Crawling and Detecting Community Structure in Online Social Networks Using Local Information

Norbert Blenn, Christian Doerr, Bas Van Kester, and Piet Van Mieghem

Department of Telecommunication,
TU Delft, Mekelweg 4, 2628CD Delft, The Netherlands
{N.Blenn,C.Doerr,S.vanKester@student.,P.F.A.VanMieghem}@tudelft.nl

Abstract. As Online Social Networks (OSNs) become an intensive subject of research for example in computer science, networking, social sciences etc., a growing need for valid and useful datasets is present. The time taken to crawl the network is however introducing a bias which should be minimized. Usual ways of addressing this problem are sampling based on the nodes (users) ids in the network or crawling the network until one "feels" a sufficient amount of data has been obtained.

In this paper we introduce a new way of directing the crawling procedure to selectively obtain communities of the network. Thus, a researcher is able to obtain those users belonging to the same community and rapidly begin with the evaluation. As all users involved in the same community are crawled first, the bias introduced by the time taken to crawl the network and the evolution of the network itself is less.

Our presented technique is also detecting communities during runtime. We compare our method called *Mutual Friend Crawling (MFC)* to the standard methods Breadth First Search (BFS) and Depth First Search (DFS) and different community detection algorithms. The presented results are very promising as our method takes only linear runtime but is detecting equal structures as modularity based community detection algorithms.

Keywords: Social Networks, Community Detection, Crawling.

1 Introduction

Analyzing human social behavior depends on observations of large scale networks. Online Social Networks (OSNs) proved to be good sources of information facilitating this kind of research, as the most popular OSNs such as Twitter, Facebook or LinkedIn consists of hundreds of millions of user accounts, thereby allowing an analysis at a sufficiently large scale.

However, it is usually not possible to obtain datasets from the operators of OSNs. Therefore, a common approach is to crawl the network per user. In this way, a user is randomly chosen and a list of his friends is downloaded. Out of the list of friends, again one user is selected and a list of friends retrieved.

This method repeats in principle until every user in the network has been visited once. This method varies in the selection of the next friend, are ranging from Breadth First Search towards Depth First Search.

This automated process of downloading users typically requires a robot (a software program) to look at the profile page of a user and store the names of all friends. Such an operation usually takes between 0.1 to 2 seconds as it includes multiple HTTP requests to a server in order to iterate through the whole list of friends. An optimistic[1] calculation shows that with one crawling computer, obtaining LinkedIn's database of 120 million users (as of Nov. 2011) would take approximately half a year. The same calculation for Facebook's dataset of 650 million users leads to a crawling time of ca. 2 years. By using massively parallel crawling techniques those times can be decreased. Clearly, by the time the last records have been obtained, the much of the retrieved information will be outdated.

A lot of work on social network analysis has been done using communities of users as a level of abstraction. A natural question is therefore whether it is possible to direct the crawling procedure in such a way that it is crawling the network community-wise. This would enable researchers to analyze useful subgraphs of the whole network while still obtaining data. In contrast, using the standard crawling methods like Breadth First Search or Depth First Search, one literally has to wait until the whole network is crawled before starting to analyze the data because there might be a few users critical to a particular single community still missing from the dataset, and their existence and criticality cannot be determined until all data is in.

In this paper we present a simple approach to crawl a network community-wise and detect communities at the same time. Our algorithm called *Mutual Friend Crawling* is compared to existing community detection methods.

The remainder of this paper is structured as follows. Section 2 summarizes the related work, in Section 3 our method of crawling is described, evaluated and the community detection is explained. Section 4 will compare the community detection with well known community detection algorithms and Section 5 will summarize our findings and give an outlook.

2 Related Work

Communities are defined in terms of the fraction of nodes of a network, that share more connections with each other than with the rest of the network. When analyzing a social network, some relevant questions are:

1. How many communities are there in the network?
2. Which nodes are in the same community?
3. How well are users in communities connected?

[1] In this context, optimistic means that no mechanisms against crawling or screen scraping are enforced.

A well known metric to capture the community structure of a network is modularity. Modularity m as defined in Clauset, Newman and Moores's work [1] is "the number of edges falling within groups minus the expected number in an equivalent network with edges placed at random." The definition of modularity is given in equation 1.

$$m = \frac{1}{2L} \sum_{i}^{N} \sum_{j}^{N} (a_{ij} - \frac{d_i d_j}{2L}) 1_{\{i \text{ and } j \text{ belong to the same community}\}} \tag{1}$$

For a given graph G with N nodes, L links and a given partition, the modularity denotes how well the community structure is expressed. The element a_{ij} denotes the element corresponding to the ith row and jth column of the adjacency matrix of G and d_i is the degree of node i. $1_{\{i \text{ and } j \text{ belong to the same community}\}}$ is the indicator function returning 1 if i and j are in the same community otherwise 0.

