2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining

Real Time Distributed Community Structure Detection in Dynamic Networks

Valerie Galluzzi
Department of Computer Science
University of Iowa
14 MacLean Hall Iowa City, IA, 52242-2429
USA
Telephone: (319)353-2037
Email: valerie-galluzzi@uiowa.edu

***Abstract*—Communities can be observed in many real-world graphs. In general, a community can be thought of as a portion of a graph in which intra-community links are dense while inter-community links are sparse. Automatic community structure detection has been well studied in static graphs. However, many practical applications of community structure involve networks in which communities change dynamically over time. Several methods of detecting the community structure of dynamic graphs have been proposed, however most treat the dynamic graph as a series of static snapshots, which creates unrealistic assumptions. Others require large amounts of computational resources or require knowledge of the dynamic graph from start to finish, relegating them to post-processing. For those who desire real-time community structure detection distributed over the observing network, these solutions are insufficient. This paper proposes a new method of community structure detection which allows for real time distributed detection of community structure.**

I. Introduction

Many common phenomena can be modeled as networks. Examples from everyday life include information networks like the internet, transportation networks, social networks both electronic and real-world, and epidemiological contact networks. In many of these networks we observe clusters of nodes which are densely connected while their connections to other nodes are comparatively few. Such clusters are known as communities, and are common features of social networks.

Automatic detection of communities is essential for some applications. One example is the understanding of disease diffusion through a social network, where a sophisticated understanding of the underlying community structure could help with better understanding disease spread and guide the deployment of interventions like vaccines or warnings to increase hand washing. Automatically detecting community structure in graphs that are static, or unchanging over time, is well explored, and a survey can be found here [1]. However, communities in social network graphs often change over time, leading to dynamic community structure which is difficult to correctly detect automatically.

Some algorithms for automatically detecting community structure in dynamic graphs have been proposed. Initial attempts tracked communities rather than detecting them in the network [2], [3], [4]. However, these could not find a newly developed community that appears in the network.

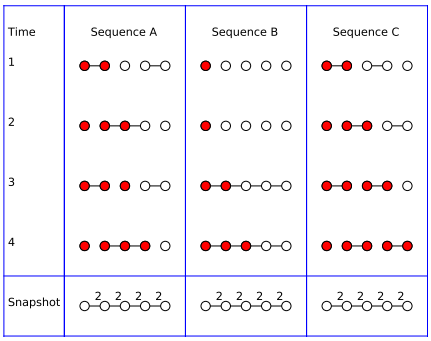


Fig. 1. Edge sequences. All sequences map to same static snapshot. If edges were weighted by number of times observed, weights would be 2.

More recent algorithms apply known static community detection algorithms to static graphs created from snapshots of the dynamic network [6], [7]. However, these need some way of capturing a dynamic network in a series of static snapshots, which can be challenging. For instance, each observed sequence of edges in figure 1 could map to the same static snapshot, even if we choose to weight edges by the number of contacts. This is despite the fact that the different sequences could mean different outcomes. For instance, in an epidemiological setting, a disease starting at the leftmost node and capable of infecting one more node with each timestep would infect 4 nodes in A, 3 nodes in B, and 5 in C. Should they all map to the same static snapshot? These questions make it difficult to create good mappings to static snapshots.

Others assume that the entire network is known from the first timestep to the last, and apply post-processing techniques to produce community structure assignments [8]. These methods preclude real time results.

However, there exist some applications for which community structure detection should happen in real time, ideally in a distributed fashion, a situation which current methods cannot accommodate. An example is the University of Iowa Computational Epidemiology group's deployment of sensors

978-0-7695-4799-2/12 $26.00 © 2012 IEEE
DOI 10.1109/ASONAM.2012.213

in the University of Iowa Hospital [9]. Sensors are small radio equipped devices, and in this case they are attached to people and objects. A contact between two sensors is detected when their radios are able to communicate with each other and so a network is created by which health care worker movements and interactions can be observed through the contacts of their attached sensors.

