The position estimation algorithm, which is a distributed algorithm inspired by Vivaldi [3].

– The community detection algorithm using K-means clustering.
– The finding of the number of communities.

Hereafter, we describe the main steps of the algorithm.

4.1 The position estimation algorithm

In the core of our proposal lies the spring metaphor which inspired the Vivaldi algorithm. As we mentioned, Vivaldi uses spring relaxation to position the nodes in a virtual space (the *n*-dimensional Euclidean space), so as the Euclidean distance of any two nodes approximates the actual distance between those nodes. In the original application of Vivaldi, the actual distances were the latencies between Internet hosts. Our algorithm is based on the idea that by providing to the algorithm with our own, appropriate, definition of distance between nodes, we can use spring relaxation to position the nodes in the *n*-dimensional space in such a manner as to reflect community membership, i.e. nodes in the same community will be placed closer in space than nodes of different communities. In other words nodes belonging to the same community will form natural clusters in space.

Let $C(x)$, $C(y)$ denote the communities' sets of two nodes $x,y \in V$, respectively, of a given graph. Since two nodes either belong to the same community ($C(x) = C(y)$) or not, ideally we should define the initial distance between two nodes x and y as:

$$d(x,y) = \begin{cases} 1, & C(x) = C(y) \\ 100, & C(x) \neq C(y) \end{cases} \tag{2}$$

Given this definition of distance, we can employ Vivaldi's spring relaxation to position the nodes appropriately in the *n*-dimensional Euclidian space ($\Re^n$). As one can expect, when Vivaldi converges nodes of the same community will be close-by in space, while nodes of different communities will be away from each other. This is the reason for the dual nature of the distance function in equation (2), otherwise all nodes, regardless of community membership would gravitate to the same point in space.

Vivaldi requires each node to select a set of other nodes to probe. Each node x calculates a "local" set $L(x)$ containing nodes of the same community, and a "foreign" set $F(x)$ containing nodes of different communities. The size of the local set as well as the size of the foreign set of a node equals the degree of the node. However, the perfect construction of these sets depends on the apriori knowledge of node community membership, which is the actual problem we are trying to solve. Even though we do not know the community each node belongs to, there are two facts we can exploit to make Vivaldi work without this apriori knowledge:

– The first fact is based on the definition of a community, according to which most of a node's neighbors belong to the same community as the node itself. This means that, if we consider all of a node's neighbors as "local" links belonging to the same community as the node itself, this assumption will be, *most of the time*, correct, which in turn means that even though some times the node may move to the wrong direction, most of the time it will move to the right direction and thus,

will eventually acquire an appropriate position in space. Thus, we let the local set $L(x)$ of a node $x \in V$, be its "neighbor set" ($L(x) = \{y \in V : x \sim y\}$). The distance from node x to nodes in $L(x)$ is set to 1 according to Equation (2).

– The second fact we exploit concerns the "foreign" set. Since we consider all of a node's links as local links, we need to find some nodes which most likely do not belong to the same community as the node itself, and therefor will be considered as foreign nodes. This can simply be done by randomly selecting a small number of nodes from the entire graph. Assuming that the number of communities in the graph is at least three, the majority of the nodes in this set will belong to a different community than the node itself. These nodes will comprise the "foreign set" $F(x)$ of node $x \in V$: $F(x) \subset \{y \in V : x \not\sim y\}$. The distance from node x to the nodes in $F(x)$ is set to 100 according to Equation (2).

The algorithm proceeds as follows: Initially each node is placed at a random position in $\Re^n$. Iteratively, each node $x \in V$ randomly selects a node y, either from $L(x)$ or $F(x)$ (see Algorithm 1). Having defined the two sets $L(x)$ and $F(x)$ for each node x and randomly selecting, at each iteration, a node y either from set $L(x)$ or from $F(x)$, we use the Vivaldi algorithm with parameters the current position of node x ($p(x)$), the current position of node y ($p(y)$) and their corresponding distance $d(x,y)$ to acquire a new position estimate for node x after Vivaldi's spring relaxation. Therefore, first we randomly select the set $L(x)$ or $F(x)$ from which y will be sampled using the procedure *randomSelection* (see line 1 of Algorithm 1). The procedure *randomSelection* gets as input a set and randomly selects a sample from this set. Next, we randomly select a node from the selected set ($L(x)$ or $F(x)$) (see lines 2-6 of Algorithm 1). Finally, the Vivaldi algorithm is executed (see line 7 of the Algorithm 1).