A modularity value of 0 defines that the number of links belonging to the same community is equal to the number a random graph would have. The higher the modularity the more pronounced the community structure, except for the trivial case of modularity $= 1$, in which all links of G are in the same community. Conversely, this means that negative values are a definition of something like an "anti-community" structure. A nice overview over modular graphs and how to achieve high modularity is given in Trajanovski [2].

As we will present an algorithm to crawl community structure in real world OSNs having linear complexity, we compared our results to crawling techniques like Breadth First Search (BFS) and Depth First Search (DFS) as described in Cormen et al. [3]. In both techniques the graph is crawled node per node adding all discovered nodes to a list of nodes to visit. The difference between BFS and DFS is based on the procedure how the next node to visit is selected. In BFS the first node of this list is selected to be visited next and removed from the list whereas in DFS the last node in the list is selected and marked as visited. Both techniques are leading the crawling procedure towards the inner core of the network due to the friendship paradaxon. This paradoxon, first observed by Field [4] stating originally that your friends have more friends than you, will force a crawl towards nodes having a high centrality in the network. A related effect, noted by Kurant et al. [5] describes that BFS and DFS is introducing an bias towards high degree nodes for an incomplete traversal of the network.

To our knowledge, BFS and DFS are the most used techniques to traverse a graph. We will show in section 3 that, in order to reveal the community structure of a graph BFS and DFS are not the best choice. One possible technique of crawling a network and detecting communities at the same time is to facilitate random walks. Random walks are known to stay inside communities as described in Pons and Latapy [6] and Lai and Lu [7]. The main idea behind random walk community detection is that a community has more links between nodes of the community than between communities. Because of this definition, a random walk would traverse nodes of the same community more often than the ones of different communities. However, a random walk allows steps backwards to already visited nodes which is increasing the time taken to crawl the network.

Different community detection algorithms like fast and greedy community detection by Clauset et al. [1], Spinglass by Reichold and Bornholdt [8], edge betweenness clustering by Girvan and Newman [9] or label propagation by Raghavan et al. [11] cannot be used to detect communities during crawling as those algorithms are meant to be applied onto the full topology of a network.

A related approach to detect community structure while crawling is presented by Nguyen et al. [12]. Their algorithm Quick Community Adaptation (QCA) assumes that the community structure is already known for a complete network and manages to calculate community structure in dynamic networks. QCA tries to maximize modularity by assigning a "force" which attracts a node towards a community. However, this method also needs the whole network including assignments of nodes into communities. In their approach the used algorithm to estimate the initial community memberships is presented by Blondel et al. [13] called the Louvain(-la-Neuve) method. This algorithm calculates a modularity maximizing partition of a given graph by using the change in modularity when discovering a new node and adding it to an existing community. If the difference is not positive the node stays in its initially assigned community.

To compare the result of different clustering algorithms we decided to use the Jaccard similarity coefficient next to the already defined modularity. The Jaccard similarity coefficient defines the similarity of sample sets by measuring the quotient of the intersection and the union of both sets.To compare the result of different cluster assignments a definition of Fortunato and Castellano [14] given in equation 2 is used.

$$I_J(s_1, s_2) = \frac{n_{11}}{n_{01} + n_{11} + n_{10}} \tag{2}$$

In equation 2, n_{11} denotes the number of node pairs found in the same community whereas n_{01} and n_{10} are the number of pairs of nodes assigned to the same community by algorithm s_1 but not s_2 and vice versa.

3 Crawling and Detecting Communities

We introduce *Mutual Friend Crawling* which crawls nodes of a network in such a way that communities are visited one after another. First we will show that our approach is crawling all nodes belonging to one community before continuing with nodes of a connected community. Afterwards we will show how to detect communities using this approach.

In contrast to BFS and DFS, our algorithm assumes the knowledge about the degree of neighboring nodes. This assumption is reasonable in OSNs as the number of friends is very easy to obtain, whereas the process of receiving the actual links towards them needs more effort. For example in the OSN Twitter, each message contains a field containing the number of followers and friends the originating author has. Also, OSNs are most commonly crawled using a technique called screen scraping. Here, the OSN is accessed the same way a user does, by using HTTP requests to analyze web pages for relevant data.