An ideal application of community structure detection in this setting would be assessments of individual risk for infection given travel between and within health care worker communities. If this assessment occurred in post processing, interventions would be difficult to apply on an individualized basis as the collected sensor data is anonymized to protect worker privacy. A centralized real time approach would depend upon good connections between sensors and a centralized community structure detector, which may be difficult to ensure given that the sensors are frequently moving. A more ideal setting would be risk levels which are calculated in real time on a health care worker's sensor based on their travel between communities and then displayed to the worker, avoiding privacy concerns and connectivity and post processing issues.

Another application would be a system that delivers ads to mobile devices. Said system might wish to differentiate those who frequent the location from those who are passing through to deliver different content. This could be accomplished by determining whether or not the user is a member of a local community. If community assessment were real time and distributed then ads could be delivered immediately and the ad system could be deployed without worrying about connectivity to a master community structure detector.

These applications require real time assessment of community structure in a distributed fashion, which cannot be realized by current community structure detection algorithms. This paper aims to introduce such an algorithm.

II. Algorithm

The following section describes our algorithm which is lightweight and customizable. It is inspired by the static network community detection algorithm proposed by [10].

A. Definitions

A dynamic graph G is a series of graphs $G = G_0, G_1, ..., G_T$ where $G_t(V, E_t)$ is the graph with vertex set V and the edge set consisting of the edges at time step $t \in 0, 1, ..., T$. Define c_i as the one community of which i is a member. Let h be the maximum number of contacts i will remember. Let a history H_i be the list of the community affiliations of node i's h most recent contacts at the time i made contact with them.

Let d be a parameter such that the most recent contact in H_i has an importance of h while the next most recent has importance of $max(h-d, 1)$, and the k^{th} most recent contact in H_i has importance of $max(h - dk, 1)$. Let $p >> h$ be a number such that $1/p$ is the probability that the node will randomly relabel itself by changing its community to its own id. Let t_r be the number of contacts that will be notified of relabeling.

TABLE I
Algorithm for Each Node

Require: H_i be the list of the community label of i's h most recent contacts at time of contact
Require: $r_i = True$ if i is currently in a relabeling phase, $False$ otherwise
Require: rc_i is the number of contacts for which i has been in the relabeling phase
Require: h, d, p, t_r defined as described in II-A
FindCommunity(i:Self ID, j:Connected Node):
if $|H_i| \geq h$ **then**
Remove the oldest contact in H_i
end if
$H_i = H_i + c_j$
if $\neg r_i$ **then**
With probability $1/p$
Let $a_i = c_i$ and $b_i = i$
Relabel c_i with b_i
$rc_i = 0$
$r_i = True$
end if
if r_i **then**
Replace all instances of a_i in H_i with b_i
end if
if r_j **then**
Replace all instances of a_j in H_i with b_j
end if
Let L_i be a list of all communities mentioned in H_i
Let all communities in L_i have weight 1
for l_k in H_i **do**
Add $max(h - dk, 1)$ to the community l_k in list L_i
end for
$c_i =$ the highest weight community in L_i
if $rc_i = t_r$ **then**
$r_i = False$
end if
if r_i **then**
$rc_i = rc_i + 1$
end if

B. Algorithm

Observe the following:

1) Individuals are more likely to interact with fellow community members than with members of other communities
2) Interactions are captured as edges between two individuals.

Let the set E_i be all edges with i as an endpoint in subset $S_{b,e} = G_b, G_{b+1}, ..., G_e$ where $[b, e]$ is the interval of interest. Given these two observations, we can see that if in set E_i node i interacts (shares edges) mostly with members of community l then it is probably a member of community l during that time. If individuals are known to change community infrequently then the current community association of the individual is likely to be predictive of future community association.

Using these insights, we can estimate community membership on an individual basis in real time using only local information with the algorithm shown in table I.