Algorithm 1: Local community finding

input : $x \in V, p(.), d(.,.)$

1 $r = randomSelection(\{0,1\})$
2 **if** $r = 0$ **then**
3 $\quad y = randomSelection(L(x))$
4 **else**
5 $\quad y = randomSelection(F(x))$
6 **end**
7 $p(x) = Vivaldi(p(x), p(y), d(x,y))$

The iterations stop when the positions of all nodes have converged. Each node continues this process until it deems its position to have stabilized as much as possible. This is done by maintaining a history of the most recent changes in the node's position, in the form of distances. When the history size reaches 110 entries, 5% of the highest and lowest history values are discarded and a stability metric is calculated. Then the 50 oldest history entries are discarded. A new value of the stability metric is calculated when the history size reaches again 110 entries. If the value of the stability metric remains below a certain threshold for 20 consecutive iterations of the algorithm, the node considers its position stable and stops updating it. The stability

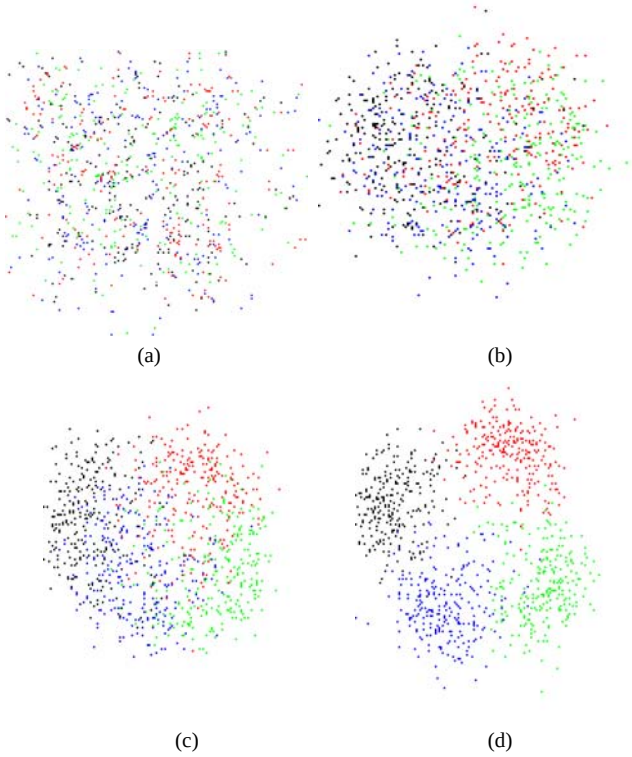


Fig. 1 Snapshots of the execution of the first phase of our algorithm for a graph with known community structure: **(a)** initialization, **(b)** after 50 iterations, **(c)** after 150 iterations, **(d)** after 400 iterations.

metric is calculated using the standard deviation of the history values, divided by the average value. The threshold value used is 1.3.

Fig. 1 shows a small time-line of the execution of our algorithm on a graph with 1024 nodes, degree 20, and a known community structure comprising four communities. We have used different colors for the nodes of each different community. Initially, nodes were randomly placed in $\Re^2$. As we can see, in the beginning all colors are dispersed on the entire space. As the algorithm progresses, we see that nodes of the same color, belonging to the same community, gradually gravitate to the same area, forming distinct clusters in space.

4.2 Community detection using K-means clustering

Having estimated a synthetic coordinate (n-dimensional position) for each node of the graph, we can use a clustering algorithm in order to identify the communities, since the nodes that belong to the same community should have been placed in proximity in $\Re^n$. To achieve this we use the K-means clustering algorithm [18,11], one

of the simplest unsupervised learning algorithms that solves the well known clustering problem [7,28,29], under the assumption that the number of clusters (number of communities) K is provided. The K-means algorithm has been successfully used in many clustering applications in networks [4,17]. In our case, K-means takes as input the positions of nodes produced by the Vivaldi algorithm and outputs a community membership for each nodes. In the remaining of the paper the terms community and cluster are used interchangeably.

K-means is a centralized clustering algorithm. However, its linear computation cost $O(KNn)$, where N denotes the number of graph nodes, makes it a good candidate even for large graphs. The main idea of K-means is to define K centroids $C_i \in \Re^n, i \in \{1,...,K\}$, one for each cluster $S_i, i \in \{1,...,K\}$. These centroids will eventually be placed at the centers of the natural clusters. The goal of K-means is to minimize the sum, over all clusters, of the within-cluster sums of node-to-cluster-centroid distances. Initially, these centroids can be randomly set. However, for our data we know that the clusters are far apart from each other due to Vivaldi and Equation (2). Therefore, we get better results when we take this property into account, selecting the initial centroids to be as far apart from each other as possible. This selection is given by the KKZ algorithm [13] in $O(KNn)$ time. According to the KKZ method, the first centroid is given as the data point with maximum norm, and the second centroid is the point farthest from the first centroid, the third centroid is the point farthest from its closest existing centroid and so on.

In each iteration of K-means the following tasks are performed, each node is assigned to the cluster with the closest centroid: subsequently the means of the resulting clusters are calculated and become the new cluster centroids.

- each node is assigned to the cluster with the closest centroid (according to the Euclidean distance, see Equation 3):
- subsequently the means of the resulting clusters are calculated and become the new cluster centroids (see Equation 4).