The number of friends is usually listed at the profile page of a user but several clicks (requests) on the list of friends are needed to obtain all node ids (friends) having a relationship with this user. If one is interested in more details of a user, for example the real name or group affiliations, this profile information will need to be obtained in any case and at the same time a friend count is also available without any additional overhead. In this way the needed crawling effort has not increased but only the order in which the data is gathered has changed.

Mutual Friend Crawling is based on the "reference score" (S_R) defined in equation 3. This score denotes the fraction of the number of already discovered links pointing to node f (references) so far in the crawling process and the total degree, i.e., total number of friends, of node f.

$$S_R = \frac{\text{number of found references to f}}{\text{degree of node f}} \tag{3}$$

During the crawl, the next node to process is chosen from the list of the already discovered nodes having the largest S_R. The full algorithm is specified in pseudo code in algorithm 1.

Algorithm 1. Mutual Friend Crawling

```
 1: create a queue $Q$
 2: create a map $R$
 3: add starting node to $Q$
 4: store starting node and 0 as number of found references in $R$
 5: while $Q$ is not empty do
 6:    for all elements in $R$ do
 7:       reference_score ← $\frac{\text{value in R}}{\text{degree of the node}}$
 8:       max_score ← $max$(max_score, reference_score)
 9:    end for
10:    next_node ← dequeue element having max_score from $Q$
11:    delete next_node from $R$
12:    if next_node has not been visited yet then
13:       for all neighbors of next_node: do
14:          add neighbor to $Q$
15:          increment number of found references to neighbor by 1 and store it in $R$
16:       end for
17:       remember that node (next_node) was visited
18:    end if
19: end while
```

Mutual Friend Crawling is based on a BFS algorithm having two major differences. The first one is a map used to store the number of found references as indicated in line 2, 4 and 15. The second difference is based on the way the next node to visit is chosen: instead of choosing simply the next one from the list as BFS does, MFC calculates the reference score (lines 6-9) and chooses the next node based on the maximum of the reference score (line 10 & 11).

The algorithm will therefore first visit nodes where to which the largest number of the overall links have already discovered. If a network has a community structure based on the definition of having more links in the community than links connecting communities our algorithm will crawl communities one after another.

In order to apply the algorithm on weighted graphs, a simple definition of the strength of a node as the sum of the weights of adjacent edges is sufficient. In this case the reference_score S_R is defined as the fraction of the sum of weights of already discovered links to the strength of the node as defined in 4.

$$S_R = \frac{\sum \text{(weights of found references to f)}}{\text{strength of node f}} \tag{4}$$

3.1 Community Crawling

Figure 1 illustrates our crawling approach intuitively using a small example. Using the definition of communities as stated above (more links "within" the group than ending "outside"), simple visual inspection already shows that there are six clusters. If we were to explore each cluster one after the other, the algorithm should first explore all nodes belonging to one color before starting with the next group of nodes. The nodes labels denote one possible order in which the graph could be crawled in order to visit communities one after another, thus leading to the intended and perfect exploration order.

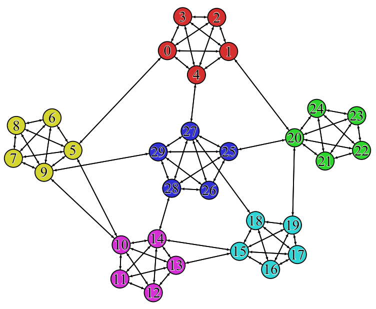

Fig. 1. A simple example graph. Nodes are labeled by the order of traversal during the crawling process. Different colors denote different communities.

In order to test the proposed algorithm on multiple graphs a metric defining how strong the community structure in a given graph is expressed was needed. A simple metric is given by the ratio of links inside communities to the total number of links. We define this value as P_{in} reflecting the probability an arbitrary chosen link is an intra community link.

We created a total of 100,000 artificial networks with different P_{in} values using a graph generator, described in van Kester [15], each with 10,000 nodes, 100,000 links and 100 equally sized communities have been generated. Two general types of graphs commonly found in network science were evaluated:

The degree distribution of the first type follows a uniform distribution and the degree distribution of the second type is approximating a power-law function. All graphs have been crawled from all possible starting nodes. During the crawl we kept track of the order when nodes are visited, which allows us to analyze if a complete community has been crawled before going to the next one.