C. Analysis

1) Runtime: If this algorithm were simulated using the algorithm in table II, it would run in $O(TEh)$ time. However, the T and E figures are not relevant to the individual node

TABLE II
ALGORITHM FOR SIMULATION

Set $c_i = i \; \forall \; i$ {Initialize each node's community to be its ID}
Set $H_i = \{i\} \; \forall \; i$ {Initialize each node's history to a meeting with itself}
Set $r_i = False \; \forall \; i$ {Initialize relabeling flag to $False$ for all nodes}
Set $rc_i = 0 \; \forall \; i$ {Initialize the relabeling counter to 0 for all nodes}
for t in $\{0, 1, ..., T\}$ **do**
  for (i, j) in E_t **do**
    FindCommunity(i,j)
  end for
end for

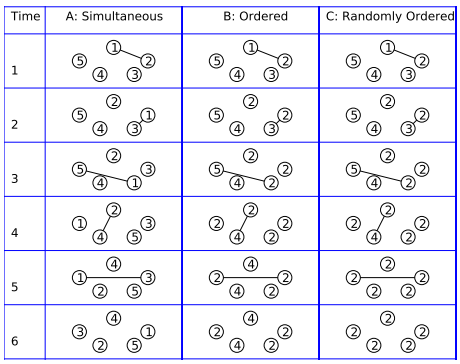


Fig. 2. Ordering Policies: $h = 1$. Depending on policy, convergance can be impossible. Labels updated according to found edge in each timestep.

when this is run in a distributed fashion, so each node will spend $O(h)$ time on this algorithm.

2) Importance of Update Order: Let us consider the conditions under which a given community can converge to having one label. One factor which is important is the order in which nodes update their community label when an edge is observed. As an example, see figure 2. In this figure the community affiliation is shown as a label on the node. We assume $h = 1$ and show how the ordering of node updates can effect convergence.

Example A shows the result if nodes can instantaneously communicate their community affiliation–in that case every meeting results in a simple swap of community affiliation. In Example B it is assumed that if an edge (i, j) exists then i will update before j if $i < j$. In this case community affiliation is pushed around the network. The only way to converge on one affiliation in Example B is to be extremely lucky with the order of edges. The only way to break these cases is by choosing the first updating point randomly, as in Example C.

All simulation experiments use the random updating strategy shown in Example C to improve convergence probability.

3) Relabeling: The purpose of randomly relabeling nodes is to enable detection of the splitting of a community. A split occurs when a community breaks into separate communities.

Without random relabeling it is possible that both communities formed by a community split will retain the same label. If that is the case, then it is impossible to tell the two communities apart. By randomly replacing a node's community label with its personal label and propagating that label through the network we ensure that a community's label is the label of a member in that community. This is essential when we need to update community labels after a split has occurred.

III. RESULTS

Unfortunately there are no dynamic benchmark graphs that could be used to evaluate the performance of this algorithm. Therefore to analyze the performance of this algorithm it was run on several artificially generated random networks. It was additionally run on a more realistic network in which nodes randomly moving through space, the results of which appear in section III-F. Although random relabeling is essential for handling community splits, it injects noise (community changes) into the system by design and is therefore not featured in many of the results to enhance the clarity of figures.

A. Networks Used

Four networks were used for testing. All networks consisted of 100 nodes. In all methods edges are generated by choosing a random node in the network and joining it to another random node in its community. A stabilization parameter n is chosen to be longer than the expected maximum convergence time of the algorithm. Between experiments the value of n could change to accommodate the longer or shorter expected maximum convergence times when using different parameters.

Join This network illustrates the joining of two communities. Initially the network is partitioned into two communities of 50 nodes each. In every iteration a random contact is generated. After n contacts have been made the two networks join together for n contacts.

Split This network illustrates the splitting of a community. In this network all nodes are in the same community for n contacts, and then the community splits into two communities of 50 nodes each for n contacts.