The final clustering is provided by assigning each node to the cluster with the closest centroid.

$$S_i = \{x \in V : ||C_i - p(x)|| \leq ||C_j - p(x)||, \forall j \in \{1,...,K\}\} \tag{3}$$

$$C_i = \frac{1}{|S_i|} \sum_{x \in S_i} p(x), \tag{4}$$

where $|S_i|$ denotes the number of nodes of cluster S_i.

It makes sense to apply K-means clustering to our data, since according to Vivaldi the nodes of the same cluster (community) are placed close to each other in the Euclidean space while nodes of different communities place themselves away from each other. Moreover, the Euclidean space that is used by K-means to compute the clusters has been also used by Vivaldi to produce the position estimates of the nodes.

4.3 Finding of the number of Communities

The selection of the number of communities could be done by the user to fit his/her specific preferences and information needs. However, it is crucial to develop a method that automatically estimates the most appropriate number of communities. This section describes a method for the automatic calculation of the number of communities.

Having estimated a synthetic coordinate (position) for each node of the graph, we execute the proposed K-means clustering algorithm [18] described in the previous section for different numbers of communities getting different clustering results. Next, we use the criterion described in [22] in order to find the most appropriate number of communities. This criterion is based on the compactness of clusters produced by K-means. Using the distance of each node to its respective cluster center, the compactness of each cluster is determined. According to this criterion, we select the number of communities that minimizes the validity ($validity$) measure. Validity is defined as the ration of intra-cluster ($intra$) to inter-cluster ($inter$) distance:

$$intra = \frac{1}{N}\sum_{i=1}^{K}\sum_{x\in S_i}||p(x)-C_i|| \tag{5}$$

$$inter = min_{i,j\in\{1,\ldots,K\}\wedge i<j}(||C_i-C_j||) \tag{6}$$

$$validity = \frac{intra}{inter} \tag{7}$$

- Intra-cluster distance is defined as the average of the distances between each node and its cluster center.
- Inter-cluster distance is given by the minimum of distances between cluster centers.

Apparently, we wish to minimize the intra-cluster distance and to maximize inter-cluster distance. Therefore, we select the clustering which gives a minimum value for the validity measure.

In order to measure the performance of the automatic selection of the number of communities, for each benchmark graph, we have executed the method as described in the "Experimental results" section of [10], searching for the most appropriate number of communities. The experimental results show that the probability of selecting the correct number of communities is related to the accuracy of the community detection algorithm. It holds that when the accuracy is greater than 90%, then the probability of selecting the correct number of communities is 87%. When the accuracy is between 80% and 90%, then the probability of selecting the correct number of communities drops to 51%. The algorithm fails when the accuracy is less than 65%.

5 Experimental results

In this section we present extensive experimental results on several benchmark graphs with known community structure which demonstrate the accuracy in identifying community membership and the computational efficiency of our algorithm. Benchmark graphs are essential for the testing of community detection algorithms since there is

an apriori knowledge of the structure of the graph and thus one is able to accurately ascertain the accuracy of the algorithm. Since there is no consensus on the definition of a community, using a real-world graph makes it more difficult to assess the accuracy of a community partition.

The first part of the experimental phase focuses on graphs with communities of the same size, while the second phase deals with graphs that contain communities with significant variations in size. Our benchmark graphs were generated randomly given the following set of parameters:

- The number of nodes N of the graph.
- The number of communities *Comm* of the graph.
- The ratio of local links (links to nodes of the same community, since the community structure of benchmark graphs is known) to node degree $local/degree$.
- The degree of nodes *degree*.

Our algorithm is experimentally compared to an efficient algorithm found in the literature namely the Lancichinetti et al. algorithmLancichinetti2 described in the following subsection.

5.1 The Lancichinetti et al. algorithm

An interesting method for community detection appears in [15]. A local algorithm is devised developing a community from a seed node and expanding around it. A community is identified as a subgraph that has a certain *fitness*. The authors provide an appropriate fitness function, whose calculation is based on the number of inter- and intra-community edges and a tunable parameter a. Starting at a node, at each iteration, the community is either expanded by a neighboring node that increases the community fitness, or shrinks by omitting a node if this action results in higher fitness for the community. The algorithm stops when the insertion of any neighboring node would lower the fitness of the community. This algorithm is local, and able to identify individual communities. The entire overlapping and hierarchical structure of a network can also be found.

For a community S of the graph, the fitness f_S is calculated as follows:

$$f_S = \frac{K_{in}^S}{(K_{in}^S + K_{out}^S)^\alpha} \tag{8}$$

where K_{in}^S and K_{out}^S refer to the total inter-community and intra-community edges of community S respectively, and α is a positive real-valued parameter which controls the size of the community. A larger value of α results to larger communities. A common value used is $\alpha = 1$.