Figure 2 shows the crawling trajectories of those crawls. The figure is expressing how many nodes have to be crawled in order to visit all nodes of a community. In order to crawl the network community-wise the optimal traversal would need to visit all nodes of one community first before visiting the next node belonging to the next community. Because all communities are equally sized, the optimal traversal of the graph would lead to a diagonal line in figure 2. A bended line is expressing the fact that nodes from different communities were visited before all nodes of the previous visited community have been finished. The order in which multiple communities are crawled is not reflected in figure 2, as we are primarily interested in obtained one completely crawled community after another and not which one is obtained first. The used colors express trajectories of different crawling methods whereas green lines belong to DFS, blue to BFS and red to our *Mutual Friend Crawling*.

For high P_{in} values, figure 2 illustrates that our algorithm performs as expected and leads the walk on the graph to all nodes contained in one community before crawling the next. For BFS and DFS a larger fraction of the network has to be crawled to finish one community. Interestingly, BFS perform "closer" to the optimum than DFS. This is because BFS explores the local neighborhood whereas DFS explores the nodes furthest away from the starting node. Thus, the "chance" of BFS to visit all nodes of one community earlier than in DFS is higher.

The proposed crawling method performs reasonably better than BFS and DFS in terms of crawling along the community structure. For P_{in} values larger 0.3,

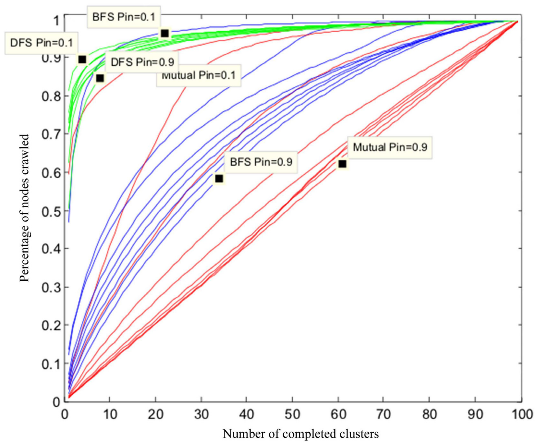

Fig. 2. Crawling communities. Depicts the percentage of nodes that have to be visited in order to crawl a full community.

the order in which the nodes are traversed fulfills our requirements. However, P_{in} values smaller than 0.5 somehow define negative communities in terms of the definition and therefore a BFS approach by chance performs better.

3.2 Community Detection

We already demonstrated empirically that our method crawls communities of a graph one after another. In order to detect communities while crawling the graph, traces of the reference_score of visited nodes can be analyzed. Figure 3 shows the trace of reference_scores performed on the example graph (figure 1) starting from node 0. As mentioned earlier the proposed method always selects the next node to visit having the highest reference_score (line 8 in algorithm 1). Hence the reference_scores inside communities should always increase or stay roughly the same while traversing the graph. When detecting a node that is interconnecting communities, a large number of links of this node are ended in the previously unknown community. Therefore, its reference_score will be smaller than the ones of nodes connected to this in the current community. As this node is selected as the last one, a drop can be observed in the trajectory of reference_scores of visited nodes. Figure 3 shows five major drops of the score. Those drops in the reference_score of the chosen nodes reflect the creation of new communities. We define the difference between the next reference_score to the previous one as $\Delta refscore$. During traversing the graph, all nodes are added to the same community as long as $\Delta refscore$ is large enough. If the score decreases a new community can be created. To prevent the creation of single node communities, we found that the drop in the reference_score should be at least half the difference between the maximum S_{max} and the minimum S_{min} of the reference_score in this community. By using this method, the *Mutual Friend Crawling* is creating six communities on the sample graph having the community assignments as indicated by different colors in figure 1.

One problem however still remains: a possibly incorrect classification of certain nodes during the first visit of a new community. In case neighbors of the starting