Connected This network contains two communities of equal size that are connected by some intermediary nodes. This network is simulated for n contacts.

B. Convergence and h

In figure 3 experiments were run on the Join network with $d = 0$ and without relabeling to show the effect of changes in h on the convergence time (the time for the number of errors to stabilize). The figure shows that larger values of h produce longer convergence times.

The intuition behind this increase is this. In the case of one node needing to decide which community it is in, it could take up to h contacts for it to decide. As h gets longer this decision time also gets longer. Therefore when a node is supposed to join with a community it belongs to, it takes longer for it to join the community, therefore lengthening convergence time.

The errors shown in the graph are summed over windows of 33 contacts. An error occurs when a pair of nodes is either in the same community when they belong to different communities or vice versa. Therefore errors are calculated over

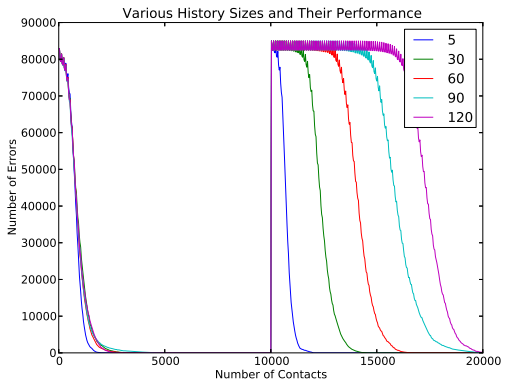


Fig. 3. Varying Values of h and Their Performance. Simulation was run on a network of 100 nodes split into two communities for 10000 contacts then joined into one community for 10000 contacts. $d = 0$, relabeling was disabled. An error is detected when there exists a pair of nodes (i, j) such that their true community is different but the algorithm labels them with the same community or vice versa. Errors summed over windows of 33 contacts.

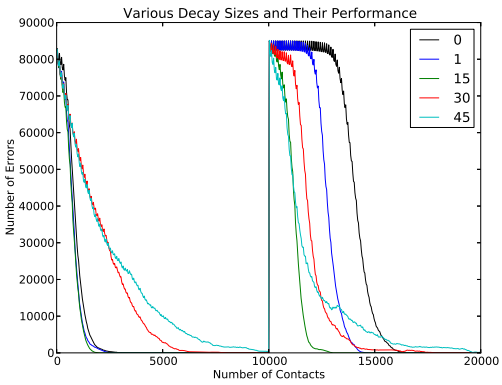


Fig. 4. Varying Values of d and Their Performance. Simulation was run on a network of 100 nodes split into two communities for 10000 contacts then joined into one community for 10000 contacts. $h = 60$, relabeling was disabled. An error is detected when there exists a pair of nodes (i, j) such that their true community is different but the algorithm labels them with the same community or vice versa. Errors summed over windows of 33 contacts.

all pairs of nodes in the network, regardless of whether or not they contacted each other recently.

The clearest performance difference occurs when the networks join together at contact 10,000. Note how as h gets larger in each case the convergence time lengthens.

The initial phase where all nodes begin with different initial labels at contact 0 can be considered to be the joining of many communities, but performance between different sizes of h is more similar in the first phase for a few reasons. The first is because the true community is smaller (50 nodes). Additionally when community joining occurs at contact 10,000 all node histories are saturated with the community label of their previous community. By contrast, in the case of many small communities joining together, the node's histories are probably not saturated with one community label, making it easier for them to convert to another label. Therefore the first phase converges more quickly than the second.

C. Convergence and d

In figure 4 experiments were run on the Join network with $h = 60$ and d at varying values. These experiments were done without relabeling to make the effect of d clearer. The error calculation and window summing is equivalent to that used for figure 3, so comparison can be made between the two figures.