The process of revealing the natural community of node is described below:

1. a loop is performed over all neighboring nodes of S not included in S;
2. the neighbor with the largest fitness is added to S, yielding a larger subgraph S';
3. the fitness of each node of S' is recalculated;

4. if a node turns out to have negative fitness, it is removed from S', yielding a new subgraph S'';
5. if (4) occurs, repeat from (3), otherwise repeat from (1) for subgraph S''.

The process terminates when there is no other candidate node with positive fitness for inclusion in the community.

In order to reveal the entire community structure of a network, each node should belong to at least one community. To achieve this goal we apply the process summarized below:

1. Select at random node A.
2. Discover the natural community of node A.
3. Randomly select a node B that has not been assigned to a community.
4. Discover the natural community of B, by exploring all the candidate nodes regardless of whether they belong to other communities (allow for overlaps).
5. Repeat from step (3).

5.2 Benchmark graphs with communities of equal size

We used the parameters values shown in Table 1 to create a set of benchmark graphs with communities of equal size.

Table 1 Benchmark graph parameters (for graph with equal size communities).

N:	1024, 4096
Comm:	2, 4, 8, 16, 32, 64
local/degree:	0.55, 0.65, 0.75, 0.85
degree:	5, 10, 20, 30, 40

The total number of nodes is divided among the various communities and then the wiring of the links is performed. The local links are selected randomly among the nodes of the same community (excluding, of course, nodes which are already neighbors, nodes with complete neighbor set and the connecting node itself). The foreign links are selected randomly among all nodes in the graph, excluding nodes in the same community as the connecting node. Concerning the dimensions of the Euclidian space, we have used $\Re^{20}$ getting high performance results. Lower values yielded worse results while higher values did not achieve better results.

In many cases, the combination of a given node degree with a local to degree ratio produced a real (float) number of local and foreign links. In these cases, the decimal value is thought of as the probability of a node having an additional link of the same type. For instance, a value of 4.3 for the local links means that 70% of the nodes have 4 local links and 30% of the nodes have 5 local links. These parameters resulted in a total number of 200 graphs, and corresponding experiments, which we present below.

5.3 Benchmark graphs with variable community sizes

We generated several synthetic benchmark graphs, with communities of variable sizes to test and compare our algorithm. In order to make graph creation more challenging, we used a power-law distribution to generate both the sizes of the communities in each graph, as well as the degree of each node of the graph. Namely, we used a Pareto distribution. For the sizes of the communities, we used a discrete Pareto distribution with a low value of one-third of the average community size (i.e. Number of nodes / Number of communities) and a high value equal to the size of the entire graph. We then calculated an appropriate slope value so as the mean value of the distribution corresponds to the average community size. Similarly, for the generation of the various node degree values of a graph, we used a discrete Pareto distribution with a low value equal to one-third of the average degree and a high value of three times the average degree. Again, the slope value was calculated in order for the mean to equal the average total degree value. Finally, all intra- and inter- community links were created in a random fashion. For instance, an intra-community link was created by uniformly selecting two nodes of the same community (given that neither node had already reached its respective short links number).

Table 2 Benchmark graph parameters (for graphs with variable size communities).

N:	1000, 5000
Comm:	5, 10, 20, 40, 80
local/degree:	0.55, 0.65, 0.75, 0.85
***degree* (average)**:	10, 20, 40, 80

The parameters used to create those graphs can be seen in Table 2. Note that not all parameter combinations make sense and thus some were omitted. For instance, one cannot create a graph of 1000 nodes with 80 communities (i.e. 12.5 nodes per community on average) and require 20 intra-community links per node.

5.4 Community detection results

In this subsection, the experimental results of the proposed algorithm are presented on the benchmark graphs. In addition, we compare the accuracy of the proposed scheme to an efficient distributed community detection algorithm found in the literature, namely the Lancichinetti algorithm [15].

In order to measure the performance of the community detection algorithms, the measure of *accuracy* has been used. Let $S_i, i \in \{1,...,Comm\}$ be the estimated and $\hat{S}_i, i \in \{1,...,Comm\}$ the corresponding actual communities. The accuracy *acc* is given by the average (over all communities) of the number of nodes that belong to the intersection of $S_i \cap \hat{S}_i$ divided by the number of nodes that belongs to the union $S_i \cup \hat{S}_i$.

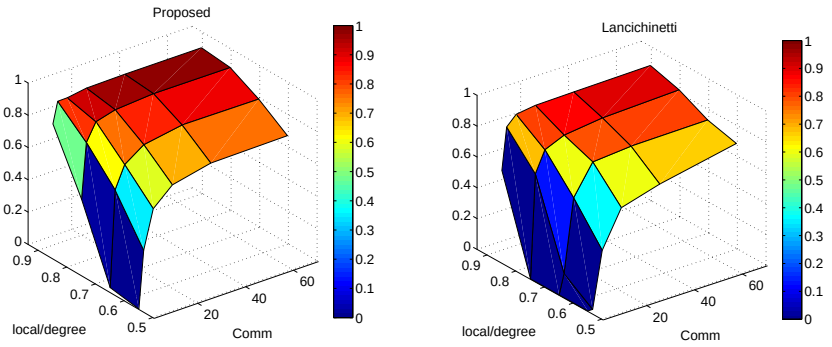
$$acc = \frac{1}{Comm} \cdot \sum_{i=1}^{Comm} \frac{|S_i \cap \hat{S}_i|}{|S_i \cup \hat{S}_i|} \tag{9}$$