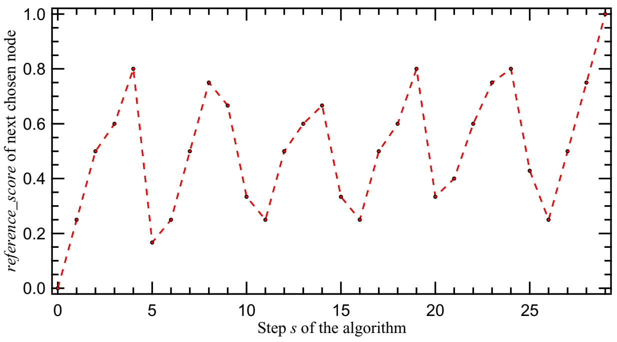

Fig. 3. Plot of reference_scores versus the number of visited nodes

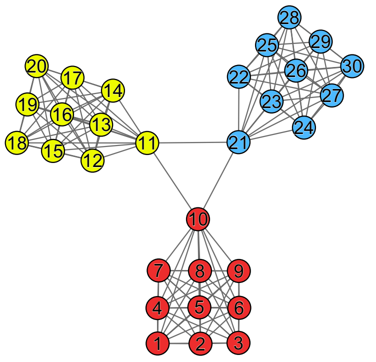

Fig. 4. Example graph where a misclassification may occur when visiting node 11

node as well as neighbors of the first node of a community have the same degree as the visited node, a node of a different community could be assigned incorrectly. Such misclassification is exemplified for the case of three equally-sized, fully-connected communities which are pairwise connected through one node as shown in figure 4. In this case when starting in one clique, all nodes of this clique are added to the first community. When reaching one of the 3 nodes connecting communities (10, 11 or 21 in figure 4) the score drops, a new community is generated and all following nodes are added to this community. When reaching for example node 10, the reference_score for the 2 peers (11 and 21) is the same. One of them is chosen to be next node to visit. But now, (e.g., by visiting node 11) 2 links towards the third node are discovered doubling its score. Having a higher score than all other neighbors of 11, 21 will be added to the same community as 11. Afterwards one of 11's or 21's neighbors are visited where the reference_score drops again leaving 11 and 21 in one community. all other nodes afterwards are correctly classified. Our solution is to check for this kind of misclassification by iterating through all nodes in a already discovered community checking if a node has more connections with another community then inside the "own" community. If so this node is merged to the connected community. As this procedure is raising the density in the community the node is merged to, the modularity value is increasing as stated in Trajanovski [2].

4 Verification and Performance Evaluation

To prove the correctness of *Mutual Friend Crawling* in terms of community detection, we compare the results of community assignments to existing approaches. However, as the variety of community detection algorithms is too large to compare against, we chose algorithms which (with slight modifications) can be used to identify communities while crawling. This means we are comparing against algorithms which iteratively assign nodes to communities without having the knowledge about the whole graph.

The chosen methods are:

1. Newman and Clauset's fast and greedy modularity maximizing method [1]
2. Pons & Latapy's random walk method [6]
3. the Louvain(-la-Neuve) method by Blondel et al. [13]

As all mentioned approaches are not directly providing a map of community ids to node ids we chose the partition resulting from the merges of nodes leading to a maximum of modularity. This partition is then compared to the output of our algorithm. For the given example graph in figure 1, all community detection algorithms found the same result as indicated by the colors in figure 1.

A comparison of the mentioned methods on some selected datasets is given in the following table 1. The datasets we compared against contain "Zachary's karate club" [16], Girvan and Newman's "American College football games" [9] and the network of all Digg users as described in Tang et. al. [17].

Table 1. Comparison of Mutual Friend Crawling to well known community detection procedures on different datasets

Dataset	Method	Number of communities	P_{in}	Modularity
Karate club	original partition	2	0.86	0.36
	Louvain method	4	0.74	0.42
	Fast and greedy method	3	0.74	0.38
	Random walk method	5	0.63	0.35
	Mutual friend crawling	2	0.86	0.36
Football	original partition	12	0.64	0.554
	Louvain method	10	0.708	0.604
	Fast and greedy method	5	0.746	0.544
	Random walk method	9	0.726	0.603
	Mutual friend crawling	9	0.736	0.57
Digg	Louvain method	26646	0.94	0.478
	Fast and greedy method	37591	0.92	0.393
	Mutual friend crawling	78308	0.83	0.142

As given in table 1, our method is comparable to existing and well known procedures when compared in terms of the P_{in} value and modularity. Except for the last dataset, a large scale directed network of all users of Digg.com where the number of detected communities is higher than given by the Louvain(-la-Neuve) or fast and greedy method.

However, this could be based on the resolution limit of modularity, as described in Fortunato and Barthélemy [18]. A partition having a high modularity could lead to a relatively small number of large communities which is not reflecting the real community structure. The communities found by *Mutual Friend Crawling* are smaller than the ones found by the other methods still having the same properties like a power law shaped community size distribution. Also the number of users in a group given by our method is reasonable. The largest community found by Mutual Friend Crawling has a size of 9443 users whereas the largest one found by the Louvain method contains 186271 users. Without further investigation one may arguments for both numbers to be better than the other one. Therefore we leave this question to be solved by further research.