Figure 4 shows the effect of d on the convergence time. Note that higher d does not always imply faster convergence. This is because after $d \geq h/2$ the performance of the algorithm becomes closer to the performance with $h = 1$. However, we know that when $h = 1$ the node must decide which community to join based solely on its last contact. Referring back to figure 2 and section III-B we can see that when $h = 1$ the convergence behavior depends on lucky choices of updating order. The reason that we can start to see longer tails when $d = 30$ and $d = 45$ is because there are a few unlucky nodes who take some time before they converge.

Worth noting is that $d = 30$ and $d = 45$ still outperforms $d = 0$ for most of the simulation until $d = 0$ converges.

Note that in the figure many of the curves appear to match one of the curves produced by $h < 60$ in figure III-B. However, d can provide useful functionality. Observe the sequence of meetings in table III. Although the value of h is lower in Example A, the algorithm is able to properly label the node more quickly in Example B where $d = 2$. This is because a longer history retains information about previous community membership, more quickly returning to an old community. In contrast, the version with smaller h takes longer to switch membership back to a former community. Additionally note that even if Example A had $h = 8$, it would still fail to switch to the new community B since there are equal numbers of A's and B's in the history. Therefore when deciding whether to utilize d or a smaller h, consider whether nodes take short trips away from their communities.

TABLE III
EXAMPLE HISTORIES GIVEN DIFFERENT PARAMETERS

Total History	B	B	A	A	A	A	B	B
Example A, h=4	B	B	A	A				
Example B, d=2,h=8	B	B	A	A	A	A	B	

D. Relabeling Split Communities

In figure 5 we can see the performance of this algorithm on a community that splits into two communities. Due to the randomness injected by relabeling, the algorithm is never able to completely converge. After the split occurs however, we see that when $rc = h = 30$ relabeling enables the community structure to begin to converge on having two communities rather than one, and errors decrease once more. However when $rc = 15$ the network is unable to converge.

This difference between convergence of the two simulations is because in the case where $rc = 15$ only 15 nodes are notified

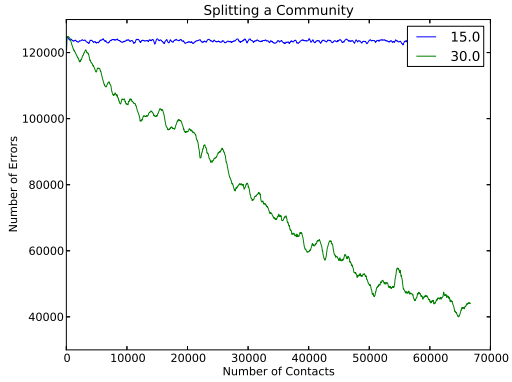


Fig. 5. Splitting a Community. $h = 60$, $d = 1$, relabeling is enabled. $p = 12000$, $rc = h$. Results are averaged over 50 simulations and are in windows of 100 contacts. Results shown immediately after split occurs.

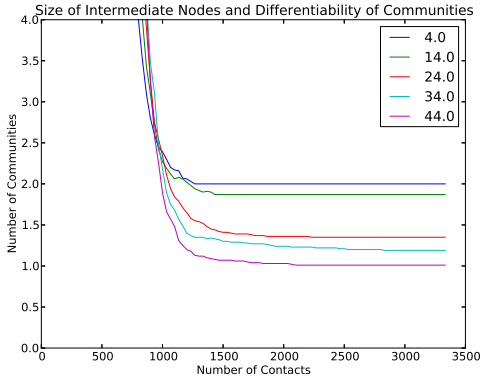


Fig. 6. The Connected Graph. Simulation run on 100 nodes split into two groups of 50 joined by given number of nodes. $h = 5$, $d = 0$ relabeling disabled. Results averaged over 50 simulations in windows of 100 contacts.

of relabeling out of the 50 in the community, so the relabeling is not able to travel through the community. By contrast when $rc = 30$ over $1/2$ of the new community is notified of the new label and the network begins to converge.