It holds that $acc \in [0,1]$, the higher the accuracy the better the results. When $acc = 1$ the community detection algorithm gives perfect results. In our experimental results, we used a modified accuracy metric acc_m based on the accuracy measured in the experiment acc and the accuracy obtained by randomly selecting nodes (that is $\frac{1}{Comm}$) in the community:

$$acc_m = \frac{acc - \frac{1}{Comm}}{1 - \frac{1}{Comm}} \tag{10}$$

This means that the modified accuracy of an experiment with the same initial accuracy as a random selection of nodes for each community is zero. The $density$ which is defined in Equation 11, measures how dense a community is.

$$density = \frac{|V| \cdot degree}{|V|(|V|-1)} = \frac{local}{\frac{N}{Comm} - 1} \tag{11}$$



(a) (b)

Fig. 2 The mean value of accuracy under different ratios of $local/degree$ and $Comm$ for **(a)** the proposed and **(b)** Lancichinetti algorithms.

We have performed experiments on all benchmark graphs using several values of the α parameter of the Lancichinetti algorithm. In the results, we have used the optimal value for our test graphs (i.e. the value of α that provided the best results). It should be noted however, that the Lancichinetti algorithm does not provide a way to calculate this value without a priori knowledge of the correct results.

Since we could not present the experimental results against all four parameter values of a single experiment, the accuracy is plotted against one or two of those parameters, averaging the actual accuracy values with the same pair of parameter values. The low dispersion value among each averaged set of values indicates that the selected pair of parameters is the one that mostly affects the algorithm's performance. These pairs are the "local/degree" and the "Comm" parameters for our algorithm and"local/degree" and the graph "density" for the Lancichinetti algorithm.

Figs. 2(a) and 2(b) illustrate the mean value of accuracy under different values of local to degree ($local/degree$) ratio and number of communities ($Comm$) for the

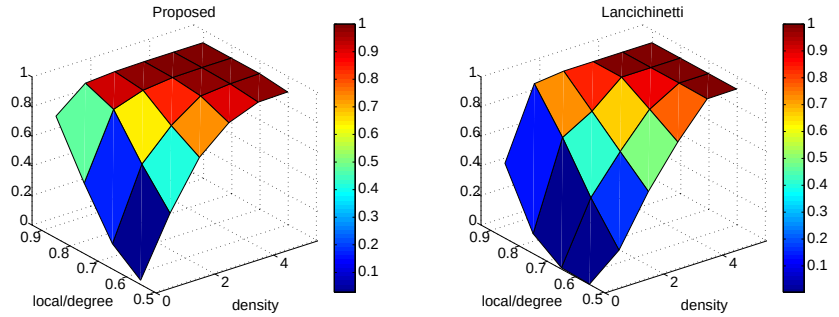


Fig. 3 The mean value of accuracy under different ratios of $local/degree$ and densities for **(a)** the proposed and **(b)** Lancichinetti algorithms.

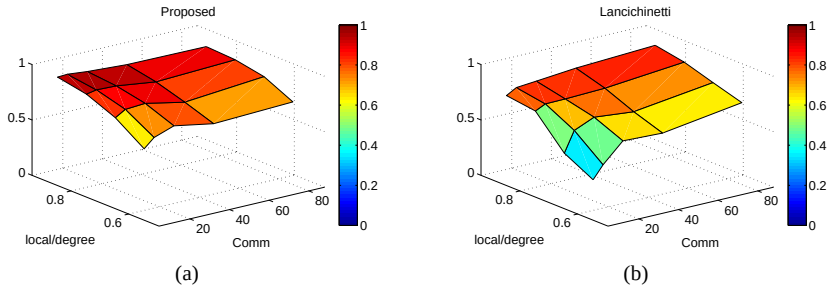
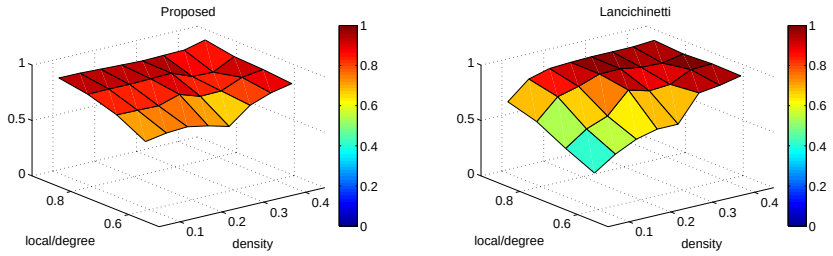