Table 2. Comparison of the partitions discovered by our method to other community detection algorithms on the college football dataset [9] using the Jaccard similarity index

method	original	Louvain	Fast and greedy	Random walk	Mutual friend crawling
original	1	0.719	0.354	0.615	0.468
Louvain		1	0.424	0.721	0.483
Fast and greedy			1	0.422	0.324
Random walk				1	0.487
Mutual friend crawling					1

As the P_{in} value and the modularity are global values which cannot be used to compare two partitions directly the Jaccard similarity index may be used. As mentioned earlier, if pairs of nodes are assigned to the same community this similarity will have a high value.

Table 2 shows the similarity of node assignment into communities between the different community detection algorithms. While this metric is not very sensitive to the number of communities it shows that our approach is equally good as well known methods. A more complete analysis of the Jaccard similarity index is given in van Kester [15].

5 Conclusion and Outlook

In this paper we presented *Mutual Friend Crawling*, an algorithm to crawl a large scale OSN in such a way that the community structure of the network is detected and communities are crawled one after another. To our knowledge this is the first analysis directing a crawling process towards community structure. We showed that our method crawls communities one after another. Especially when obtaining large scale networks, researchers could begin to analyze datasets based on communities while the crawling process is still running.

Further work is needed if the community consists of overlapping groups as the partition of a large scale network into clearly separated communities does not make sense in most Online Social Networks. Also the application of standard metrics (like modularity or the used P_{in} value) to compare partitions to real world community structure should be the focus of additional, future research.

Acknowledgements: We thank Wynand Winterbach and Stojan Trajanowski for useful discussions. This project was partially supported by the Transsector Research Academy for Complex Networks and Services (TRANS).

References

1. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Physical Review E 70 (December 2004)
2. Trajanovski, S., Wang, H., Van Mieghem, P.: Maximum Modular Graphs. European Physics Journal B (2011) (submitted)

3. Cormen, T.: Introduction to algorithms. MIT electrical engineering and computer science series. MIT Press (2001)
4. Feld, S.L.: Why Your Friends Have More Friends Than You Do. American Journal of Sociology 96(6), 1464–1477 (1991)
5. Kurant, M., Markopoulou, A., Thiran, P.: On the bias of BFS (Breadth First Search). In: 22nd International Teletraffic Congress (ITC), pp. 1–8. IEEE (2010)
6. Pons, P., Latapy, M.: Computing Communities in Large Networks Using Random Walks. In: Yolum, p., Güngör, T., Gürgen, F., Özturan, C. (eds.) ISCIS 2005. LNCS, vol. 3733, pp. 284–293. Springer, Heidelberg (2005)
7. Lai, D., Lu, H., Nardini, C.: Enhanced modularity-based community detection by random walk network preprocessing. Phys. Rev. E 81, 066118 (2010)
8. Reichardt, J., Bornholdt, S.: Statistical mechanics of community detection. Phys. Rev. E Stat. Nonlin. Soft. Matter Phys. 74 (July 2006)
9. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proceedings of the National Academy of Sciences 99, 7821–7826 (2002)
10. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69, 026113 (2004)
11. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Physical Review E 76, 036106+ (2007)
12. Nguyen, N., Dinh, T., Xuan, Y., Thai, M.: Adaptive algorithms for detecting community structure in dynamic social networks. In: 2011 Proceedings IEEE INFOCOM, pp. 2282–2290 (April 2011)
13. Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast unfolding of community hierarchies in large networks. CoRR, abs/0803.0476 (2008)
14. Fortunato, S., Castellano, C.: Community structure in graphs (2007)
15. Van Kester, S.: Efficient Crawling of Community Structures in Online Social Networks. PVM 2011-071, Tu Delft (September 2011)
16. Zachary, W.W.: An Information Flow Model for Conflict and Fission in Small Groups. Journal of Anthropological Research 33(4) (1977)
17. Tang, S., Blenn, N., Doerr, C., Van Mieghem, P.: Digging in the Digg Social News Website. IEEE Transactions on Multimedia 13, 1163–1175 (2011)
18. Fortunato, S., Barthélemy, M.: Resolution limit in community detection. Proceedings of the National Academy of Sciences 104, 36–41 (2007)