These results illustrate the importance of considering the underlying network when choosing the value of rc. A value that is too small will create relabeling attempts which die out too quickly, preventing relabeling. A value that is too large would allow a relabeling to spread outside of a community if the community is still connected to other communities, and so could cause the relabeling to spread outside of the community and create the same problem relabeling is designed to fix– two communities with the same label. In this case, rc greater than $1/2$ of the number of nodes in the community enabled convergence.

E. Shared Nodes and Their Effect on Communities

One example which illustrates the ability of this algorithm to distinguish between communities is an example wherein two groups of nodes share a small number of nodes between them in the Connected network. In figure 6 we see one such network where the number of intermediary nodes range from 4 to 44. The figure reported is the average number of communities over all 50 simulation runs. For instance, if the value is 1.5 then in half of the runs there were 2 communities at that number of contacts and in the other half there was 1 community at that number of contacts. We can see that as the number of intermediary nodes grows, the two groups are more likely to be judged as belonging to the same community.

This result matches up with our definition of community, which states that nodes are likely to share edges with their community members and unlikely to share edges with non-community members. As the number of nodes who have edges connecting both groups grows, the likelihood that both groups belong to the same community grows as well.

However, it is interesting to note that the relation does not appear to be linear. There is a big jump between 14 and 24 where when 14 nodes were shared the communities were likely to be judged as two separate communities, where when 24 nodes were shared communities were more likely to be judged as one community.

F. Finding Communities in a Random Spatial Graph

The algorithm was also run on a random spatial network. In this network, 30 nodes are randomly seeded on a 30x30 unit square. In each iteration an edge is discovered between each node and its three nearest neighbors. Afterward the nodes move a small distance with random probability.

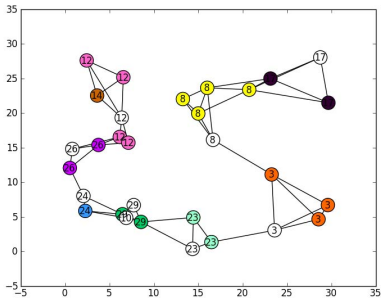
The simulation was run on the generated graph with $h = 20$, $d = 1$, $p = 4000$, relabeling turned on, and $rc = 20$. The resulting network has been graphed in figure III-F. The nodes are all labeled and colored by their community label. White nodes are nodes who have a community label that is the same as their node ID. That is desirable because we would like for the label of a community to be the ID of a node within it in order to ensure that communities have different labels.

In this network colocated groups of nodes should be in the same community. Also there should be many connections within communities and few connections between them. In figure 7(b) we can see that is true for all. [1]. Additionally most communities have a white node within them so most community labels are the ID of a node within that community.

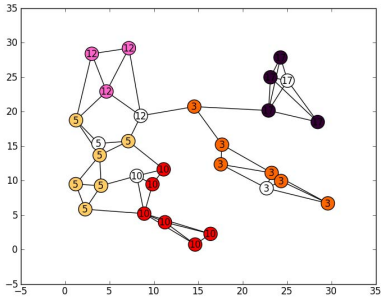
1) Relabeling Event: In figures 7(b) and 7(c) we witness a relabeling event occurring where community 5 becomes two communities, one labeled with 5 and the other labeled with 24. Note that this enables two communities that originally had the same label to become labeled differently.

2) Community Growth and Shrinkage: Communities grow and shrink depending on dynamic changes in the network. In figure 7(c) note that community 12 has grown by one node. In this case community 12 grew while community 5 shrunk.

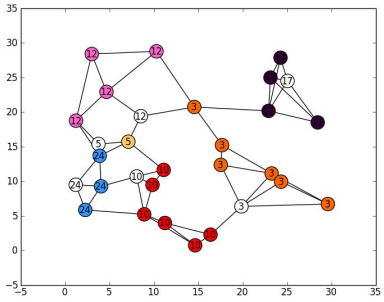
[1]A slideshow of this community detection simulation is available at http://www.divms.uiowa.edu/~vgalluzz/RTDCD.html



(a) Timestep 0



(b) Timestep 62



(c) Timestep 63

Fig. 7. The Moving Graph (selected timesteps). Colors and labels indicate community membership. White color indicates community label is ID.