Fig. 4 The mean value of accuracy under different ratios of $local/degree$ and $Comm$ for **(a)** the proposed and **(b)** Lancichinetti algorithms.

proposed and Lancichinetti algorithms in graphs with communities of equal size, respectively. Figs. 3(a) and 3(b) illustrate the mean value of accuracy under different values of local to degree ($local/degree$) ratio and density for the proposed and Lancichinetti algorithms, respectively. Figs. 4(a) and 4(b) illustrate the mean value of accuracy under different values of local to degree ($local/degree$) ratio and number of communities ($Comm$) for the proposed and Lancichinetti algorithms in graphs with variable community sizes, respectively. Finally, Figs. 5(a) and 5(b) illustrate the mean value of accuracy under different values of local to degree ($local/degree$) ratio and density for the proposed and Lancichinetti algorithms in graphs with variable community sizes, respectively.

According to these figures, it holds that the proposed method outperforms the Lancichinetti, since the accuracy of the proposed method is higher than the corresponding accuracy of Lancichinetti under any case. Over all 200 experiments of graphs with communities of equal size, our algorithm achieved an average accuracy of 72% whereas the Lancichinetti's algorithm respective value was 58%. Over all 128 graphs with variable community sizes, our algorithm achieved an average accuracy of 85.9% whereas the Lancichinetti's algorithm respective value was 72.03%. The rea-



(a) (b)

Fig. 5 The mean value of accuracy under different ratios of $local/degree$ and densities for **(a)** the proposed and **(b)** Lancichinetti algorithms.

son for this is that in cases of less dense communities (large community population and/or small degree) it is more difficult for Lancichinetti to locate triangles of nodes (circles of three edges), the existence of which facilitate its operation. Our algorithm is based on the cooperation of all nodes in the community (pulling each other to the right direction) and thus is not depended on the existence of triangles.

Fig. 6(a) illustrates the computation cost (in seconds) under different degrees for the proposed and Lancichinetti algorithms. Our algorithm's running time only depends on the size of the graph. This is because the number of updates per node is irrelevant to the size of the foreign and local sets. Larger foreign and local sets simply mean that the algorithm has more nodes to choose from when updating. Finally, on average, our algorithm requires 6.5 seconds to find all communities, while Lancichinetti's algorithm requires 41 seconds. Fig. 6(b) displays a different success metric. Given a modified accuracy threshold, we show the ratio of all 200 experiments with modified accuracy higher than the threshold. The difference between the two methods is more apparent in more difficult test cases (where obtaining a high accuracy value is more difficult) where Lancichinetti fails more often.

Fig. 6(c) illustrates the modified accuracy in each of the experiments sorted in ascending order by the Lancichinetti algorithm modified accuracy in graphs with variable community sizes. Similarly to Fig. 6(b), Fig. 6(d) displays the ratio of all 128 graphs with variable community sizes with modified accuracy higher than the threshold. In both Figures, it is evident that the proposed method clearly outperforms Lancichinetti's algorithm.

5.5 Experiments on real dataset graphs

In this section we try to assess the effectiveness of our algorithm, as opposed to Lancichinetti's et al. [15] trying to identify communities in two graphs based on real datasets from the Stanford Large Network Dataset Collection. The first one is a communications network graph, namely Enron which counts 36,692 nodes and 367,662 edges, and represents the email communications network of Enron. The second one, named Epinions, is a social network graph which counts 75,879 nodes and 508,837 edges and represents the Who-trusts-whom network of Epinions.com.

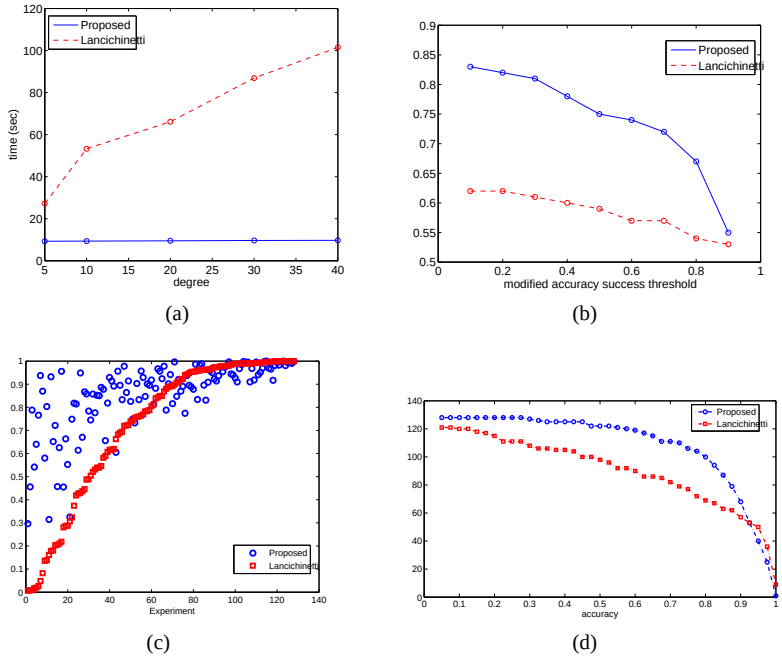


Fig. 6 **(a)** Time required for the detection of all communities, in graphs of 4096 nodes and all degree values in graphs with communities of equal size. **(b)** Percentage of experiments with modified accuracy over a threshold in graphs with communities of equal size. **(c)** The modified accuracy in each of the experiments sorted in ascending order by the Lancichinetti algorithm modified accuracy in graphs with variable community sizes. **(d)** Percentage of experiments with modified accuracy over a threshold in graphs with variable community sizes.

The main conclusion drawn from Fig 7 is the fact that our algorithm locates denser communities than Lancichinetti's et al. since the short/total average values are consistently higher for our algorithm. In addition, one can see that although the density of the communities does seem to vary, there is no apparent correlation between density and community size.

From Fig 8, we deduce that our algorithm appears to locate communities of somewhat larger sizes. In general, this is a positive sign in an algorithm's behavior since it is a usual problem in the operation of community detection algorithms to degenerate into locating very small communities of one or two nodes.

6 Conclusions

We presented a community finding algorithm which is based on the Vivaldi network coordinate system. The proposed algorithm has been tested on a large number of benchmark graphs with known community structure, and two real dataset graphs, comparing it with the algorithm found in [15] and proving its effectiveness and

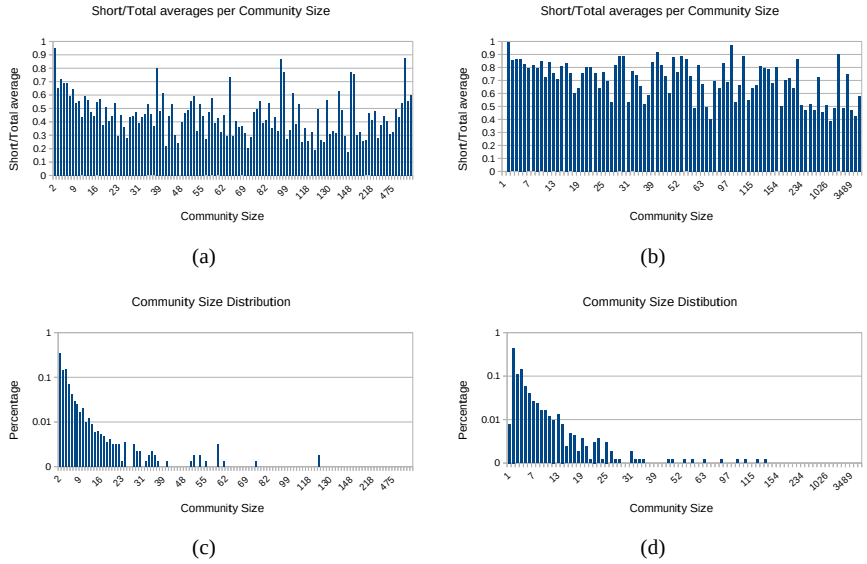


Fig. 7 Enron graph: (a),(b) Short/total average per community size for Lancichinetti's and the proposed algorithm, respectively. (c),(d) Community size distribution for Lancichinetti's and the proposed algorithm, respectively.

computational cost. An improvement would be to intelligently recalculate the local and foreign sets during the execution of the algorithm, in light of the partial results achieved, fact that would allow the algorithm to converge earlier. Furthermore, an interesting direction would be to use the distributed K-means [5] in order to have a fully distributed system. It would be beneficial to test the algorithm on more real-world graphs such as web graphs, social network graphs and more.

References

1. J. P. Bagrow and E. M. Bollt. Local method for detecting communities. *Physical Review E*, 72(4):46–108, Oct 2005.
2. B. Buter, N. Dijkshoorn, D. Modolo, Q. Nguyen, S. van Noort, B. van de Poel, A. Ali, and A. Salah1. Explorative visualization and analysis of a social network for arts: The case of deviantart. *Journal of Convergence Volume*, 2(1), 2011.
3. F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceedings of the ACM SIGCOMM '04 Conference*, August 2004.
4. S. Datta, C. R. Giannella, and H. Kargupta. K-means clustering over a large, dynamic network. *Proc. SIAM Int'l Conf. Data Mining*, pages 153–164, 2006.
5. S. Datta, C. R. Giannella, and H. Kargupta. Approximate distributed k-means clustering over a peer-to-peer network. *IEEE Transactions on Knowledge and Data Engineering*, 21:1372–1388, 2009.
6. I. Derényi, G. Palla, and T. Vicsek. Clique Percolation in Random Networks. *Physical Review Letters*, 94(16):160–202, Apr 2005.
7. A. Dutta, I. Ghosh, and D. Mukhopadhyay. An advanced partitioning approach of web page clustering utilizing content & link structure. *Journal of Convergence Information Technology*, 4(3):65–71, 2009.
8. G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35:66–71, March 2002.