IV. Conclusion

This paper offers a real time distributed algorithm for estimating community structure. It is well suited for distributed computing, with a runtime that is linear in h, a history length parameter. h is adjustable and therefore the cost of the algorithm can be adjusted to suit the user.

This algorithm was simulated on artificially generated networks and correctly estimates the artificial communities. In addition it finds a reasonable community structure on a randomly generated network.

Finding community structure accurately depends on user adjustment of several key parameters $h, d, p,$ and rc which were explored in simulation. Of these parameters h is the most important since it effects both runtime and convergence time. However, the right value of d can help to improve convergence time and the correct value of rc enables relabeling to occur.

V. Future Work

Future developments could include allowing for multiple community affiliations for each node.

Additionally this algorithm does not allow for cases where one portion of the network changes community affiliation often while another is more stable. Future work includes adjusting h and other parameters on each node to create custom behavior in different portions of the network.

We hope to deploy this algorithm in the University of Iowa Computational Epidemiology group's hospital sensor network in an infection risk detection application as described in the introduction. In addition to detection of individual risk we could also create a community-wide measure of risk that is calculated in a distributed fashion, reflecting the risk brought to community members when other members of their community bring diseases from outside of the community to them.

References

[1] S. Fortunato, "Community detection in graphs," *Physics Reports*, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0370157309002841

[2] C. C. Aggarwal and P. S. Yu, "Online Analysis of Community Evolution in Data Streams," *Proceeding of the 5th SIAM International Conference on Data Mining*, pp. 56–67, 2005. [Online]. Available: http://www.siam.org/proceedings/datamining/2005/dm05_06aggarwalc.pdf

[3] T. Falkowski, J. Bartelheimer, and M. Spiliopoulou, "Mining and Visualizing the Evolution of Subgroups in Social Networks," *Proceedings of the 2006 IEEEWICACM International Conference on Web Intelligence*, pp. 52–58, 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4061341

[4] M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult, "Monic: modeling and monitoring cluster transitions," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, T. Eliassi-Rad, L. H. Ungar, M. Craven, and D. Gunopulos, Eds., vol. 6. ACM, 2006, pp. 706–711. [Online]. Available: http://portal.acm.org/citation.cfm?id=1150491

[5] D. Greene, "Tracking the Evolution of Communities in Dynamic Social Networks," *Methods*, pp. 1–18, 2011.

[6] T. Y. Berger-Wolf and J. Saia, "A framework for analysis of dynamic social networks," *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining KDD 06*, vol. 06, no. 191445, p. 523, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1150402.1150462

[7] T. Falkowski and M. Spiliopoulou, "Community Analysis in Dynamic Social Networks," *Thèse*, pp. 125–150, 2009. [Online]. Available: http://edoc.bibliothek.uni-halle.de/servlets/MCRFileNodeServlet/HALCoRe_derivate_00003267/Tanja_Falkowsk

[8] T. Berger-wolf, C. Tantipathananandh, and D. Kempe, "Link Mining: Models, Algorithms, and Applications," *Networks*, pp. 45–47, 2010. [Online]. Available: http://www.springerlink.com/index/10.1007/978-1-4419-6515-8

[9] T. Herman, S. V. Pemmaraju, A. M. Segre, P. M. Polgreen, and D. E. Curtis, "Wireless Applications for Hospital Epidemiology Categories and Subject Descriptors," pp. 45–50, 2003.

[10] I. X. Y. Leung, P. Hui, P. Liò, and J. Crowcroft, "Towards real-time community detection in large networks," *Physical Review E - Statistical, Nonlinear and Soft Matter Physics*, vol. 79, no. 6 Pt 2, p. 10, 2008. [Online]. Available: http://arxiv.org/abs/0808.2633