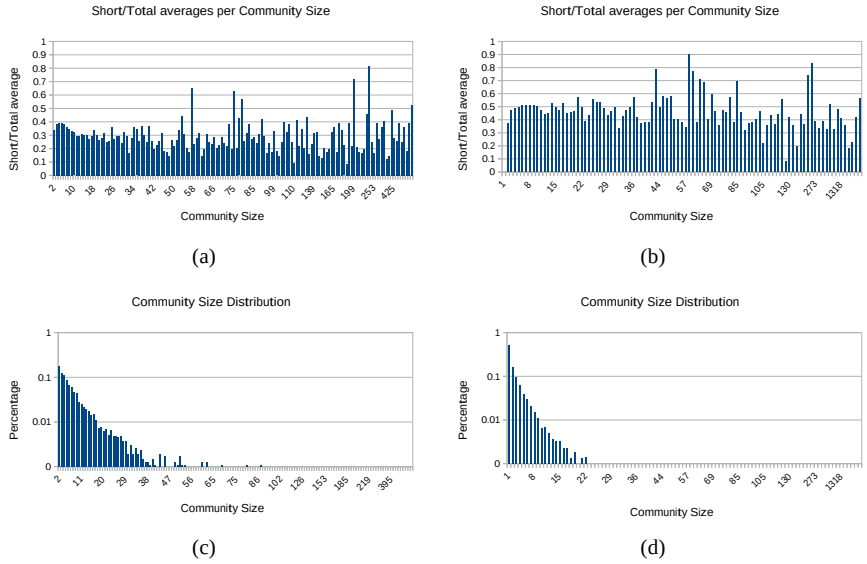


Fig. 8 Epinions graph: (a),(b) Short/total average per community size for Lancichinetti's and the proposed algorithm, respectively. (c),(d) Community size distribution for Lancichinetti's and the proposed algorithm, respectively.

9. M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, June 2002.
10. P. F. Harris Papadakis and C. Panagiotakis. Distributed community detection: Finding neighborhoods in a complex world using synthetic coordinates. In *IEEE symposium on Computers and Communications*, pages 1145–1150, 2011.
11. T. Jo. Inverted index based modified version of k-means algorithm for text clustering. *Journal of Information Processing Systems*, 4(2), 2008.
12. D. Katsaros, G. Pallis, K. Stamos, A. Vakali, A. Sidiropoulos, and Y. Manolopoulos. Cdns content outsourcing via generalized communities. *IEEE Transactions on Knowledge and Data Engineering*, 21:137–151, 2009.
13. I. Katsavounidis, C.-C. J. Kuo, and Z. Zhang. A new initialization technique for generalized lloyd iteration. *IEEE Signal Processing Letters*, 1(10):144–146, 1994.
14. A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis. *Physical Review E*, 80(5 Pt 2):056117, Sep 2009.
15. A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015+, March 2009.
16. A. Lancichinetti and S. Fortunato1. Community detection algorithms: A comparative analysis. *Physical Review E*, 80(5):056117, Nov 2009.
17. Y. Liu, W. Li, and Y.-C. Li. Network traffic classification using k-means clustering. *Computer and Computational Sciences, International Multi-Symposiums on*, 0:360–365, 2007.
18. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.
19. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, Feb 2004.
20. G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society, June 2005.

21. S. Papadopoulos, A. Skusa, A. Vakali, Y. Kompatsiaris, and N. Wagner. Bridge bounding: A local approach for efficient community discovery in complex networks. Technical Report arXiv:0902.0871, Feb 2009.
22. S. Ray and R. Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. In *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques*, pages 137–143, 1999.
23. S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27 – 64, 2007.
24. SNAP. Stanford large network dataset collection. In *http://snap.stanford.edu*.
25. S. Van Dongen. Graph clustering via a discrete uncoupling process. *SIAM J. Matrix Anal. Appl.*, 30:121–141, February 2008.
26. S. Wang, Y. Tsai, C. Shen, and P. Chen. Hierarchical key derivation scheme for group-oriented communication systems. *International Journal of Information Technology, Communications and Convergence*, 1(1):66–76, 2010.
27. F. Wu and B. A. Huberman. Finding communities in linear time: a physics approach. *The European Physical Journal B - Condensed Matter and Complex Systems*, 38(2):331–338, March 2004.
28. Y. Ye, X. Li, B. Wu, and Y. Li. A comparative study of feature weighting methods for document co-clustering. *International Journal of Information Technology, Communications and Convergence*, 1(2):206–220, 2011.
29. F. Yu, D. Oyana, W. Hou, and M. Wainer. Approximate clustering on data streams using discrete cosine transform. *Journal of Information Processing Systems*, 6(1):67–78, 2010.